

DLA
ABSOLWENTÓW
SZKÓŁ
PODSTAWOWYCH

Informatyka na czasie

Podręcznik dla liceum ogólnokształcącego i technikum

3

Zakres rozszerzony

nowa
era

Droga Nowa Ero,

Nigdy bym nie publikowała publicznie książek wydawnictw, które działają na uczciwych zasadach.

Wasza firma jednak promuje masowy dodruk, całkowicie niepotrzebnych książek, które mogłyby zastąpione wersjami elektronicznymi!

Co prawda e-booki są dostępne na waszej stronie, jednak:

- W przeciwieństwie do fizycznej książki, licencja na e-book kończy się po roku. Oznacza to, że jeżeli moja córka chciałaby powtórzyć sobie całą wiedzę do matury, musiałabym jej kupić wszystkie wasze książki od nowa.
- Waszych e-booków nie da się pobrać! Wymagają one dostęp do internetu, co uniemożliwia ich użycie na naszej wsi, gdzie zasięg jest ograniczony.
- Wasze e-booki nie działają na telefonach komórkowych!!!
- Wasze e-booki sprzedawane są po tej samej (albo wyższej) cenie co regulame książki. Cena e-booka powinna być niższa, gdyż e-booki wymagają elektronicznego czytnika (tabletu)!

Czas rozpocząć nową erę (o ironio), w której papier nie jest beczelnie marnowany dla pieniędzy. Przedstawiam e-book, który spełnia wszystkie oczekiwania uczniów.

Dbajmy o środowisko, zróbmy to dla młodych pokoleń.

Spis treści

1. Rozwiązywanie problemów z wykorzystaniem dynamicznych struktur danych	7
Kartka z historii	8
1. Odwrotna notacja polska (ONP)	10
1.1. Sposoby zapisu wyrażeń algebraicznych	10
1.2. Czym jest stos?	11
1.3. Algorytm zamiany wyrażenia z notacji tradycyjnej na ONP	12
1.4. Implementacja algorytmu zamiany wyrażenia z notacji tradycyjnej na ONP	16
1.5. Obliczanie wartości wyrażenia arytmetycznego zapisanego w ONP	20
2. Znajdowanie drogi wyjścia z labiryntu	28
2.1. Przygotowanie planszy z labiryntem	28
2.2. Rekurencyjny algorytm znajdowania wyjścia z labiryntu	31
2.3. Czym jest kolejka?	39
2.4. Iteracyjny algorytm znajdowania wyjścia z labiryntu	40
3. Wykorzystanie list w rozwiązywaniu problemów	47
3.1. Czym jest lista?	47
3.2. Symulacja problemu Flawiusza	49
3.3. Program symulujący problem Flawiusza	50
3.4. Sortowanie leksykograficzne	53
3.5. Ocena złożoności obliczeniowej algorytmu rozwiązującego problem Flawiusza i algorytmu sortowania leksykograficznego	59
4. Grafy. Znajdowanie najkrótszej drogi	62
4.1. Czym jest graf?	62
4.2. Reprezentacja grafu	64
4.3. Przeszukiwanie grafu w głąb	69
4.4. Przeszukiwanie grafu w szerz	71
4.5. Algorytm Dijkstry	73
Wiesz, umiesz, zdasz	85
2. Algorytmy numeryczne	95
Kartka z historii	96
5. Reprezentacja liczb rzeczywistych w komputerze	98
5.1. Rozwinięcie binarne ułamka właściwego	98
5.2. Reprezentacja stało- i zmiennoprzecinkowa liczb rzeczywistych	103
5.3. Działania arytmetyczne na liczbach zmiennoprzecinkowych	109
6. Błędy w obliczeniach	112
6.1. Źródła błędów w obliczeniach	112
6.2. Błąd bezwzględny i względny	115
6.3. Algorytmy znajdowania pierwiastków równania kwadratowego	115
7. Obliczanie wartości wielomianu	122
7.1. Sformułowanie problemu obliczania wartości wielomianu	122
7.2. Obliczanie wartości wielomianu algorytmem naiwnym	123
7.3. Obliczanie wartości wielomianu za pomocą schematu Hornera	124
7.4. Inne zastosowania schematu Hornera	126

8. Metody obliczeń przybliżonych.....	129
8.1. Znajdowanie miejsc zerowych funkcji	129
8.2. Obliczanie pierwiastka kwadratowego.....	133
8.3. Obliczanie pól obszarów zamkniętych.....	137
8.4. Znajdowanie przybliżenia liczby π	141
8.5. Symulacja ruchów Browna	145
9. Algorytmy badające własności geometryczne	149
9.1. Położenie punktów względem prostej i odcinka	149
9.2. Przycinanie się dwóch odcinków	156
9.3. Przynależność punktu do wielokąta wypukłego	159
10. Fraktale	165
10.1. Zbiór Cantora	165
10.2. Płatek Kocha.....	173
10.3. Drzewo binarne.....	178
10.4. Dywan Sierpińskiego.....	180
10.5. Przekształcenia afiniczne	183
Wiesz, umiesz, zdasz	190
 3. Zaawansowane algorytmy i techniki programistyczne.....	 199
Kartka z historii.....	200
11. Wyszukiwanie wzorca w tekście.....	202
11.1. Algorytm naiwny	202
11.2. Metoda haszowania.....	205
11.3. Algorytm Karpa–Rabina	207
12. Szyfrowanie z kluczem publicznym. Algorytm RSA.....	214
12.1. Algorytm RSA	215
12.2. Program generujący klucz publiczny i klucz prywatny.....	217
12.3. Program szyfrujący informacje kluczem publicznym	219
12.4. Program deszyfrujący kryptogram kluczem prywatnym.....	222
13. Programowanie obiektowe	225
13.1. Podstawowe pojęcia.....	226
13.2. Konstruktor i przeciążanie metod	229
13.3. Hierarchia klas.....	232
13.4. Polimorfizm i metody wirtualne.....	235
Wiesz, umiesz, zdasz	241
Z informatyką w przyszłość.....	250
 4. Relacyjne bazy danych	 251
Kartka z historii.....	252
14. Wprowadzenie do relacyjnych baz danych.....	254
14.1. Projektowanie bazy danych	254
14.2. Od projektu do bazy danych	258
14.3. Formularze – zarządzanie danymi i ich przeglądanie	268
14.4. Kwerendy – wyszukiwanie i łączenie danych.....	270
14.5. Raporty – wizualne przedstawienie danych.....	272

15. Wykorzystanie danych pochodzących z kwerend.....	277
15.1. Łączenie danych za pomocą tabeli pomostowej.....	277
15.2. Parametry i kryteria kwerend wybierających	279
15.3. Agregowanie danych w kwerendach krzyżowych.....	282
16. Podstawy języka SQL	289
16.1. Pojęcia używane w różnych systemach zarządzania bazą danych	290
16.2. Tworzenie i modyfikowanie relacyjnej bazy danych.....	292
16.3. Zabezpieczanie bazy danych	302
17. Zapytania w języku SQL	305
17.1. Zapytania zwracające dane	305
17.2. Zapytania wykorzystujące związki między tabelami.....	312
17.3. Stosowanie języka SQL w kwerendach MS Access.....	316
18. Tworzenie aplikacji korzystającej z sieciowej bazy danych	321
18.1. Projektowanie aplikacji internetowej	321
18.2. Realizacja oprogramowania po stronie front-end	324
18.3. Realizacja oprogramowania po stronie back-end.....	329
Wiesz, umiesz, zdasz	336
Z informatyką w przyszłość	342
Projekt zespołowy	343
 5. Rozwiązywanie różnych problemów z wykorzystaniem komputera	 345
Kartka z historii.....	346
19. Sterujemy robotem	348
19.1. Czym są roboty?	348
19.2. Roboty w edukacji.....	349
19.3. Robot LEGO MINDSTORMS EV3.....	352
19.4. Mikrokontroler BBC micro:bit	354
19.5. Programowanie robotów i symulatory	354
19.6. Symulacja robota EV3 w środowisku Open Roberta Lab.....	356
19.7. Symulacja płytki micro:bit w środowisku Microsoft MakeCode.....	360
19.8. Środowisko do symulacji VEXcode VR.....	362
20. Sztuka publikowania w sieci	368
20.1. Jak zainteresować odbiorców swoimi publikacjami?	368
20.2. Teksty informacyjne.....	373
20.3. Grafika i identyfikacja wizualna.....	375
20.4. Nagrania audio i wideo oraz wystąpienia na żywo	377
20.5. Przygotowanie materiału do publikacji	383
20.6. Jak poznać (lepiej) swojego odbiorcę?	385
21. Grafiki informacyjne	387
21.1. Sposoby przekazywania informacji	387
21.2. Piktogramy	389
21.3. Przykłady infografik	390
21.4. Jak stworzyć poprawną infografikę?	393
21.5. Tworzymy własną infografikę.....	400
Z informatyką w przyszłość	404
Projekt zespołowy	405
Dodatki.....	407
Indeks	437

O czym jest podręcznik?

Podręcznik *Informatyka na czasie 3* pomoże ci pogłębić wiedzę na temat algorytmiki i programowania w języku C++. Narysujesz fraktale z użyciem języka JavaScript, nauczysz się korzystać z baz danych, a także poznasz podstawy robotyki, przygotujesz treści do publikacji w sieci i utworzysz infografiki.



Czemu służą poszczególne elementy podręcznika?

Cele lekcji

Informacje na temat tego, czym będziesz się zajmować na danej lekcji.

Ćwiczenie 1

Proste zadania w toku lekcji. Odpowiedzi nie należy zapisywać w podręczniku.

Zadania

Zestaw poleceń o różnym stopniu trudności. Odpowiedzi nie należy zapisywać w podręczniku.

Dobra rada

Podpowiedzi ułatwiające wykonanie omawianych czynności.

Dobra rada

Wskazówki szczególnie przydatne w kształceniu kluczowych umiejętności.

A to ciekawe

Ciekawostki pokazujące omawiane zagadnienia z zaskakującej perspektywy.

Warto wiedzieć

Informacje poszerzające wiedzę o omawianych zagadnieniach.

Podsumowanie

Najważniejsze informacje z lekcji podane w punktach, porządkujące nowo zdobytą wiedzę.

Projekt zespołowy

Zadanie do wykonania w grupie, z wykorzystaniem wiedzy z kilku rozdziałów.

Kartka z historii

Wydarzenia najważniejsze dla rozwoju danej dziedziny informatyki.

Zapamiętaj

Najważniejsze pojęcia omawiane na lekcji, które należy zapamiętać.

Z informatyką w przyszłość

Przykłady zastosowania treści omawianych w podręczniku w przyszłym życiu zawodowym.

Oznaczenia w podręczniku

📄 Odsyłacz do zagadnienia omawianego szerzej w innym miejscu

🔗 Wskaźnik pojęcia; pojęcia obok wskaźników układają się w siatkę pojęć z danej lekcji

★★★ Oznaczenie stopnia trudności zadań



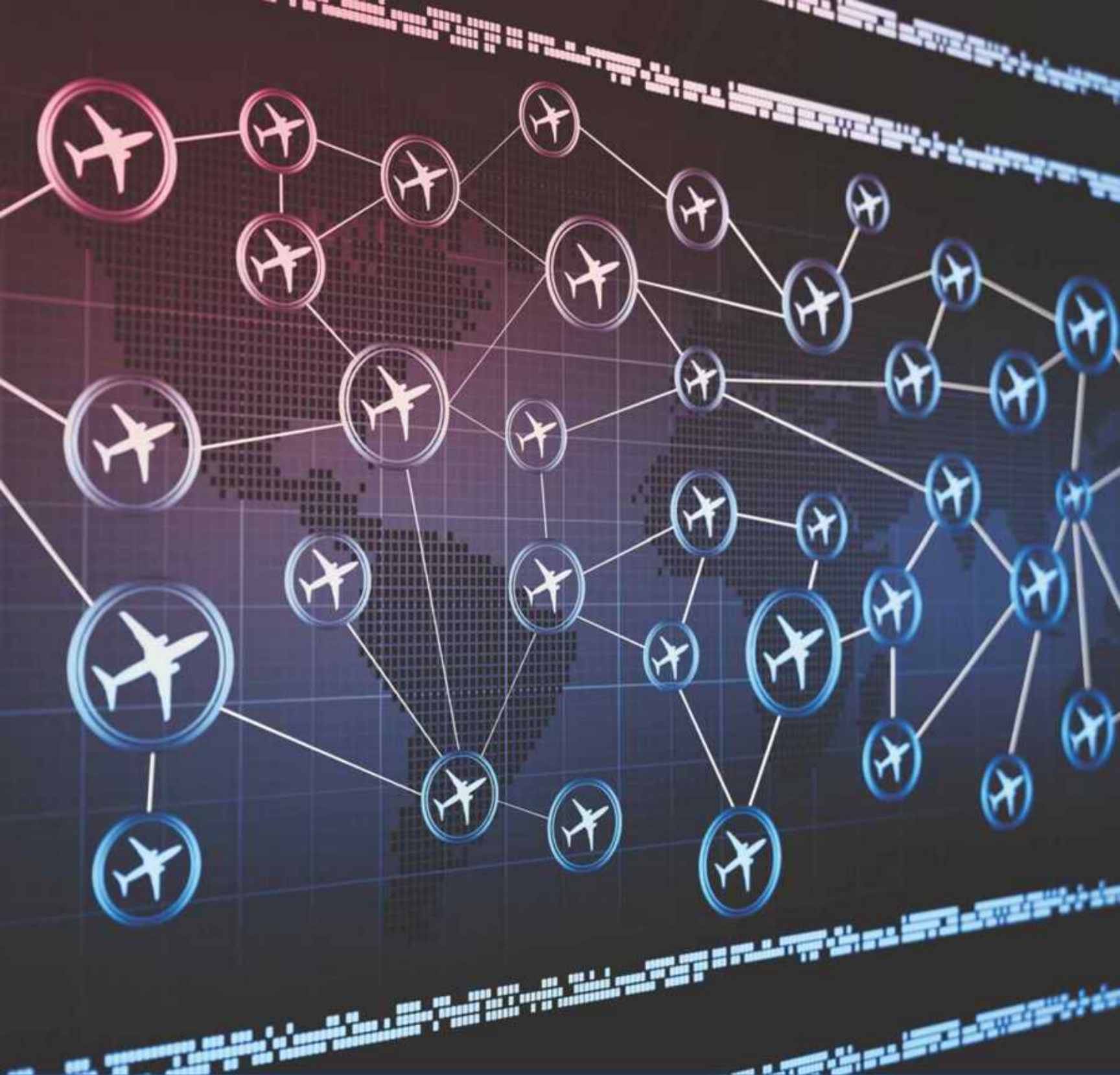
WIESZ, UMIESZ, ZDASZ

Sposób na zadania

Zadania rozwiązane krok po kroku, które ułatwiają opanowanie różnych typów zadań.

Trening

Zestaw zadań pozwalających utrwalić wiedzę z danego rozdziału.



1 Rozwiązywanie problemów z wykorzystaniem dynamicznych struktur danych

1. Odwrotna notacja polska (ONP)
 2. Znajdowanie drogi wyjścia z labiryntu
 3. Wykorzystanie list w rozwiązywaniu problemów
 4. Grafy. Znajdowanie najkrótszej drogi
- Wiesz, umiesz, zdasz

Algorytmika i programowanie



W 1952 r. Amerykanka Grace Hopper opracowała program tłumaczący polecenia zapisane jako słowa języka angielskiego na binarny kod maszynowy komputera. Nazwała go A-0. Program ten uznaje się za pierwszy **kompilator**. Jego udoskonalona wersja A-2 była pierwszym powszechnie stosowanym kompilatorem. Hopper znana jest także jako programistka komputerów Harvard Mark I i Harvard Mark II. Przypisuje się jej m.in. spopularyzowanie w słownictwie informatycznym pojęcia „bug”, które oznacza błąd oprogramowania.

1952 r.

W 1954 r. zespół badawczy firmy IBM pod kierownictwem Johna Backusa zaprojektował język programowania **FORTAN** (od ang. *FORmula TRANslation*). Język ten pierwotnie opracowano dla komputerów IBM 704 (na zdjęciu), a następnie udoskonalano. Do lat 80. XX w. był podstawowym językiem używanym do obliczeń numerycznych. Stosuje się go nadal, np. w programowaniu superkomputerów wykorzystywanych do przeprowadzania symulacji. FORTRAN jest uważany za pierwszy język programowania wysokiego poziomu.



1954 r.

1953 r.

Hans Peter Luhn, naukowiec pochodzenia niemieckiego, który pracował w IBM, jest uważany za autora **metody haszowania**. Ułatwia ona wyszukiwanie informacji w wielkich zbiorach danych. W 1953 r. Luhn zaproponował funkcję matematyczną przyporządkującą argumentowi pewną wartość. Wyszukanie informacji w dużo mniejszym zbiorze danych o tej samej wartości funkcji było o wiele szybsze. Luhn jest też twórcą algorytmu wykorzystywanego do sprawdzania poprawności ciągów liczb. Algorytmu Luhna używa się np. do weryfikacji numeru karty kredytowej albo numeru IMEI.



1956 r.



W 1956 r. holenderski naukowiec Edsger Dijkstra wymyślił algorytm, który wyznacza najmniejsze koszty dojścia z wybranego wierzchołka grafu do pozostałych wierzchołków oraz wskazuje, przez które wierzchołki prowadzą drogi o najmniejszych kosztach. Algorytm został nazwany od nazwiska jego twórcy **algorytmem Dijkstry**. Edsger Dijkstra brał udział też m.in. w opracowywaniu języka ALGOL. Za swoje osiągnięcia otrzymał w 1972 r. nagrodę Turinga.



W 1958 r. John McCarthy, amerykański informatyk z MIT, stworzył język programowania **LISP** (od ang. *LISt Processing*), ukierunkowany na rekurencyjne przetwarzanie struktur listowych. Pierwsza implementacja języka powstała dla komputera IBM 704. LISP był używany przez naukowców pracujących nad sztuczną inteligencją. To właśnie John McCarthy jest autorem terminu *artificial intelligence*. Za wkład w rozwój sztucznej inteligencji otrzymał w 1971 r. nagrodę Turinga.

1958 r.

W 1959 r. z inicjatywy rządu USA i grupy amerykańskich firm powstał język przeznaczony dla biznesu, niezależny od architektury konkretnej maszyny. Językiem tym był **COBOL** (od ang. *COmmon Business-Oriented Language*). COBOL służył do wykonywania obliczeń administracyjno-ekonomicznych i przetwarzania dużych zbiorów danych. Jego składnia jest zbliżona do składni języka naturalnego.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO-WORLD.
*
ENVIRONMENT DIVISION.
*
DATA DIVISION.
*
PROCEDURE DIVISION.
PARA-1.
    DISPLAY "Hello, world.".
*
EXIT PROGRAM.
END PROGRAM HELLO-WORLD.
```

1959 r.

1959 r.

W 1959 r. angielski informatyk Charles Antony Richard Hoare stworzył jeden z najpopularniejszych algorytmów sortowania oparty na metodzie „dziel i zwyciężaj” – znany pod nazwą **QuickSort**. Wymyślił go, gdy pracował w Moskwie nad koncepcją maszynowego przekładu tekstu z języka rosyjskiego na angielski. Uczony znany jest także dzięki tzw. logice Hoare’a, służącej do opisu poprawności algorytmów. Hoare został uhonorowany nagrodą Turinga (w 1980 r.) oraz tytułem doktora honoris causa m.in. Uniwersytetu Warszawskiego.



1960 r.



W 1960 r. międzynarodowy zespół naukowców, w skład którego wchodził m.in. Duńczyk Peter Naur i Amerykanin John Backus (na zdjęciu), opracował język **ALGOL 60** (od ang. *ALGOrithmic Language*). Jest on uważany za pierwszy język ogólnego przeznaczenia, niepowiązany z architekturą konkretnego komputera. Oznacza to, że programy napisane w tym języku można było przenosić między komputerami. ALGOL 60 był pierwowzorem wielu innych języków programowania, np. języka Pascal.

1. Odwrotna notacja polska (ONP)

W znanym ci sposobie zapisywania wyrażeń algebraicznych znaki działań umieszcza się pomiędzy argumentami, a o kolejności wykonywania obliczeń decydują nawiasy oraz to, jak wysoki priorytet ma dane działanie. Nie jest to jednak jedyny sposób przedstawiania wyrażeń algebraicznych. Istnieją notacje, w których nie używa się nawiasów, a mimo to można właściwie określić kolejność wykonywania działań. Jedną z nich jest odwrotna notacja polska.

Cele lekcji

- Poznasz sposoby zapisu wyrażeń algebraicznych bez użycia nawiasów, w tym odwrotną notację polską.
- Dowiesz się, czym są dynamiczne struktury danych.
- Poznasz i zastosujesz dynamiczną strukturę danych o nazwie stos.
- Zamienisz wyrażenie algebraiczne zapisane w notacji tradycyjnej na zapis w odwrotnej notacji polskiej.
- Wykorzystasz odwrotną notację polską do obliczenia wartości wyrażenia arytmetycznego.

1.1. Sposoby zapisu wyrażeń algebraicznych

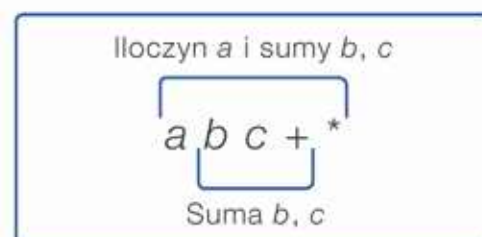
Stosowany powszechnie zapis wyrażeń algebraicznych, w którym znak działania występuje pomiędzy argumentami, to **notacja infiksowa**. Na przykład sumę a i b w tej notacji zapisujemy: $a + b$.

- Innym sposobem zapisywania wyrażeń algebraicznych jest
- Notacja prefiksowa** (notacja polska) • **notacja prefiksowa**, zwana też **notacją polską**. W jej przypadku znak działania stawia się przed argumentami, których dotyczy. Suma a i b będzie wyglądała następująco: $+ a b$. Znak działania można też postawić za argumentami: $a b +$. Ten sposób zapisu nazywa się
- Notacja sufiksowa** (odwrotna notacja polska – ONP) • **notacją sufiksową** lub **odwrotną notacją polską (ONP)**. W notacjach prefiksowej i sufiksowej nie używa się nawiasów, a mimo to można określić kolejność wykonywania działań zgodnie z ich priorytetami.

Sposób interpretacji wyrażenia algebraicznego zapisanego w odwrotnej notacji polskiej (notacji sufiksowej) ilustruje rysunek 1.1.

Warto wiedzieć

Notację polską wymyślił polski matematyk Jan Łukasiewicz. Odwrotna notacja polska została zaproponowana przez australijskiego naukowca C.L. Hamblina i nazwana odwrotną notacją polską jako wyraz uznania dla osiągnięć Łukasiewicza.



Rys. 1.1. Wyrażenie algebraiczne $a * (b + c)$ przedstawione w odwrotnej notacji polskiej (notacji sufiksowej)

Na wyrażenia zapisane w odwrotnej notacji polskiej należy patrzeć od prawej do lewej. Wyrażenie algebraiczne $a b c + *$ z rysunku 1.1 oznacza iloczyn, którego pierwszym czynnikiem jest a , natomiast drugim suma b i c ($b c +$ oznacza sumę b i c , ponieważ znak $+$ znajduje się bezpośrednio za b i c). Zatem zapis $a b c + *$ w odwrotnej notacji polskiej odpowiada wyrażeniu $a * (b + c)$ w notacji tradycyjnej.

Tabela 1.1 przedstawia przykłady wyrażen algebraicznych w notacji tradycyjnej i odwrotnej notacji polskiej.

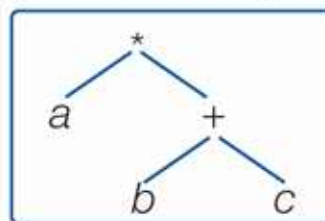
Warto wiedzieć

Odwrotna notacja polska była stosowana w niektórych kalkulatorach.

Notacja tradycyjna (infiksowa)	Odwrotna notacja polska (notacja sufiksowa)
$a + b * c$	$a b c * +$
$a * b + c$	$a b * c +$
$(a - b) * c$	$a b - c *$
$a / (b + c)$	$a b c + /$

Tabela 1.1. Przykłady wyrażen algebraicznych zapisanych w notacji tradycyjnej i odwrotnej notacji polskiej

Na podstawie zapisu w odwrotnej notacji polskiej można łatwo przedstawić wyrażenie w postaci graficznej, jak na rysunku 1.2. Taki sposób zapisu nazywamy **drzewem wyrażenia algebraicznego**.



Rys. 1.2. Drzewo wyrażenia algebraicznego $a b c + *$

Drzewo wyrażenia algebraicznego

Ćwiczenie 1

Zapisz wyrażenie $(a + b) * (c + d)$ w odwrotnej notacji polskiej oraz utwórz drzewo tego wyrażenia algebraicznego.

1.2. Czym jest stos?

W algorytmie zamieniającym wyrażenia z notacji tradycyjnej na ONP wykorzystamy strukturę danych o nazwie **stos** (ang. *stack*). Można ją porównać do stosu książek (rys. 1.3). Kolejną książkę dokłada się tylko na wierzch stosu i zdjąć można jedynie książkę leżącą na wierzchu. Podobnie w strukturze danych ostatnio dodany element jest wierzchołkiem stosu, a usunąć można tylko element będący wierzchołkiem. Na stosie można przecho-



Rys. 1.3. Stos książek

wywać wyłącznie dane tego samego typu. Jest on strukturą danych, z której pobieramy elementy w kolejności odwrotnej do ich wstawiania. Taką strategię dostępu nazywa się **LIFO** (od ang. *last in, first out* – ostatni wchodzi, pierwszy wychodzi).

Operacje na stosie • Możemy wyróżnić następujące **operacje na stosie**:

- ▶ *push* – umieszczenie elementu na stosie,
- ▶ *pop* – usunięcie elementu będącego wierzchołkiem stosu,
- ▶ *top* – odczytanie wartości elementu będącego wierzchołkiem stosu,
- ▶ *empty* – sprawdzenie, czy stos jest pusty.

Dynamiczna struktura danych • Stos jest przykładem **dynamicznej struktury danych**, czyli takiej, która może zmieniać swój rozmiar podczas działania programu. Liczbę elementów pamiętanych w dynamicznej strukturze danych można zwiększać lub zmniejszać w zależności od potrzeb.

Dobra rada

Jeśli nie wiesz, jak wymawiać obcojęzyczne nazwy umieszczone w podręczniku, skorzystaj z translatora, który daje możliwość odsłuchania tekstu, np. Tłumacza Google lub Bing Microsoft Translator.

Zapamiętaj

Stos jest dynamiczną strukturą danych, czyli taką, której rozmiar można zmieniać w trakcie działania programu. Operacje wykonuje się tylko na jednym końcu stosu. Do operacji tych należą: umieszczenie elementu na stosie (*push*), zdjęcie elementu ze stosu (*pop*), odczytanie wartości elementu będącego wierzchołkiem stosu (*top*) i sprawdzenie, czy stos jest pusty (*empty*).

1.3. Algorytm zamiany wyrażenia z notacji tradycyjnej na ONP

W algorytmie zamiany wyrażenia algebraicznego zapisanego w notacji tradycyjnej na zapis w odwrotnej notacji polskiej ograniczymy się do wyrażeń, w których mogą występować tylko:

- ▶ podstawowe działania algebraiczne dwuargumentowe – dodawanie, odejmowanie, mnożenie i dzielenie,
- ▶ dowolnie zagnieżdżone nawiasy okrągłe,
- ▶ jednoznakowe argumenty (litery, liczby jednocyfrowe).

Zakładamy także, że wyrażenie w notacji tradycyjnej jest zapisane bez błędów. Oto specyfikacja problemu:

Specyfikacja

Dane: *w* – napis reprezentujący wyrażenie algebraiczne w notacji tradycyjnej, w którym mogą występować znaki +, −, *, / jako znaki działań dwuargumentowych, nawiasy okrągłe oraz litery i liczby jednocyfrowe jako argumenty jednoznakowe.

Wynik: *onp* – napis reprezentujący wyrażenie zapisane w odwrotnej notacji polskiej.

Algorytm zamiany będzie polegał na przeglądaniu wyrażenia zapisanego w notacji tradycyjnej od lewej do prawej i w zależności od rodzaju napotkanego znaku – podejmowaniu odpowiedniej decyzji.

Niektóre znaki podczas analizy wyrażenia wejściowego trzeba przechować przez pewien czas. Dotyczy to znaków działań (do czasu rozpatrzenia drugiego argumentu) oraz nawiasu otwierającego (do czasu napotkania w wyrażeniu odpowiadającego mu nawiasu zamykającego). Dane te przechowujemy na stosie. Sposób postępowania ilustrują tabele 1.2 i 1.3 (s. 14) na przykładzie wyrażeń $a + b * c$ oraz $a - (b + c) * d + e$. W ostatniej kolumnie każdej tabeli elementy stosu zapisane są w kolejności od dna stosu do wierzchołka stosu (wierzchołkiem stosu jest pierwszy znak od prawej).

Tabela 1.2 przedstawia kolejne kroki prowadzące do zamiany wyrażenia $a + b * c$ na zapis w ONP. W kroku 4 znak mnożenia jest wkładany na stos po sprawdzeniu, jaki znak jest wierzchołkiem stosu. Jest to znak dodawania, a więc działania o priorytecie niższym od mnożenia. Gdyby było to działanie o priorytecie wyższym lub równym, przed włożeniem na stos należałoby znak takiego działania zdjąć ze stosu i dopisać do wyrażenia wynikowego. Po przeanalizowaniu wszystkich znaków wyrażenia wejściowego należy znaki działań pozostałe na stosie dopisać do wyrażenia wynikowego (kroki 6 i 7).

Krok	Rozpatrywany znak	Operacja/operacje	Wyrażenie w ONP	Elementy na stosie
1.	a	Dopisanie do ONP	a	
2.	$+$	Włożenie na stos	a	$+$
3.	b	Dopisanie do ONP	$a b$	$+$
4.	$*$	Włożenie na stos	$a b$	$+ *$
5.	c	Dopisanie do ONP	$a b c$	$+ *$
6.		Zdjęcie ze stosu i dopisanie do ONP znaku $*$	$a b c *$	$+$
7.		Zdjęcie ze stosu i dopisanie do ONP znaku $+$	$a b c * +$	

Warto wiedzieć

Drzewo wyrażenia algebraicznego łatwiej jest utworzyć na podstawie zapisu wyrażenia w ONP niż na podstawie zapisu w notacji tradycyjnej.

Tabela 1.2. Kolejne kroki podczas zamiany wyrażenia $a + b * c$ na zapis w ONP

Tabela 1.3 na s. 14 przedstawia kolejne kroki prowadzące do zamiany wyrażenia $a - (b + c) * d + e$ na zapis w ONP. W kroku 7 ze stosu zdejmujemy znak działania, który znajduje się nad nawiasem otwierającym. Znak ten dopisujemy do wyrażenia wynikowego, następnie zdejmujemy ze stosu nawias otwierający i go gubimy. W ten sposób zapiszemy w odwrotnej notacji polskiej fragment wyrażenia ujęty w nawiasy. W kroku 10, po napotkaniu w wyrażeniu znaku $+$, ze stosu zdejmujemy dwa znaki – jeden o priorytecie wyższym niż napotkany znak, drugi o priorytecie takim samym co napotkany znak – i dopisujemy je do wyrażenia wynikowego. Następnie znak $+$ wkładamy na stos.

Ponieważ w wyrażeniu wejściowym nie ma już znaków do rozpatrzenia, zdejmujemy znak + ze stosu i dopisujemy go do wyrażenia wynikowego (krok 12).

Krok	Rozpatrywany znak	Operacja/operacje	Wyrażenie w ONP	Elementy na stosie
1.	a	Dopisanie do ONP	a	
2.	$-$	Włożenie na stos	a	$-$
3.	$($	Włożenie na stos	a	$-($
4.	b	Dopisanie do ONP	$a b$	$-($
5.	$+$	Włożenie na stos	$a b$	$-(+$
6.	c	Dopisanie do ONP	$a b c$	$-(+$
7.	$)$	Zdjęcie ze stosu i dopisanie do ONP znaku + Zdjęcie ze stosu znaku $($	$a b c +$	$-$
8.	$*$	Włożenie na stos	$a b c +$	$- *$
9.	d	Dopisanie do ONP	$a b c + d$	$- *$
10.	$+$	Zdjęcie ze stosu i dopisanie do ONP znaku * Zdjęcie ze stosu i dopisanie do ONP znaku - Włożenie na stos znaku +	$a b c + d * -$	$+$
11.	e	Dopisanie do ONP	$a b c + d * - e$	$+$
12.		Zdjęcie ze stosu i dopisanie do ONP znaku +	$a b c + d * - e +$	

Tabela 1.3. Kolejne kroki podczas zamiany wyrażenia $a - (b + c) * d + e$ na zapis w ONP

Tabela 1.4 zawiera zestawienie operacji wykonywanych w algorytmie zamiany wyrażenia algebraicznego z notacji tradycyjnej na odwrotną notację polską w zależności od napotkanego znaku.

Rozpatrywany znak	Operacja/operacje
Litera/liczba jednocyfrowa	Dopisanie litery/liczby jednocyfrowej do wyniku
$($	Włożenie nawiasu na stos
$)$	Zdjęcie ze stosu znaków działań do napotkania na stosie nawiasu otwierającego i dopisanie tych znaków działań do wyniku Zdjęcie ze stosu nawiasu otwierającego
Znak działania	Zdjęcie ze stosu znaków działań o priorytecie wyższym lub równym priorytetowi napotkanego znaku działania i dopisanie tych znaków działań do wyniku (jeśli takie znaki są) Włożenie rozpatrywanego znaku działania na stos
Koniec wyrażenia	Zdjęcie ze stosu i dopisanie do wyniku znaków działań, które pozostały na stosie

Tabela 1.4. Operacje w algorytmie zamiany wyrażenia algebraicznego z postaci tradycyjnej na zapis w ONP

Ćwiczenie 2

Zapisz kolejne kroki zamiany podanego wyrażenia na zapis w ONP.

- a. $a \cdot b + c$
- b. $a \cdot (b + c)$
- c. $(a + b) \cdot (c + d)$
- d. $a + b \cdot c - d$

Algorytm zamiany wyrażenia algebraicznego z notacji tradycyjnej na odwrotną notację polską, zgodny ze specyfikacją podaną na s. 12, można zapisać w pseudokodzie następująco:

```

onp ← ""
dla i ← 0, ..., długość w - 1 wykonuj
    jeśli w[i] = '(' to
        push(w[i])
        przejdź do następnego powtórzenia pętli
    jeśli w[i] = ')' to
        dopóki top() ≠ '(' wykonuj
            onp ← onp + top()
            pop()
        pop() // zdjęcie ze stosu znaku (
        przejdź do następnego powtórzenia pętli
    jeśli w[i] = '+' lub w[i] = '-' to
        dopóki nie empty() oraz top() ≠ '(' wykonuj
            onp ← onp + top()
            pop()
        push(w[i])
        przejdź do następnego powtórzenia pętli
    jeśli w[i] = '*' lub w[i] = '/' to
        jeśli nie empty() oraz
            (top() = '*' lub top() = '/') to
            onp ← onp + top()
            pop()
        push(w[i])
    w przeciwnym przypadku onp ← onp + w[i]
dopóki nie empty() wykonuj
    onp ← onp + top()
    pop()

```

Zakładamy, że mamy do dyspozycji **stos** oraz **operacje na stosie**. Zgodnie ze specyfikacją w wyrażeniu mogą wystąpić tylko cztery działania: dwa o niższym priorytecie i dwa o wyższym. Napotkanie dodawania lub odejmowania powoduje zdjęcie ze stosu i dopisanie do wyniku działań znajdujących się na stosie aż do napotkania ewentualnego nawiasu otwierającego (ponieważ nie ma działań o niższym priorytecie). Natomiast napotkanie mnożenia lub dzielenia powoduje zdjęcie ze stosu i dopisanie do wyniku tylko innego mnożenia lub dzielenia (ponieważ nie ma działań o wyższym priorytecie).

Stos,
s. 11 [🔗](#)

Operacje na stosie,
s. 12 [🔗](#)

Jeśli napotkany znak nie jest znakiem działania ani nawiasem, to musi być argumentem. Dopisanie argumentu do wyniku realizuje instrukcja podana jako przeciwny przypadek do ostatniej instrukcji **jeśli**.

Ćwiczenie 3

Zamień poniższe wyrażenia na zapis w odwrotnej notacji polskiej zgodnie z omówionym algorytmem zapisanym w pseudokodzie.

a. $((a + b) \cdot c - d) \cdot e + f \cdot g$

b.
$$\frac{a \cdot (b + c) - d \cdot (e + f)}{g + h}$$

1.4. Implementacja algorytmu zamiany wyrażenia z notacji tradycyjnej na ONP

Zaimplementujemy teraz w języku C++ omówiony algorytm zamiany wyrażenia algebraicznego z notacji tradycyjnej na odwrotną notację polską. Wykorzystamy w tym celu szablon stosu z **biblioteki STL** (ang. *Standard Template Library* – standardowa biblioteka szablonów). Aby móc skorzystać z szablonu, na początku programu należy dołączyć bibliotekę **stack** – dyrektywa `#include <stack>`. Ogólna

Deklaracja stosu • **deklaracja stosu** jest następująca:

```
stack<typ elementów stosu> nazwa_stosu;
```

W naszym programie elementami przechowywanymi na stosie będą znaki, a więc deklaracja zmiennej typu **stack** o nazwie `stos` będzie miała postać:

```
stack<char> stos;
```

Metody push, pop, top, empty dla klasy **stack**

Operacje na stosie,
s. 12 

Typ **stack** jest typem obiektowym, zatem udostępnia metody do przetwarzania danych. Należą do nich m.in. **metody** push, pop, top, **empty**. Ich działanie odpowiada wymienionym w tym temacie **operacjom na stosie**. Parametrem metody push jest element, który chcemy wstawić na stos. Metody top, pop, **empty** są bezparametrowe. Korzystając z metody, stosujemy notację z kropką – po nazwie stosu zapisujemy kropkę i nazwę metody, np. `stos.top()`.

Odwołanie do elementu stosu, gdy stos jest pusty, jest błędem. Żeby przy odwołaniu do wierzchołka stosu nie sprawdzać, czy stos jest pusty, wprowadzimy **wartownika**. Będzie on pełnił funkcję dna stosu. Za wartownika przyjmiemy znak, który nie może wystąpić w wyrażeniu wejściowym (np. znak #).

Wartownik to element ułatwiający odnalezienie końca danych.

Kod źródłowy funkcji realizującej algorytm zamiany wyrażenia algebraicznego z postaci tradycyjnej na odwrotną notację polską, zapisany wcześniej w pseudokodzie, może wyglądać następująco.

```

1. string ONP(string w)
2. {
3.     stack<char> stos;
4.     stos.push('#');
5.     string onp="";
6.     for (int i=0;i<w.size();i++)
7.     {
8.         if (w[i]=='(')
9.         {
10.            stos.push('(');
11.            continue;
12.        }
13.        if (w[i]==')')
14.        {
15.            while (stos.top()!='(')
16.            {
17.                onp=onp+stos.top();
18.                stos.pop();
19.            }
20.            stos.pop();
21.            continue;
22.        }
23.        if (w[i]=='+' || w[i]=='-')
24.        {
25.            while (stos.top()!='#' && stos.top()!='(')
26.            {
27.                onp=onp+stos.top();
28.                stos.pop();
29.            }
30.            stos.push(w[i]);
31.            continue;
32.        }
33.        if (w[i]=='*' || w[i]=='/')
34.        {
35.            if (stos.top()=='*' || stos.top()=='/')
36.            {
37.                onp=onp+stos.top();
38.                stos.pop();
39.            }
40.            stos.push(w[i]);
41.        }
42.        else onp=onp+w[i];
43.    }
44.    while (stos.top()!='#')
45.    {
46.        onp=onp+stos.top(); stos.pop();
47.    }
48.    stos.pop();
49.    return onp;
50. }

```

• Kod źródłowy funkcji zamieniającej wyrażenie algebraiczne z postaci tradycyjnej na odwrotną notację polską (z wielokrotnym wykorzystaniem instrukcji warunkowej)

Dobra rada

Kiedy korzystasz z typu `stack`, pamiętaj o dołączeniu do programu biblioteki `stack`.

W linii 3 została zadeklarowana zmienna `stos`, a w linii 4 wstawiono na `stos` znak `#` jako dno stosu (wartownik). W linii 5 zadeklarowana jest zmienna `onp` i nadana jest jej wartość – napis pusty. W liniach 6–43 znajduje się główna pętla przeglądająca kolejne znaki wyrażenia algebraicznego i w zależności od rozpatrywanego znaku wykonująca operacje zgodnie z omówionym algorytmem. Po rozpoznaniu znaku i wykonaniu odpowiadających mu czynności należy przejść do analizy kolejnego znaku, pomijając pozostałe instrukcje w pętli. Aby to zrobić,

Instrukcja `continue` • wykorzystaliśmy **instrukcję `continue`**, która powoduje pominięcie instrukcji do końca pętli i przejście do następnego powtórzenia pętli. Pętla w liniach 44–47 zdejmuję ze `stosu` i dopisuje do wyniku znaki działań, które zostały na `stosie` po zakończeniu analizy wyrażenia algebraicznego. W linii 48 ze `stosu` usuwany jest wartownik (`stos` będzie pusty).

Rysunek 1.4 przedstawia przykład wywołania programu wykorzystującego funkcję `ONP`, przedstawioną na s. 17, i zamieniającego zapis wyrażenia z postaci tradycyjnej na odwrotną notację polską.

Warto wiedzieć

Zamiast korzystać z instrukcji `continue`, można użyć instrukcji: `if ...`
`else if ...`

Podaj wyrażenie: $(a+b)/c-(b+d)$
ONP: `ab+c/bd+-`

Rys. 1.4. Przykład wywołania programu zamieniającego wyrażenie zapisane w postaci tradycyjnej na zapis w ONP

Ćwiczenie 4

Napisz program zamieniający wyrażenie zapisane w notacji tradycyjnej na zapis w odwrotnej notacji polskiej zgodnie ze specyfikacją podaną na s. 12.

Instrukcja wyboru • W głównej pętli funkcji `ONP` (pętli `for`) zamiast wielokrotnie używać instrukcji warunkowej, można skorzystać z **instrukcji wyboru `switch`**. Składnia instrukcji `switch` jest następująca:

```
switch (zmienna)
{
    case wartość_1:
        // instrukcje, gdy zmienna==wartość_1
        break;
    case wartość_2:
        // instrukcje, gdy zmienna==wartość_2
        break;
    ...
    case wartość_n:
        // instrukcje, gdy zmienna==wartość_n
        break;
    default:
        // instrukcje, gdy zmienna nie przyjmuje
        // żadnej z wymienionych wcześniej wartości
}
```

Aby skorzystać z instrukcji **switch**, wartości, jakie może przyjąć zmienna, muszą być typu całkowitoliczbowego. W zależności od wartości zmiennej wykonywana jest jedna z części kodu zaczynających się od słowa kluczowego **case**. Na przykład gdy zmienna przyjmie wartość `wartość_2`, program wykona instrukcje od części rozpoczynającej się od **case** `wartość_2`. Część **default** jest opcjonalna (może nie wystąpić) – podobnie jak część **else** w instrukcji warunkowej **if**. Dzięki **instrukcji break** można przerwać wykonywanie kolejnych instrukcji zapisanych w **switch**. Gdyby zabrakło instrukcji **break**, po wykonaniu instrukcji dla danej wartości zmiennej program przeszedłby do kolejnych linii w instrukcji **switch**. Zauważ, że jeśli nie wystąpi część **default**, nie ma potrzeby używania instrukcji **break** w części z ostatnim **case**. Instrukcja **break** ma także zastosowanie w pętlach – dzięki niej można zatrzymać wykonywanie instrukcji w pętli i wyjść z pętli.

Fragment kodu źródłowego funkcji ONP z wykorzystaniem instrukcji **switch** (zamiast linii 7–43) może wyglądać tak jak poniżej. Zauważ, że zmienną jest znak, któremu w języku C++ odpowiada kod ASCII. Zmienna przyjmuje zatem wartości, które są liczbami całkowitymi.

```

1.  switch (w[i])
2.  {
3.      case '(':
4.          stos.push('('); break;
5.      case ')':
6.          while (stos.top()!='(')
7.          {
8.              onp=onp+stos.top(); stos.pop();
9.          }
10.         stos.pop(); break;
11.     case '+':
12.         while (stos.top()!='#' && stos.top()!='(')
13.         {
14.             onp=onp+stos.top(); stos.pop();
15.         }
16.         stos.push('+'); break;
17.     case '-':
18.         while (stos.top()!='#' && stos.top()!='(')
19.         {
20.             onp=onp+stos.top(); stos.pop();
21.         }
22.         stos.push('-'); break;
23.     case '*':
24.         if (stos.top()=='*' || stos.top()=='/')
25.         {
26.             onp=onp+stos.top(); stos.pop();
27.         }
28.         stos.push('*'); break;

```

● Fragment kodu źródłowego funkcji zamieniającej wyrażenie algebraiczne z postaci tradycyjnej na odwrotną notację polską (z wykorzystaniem instrukcji **switch**)

Warto wiedzieć

Stos w programie zamieniającym wyrażenie z notacji tradycyjnej na ONP można też zaimplementować, wykorzystując do tego zmienną typu `string`.

```

29.     case '/':
30.         if (stos.top()=='*' || stos.top()=='/')
31.         {
32.             onp=onp+stos.top(); stos.pop();
33.         }
34.         stos.push('/'); break;
35.     default:
36.         onp=onp+w[i];
37. }
```

Ćwiczenie 5

Napisz program zamieniający wyrażenie zapisane w notacji tradycyjnej na zapis w odwrotnej notacji polskiej z wykorzystaniem instrukcji wyboru `switch`.

Zapamiętaj

Do zamiany wyrażenia z notacji tradycyjnej na odwrotną notację polską wykorzystuje się strukturę danych o nazwie `stos`. Aby wybrać jeden przypadek z wielu, można wielokrotnie zastosować instrukcję warunkową `if` lub skorzystać z instrukcji wyboru `switch`.

1.5. Obliczanie wartości wyrażenia arytmetycznego zapisanego w ONP

Zajmiemy się teraz problemem obliczania wartości wyrażenia zapisanego w odwrotnej notacji polskiej zgodnie z poniższą specyfikacją.

Specyfikacja

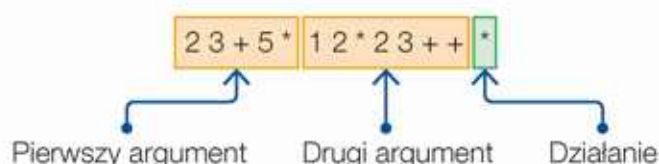
Dane: `onp` – napis reprezentujący wyrażenie arytmetyczne w odwrotnej notacji polskiej, w którym mogą występować znaki `+`, `-`, `*`, `/` jako znaki działań dwuargumentowych oraz liczby jednocyfrowe. Znak `/` oznacza wyznaczanie części całkowitej z dzielenia.

Wynik: `wart` – liczba całkowita będąca wartością wyrażenia `onp`.

Algorytm rekurencyjny

Najpierw **rekurencyjnie** obliczymy wartości argumentów działania, które trzeba wykonać jako ostatnie. W odwrotnej notacji polskiej znak tego działania jest ostatnim znakiem zapisu. Rysunek 1.5 przedstawia przykładowe wyrażenie z podziałem na fragmenty, których wartości trzeba wyznaczyć rekurencyjnie.

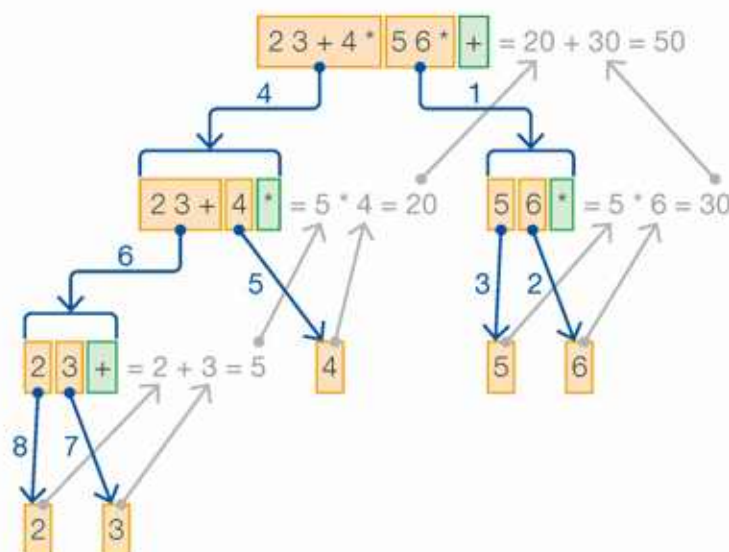
Rekurencja (rekursja),
podręcznik
Informatyka na czasie 2.
Zakres rozszerzony,
s. 236 



Rys. 1.5. Podział wyrażenia na argumenty wyznaczane rekurencyjnie

Jeśli w odwrotnej notacji polskiej ostatni znak wyrażenia jest cyfrą, to dane wyrażenie jest stałą. W takim przypadku zgodnie ze specyfikacją stałą tą jest liczba jednocyfrowa. Nie ma wówczas wywołania rekurencyjnego, a wynikiem jest dana liczba.

Na rysunku 1.6 są przedstawione wywołania rekurencyjne dla wyrażenia $2\ 3\ +\ 4\ *\ 5\ 6\ +\ +$. Liczby przy strzałkach oznaczają kolejność tych wywołań. Zielony prostokąt wskazuje ostatni znak badanego wyrażenia, a pomarańczowe prostokąty – argumenty, które zostaną obliczone rekurencyjnie. Szarym kolorem oznaczone są zwracane wyniki.



Rys. 1.6. Schemat rekurencyjnego obliczania wartości wyrażenia zapisanego w ONP

Oto pseudokod funkcji realizującej algorytm rekurencyjny:

```
funkcja Oblicz(onp)
    znak ← ostatni znak wyrażenia w onp
    onp ← onp bez ostatniego znaku
    jeśli znak jest cyfrą to
        zwróć wartość(znak) i zakończ
    w przeciwnym przypadku
        arg2 ← Oblicz(onp)
        arg1 ← Oblicz(onp)
        jeśli znak = '+' to
            zwróć arg1 + arg2 i zakończ
        jeśli znak = '-' to
            zwróć arg1 - arg2 i zakończ
        jeśli znak = '*' to
            zwróć arg1 * arg2 i zakończ
        jeśli znak = '/' to
            zwróć arg1 div arg2 i zakończ
```

Warto wiedzieć

Funkcja `Oblicz` musi modyfikować zmienną `onp`, aby w kolejnych wywołaniach rekurencyjnych parametrem tej funkcji było odpowiednio skrócone wyrażenie.

Wartością funkcji `Oblicz` jest liczba. Dlatego w przypadku, gdy ostatni znak wyrażenia jest cyfrą, dokonujemy konwersji znaku cyfry na liczbę jednocyfrową. Operacja ta oznaczona jest przez `wartość(znak)`.

Dla wyrażenia przedstawionego na rysunku 1.6 ze s. 21 następują poniższe wywołania funkcji `Oblicz`:

```
Oblicz("23+4*56*+") = 50
  Oblicz("23+4*56*") = 30
    Oblicz("23+4*56") = 6, warunek początkowy
    Oblicz("23+4*5") = 5, warunek początkowy
  Oblicz("23+4*") = 20
    Oblicz("23+4") = 4, warunek początkowy
    Oblicz("23+") = 5
      Oblicz("23") = 3, warunek początkowy
      Oblicz("2") = 2, warunek początkowy
```

Zapisując funkcję `Oblicz` w języku C++, należy pamiętać, że zmiany dokonane na parametrze `onp` (usunięcie ostatniego znaku z wyrażenia) muszą zostać zachowane. Parametr jest więc przekazywany przez referencję. W tym celu przed nazwą parametru stawiamy znak `&`. Kod źródłowy funkcji może wyglądać następująco:

Kod źródłowy 
funkcji obliczającej
wartość wyrażenia
arytmetycznego
zapisanego w ONP,
realizującej algorytm
rekurencyjny

```
1. int Oblicz(string &onp)
2. {
3.     char znak=onp[onp.size()-1];
4.     onp.erase(onp.size()-1,1);
5.     if (znak>='0' && znak<='9')
6.         return znak-48;
7.     else
8.     {
9.         int arg2=Oblicz(onp);
10.        int arg1=Oblicz(onp);
11.        switch (znak)
12.        {
13.            case '+':
14.                return arg1+arg2;
15.            case '-':
16.                return arg1-arg2;
17.            case '*':
18.                return arg1*arg2;
19.            case '/':
20.                return arg1/arg2;
21.        }
22.    }
23. }
```

Dobra rada

Znaki parametru `onp` są usuwane z wyrażenia przez funkcję rekurencyjną `Oblicz`. Jeśli po jej wykonaniu potrzebny byłby jeszcze zapis wyrażenia, zapamiętaj go w zmiennej pomocniczej przed wywołaniem funkcji `Oblicz`.

W linii 3 zmienna `znak` przyjmuje wartość ostatniego znaku wyrażenia `onp` będącego parametrem, a w linii 4 ten znak jest usuwany z wyrażenia. Jeśli jest on cyfrą, to funkcja `Oblicz` zwróci liczbę jednocyfrową.

Aby zwrócić tę liczbę, od kodu ASCII cyfry odejmowana jest wartość 48, czyli kod ASCII znaku '0' (linia 6). W liniach 9 i 10 obliczane są rekurencyjnie argumenty działania (najpierw prawy argument). Następnie w instrukcji **switch** obliczany jest wynik funkcji (linie 11–21). Instrukcja **return** kończy działanie funkcji. Ponieważ została użyta w instrukcji **switch**, nie jest już potrzebna instrukcja **break**.

Rysunek 1.7 przedstawia wywołanie programu obliczającego wartość wyrażenia arytmetycznego podanego w odwrotnej notacji polskiej.

Podaj wyrażenie w ONP: 65*42-1+/
Wartosc wyrażenia: 10

Rys. 1.7. Przykład wywołania programu obliczającego wartość wyrażenia arytmetycznego podanego w odwrotnej notacji polskiej

Ćwiczenie 6

Napisz program obliczający rekurencyjnie wartość wyrażenia zapisanego w ONP zgodnie ze specyfikacją podaną na s. 20.

Algorytm z wykorzystaniem stosu

Będziemy teraz analizować wyrażenie od lewej do prawej. Ponieważ w zapisie ONP znak działania znajduje się za argumentami, których dotyczy działanie, argumenty musimy zapamiętać do momentu, aż odczytamy znak działania. Dlatego przechowamy je na stosie. Działanie dotyczy dwóch argumentów znajdujących się na górze stosu (prawy argument działania to wierzchołek stosu, a lewy to element znajdujący się bezpośrednio pod nim). Należy więc zdjąć dwa elementy ze stosu, wykonać działanie arytmetyczne i wynik ponownie włożyć na stos. Po przejrzaniu całego wyrażenia na stosie pozostanie jeden element – wynik działania. Elementami stosu będą liczby całkowite.

Dobra rada

Zapis **return znak-48** w linii 6 możesz zastąpić instrukcją:
return znak-'0'

Warto wiedzieć

W omówionym algorytmie rekurencyjnym, znajdującym wartość wyrażenia arytmetycznego zapisanego w ONP, wyrażenie było przeglądane od prawej do lewej. Jest to ogólna cecha algorytmów rekurencyjnych – przeglądają one dane w odwrotnej kolejności niż algorytmy iteracyjne.

A to ciekawe

Stos w translatorach języków programowania

Twórcą koncepcji stosu jako struktury danych jest niemiecki matematyk Friedrich Bauer (na zdjęciu). Przedstawił ją w roku 1955. Stos jest wykorzystywany m.in. w translatorach języków programowania do pamiętania zmiennych lokalnych i parametrów funkcji. Właśnie z tego powodu przy obliczaniu złożoności pamięciowej algorytmów rekurencyjnych dodaje się składnik odpowiedzialny za rozmiar stosu. Friedrich Bauer jest też autorem popularnego terminu „inżynieria oprogramowania”.



Tabela 1.5 przedstawia kolejne kroki podczas wyznaczania wartości wyrażenia $2 \cdot 3 \cdot 4 \cdot 5 \cdot +$ z wykorzystaniem stosu.

Krok	Rozpatrywany znak	Operacja/operacje	Elementy na stosie
1.	2	Włożenie na stos	2
2.	3	Włożenie na stos	2 3
3.	*	Zdjęcie ze stosu 3 Zdjęcie ze stosu 2 Włożenie na stos iloczynu 2 i 3	6
4.	4	Włożenie na stos	6 4
5.	5	Włożenie na stos	6 4 5
6.	*	Zdjęcie ze stosu 5 Zdjęcie ze stosu 4 Włożenie na stos iloczynu 4 i 5	6 20
7.	+	Zdjęcie ze stosu 20 Zdjęcie ze stosu 6 Włożenie na stos sumy 6 i 20	26
8.		Zdjęcie ze stosu wyniku wyrażenia	

Tabela 1.5. Kolejne kroki podczas obliczania wartości wyrażenia $2 \cdot 3 \cdot 4 \cdot 5 \cdot +$ z wykorzystaniem stosu

Algorytm obliczający wartość wyrażenia arytmetycznego z wykorzystaniem stosu zgodnie ze specyfikacją podaną na s. 20 można zapisać w pseudokodzie następująco:

```

dla i ← 0, ..., długość onp – 1 wykonuj
    jeśli onp[i] jest cyfrą to push(wartość(onp[i]))
    w przeciwnym przypadku
        arg2 ← top()
        pop()
        arg1 ← top()
        pop()
        jeśli znak = '+' to
            push(arg1 + arg2)
            przejdź do następnego powtórzenia pętli
        jeśli znak = '-' to
            push(arg1 - arg2)
            przejdź do następnego powtórzenia pętli
        jeśli znak = '*' to
            push(arg1 * arg2)
            przejdź do następnego powtórzenia pętli
        jeśli znak = '/' to
            push(arg1 div arg2)
wart ← top()
pop()

```

Warto wiedzieć

Wierzchołkiem stosu jest prawy argument działania. Kolejność zdejmowania argumentów ze stosu ma znaczenie dla działań, które nie są przemienne.

Oto kod źródłowy funkcji realizującej podany algorytm:

```

1. int Oblicz(string onp)
2. {
3.     stack<int> stos;
4.     for(int i=0; i<onp.size(); i++)
5.         if (onp[i]>='0' && onp[i]<='9')
6.             stos.push(onp[i]-48);
7.         else
8.         {
9.             int arg2=stos.top(); stos.pop();
10.            int arg1=stos.top(); stos.pop();
11.            switch (onp[i])
12.            {
13.                case '+':
14.                    stos.push(arg1+arg2);
15.                    break;
16.                case '-':
17.                    stos.push(arg1-arg2);
18.                    break;
19.                case '*':
20.                    stos.push(arg1*arg2);
21.                    break;
22.                case '/':
23.                    stos.push(arg1/arg2);
24.            }
25.        }
26.        int wart=stos.top();
27.        stos.pop();
28.        return wart;
29.    }

```

• Kod źródłowy funkcji obliczającej wartość wyrażenia arytmetycznego zapisanego w ONP, wykorzystującej stos

W linii 3 deklarowany jest stos, którego elementami będą liczby całkowite (zmienna `stos` typu `stack` o elementach typu `int`). W pętli (linie 4–25) przeglądane są kolejne znaki wyrażenia od lewej do prawej. Jeśli znak jest cyfrą, to jest ona zapisywana na stosie jako liczba jednocyfrowa (linie 5–6). W przeciwnym przypadku ze stosu są zdejmowane dwie liczby, które są zapamiętywane odpowiednio w zmiennych `arg2` i `arg1`, czyli najpierw prawy argument działania, potem lewy (linie 9–10). W liniach 11–24 wkładany jest na stos wynik analizowanego działania. Po zakończeniu działania pętli na stosie znajduje się jedna liczba, która jest obliczoną wartością wyrażenia. Jest ona zdejmowana ze stosu i zapamiętywana w zmiennej `wart` (linie 26–27), której wartość jest wynikiem funkcji `Oblicz` (linia 28).

Ćwiczenie 7

Napisz program obliczający wartość wyrażenia zapisanego w ONP zgodnie ze specyfikacją podaną na s. 20. Wykorzystaj stos.

Podsumowanie

- Sposób zapisu wyrażeń algebraicznych, w którym znak działania występuje pomiędzy argumentami, nazywamy notacją infiksową.
- W notacji prefiksowej (notacji polskiej) znak działania występuje przed argumentami, natomiast w notacji sufiksowej (odwrotnej notacji polskiej) znak działania zapisuje się po argumentach.
- W notacji polskiej oraz odwrotnej notacji polskiej nie używa się nawiasów.
- Dynamiczne struktury danych charakteryzują się tym, że ich rozmiar (liczba elementów) może się zmieniać podczas działania programu.
- Stos to dynamiczna struktura danych typu LIFO (ang. *last in, first out*), w której operacje są wykonywane na jej jednym końcu. Przykładowe operacje na stosie to: *push* (umieszczenie elementu na stosie), *pop* (zdjęcie elementu ze stosu), *top* (odczytanie wartości elementu będącego wierzchołkiem stosu) i *empty* (sprawdzenie, czy stos jest pusty).
- Algorytm zamiany wyrażenia z notacji tradycyjnej na odwrotną notację polską wykorzystuje stos. Na stosie przechowywane są nawiasy otwierające oraz znaki działań.
- Algorytm obliczania wartości wyrażenia arytmetycznego zapisanego w odwrotnej notacji polskiej może wykorzystywać stos. Wówczas na stosie przechowywane są argumenty działań arytmetycznych.
- Aby wybrać jeden przypadek z wielu, można wielokrotnie użyć instrukcji warunkowej **if** lub instrukcji wyboru **switch**.

Zadania

- ★ **1** Napisz program, który wczyta napis złożony wyłącznie z nawiasów okrągłych oraz sprawdzi, czy są one właściwie sparowane. Program powinien wypisać słowo „TAK” lub „NIE”. Na przykład dla napisu „(())” program zwróci „TAK”, a dla napisu „)(())” – „NIE”.
- ★ **2** Napisz program zamieniający wyrażenie algebraiczne zapisane w notacji tradycyjnej na wyrażenie w odwrotnej notacji polskiej. W wyrażeniu mogą wystąpić argumenty jednoznakowe (litery lub liczby jednocyfrowe), nawiasy okrągłe oraz dwuargumentowe działania arytmetyczne: dodawanie (+), odejmowanie (–), mnożenie (*), wyznaczanie części całkowitej z dzielenia (/) oraz reszty z dzielenia (%).
- ★ **3** Napisz program obliczający wartość wyrażenia arytmetycznego zapisanego w odwrotnej notacji polskiej. Wyrażenie może się składać z cyfr jako argumentów jednoznakowych oraz z dwuargumentowych działań arytmetycznych: dodawania (+), odejmowania (–), mnożenia (*), wyznaczania części całkowitej z dzielenia (/) oraz reszty z dzielenia (%).
- ★★ **4** Napisz program, który wczyta liczbę całkowitą dodatnią, a następnie wypisze tworzące ją cyfry w kolejności od cyfry najbardziej znaczącej do cyfry jedności, każdą w oddzielnym wierszu. Wykorzystaj stos liczb całkowitych.

- ★★ **5** Napisz program, który wczyta napis złożony z maksymalnie trzech rodzajów nawiasów: (), [] oraz {}, a następnie sprawdzi, czy są one właściwie sparowane. Kolejność par nawiasów jest dowolna. Na przykład napis „{()}" uznajemy za poprawny, a napis „{())" – za niepoprawny. Program powinien wypisać słowo „TAK” lub „NIE”.
- ★★ **6** Napisz program zamieniający wyrażenie algebraiczne zapisane w notacji tradycyjnej na wyrażenie zapisane w odwrotnej notacji polskiej. Wyrażenie może się składać z argumentów jednoznakowych (liter i liczb jednocyfrowych), nawiasów okrągłych oraz dwuargumentowych działań arytmetycznych: dodawania (+), odejmowania (–), mnożenia (*), wyznaczania części całkowitej z dzielenia (/) oraz potęgowania (^).
- ★★ **7** Napisz program wczytujący napis składający się z cyfr jako argumentów jednoznakowych oraz dwuargumentowych działań arytmetycznych: dodawania (+), odejmowania (–), mnożenia (*) oraz wyznaczania części całkowitej z dzielenia (/) i sprawdzający, czy jest to prawidłowe wyrażenie arytmetyczne zapisane w odwrotnej notacji polskiej. Program powinien wypisać słowo „TAK” lub „NIE”.
- ★★★ **8** Napisz program, który wczyta napis złożony z maksymalnie trzech rodzajów nawiasów: (), [] oraz {}, a następnie sprawdzi, czy są one właściwie sparowane. Kolejność par nawiasów ma znaczenie i jest następująca (od najbardziej wewnętrznego): (), [], {}. Na przykład napis „{()}" jest poprawny, a napis „{()}" nie jest. Program powinien wypisać słowo „TAK” lub „NIE”.
- ★★★ **9** Zapisz w wybranej przez siebie notacji (lista kroków, schemat blokowy, pseudokod, język programowania) algorytm sortowania szybkiego bez wykorzystania rekurencji. Skorzystaj ze stosu.

2. Znajdowanie drogi wyjścia z labiryntu

Zapewne nie raz spotkaliście się z grą, w której trzeba przeprowadzić bohatera przez labirynt korytarzy. Być może w szkole podstawowej pisaliście programy umożliwiające sterowanie postacią na planszy. W tym temacie poznasz algorytmy sprawdzające, czy istnieje droga prowadząca do wyjścia z labiryntu, oraz znajdziesz tę drogę.

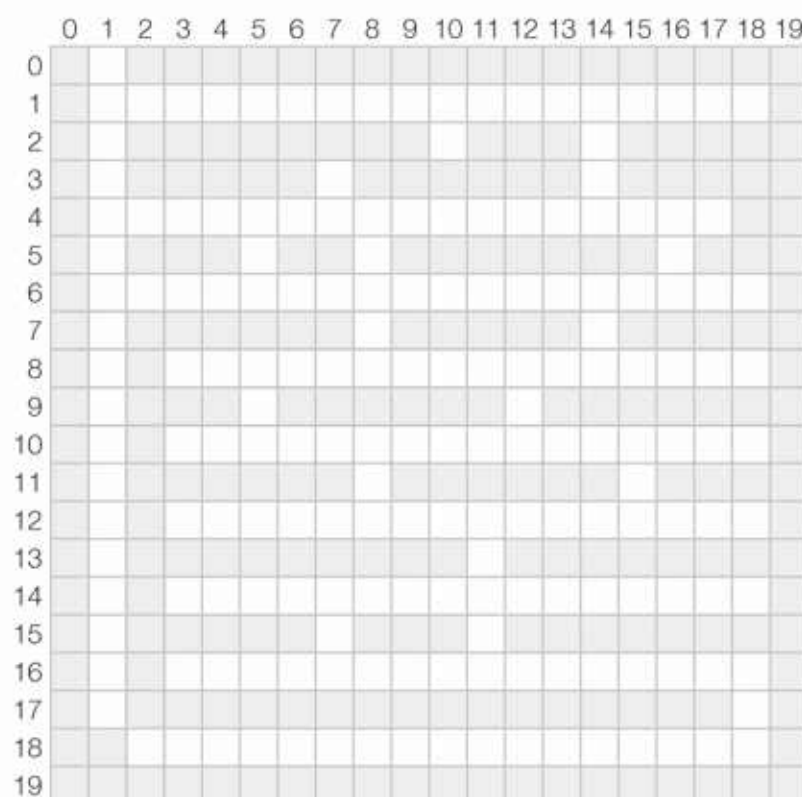
Cele lekcji

- Poznasz algorytmy znajdowania wyjścia z labiryntu.
- Znajdziesz najkrótszą drogę prowadzącą do wyjścia z labiryntu.
- Zastosujesz dynamiczną strukturę danych o nazwie kolejka.

2.1. Przygotowanie planszy z labiryntem

Tablica dwuwymiarowa,
podręcznik
Informatyka na czasie 2.
Zakres rozszerzony,
s. 219 

Planszę z labiryntem będziemy reprezentować za pomocą **tablicy dwuwymiarowej**, w której liczba kolumn i liczba wierszy są równe. Przykładowy labirynt przedstawia rysunek 2.1. Pola szare oznaczają ściany, pola białe tworzą korytarze. Przyjmiemy, że przy krawędzi może się znajdować tylko jedno białe pole – będzie to wyjście. Polem startowym będzie dowolne białe pole. W szczególnym przypadku wyjście i pole startowe mogą być w tym samym miejscu.



Rys. 2.1. Przykładowa plansza z labiryntem

Opis planszy wygodnie jest przygotować w pliku tekstowym. Pola można oznaczyć konkretnymi znakami, np. pole szare wielką literą X, a pole białe – wielką literą W. Plansza na rysunku 2.1 ma wymiary 20×20 , więc plik tekstowy ją opisujący będzie się składał z 20 wierszy, a każdy wiersz z 20 znaków (wielkich liter X lub W). Trzy pierwsze wiersze pliku dla planszy z rysunku 2.1 są następujące:

```
XWXXXXXXXXXXXXXXXXXXXXX
XWWWWWWWWWWWWWWWWWWX
XWXXXXXXXXXXWXXXWXXXXX
```

Algorytmy poszukujące drogi wyjścia z labiryntu będą oznaczały pola już rozpatrzone. Wykorzystają do tego liczby całkowite. Dlatego w tablicy reprezentującej planszę przechowamy takie właśnie liczby. Pola tworzące ściany oznaczymy liczbą -1 . Liczbę 0 przypiszemy polom korytarzy, które nie były jeszcze rozpatrywane. Liczby dodatnie wykorzystamy przy poszukiwaniu drogi, a liczbę -2 do oznaczenia znalezionej drogi. Wartości tablicy reprezentującej planszę z rysunku 2.1 przedstawia rysunek 2.2.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
2	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1
3	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1
4	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-1
5	-1	0	-1	-1	-1	0	-1	-1	0	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1
6	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
7	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1
8	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
9	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1
10	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
11	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1
12	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
13	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
14	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
15	-1	0	-1	-1	-1	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
16	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
17	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1
18	-1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
19	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Rys. 2.2. Wartości tablicy reprezentującej planszę z rysunku 2.1

Funkcja wczytująca do programu opis labiryntu z pliku tekstowego powinna zamieniać odczytane znaki (X lub W) na odpowiednie wartości liczbowe. Parametrem funkcji będzie tablica dwuwymiarowa `Lab`. Liczbę wierszy i kolumn określimy za pomocą zdefiniowanej w programie stałej `N`. Należy pamiętać o dodaniu w kodzie programu dyrektywy `#include <fstream>` – powoduje ona dołączenie biblioteki służącej do obsługi plików. Kod źródłowy funkcji wczytującej opis labiryntu z pliku tekstowego *labirynt.txt* może być następujący.

Warto wiedzieć

Istnieją algorytmy pozwalające wygenerować labirynt, w którym z dowolnego miejsca można wyznaczyć drogę prowadzącą do wyjścia.

Warto wiedzieć

Informacja o tym, czy pole jest oznaczone kolorem szarym czy białym, to informacja zero-jedynkowa (logiczna).

Dobra rada

Pamiętaj, że definicję stałej rozpoczyna się od słowa kluczowego `const`. Po tym słowie podaj typ stałej i jej nazwę, a po znaku równości – wartość stałej.

Obsługa plików tekstowych w języku C++, s. 417–419 

Kod źródłowy funkcji 
wczytującej do
programu opis planszy
z labiryntem z pliku
tekstowego

Dobra rada

Gdy parametrem funkcji jest tablica dwuwymiarowa, nie możesz pominąć w nawiasach jej drugiego wymiaru (liczby kolumn).

```
1. void WczytajLabirynt(int Lab[][N])
2. {
3.     string s;
4.     ifstream we("labirynt.txt");
5.     for (int i=0;i<N;i++)
6.     {
7.         we>>s;
8.         for (int j=0;j<N;j++)
9.             if (s[j]=='X') Lab[i][j]=-1;
10.            else Lab[i][j]=0;
11.        }
12.    we.close();
13. }
```

Dobra rada

Pamiętaj, że tablice są przekazywane do funkcji przez wskaźnik. Oznacza to, że operacje są wykonywane na oryginalnej tablicy (nie jest tworzona kopia takiego parametru), a dokonane w niej zmiany zostają zachowane po zakończeniu działania funkcji.

W linii 4 deklarujemy zmienną plikową `we` typu `ifstream` i otwieramy do odczytu skojarzony z nią plik zawierający opis planszy. W linii 7 program wczytuje do zmiennej `s` wiersz pliku reprezentowanego przez zmienną `we`. Pętla w liniach 8–10 przegląda znaki wczytanego napisu i jeśli dany znak jest literą `X`, to odpowiedniemu elementowi tablicy przypisuje wartość `-1`, w przeciwnym przypadku – wartość `0`. W linii 12 przy użyciu metody `close` zamykamy plik skojarzony ze zmienną `we`.

Napiszemy kod źródłowy funkcji, która na podstawie wartości zapisanych w tablicy `Lab` wypisze planszę z labiryntem na ekranie. Pole korytarza wyświetli się jako puste, a dla pola oznaczającego ścianę wypiszemy znak `X`. Oprócz labiryntu wypiszemy numery wierszy oraz kolumn. Na każde pole przeznaczymy trzy znaki. Oto kod źródłowy funkcji:

Kod źródłowy funkcji 
wypisującej planszę
z labiryntem na
podstawie wartości
elementów tablicy

```
1. void WypiszLabirynt(int Lab[][N])
2. {
3.     cout<<" ";
4.     for (int j=0;j<N;j++)
5.         cout<<setw(3)<<j;
6.     cout<<endl;
7.     for (int i=0;i<N;i++)
8.     {
9.         cout<<setw(3)<<i;
10.        for (int j=0;j<N;j++)
11.            if (Lab[i][j]==-1) cout<<" X";
12.            else cout<<" ";
13.        cout<<endl;
14.    }
15. }
```

Na ekranie najpierw zostaną wypisane trzy spacje (linia 3), a następnie numery kolumn (linie 4–5). Przy wypisywaniu numeru kolumny wykorzystujemy **funkcję** `setw` z biblioteki `iomanip`, formatującą wypisywane wartości. Jej parametrem jest liczba całkowita, która określa, ile znaków

Funkcja `setw` 

będzie przeznaczonych na wyświetlenie napisu. Dla parametru równego 3 liczba jednocyfrowa zostanie poprzedzona dwiema spacjami, a liczba dwucyfrowa – jedną spacją. Jeśli wyświetlany napis liczyłby więcej znaków, niż określa to parametr funkcji, to szerokość pola zostałaby odpowiednio zwiększona. Funkcja `setw` jest przykładem **manipulatora strumienia**. Manipulatory strumieni pozwalają m.in. określić sposób interpretowania wczytywanych danych i wyświetlania informacji na ekranie.

Pętla w liniach 7–14 odpowiada za wypisanie znaków w kolejnych wierszach. Najpierw wypisywany jest numer wiersza (linia 9) – tu także wykorzystana jest funkcja `setw`. Pętla w liniach 10–12 wypełnia wiersz na podstawie tablicy Lab: dla wartości elementu `-1` (ściana) wypisuje znak X poprzedzony dwiema spacjami, w przeciwnym przypadku – trzy spacje (wolne pole). Rysunek 2.3 przedstawia efekt działania funkcji `WypiszLabirynt` dla wartości tablicy takich jak na rysunku 2.2 ze s. 29.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
1	X																			X
2	X		X	X	X	X	X	X	X	X		X	X	X		X	X	X	X	X
3	X		X	X	X	X	X		X	X	X	X	X	X		X	X	X	X	X
4	X																			X
5	X		X	X	X		X	X		X	X	X	X	X	X	X		X	X	X
6	X																			X
7	X		X	X	X	X	X	X		X	X	X	X	X		X	X	X	X	X
8	X		X																	X
9	X		X	X	X		X	X	X	X	X		X	X	X	X	X	X	X	X
10	X		X																	X
11	X		X	X	X	X	X		X	X	X	X	X	X		X	X	X	X	X
12	X		X																	X
13	X		X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X
14	X		X																	X
15	X		X	X	X	X	X		X	X	X		X	X	X	X	X	X	X	X
16	X		X																	X
17	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
18	X	X																		X
19	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Rys. 2.3. Efekt działania funkcji `WypiszLabirynt` dla wartości tablicy z rysunku 2.2

Ćwiczenie 1

Napisz program, który wczyta opis labiryntu z pliku otrzymanego od nauczyciela (np. *labirynt.txt*) i wypisze planszę z labiryntem na ekranie. Wykorzystaj funkcje `WczytajLabirynt` oraz `WypiszLabirynt`.

2.2. Rekurencyjny algorytm znajdowania wyjścia z labiryntu

Sprawdźmy najpierw, czy istnieje droga prowadząca z pola startowego do wyjścia. Następnie wyznaczmy taką drogę.

Manipulator strumienia

Dobra rada

Aby korzystać w programie z funkcji zdefiniowanych w bibliotece `iomanip`, pamiętaj o dodaniu dyrektywy `#include <iomanip>`.

Warto wiedzieć

W bibliotece `iomanip` dostępnych jest wiele manipulatorów strumienia. Manipulatory zdefiniowano także w bibliotece `iostream`. Standard języka C++ nie precyzuje, jak manipulatory zachowują się w przypadku nieprawidłowego użycia.

Warto wiedzieć

Dzięki funkcji `setw` można też określić sposób wyświetlania wartości innego typu niż liczby całkowite, np. znaków. Poniższa instrukcja wyświetli znak X poprzedzony dwiema spacjami:
`cout<<setw(3)<<'X';`

Rekurencyjny algorytm sprawdzający, czy istnieje droga wyjścia z labiryntu

Specyfikacja problemu jest następująca:

Dobra rada

Staraj się nadawać zmiennym takie nazwy, które podpowiadają, co będzie w nich pamiętane. Dzięki temu kod źródłowy programu jest czytelniejszy i łatwiej go poprawiać. Na przykład w naszym programie nazwa `w` jest pierwszą literą słowa „wiersz”, a `k` – pierwszą literą słowa „kolumna”.

Specyfikacja

Dane: n – liczba całkowita dodatnia określająca liczbę wierszy i liczbę kolumn tablicy `Lab`,

`Lab[0..n-1][0..n-1]` – tablica o n wierszach i n kolumnach, której elementami są liczby całkowite opisujące planszę z labiryntem, `Lab[i][j] = -1`, gdy pole (i, j) jest ścianą, `Lab[i][j] = 0` – w przeciwnym przypadku, $0 \leq i < n, 0 \leq j < n$,

(w, k) – pole startowe, `Lab[w][k] = 0`, $0 \leq w < n, 0 \leq k < n$.

Wynik: wartość logiczna **prawda**, gdy istnieje droga od pola (w, k) do wyjścia, czyli pola (w_1, k_1) , gdzie $w_1 = 0$ lub $w_1 = n - 1$ lub $k_1 = 0$ lub $k_1 = n - 1$, **fałsz** – w przeciwnym przypadku.

Jeśli pole startowe leży w skrajnym wierszu lub skrajnej kolumnie, to znaleźliśmy wyjście z labiryntu. Algorytm kończy wówczas działanie. W przeciwnym przypadku należy sprawdzić pola sąsiadujące z tym, na którym się aktualnie znajdujemy. Jeśli wolno stanąć na sąsiednim polu, to potraktujemy je jako nowe pole startowe i wywołamy rekurencyjnie funkcję dla tego pola.

Odwiedzone pola należy oznaczać, inaczej wielokrotnie moglibyśmy poszukiwać drogi z tego samego pola (wywoływać funkcję rekurencyjnie dla tych samych parametrów, a więc mielibyśmy do czynienia z rekurencją nieskończoną). Do oznaczenia odwiedzonych pól wykorzystamy wartość 1. Jeśli po rekurencyjnym sprawdzeniu, czy istnieje droga z sąsiednich pól, nie dotrzemy do wyjścia, to nie istnieje droga z pola startowego do wyjścia. Funkcja zwróci wówczas wartość **fałsz**.

Oto zapis w pseudokodzie funkcji sprawdzającej, czy istnieje droga prowadząca do wyjścia z labiryntu:

```
funkcja Droga(Lab[][], w, k)
    Lab[w][k] ← 1
    jeśli w = 0 lub w = n - 1 lub k = 0 lub k = n - 1 to
        zwróć prawda i zakończ
    jeśli Lab[w-1][k] = 0 oraz Droga(Lab, w-1, k) to
        zwróć prawda i zakończ
    jeśli Lab[w+1][k] = 0 oraz Droga(Lab, w+1, k) to
        zwróć prawda i zakończ
    jeśli Lab[w][k-1] = 0 oraz Droga(Lab, w, k-1) to
        zwróć prawda i zakończ
    jeśli Lab[w][k+1] = 0 oraz Droga(Lab, w, k+1) to
        zwróć prawda i zakończ
    zwróć fałsz i zakończ
```

Dla labiryntu z rysunku 2.2 ze s. 29 oraz pola startowego (1,14) podany algorytm oznaczy liczbą 1 pola labiryntu wyróżnione na rysunku 2.4 kolorem zielonym. Jeśli w algorytmie zmienimy kolejność wywołań rekurencyjnych, liczbą 1 mogą zostać oznaczone inne pola.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	-1
2	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	1	-1	-1	-1	-1	-1
3	-1	1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1
4	-1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	-1	-1
5	-1	0	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1
6	-1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1
7	-1	0	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1
8	-1	0	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1
9	-1	0	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1
10	-1	0	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1
11	-1	0	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1
12	-1	0	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1
13	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1
14	-1	0	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1
15	-1	0	-1	-1	-1	-1	-1	1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1
16	-1	0	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1
17	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1
18	-1	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1
19	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Rys. 2.4. Labirynt z polami oznaczonymi przez algorytm rekurencyjny liczbą 1, gdy polem startowym jest pole (1,14)

Warto wiedzieć

Algorytm sprawdzający, czy istnieje droga wyjścia z labiryntu, nie musi rozpatrzeć wszystkich wolnych pól planszy. Na przykład na rysunku 2.4 widać, że pola (1,13) i (1,15), sąsiadujące z polem startowym (1,14), nie zostały odwiedzone. Stało się tak dlatego, że przed nimi było rozpatrywane pole (2,14), a po przejściu przez nie można znaleźć drogę do wyjścia z labiryntu.

Ćwiczenie 2

Napisz program, który wczyta opis labiryntu z pliku otrzymanego od nauczyciela (np. *labirynt.txt*) oraz wypisze planszę z labiryntem na ekranie. Następnie wczyta współrzędne pola startowego i sprawdzi, czy istnieje droga prowadząca z tego pola do wyjścia. Jeśli droga istnieje, program wypisze „TAK”, w przeciwnym przypadku – „NIE”.

Rekurencyjny algorytm wyznaczający drogę do wyjścia z labiryntu

Oznaczenie odwiedzonych pól taką samą liczbą (w naszym przypadku liczbą 1) nie wystarczy, aby wskazać drogę prowadzącą do wyjścia. W algorytmie wyznaczającym drogę odwiedzone pola będziemy oznaczać kolejnymi liczbami całkowitymi dodatnimi, zaczynając od 1. Pole startowe otrzyma zatem wartość 1, następne odwiedzone pole – wartość 2 itd. W ten sposób droga przejścia będzie oznaczona kolejnymi liczbami całkowitymi dodatnimi. W funkcji Droga dodamy parametr *x* określający wartość, którą należy oznaczyć aktualnie rozpatrywane pole. Oto zmodyfikowana funkcja zapisana w pseudokodzie.


```

funkcja Droga(Lab[[]],w,k,x)
  Lab[w][k] ← x
  jeśli w = 0 lub w = n - 1 lub k = 0 lub k = n - 1 to
    zwróć prawda i zakończ
  jeśli Lab[w-1][k] = 0 oraz Droga(Lab,w-1,k,x+1) to
    zwróć prawda i zakończ
  jeśli Lab[w+1][k] = 0 oraz Droga(Lab,w+1,k,x+1) to
    zwróć prawda i zakończ
  jeśli Lab[w][k-1] = 0 oraz Droga(Lab,w,k-1,x+1) to
    zwróć prawda i zakończ
  jeśli Lab[w][k+1] = 0 oraz Droga(Lab,w,k+1,x+1) to
    zwróć prawda i zakończ
  zwróć fałsz i zakończ

```

Rysunek 2.5 przedstawia labirynt z polami, które algorytm oznaczy kolejnymi liczbami całkowitymi dodatnimi, gdy startujemy z pola (1,14).

Warto wiedzieć

Plansza może wyglądać inaczej, jeśli w algorytmie zmienimy kolejność wywołań rekurencyjnych.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	-1	25	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	24	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	-1
2	-1	23	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	2	-1	-1	-1	-1	-1
3	-1	22	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	3	-1	-1	-1	-1	-1
4	-1	21	20	19	18	17	0	0	10	9	8	7	6	5	4	47	46	47	-1	-1
5	-1	0	-1	-1	-1	16	-1	-1	11	-1	-1	-1	-1	-1	-1	-1	45	-1	-1	-1
6	-1	0	0	0	0	15	14	13	12	47	46	45	44	43	42	43	44	45	46	-1
7	-1	0	-1	-1	-1	-1	-1	-1	13	-1	-1	-1	-1	-1	41	-1	-1	-1	-1	-1
8	-1	0	-1	19	18	17	16	15	14	41	40	39	38	39	40	41	42	43	44	-1
9	-1	0	-1	-1	-1	18	-1	-1	-1	-1	-1	-1	37	-1	-1	-1	-1	-1	-1	-1
10	-1	0	-1	21	20	19	20	21	22	39	38	37	36	35	34	33	34	35	36	-1
11	-1	0	-1	-1	-1	-1	-1	-1	23	-1	-1	-1	-1	-1	-1	32	-1	-1	-1	-1
12	-1	0	-1	29	28	27	26	25	24	25	26	27	28	29	30	31	32	33	34	-1
13	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	28	-1	-1	-1	-1	-1	-1	-1
14	-1	0	-1	41	40	39	38	37	38	39	40	29	30	31	32	33	34	35	36	-1
15	-1	0	-1	-1	-1	-1	-1	36	-1	-1	-1	30	-1	-1	-1	-1	-1	-1	-1	-1
16	-1	0	-1	39	38	37	36	35	34	33	32	31	32	33	34	35	36	37	38	-1
17	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	39	-1
18	-1	-1	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	-1
19	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Rys. 2.5. Labirynt z polami oznaczonymi przez algorytm rekurencyjny kolejnymi liczbami całkowitymi dodatnimi, gdy polem startowym jest (1,14)

Zwróć uwagę, że na planszy powtarzają się niektóre liczby. Spójrz np. na pole (6,8) oznaczone wartością 12. Dwa pola obok niego – pola (6,7) oraz (7,8) – są oznaczone wartością 13. Zgodnie z algorytmem po polu (6,8) najpierw rozpatrywane jest pole (7,8). Ponieważ po przejściu przez to pole nie uda się dotrzeć do wyjścia (są ślepe korytarze lub algorytm napotka pole wcześniej odwiedzone), algorytm wróci do pola (6,8) i poszuka drogi przechodzącej przez pole po lewej – pole (6,7).

Żeby wskazać drogę, należy ją ponownie przejrzeć w odwrotnym kierunku – od wyjścia do pola startowego. Gdybyśmy zaczęli od pola startowego, po dotarciu do pola, dla którego sąsiednie pola mają te same

wartości, np. pola (6,8), nie byłoby wiadomo, na które z sąsiednich pól przejść. Drogę można oznaczyć konkretną wartością, np. liczbą -2. Oto zmodyfikowana specyfikacja problemu:

Specyfikacja

Dane: n – liczba całkowita dodatnia określająca liczbę wierszy i liczbę kolumn tablicy Lab ,

$Lab[0..n-1][0..n-1]$ – tablica o n wierszach i n kolumnach, której elementami są liczby całkowite opisujące planszę z labiryntem, $Lab[i][j] = -1$, gdy pole (i,j) jest ścianą, $Lab[i][j] = 0$ – w przeciwnym przypadku, $0 \leq i < n$, $0 \leq j < n$,

(w, k) – pole startowe, $Lab[w][k] = 0$, $0 \leq w < n$, $0 \leq k < n$.

Wynik: wartość logiczna **prawda**, gdy istnieje droga od pola (w,k) do wyjścia, czyli pola $(w1,k1)$, gdzie $w1 = 0$ lub $w1 = n - 1$ lub $k1 = 0$ lub $k1 = n - 1$, **fałsz** – w przeciwnym przypadku,

Lab – tablica z drogą oznaczoną liczbą -2: od pola (w,k) do pola $(w1,k1)$, jeśli taka droga istnieje.

Zapis funkcji oznaczającej drogę może być taki jak poniżej. Parametry w i k oznaczają tym razem nie pole startowe, ale znalezione wyjście.

```
funkcja OznaczDroge(Lab[][],w,k)
  x ← Lab[w][k]
  Lab[w][k] ← -2
  dopóki x > 1 wykonuj
    x ← x - 1
    jeśli w > 0 oraz Lab[w-1][k] = x to
      w ← w - 1
    w przeciwnym przypadku
      jeśli w < n - 1 oraz Lab[w+1][k] = x to
        w ← w + 1
      w przeciwnym przypadku
        jeśli k > 0 oraz Lab[w][k-1] = x to
          k ← k - 1
        w przeciwnym przypadku
          k ← k + 1
  Lab[w][k] ← -2
```

Warto wiedzieć

Funkcję `OznaczDroge` można również zapisać rekurencyjnie.

Zmienna x przyjmuje wartość pamiętaną dla pola, które jest znalezionym wyjściem. Polu temu następnie przypisujemy wartość -2. Dopóki nie dojdziemy do pola startowego (wartość zmiennej x jest większa od 1), poszukujemy sąsiedniego pola o wartości o 1 mniejszej względem aktualnie rozpatrywanego pola, które następnie oznaczamy jako należące do drogi. Jeśli dwa sąsiednie pola są oznaczone taką samą wartością, to nie ma znaczenia, które z nich wybierzemy – przez obydwa prowadzą drogi do wyjścia z labiryntu.

Kody źródłowe funkcji Droga oraz OznaczDroge w programie wyznaczającym drogę wyjścia z labiryntu mogą być następujące:

Fragment kodu
źródłowego programu
wyznaczającego
rekurencyjnie
drogę do wyjścia
z labiryntu – funkcje
Droga i OznaczDroge

```

1.  bool Droga(int Lab[][N], int w, int k, int x,
2.           int &w1, int &k1)
3.  {
4.      Lab[w][k]=x;
5.      if (w==0 || w==N-1 || k==0 || k==N-1)
6.      {
7.          w1=w; k1=k;
8.          return true;
9.      }
10.     if (Lab[w-1][k]==0 && Droga(Lab,w-1,k,x+1,w1,k1))
11.         return true;
12.     if (Lab[w+1][k]==0 && Droga(Lab,w+1,k,x+1,w1,k1))
13.         return true;
14.     if (Lab[w][k-1]==0 && Droga(Lab,w,k-1,x+1,w1,k1))
15.         return true;
16.     if (Lab[w][k+1]==0 && Droga(Lab,w,k+1,x+1,w1,k1))
17.         return true;
18.     return false;
19. }
20.
21. void OznaczDroge(int Lab[][N], int w, int k)
22. {
23.     int x=Lab[w][k];
24.     Lab[w][k]=-2;
25.     while (x>1)
26.     {
27.         x--;
28.         if (w>0 && Lab[w-1][k]==x) w--;
29.         else if (w<N-1 && Lab[w+1][k]==x) w++;
30.         else if (k>0 && Lab[w][k-1]==x) k--;
31.         else k++;
32.         Lab[w][k]=-2;
33.     }
34. }
```



Dobra rada

Pamiętaj, że przy przekazaniu parametru przez referencję zmiany nanoszone są bezpośrednio na parametrze, nie na jego kopii. Dzięki temu zmieniona wartość parametru jest pamiętana po zakończeniu działania funkcji.

Zwróć uwagę na nagłówek funkcji Droga. Ma ona dwa dodatkowe parametry w1 i k1, przekazywane przez referencję. Za ich pomocą funkcja zwróci współrzędne znalezionej drogi. Są one potrzebne do oznaczenia drogi – ich wartości będą parametrami aktualnymi funkcji OznaczDroge.

Do wypisania labiryntu ze znaną drogą wykorzystamy zdefiniowaną wcześniej funkcję WypiszLabirynt. Zmodyfikujemy ją tak, aby dla wartości elementu tablicy Lab równej -2 była wypisywana litera D. Oto instrukcja warunkowa odpowiedzialna za wyświetlenie pola labiryntu w zależności od wartości elementu tablicy Lab.

Kod źródłowy funkcji
wypisującej planszę
z labiryntem na
podstawie wartości
elementów w tablicy,
s. 30

```

1. if (Lab[i][j]==-1) cout<<" X";
2. else if (Lab[i][j]==-2) cout<<" D";
3.     else cout<<" ";

```

❶ Zmodyfikowany fragment kodu źródłowego funkcji wypisującej planszę z labiryntem na podstawie wartości elementów tablicy

Kod źródłowy funkcji main programu znajdującego i wypisującego drogę wyjścia z labiryntu może być następujący:

```

1. int main()
2. {
3.     int w, k, w1, k1;
4.     int Lab[N][N];
5.     WczytajLabirynt(Lab);
6.     WypiszLabirynt(Lab);
7.     cout<<"Wspolrzedne pola startowego: "<<endl;
8.     cout<<"w = "; cin>>w;
9.     cout<<"k = "; cin>>k;
10.    if (Droga(Lab,w,k,1,w1,k1))
11.    {
12.        OznaczDroge(Lab,w1,k1);
13.        WypiszLabirynt(Lab);
14.    }
15.    else
16.        cout<<"Brak drogi";
17.    return 0;
18. }

```

❷ Fragment kodu źródłowego programu wyznaczającego rekurencyjnie drogę do wyjścia z labiryntu – funkcja main

W liniach 7–9 wczytujemy współrzędne pola startowego. Jeśli droga do wyjścia zostanie odnaleziona (funkcja Droga przyjmie wartość **true** – linia 10), to program oznacza drogę (linia 12) oraz wypisuje labirynt z zaznaczoną drogą (linia 13). Gdy nie ma drogi prowadzącej do wyjścia, wyświetli się odpowiedni komunikat (linia 16). Przykład działania programu przedstawia rysunek 2.6 na s. 38.

A to ciekawe

W sieci korytarzy

Motyw labiryntu jest znany od tysięcy lat i ma bogatą symbolikę. Pojawił się np. w mitcie o Tezeuszu, w którym główny bohater wydostaje się z tej pułapki dzięki nici Ariadny. Niektórzy uważają, że mityczny labirynt znajdował się w pałacu w Knossos na Krecie. Labirynty budowano m.in. po to, aby utrudnić dotarcie do grobowców i skarbców. Pełniły także funkcję dekoracyjną – do dziś zdobią wnętrza wielu świątyń (np. katedr w Amiens i Chartres we Francji), były także bardzo popularne w renesansowych ogrodach.



Współrzędne pola startowego:
w = 1
k = 14

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	X	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
1	X	D													D					X
2	X	D	X	X	X	X	X	X	X	X		X	X	X	D	X	X	X	X	X
3	X	D	X	X	X	X	X		X	X	X	X	X	X	D	X	X	X	X	X
4	X	D	D	D	D	D			D	D	D	D	D	D	D					X
5	X		X	X	X	D	X	X	D	X	X	X	X	X	X	X		X	X	X
6	X					D	D	D	D											X
7	X		X	X	X	X	X	X		X	X	X	X	X		X	X	X	X	X
8	X		X																	X
9	X		X	X	X		X	X	X	X	X		X	X	X	X	X	X	X	X
10	X		X																	X
11	X		X	X	X	X	X		X	X	X	X	X	X	X		X	X	X	X
12	X		X																	X
13	X		X	X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X
14	X		X																	X
15	X		X	X	X	X	X		X	X	X		X	X	X	X	X	X	X	X
16	X		X																	X
17	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X
18	X	X																		X
19	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Rys. 2.6. Przykład działania programu znajdującego drogę do wyjścia z pola (1,14)

Ćwiczenie 3

Napisz program, który z pliku przekazanego ci przez nauczyciela (np. *labirynt.txt*) wczyta informacje o planszy z labiryntem, a następnie wypisze tę planszę na ekranie. Po wczytaniu współrzędnych pola startowego program wypisze planszę z oznaczoną drogą do wyjścia, jeśli taka istnieje, lub komunikat informujący, że drogi do wyjścia nie ma. Znaleziona droga nie musi być najkrótszą możliwą. Przetestuj działanie programu dla różnych pól startowych.

Algorytm przeszukiwania z nawrotami

Opisany rekurencyjny algorytm poszukiwania drogi wyjścia z labiryntu jest przykładem **algorytmu przeszukiwania z nawrotami**. Jest to ogólna technika rozwiązywania problemów polegająca na usystematyzowanym (nielosowym) przeglądaniu możliwych rozwiązań (w naszym przypadku dróg w labiryncie). Gdy okazuje się, że kandydat na rozwiązanie nie jest właściwy, algorytm powraca do punktu, w którym może modyfikować rozwiązanie. W naszym programie, gdy okazuje się, że wytyczana droga nie prowadzi do wyjścia, program wraca do pola, w którym może modyfikować drogę, i sprawdza kolejnego kandydata.

Zwróć uwagę, że jeśli istnieje droga prowadząca do wyjścia z labiryntu, to omówiony algorytm ją znajdzie, ale nie zawsze będzie to droga najkrótsza. Na przykład dla planszy i pola startowego przedstawionych na rysunku 2.6 można wskazać krótszą drogę.

Warto wiedzieć

Innym przykładem problemu, który można rozwiązać metodą z nawrotami, jest problem ustawienia na szachownicy 8 hetmanów tak, aby się nawzajem nie szachowały.

Zapamiętaj

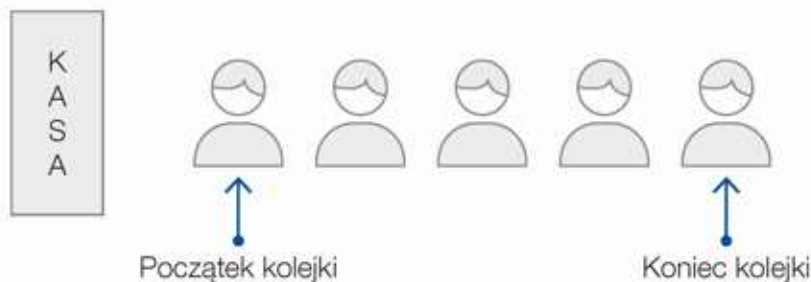
Rekurencyjny algorytm znajdowania drogi do wyjścia z labiryntu jest przykładem algorytmu z nawrotami. Jeśli kandydat na rozwiązanie nie jest właściwy, algorytm powraca do punktu, w którym może modyfikować rozwiązanie. Podany algorytm zawsze znajdzie drogę, o ile ona istnieje, ale niekoniecznie będzie to droga najkrótsza.

2.3. Czym jest kolejka?

Drogę do wyjścia z labiryntu można znaleźć za pomocą innego algorytmu, który wykorzystuje strukturę danych nazywaną **kolejką**. Kolejka (ang. *queue*) to **dynamiczna struktura danych**, w której nowy element można dodać tylko na końcu, a usunąć można tylko element znajdujący się z przodu. W kolejce przechowuje się dane tego samego typu. Zasada przy dodawaniu i usuwaniu elementów jest taka jak w przypadku kolejki klientów do kasy. Ten klient, który pierwszy stanął w kolejce, jako pierwszy będzie obsłużony. Nowy klient może stanąć jedynie na końcu kolejki (rys. 2.7).

Kolejka

Dynamiczna struktura danych, s. 12 



Rys. 2.7. Kolejka do kasy

Kolejka jest strukturą danych, która realizuje strategię **FIFO** (od ang. **F**irst **I**n, **F**irst **O**ut – pierwszy wchodzi, pierwszy wychodzi). Możemy wyróżnić następujące **operacje na kolejce**:

Operacje na kolejce

- ▶ *push* – umieszczenie elementu na końcu kolejki,
- ▶ *pop* – usunięcie elementu z początku kolejki,
- ▶ *front* – pobranie wartości elementu z początku kolejki,
- ▶ *empty* – sprawdzenie, czy kolejka jest pusta.

Zapamiętaj

Kolejka jest dynamiczną strukturą danych, a więc jej rozmiar może się zmieniać w trakcie działania programu. Operacje na kolejce wykonuje się na jej początku i końcu. Do operacji tych należą: dodanie elementu na końcu kolejki (*push*), usunięcie elementu z początku kolejki (*pop*), pobranie wartości elementu z początku kolejki (*front*) oraz sprawdzenie, czy kolejka jest pusta (*empty*).

Warto wiedzieć

Kolejkę często wykorzystuje się w systemach operacyjnych. Ustawia się w niej m.in. zadania oczekujące na wydrukowanie lub przetworzenie przez procesor.

2.4. Iteracyjny algorytm znajdowania wyjścia z labiryntu

Wadą rekurencyjnego algorytmu znajdującego drogę prowadzącą do wyjścia z labiryntu jest nierówne traktowanie poszczególnych pól. Uprzywilejowane jest to sąsiednie pole, dla którego najpierw następuje wywołanie rekurencyjne. Jeśli przy tym wywołaniu algorytm znajdzie drogę do wyjścia, to ta droga jest wynikiem działania algorytmu. Nie musi być to jednak droga najkrótsza. Być może krótszą drogę udało by się znaleźć, jeśli przeszlibyśmy na inne sąsiednie pole. Wszystkie sąsiednie pola powinny być więc traktowane tak samo.

Dla danego pola istnieją maksymalnie cztery sąsiednie pola, na które można przejść. Wszystkich ich równocześnie nie rozpatrzymy, więc współrzędne tych, na które można przejść, umieścimy w kolejce. Odwiedzone pola tak jak poprzednio będziemy oznaczać kolejnymi liczbami całkowitymi dodatnimi. Na początku do kolejki wstawiamy współrzędne pola startowego i oznaczamy to pole wartością 1. Następnie, dopóki nie znajdziemy wyjścia i są jakieś pola w kolejce do rozpatrzenia (kolejka nie jest pusta), pobieramy pole z kolejki i sprawdzamy, czy jest wyjściem. Jeśli nie jest, to rozpatrujemy sąsiednie pola. Jeśli nie zostały jeszcze odwiedzone, to oznaczamy je odpowiednią liczbą i umieszczamy w kolejce.

Oto zapis algorytmu w pseudokodzie:

```

wyjście ← fałsz
wstaw pole (w,k) do kolejki
Lab[w][k] ← 1
dopóki nie wyjście oraz nie pusta kolejka wykonuj
    przypisz w i k współrzędne pola z początku kolejki
    usuń pole z kolejki
    jeśli w = 0 lub w = n - 1 lub k = 0 lub k = n - 1 to
        wyjście ← prawda
    w przeciwnym przypadku
        jeśli Lab[w-1][k] = 0 to
            Lab[w-1][k] ← Lab[w][k] + 1
            wstaw pole (w-1,k) do kolejki
        jeśli Lab[w+1][k] = 0 to
            Lab[w+1][k] ← Lab[w][k] + 1
            wstaw pole (w+1,k) do kolejki
        jeśli Lab[w][k-1] = 0 to
            Lab[w][k-1] ← Lab[w][k] + 1
            wstaw pole (w,k-1) do kolejki
        jeśli Lab[w][k+1] = 0 to
            Lab[w][k+1] ← Lab[w][k] + 1
            wstaw pole (w,k+1) do kolejki

```

Rysunek 2.8 przedstawia labirynt z polami, które algorytm wykorzystujący kolejkę oznaczy liczbami całkowitymi dodatnimi, startując z pola (1,14).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	-1	15	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	14	13	12	11	10	9	8	7	6	5	4	3	2	1	2	3	4	5	-1
2	-1	15	-1	-1	-1	-1	-1	-1	-1	-1	6	-1	-1	-1	2	-1	-1	-1	-1	-1
3	-1	0	-1	-1	-1	-1	-1	12	-1	-1	-1	-1	-1	-1	3	-1	-1	-1	-1	-1
4	-1	0	16	15	14	13	12	11	10	9	8	7	6	5	4	5	6	7	-1	-1
5	-1	0	-1	-1	-1	14	-1	-1	11	-1	-1	-1	-1	-1	-1	-1	7	-1	-1	-1
6	-1	0	0	0	16	15	14	13	12	13	14	13	12	11	10	9	8	9	10	-1
7	-1	0	-1	-1	-1	-1	-1	-1	13	-1	-1	-1	-1	-1	11	-1	-1	-1	-1	-1
8	-1	0	-1	0	0	0	16	15	14	15	16	15	14	13	12	13	14	15	16	-1
9	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	15	-1	-1	-1	-1	-1	-1	-1
10	-1	0	-1	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	-1
11	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1
12	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
13	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
14	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
15	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1
16	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
17	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1
18	-1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
19	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Rys. 2.8. Labirynt z polami, które algorytm wykorzystujący kolejkę oznaczy liczbami całkowitymi dodatnimi, gdy polem startowym jest (1,14)

Tabela 2.1 przedstawia elementy w kolejce dla 14 początkowych iteracji pętli, gdy polem startowym jest pole (1,14).

Iteracja pętli	Rozpatrywane pole	Stan kolejki
–	–	(1,14)
1	(1,14)	(2,14); (1,13); (1,15)
2	(2,14)	(1,13); (1,15); (3,14)
3	(1,13)	(1,15); (3,14); (1,12)
4	(1,15)	(3,14); (1,12); (1,16)
5	(3,14)	(1,12); (1,16); (4,14)
6	(1,12)	(1,16); (4,14); (1,11)
7	(1,16)	(4,14); (1,11); (1,17)
8	(4,14)	(1,11); (1,17); (4,13); (4,15)
9	(1,11)	(1,17); (4,13); (4,15); (1,10)
10	(1,17)	(4,13); (4,15); (1,10); (1,18)
11	(4,13)	(4,15); (1,10); (1,18); (4,12)
12	(4,15)	(1,10); (1,18); (4,12); (4,16)
13	(1,10)	(1,18); (4,12); (4,16); (2,10); (1,9)
14	(1,18)	(4,12); (4,16); (2,10); (1,9)

Tabela 2.1. Elementy w kolejce przy pierwszych 14 iteracjach pętli dla przykładu z rysunku 2.8

Warto wiedzieć

Algorytm znajdujący najkrótszą drogę do wyjścia z labiryntu z wykorzystaniem kolejki rozpatruje tylko pola znajdujące się w odległości od pola startowego nie większej niż długość najkrótszej drogi. W przykładzie z rysunku 2.8 pola z liczbą 16 zostały oznaczone tą liczbą i dołączone do kolejki, ale nie były rozpatrywane.

Pierwszy wiersz tabeli 2.1 ze s. 41 to stan przed rozpoczęciem wykonywania pętli. Rozpatrzenie pola startowego powoduje dodanie trzech elementów do kolejki. Zbadanie pól (4,14) oraz (1,10) sprawia, że do kolejki dodajemy za każdym razem dwa elementy. Po rozpatrzeniu pola (1,18) nie dopiszemy do kolejki żadnego pola. Pozostałe pola z tabeli 2.1 powodują, że do kolejki dodajemy po jednym polu.

Aby zaimplementować omówiony algorytm wyznaczania najkrótszej drogi do wyjścia z labiryntu, wykorzystamy szablon kolejki z biblioteki STL. Należy wtedy dodać do programu bibliotekę `queue` za pomocą

Deklaracja kolejki • dyrektywy `#include <queue>`. Ogólna postać **deklaracji kolejki** jest następująca:

```
queue<typ elementów kolejki> nazwa_kolejki;
```

Typ `queue` jest typem obiektowym, a więc udostępnia metody do przetwarzania danych. Należą do nich m.in. **metody** `push`, `pop`, `front`, `empty` dla klasy `queue`. Działanie tych metod odpowiada wymienionym wcześniej **operacjom na kolejce**. Parametrem metody `push` jest element, który chcemy umieścić na końcu kolejki. Metody `pop`, `front`, `empty` są bezparametrowe.

Operacje na kolejce,
s. 39 [🔗](#)

Elementami pamiętanymi w kolejce są współrzędne pól, a więc dwie liczby całkowite. Zdefiniujemy w związku z tym typ reprezentujący pojedyncze pole jako **strukturę**:

Struktura, podręcznik
Informatyka na czasie 2.
Zakres rozszerzony,
s. 109 [🔗](#)

```
struct pole
{
    int w, k;
};
```

Oto kod źródłowy funkcji `Droga`, realizującej omawiany algorytm:

Fragment kodu •
źródłowego programu
znajdującego
najkrótszą drogę
do wyjścia z labiryntu
(z wykorzystaniem
kolejki) – funkcja `Droga`

```
1. bool Droga(int Lab[][N], pole p1, pole &p2)
2. {
3.     int w, k;
4.     bool wyjscie=false;
5.     queue<pole> Q;
6.     Q.push(p1);
7.     Lab[p1.w][p1.k]=1;
8.     while (!wyjscie && !Q.empty())
9.     {
10.         p2=Q.front(); Q.pop(); w=p2.w; k=p2.k;
11.         if (w==0 || w==N-1 || k==0 || k==N-1)
12.             wyjscie=true;
13.         else
14.         {
15.             if (Lab[w-1][k]==0)
16.             {
17.                 Lab[w-1][k]=Lab[w][k]+1;
18.                 p2.w=w-1; p2.k=k; Q.push(p2);
19.             }
```

```

20.         if (Lab[w+1][k]==0)
21.         {
22.             Lab[w+1][k]=Lab[w][k]+1;
23.             p2.w=w+1; p2.k=k;
24.             Q.push(p2);
25.         }
26.         if (Lab[w][k-1]==0)
27.         {
28.             Lab[w][k-1]=Lab[w][k]+1;
29.             p2.w=w; p2.k=k-1;
30.             Q.push(p2);
31.         }
32.         if (Lab[w][k+1]==0)
33.         {
34.             Lab[w][k+1]=Lab[w][k]+1;
35.             p2.w=w; p2.k=k+1;
36.             Q.push(p2);
37.         }
38.     }
39. }
40. return wyjscie;
41. }

```

Warto wiedzieć

W przedstawionym programie w każdej z kolejnych instrukcji **if** trzeba aktualizować obie współrzędne pola p2, ponieważ mogły zostać zmienione w poprzedniej instrukcji **if**.

Zwróć uwagę na parametry funkcji Droga (linia 1). Parametr p1 (przekazany przez wartość) jest polem startowym, a parametr p2 (przekazany przez referencję) to znalezione współrzędne pola będącego wyjściem z labiryntu, pod warunkiem że droga istnieje, czyli wartością funkcji **Droga** jest **true**.

W linii 5 jest zadeklarowana kolejka o nazwie Q, służąca do pamiętania elementów typu pole. W linii 6 parametr p1 (przechowujący współrzędne pola startowego) jest wstawiany do kolejki, a w linii 7 pole startowe jest oznaczane liczbą 1. Pętla w liniach 8–39 pobiera najpierw z kolejki pierwszy element i w pomocniczych zmiennych w i k zapisuje współrzędne aktualnie rozpatrywanego pola (linia 10). Jeśli jest to pole będące wyjściem z labiryntu, pętla kończy działanie (zmienna logiczna wyjscie przyjmuje wartość **true** – linia 12). W przeciwnym przypadku badane są cztery sąsiednie pola (linie 13–38). Jeśli dane pole nie zostało jeszcze odwiedzone (jest oznaczone liczbą 0), to jest mu przypisywana wartość o jeden większa od wartości zapisanej dla bieżącego pola, a następnie pole jest dopisywane na końcu kolejki.

Kod źródłowy funkcji main wykorzystującej funkcję Droga znajduje się na s. 44. Funkcje WczytajLabirynt, WypiszLabirynt i OznaczDroge są takie same jak w programie rekurencyjnym szukającym drogi w labiryncie.

Rysunek 2.9 na s. 44 przedstawia przykład wywołania programu znajdującego najkrótszą drogę prowadzącą do wyjścia z labiryntu, jeśli polem startowym jest pole (16,10).

Fragment kodu
źródłowego programu
znajdującego
najkrótszą drogę
do wyjścia z labiryntu
(z wykorzystaniem
kolejki) – funkcja main

```
1. int main()
2. {
3.     int Lab[N][N];
4.     pole p1, p2;
5.     WczytajLabirynt(Lab);
6.     WypiszLabirynt(Lab);
7.     cout<<"Wspolrzedne pola startowego: "<<endl;
8.     cout<<"w= "; cin>>p1.w;
9.     cout<<"k= "; cin>>p1.k;
10.    if (Droga(Lab,p1,p2))
11.    {
12.        OznaczDroge(Lab,p2.w,p2.k);
13.        WypiszLabirynt(Lab);
14.    }
15.    else cout<<"Brak drogi";
16.    return 0;
17. }
```

Warto wiedzieć

Może istnieć kilka najkrótszych dróg, jednak algorytm znajdzie tylko jedną z nich. Na przykład na rysunku 2.9 z pola (6,8) można przejść na pole (5,8), potem na pole (4,8), a następnie w lewo do pola (4,1) i dalej do wyjścia.

Wspolrzedne pola startowego:
w = 16
k = 10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	X	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
1	X	D																		X
2	X	D	X	X	X	X	X	X	X	X		X	X	X		X	X	X	X	X
3	X	D	X	X	X	X	X		X	X	X	X	X	X		X	X	X	X	X
4	X	D																	X	X
5	X	D	X	X	X		X	X		X	X	X	X	X	X	X		X	X	X
6	X	D	D	D	D	D	D	D	D											X
7	X		X	X	X	X	X	X	D	X	X	X	X	X		X	X	X	X	X
8	X		X			D	D	D	D							X	X	X	X	X
9	X		X	X	X	D	X	X	X	X	X		X	X	X	X	X	X	X	X
10	X		X			D	D	D	D											X
11	X		X	X	X	X	X	X	D	X	X	X	X	X	X		X	X	X	X
12	X		X						D	D	D	D								X
13	X		X	X	X	X	X	X	X	X	X	D	X	X	X	X	X	X	X	X
14	X		X									D								X
15	X		X	X	X	X	X		X	X	X	D	X	X	X	X	X	X	X	X
16	X		X								D	D								X
17	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X
18	X	X																		X
19	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Rys. 2.9. Labirynt z zaznaczoną najkrótszą drogą do wyjścia z pola (16,10)

Ćwiczenie 4

Napisz program, który znajdzie najkrótszą drogę prowadzącą od wczytanego pola do wyjścia z labiryntu oraz wypisze planszę ze znalezioną drogą. Jeśli droga prowadząca do wyjścia nie istnieje, program wyświetli odpowiedni komunikat. Opis labiryntu wczytaj z pliku otrzymanego od nauczyciela (np. *labirynt.txt*).

Podsumowanie

- Rekurencyjny algorytm znajdowania drogi wyjścia z labiryntu polega na rekurencyjnym wywoływaniu funkcji poszukującej drogi dla tych sąsiednich pól, na które można stanąć i które nie były jeszcze rozpatrywane.
- Algorytm rekurencyjny znajdowania drogi wyjścia z labiryntu nie musi znaleźć najkrótszej możliwej drogi.
- Rekurencyjny algorytm poszukujący drogi wyjścia z labiryntu jest przykładem algorytmu przeszukiwania z nawrotami. Algorytm taki przegląda możliwe rozwiązania, a gdy okazuje się, że kandydat na rozwiązanie nie jest właściwy, powraca do punktu, w którym może modyfikować rozwiązanie.
- Kolejka to dynamiczna struktura danych typu FIFO (ang. *first in, first out*), w której operacje są wykonywane na jej początku i końcu. Z kolejki można pobrać jedynie element znajdujący się na początku. Nowy element można dopisać tylko na końcu kolejki.
- Przykładowe operacje na kolejce to: umieszczenie elementu na końcu kolejki (*push*), usunięcie elementu z początku kolejki (*pop*), pobranie wartości pierwszego elementu (*front*) i sprawdzenie, czy kolejka jest pusta (*empty*).
- Najkrótszą drogę do wyjścia z labiryntu można znaleźć za pomocą algorytmu iteracyjnego z wykorzystaniem kolejki. W kolejce przechowywane są współrzędne pól, które należy jeszcze rozpatrzyć.

Zadania

- * **1** Napisz program symulujący kolejkę do kasy. Wprowadzenie z klawiatury liczby 1 oznacza, że na końcu kolejki staje klient, a wpisanie liczby -1 oznacza, że klient z początku kolejki został obsłużony. Podanie liczby 0 kończy wczytywanie. Program powinien wypisać największą liczbę klientów, którzy jednocześnie stali w kolejce. Na przykład dla liczb 1 1 1 -1 1 1 -1 0 program zwróci liczbę 4. Załóż, że wprowadzane dane są poprawne, a więc nie zostanie podana liczba -1, gdy w kolejce nikt nie stoi.
- * **2** Przedstawiony w tym temacie algorytm znajdujący najkrótszą drogę do wyjścia z labiryntu zmodyfikuj tak, aby do przechowania pól labiryntu, które należy rozpatrzyć, zamiast kolejki zastosować stos. Czy zmodyfikowany algorytm zawsze znajdzie drogę do wyjścia w labiryncie, jeśli ona istnieje? Czy będzie to najkrótsza droga?
- * **3** Zaprojektuj planszę z labiryntem o n wierszach i m kolumnach, a następnie utwórz plik tekstowy opisujący tę planszę. Napisz program, który znajdzie najkrótszą drogę w labiryncie o n wierszach i m kolumnach, a następnie sprawdź jego działanie, korzystając z przygotowanego pliku.

- ★★ **4** W omówionym w tym temacie programie znajdującym najkrótszą drogę do wyjścia z labiryntu zastąp kolejkę stosem. Wkładaj na stos pola w takiej kolejności, aby uzyskać taką samą drogę jak za pomocą programu rekurencyjnego przedstawionego w tym temacie (dla takiej samej planszy). Plik tekstowy z opisem planszy przekaże ci nauczyciel (np. *labirynt.txt*).

- ★★ **5** Przyjmij następującą reprezentację planszy z labiryntem: tablica dwuwymiarowa składająca się z elementów typu `pole`.

```
struct pole
{
    int x;
    bool D, G, L, P;
};
```

Na każdym polu można stanąć. O tym, czy można przejść na sąsiednie pole, decydują wartości logiczne określające cztery kierunki: D – dół, G – góra, L – lewo, P – prawo. Napisz program znajdujący najkrótszą drogę wyjścia z labiryntu. Dane planszy odczytaj z pliku tekstowego, który otrzymasz od nauczyciela (np. *labirynt_logiczny.txt*). W pliku znajduje się opis planszy o wymiarach 10×10 . Jeden wiersz pliku opisuje jeden wiersz planszy i składa się z 40 znaków. Jedno pole opisane jest czterema cyframi 0 lub 1 określającymi kolejno wartości D, G, L, P.

- ★★★ **6** Poszukaj w dostępnych źródłach informacji na temat algorytmów automatycznego generowania plansz labiryntów, a następnie napisz program generujący taką planszę. Przyjmij następującą reprezentację planszy: tablica dwuwymiarowa składająca się z elementów typu `pole`.

```
struct pole
{
    int x;
    bool D, G, L, P;
};
```

Na każdym polu można stanąć. O tym, czy można przejść na sąsiednie pole, decydują wartości logiczne określające cztery kierunki: D – dół, G – góra, L – lewo, P – prawo.

- ★★★ **7** Napisz program rozwiązujący problem ustawienia hetmanów na szachownicy o wymiarach $n \times n$ (n – dana liczba całkowita dodatnia). Należy na niej ustawić n figur hetmanów tak, aby się nie szachowały. Program powinien wypisać współrzędne n pól, na których należy ustawić hetmanów.

3. Wykorzystanie list w rozwiązywaniu problemów

Na pewno z dzieciństwa znacie wiele wyliczanek. Być może podczas zabawy używaliście ich do wytypowania osoby, która ma wykonać jakieś zadanie. Pewna znana z historii wyliczanka stała się inspiracją do sformułowania problemu matematycznego. Zajmiemy się nim w tym temacie. Wykorzystamy przy tym dynamiczną strukturę danych o nazwie lista. Użyjemy jej także do sortowania słów według porządku, jaki stosuje się m.in. w słownikach.

Cele lekcji

- Dowiesz się, czym jest lista, i poznasz różne rodzaje list.
- Zrozumiesz, na czym polega problem Flawiusza.
- Wykorzystasz listę do symulacji problemu Flawiusza oraz porządkowania słów leksykograficznie.

3.1. Czym jest lista?

Lista (ang. *list*) to **dynamiczna struktura danych**, w której dostęp do elementów jest sekwencyjny. Oznacza to, że od danego elementu listy można przejść bezpośrednio do elementu sąsiedniego. Aby dostać się do elementu znajdującego się na i -tym miejscu listy, trzeba przejść przez wszystkie elementy, które nas od niego oddzielają. Nowy element można wstawić w dowolne miejsce listy, można też usunąć dowolny element z listy. W liście przechowuje się dane tego samego typu. Struktury danych **stos** i **kolejka** są szczególnymi przypadkami listy.

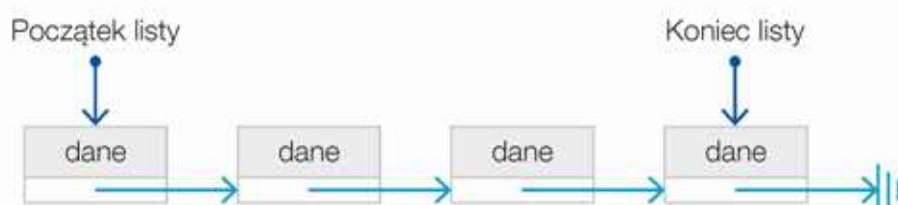
Dynamiczna struktura danych,
s. 12 [🔗](#)

Stos,
s. 11 [🔗](#)
Kolejka,
s. 39 [🔗](#)

Rodzaje list

Rysunek 3.1 przedstawia przykład **listy jednokierunkowej**. W takiej liście każdy element przechowuje oprócz danych informację o tym, który element jest następny. Listę jednokierunkową można przeglądać tylko w jedną stronę: od początku do końca. Zatem z danego elementu listy możemy uzyskać dostęp tylko do następnych elementów, poprzednie nie są dostępne.

Lista jednokierunkowa



Rys. 3.1. Przykład listy jednokierunkowej złożonej z czterech elementów

Warto wiedzieć

W graficznym przedstawieniu listy do zaznaczenia końca danych czasami używa się znanego np. z elektroniki symbolu uziemienia.

Jeśli ostatni element listy jednokierunkowej zawiera informację o pierwszym elemencie, to taka lista jest **listą jednokierunkową cykliczną** (rys. 3.2).



Rys. 3.2. Przykład listy jednokierunkowej cyklicznej złożonej z czterech elementów

Innym przykładem listy jest **lista dwukierunkowa**. Można ją przeglądać w dwóch kierunkach: od początku do końca i od końca do początku. Każdy element takiej listy zawiera informacje zarówno o tym, który element jest następny, jak i o tym, który jest poprzedni (rys. 3.3).

Warto wiedzieć

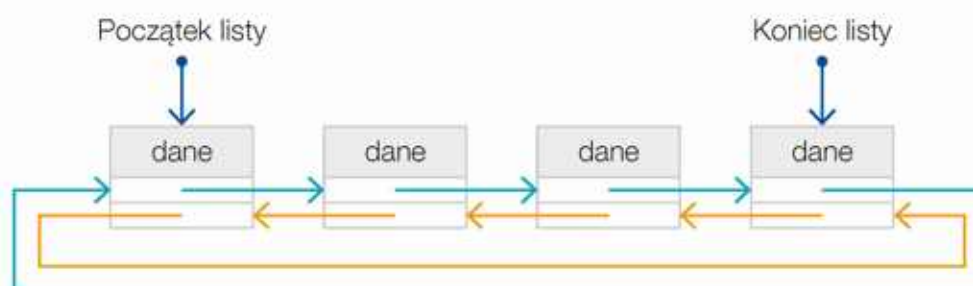
Lista dwukierunkowa zajmuje więcej pamięci komputera od listy jednokierunkowej, ponieważ w każdym elemencie oprócz informacji o elemencie następnym trzeba przechowywać informację o poprzednim.



Rys. 3.3. Przykład listy dwukierunkowej złożonej z czterech elementów

Jeśli w liście dwukierunkowej pierwszy element zawiera informację o tym, który element jest ostatni, a ostatni o tym, który jest pierwszy,

to mamy do czynienia z **listą dwukierunkową cykliczną** (rys. 3.4).



Rys. 3.4. Przykład listy dwukierunkowej cyklicznej złożonej z czterech elementów

Warto wiedzieć

Lista nie wymaga ciągłego obszaru pamięci komputera, ponieważ jej elementy nie muszą być przechowywane obok siebie. Nawet jeśli są tak przechowywane, to kolejność elementów na liście nie musi być taka sama jak w pamięci. Inaczej jest w przypadku tablicy – jej kolejne elementy muszą być zapisane w sąsiednich miejscach pamięci komputera.

Zapamiętaj

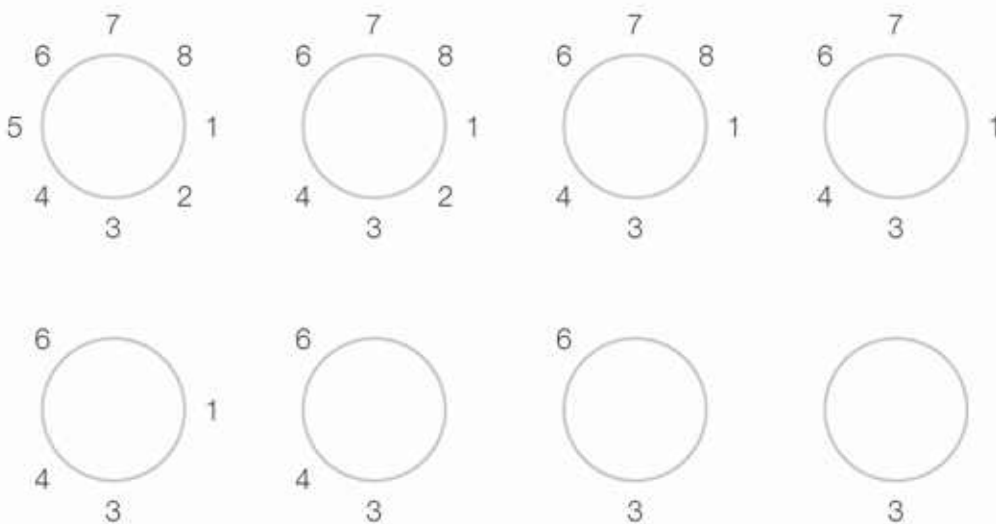
Lista jest dynamiczną strukturą danych o sekwencyjnym dostępie do elementów. Nowy element można wstawić w dowolnym miejscu listy, można też usunąć z niej dowolny element. W zależności od tego, jak możemy przeglądać elementy listy, wyróżniamy listy jedno- i dwukierunkowe. Dodatkowo oba rodzaje list mogą być listami cyklicznymi.

3.2. Symulacja problemu Flawiusza

Wyobraź sobie, że przy okrągłym stole siedzi n osób. Krzesła, na których siedzą, są ponumerowane od 1 do n . Cyklicznie co k -ta osoba wstaje, odstawia swoje krzesło i wychodzi. Odliczanie rozpoczynamy od pierwszej osoby, a więc jako pierwsza wyjdzie osoba siedząca na krześle o numerze k . Dalej odliczamy w tym samym kierunku. Należy wskazać osobę (numer jej krzesła), która jako ostatnia wstanie od stołu.

To zadanie jest jedną z wersji **problemu Józefa Flawiusza**. Ogólnie polega on na tym, że ze zbioru n elementów usuwa się co k -ty i trzeba określić, który element zostanie usunięty jako ostatni. Rysunek 3.5 przedstawia symulację problemu dla 8 osób, gdy od stołu wstaje co 5. osoba.

• Problem Józefa Flawiusza



Rys. 3.5. Przykład symulacji problemu dla 8 osób, gdy co 5. wstaje od stołu

Ćwiczenie 1

Rozważ opisany problem dla 7 osób, gdy co 3. wstaje od stołu.

A to ciekawe

Wyliczanka na śmierć i życie

Nazwa problemu Józefa Flawiusza pochodzi od imienia i nazwiska żydowskiego historyka z I w. n.e. W powstaniu Żydów przeciwko Rzymianom dowodził on obroną Jotapaty. Według niektórych źródeł, gdy miasto się poddało, ukrył się wraz z grupą powstańców. Ponieważ nie chcieli trafić do niewoli, a religia zabraniała im samobójstwa, zdecydowali, że będą losować, kto kogo ma zabić. Odliczali co określoną liczbę i każda wyznaczona osoba miała zginąć z rąk następnej wyznaczonej osoby. Ten, kto zostanie, miał sam odebrać sobie życie. Gdy jednak ostatnimi dwoma okazali się Flawiusz z jednym z towarzyszy, obaj wybrali niewolę.



Warto wiedzieć

Według niektórych źródeł pierwszą osobą, która przekształciła historię Józefa Flawiusza w problem matematyczny, był szesnastowieczny francuski matematyk Claude-Gaspard Bachet de Méziriac.

Oto specyfikacja problemu Flawiusza:

Specyfikacja

Dane: n – liczba całkowita dodatnia określająca liczbę elementów położonych na okręgu,

k – liczba całkowita dodatnia określająca, co który element jest usuwany.

Wynik: m – liczba całkowita dodatnia określająca numer elementu, który zostanie usunięty jako ostatni.

Założmy, że mamy cykliczną listę jednokierunkową o nazwie *lista* złożoną z n elementów pamiętających kolejne liczby całkowite dodatnie od 1 do n . Rysunek 3.6 przedstawia przykład takiej listy dla $n = 8$.



Rys. 3.6. Cykliczna lista jednokierunkowa złożona z liczb od 1 do 8

Ponieważ chcemy wyznaczyć element, który będzie usunięty jako ostatni, trzeba usunąć z listy $n - 1$ elementów, czyli doprowadzić do sytuacji, że lista będzie się składać z jednego elementu. Dlatego w algorytmie symulującym problem Flawiusza liczba powtórzeń zewnętrznej pętli będzie równa $n - 1$. Żeby wyznaczyć kolejny element do usunięcia, trzeba przesunąć się na liście o $k - 1$ elementów. Na przykład jeśli $k = 2$, to za pierwszym razem usuwamy drugi element, czyli należy się ustawić na następnym elemencie za pierwszym (przesunąć o jedną pozycję). Zapis algorytmu w pseudokodzie może być następujący:

```

dla i ← 1, 2, ..., n - 1 wykonuj
    dla j ← 1, 2, ..., k - 1 wykonuj
        przejdź do następnego elementu listy
    usuń bieżący element z listy
  
```

3.3. Program symulujący problem Flawiusza

W programie symulującym problem Flawiusza użyjemy szablonu **list**

Deklaracja listy z biblioteki STL. Ogólna **deklaracja listy** jest następująca:

```
list<typ elementów listy> nazwa_listy;
```

Korzystanie z typu **list** jest możliwe po dołączeniu do programu biblioteki **list** (dyrektywa `#include <list>`). Typ **list** reprezentuje **listę dwukierunkową**. My będziemy przeglądać listę tylko w jednym kierunku. Ponieważ typ **list** nie reprezentuje listy cyklicznej, po rozpoznaniu końca listy zaczniemy ponowne jej przeglądanie od początku.

Lista dwukierunkowa,
s. 48

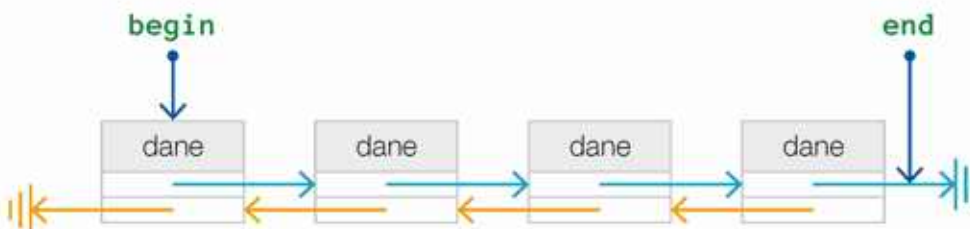
Dostęp do elementów listy jest możliwy dzięki iteratorom. **Iterator** • **Iterator** jest zmienną pozwalającą na sekwencyjne przeglądanie zawartości struktury danych z biblioteki STL. Najczęściej zawiera wskaźnik (adres w pamięci) danego elementu struktury danych.

Ogólna postać **deklaracji iteratora dla typu list** jest następująca: • **Deklaracja iteratora dla typu list**

`list<typ elementów listy>::iterator nazwa_iteratora;`

Dostępne są różne metody, które zwracają iteratory. Dwie z nich, obie bezparametrowe, to:

- ▶ **metoda begin** – zwraca iterator wskazujący na pierwszy element listy; jeśli lista jest pusta, metoda zwróci ten sam iterator co metoda **end**; • **Metoda begin dla klasy list**
- ▶ **metoda end** – zwraca iterator wskazujący miejsce za ostatnim elementem na liście; ponieważ iterator nie wskazuje żadnego elementu listy, nie należy wykonywać operacji na wyniku metody **end**. • **Metoda end dla klasy list**



Rys. 3.7. Wskazania iteratorów **begin** i **end** dla listy dwukierunkowej

Na przykład przejrzanie elementów listy o nazwie `lista` i wypisanie wartości wszystkich tych elementów zrealizują następujące instrukcje:

```
list<int> lista;
list<int>::iterator it;
...
for (it=lista.begin(); it!=lista.end(); it++)
    cout<<*it;
```

Żeby odwołać się do wartości elementu wskazywanego przez iterator, należy przed jego nazwą umieścić znak `*`. Oznacza to pobranie wartości wskazywanej przez iterator (pamiętanej pod danym adresem). Inkrementacja iteratora (czyli powiększenie jego wartości o 1: `it++`) w pętli **for** oznacza przejście do następnego elementu listy.



Rys. 3.8. Fragment listy z wartościami iteratorów `it`, `it - 1` oraz `it + 1`

Warto wiedzieć

W przypadku listy dwukierunkowej (typu `list`) można wykonywać także dekrementację iteratora (pomniejszanie go o 1), co oznacza przejście do poprzedniego elementu listy.

Kod źródłowy funkcji `main` programu symulującego problem Flawiusza, zgodnie ze specyfikacją podaną na s. 50, może być następujący.

Fragment kodu 
źródłowego programu
symulującego problem
Flawiusza – funkcja main

Warto wiedzieć

Aby wypisać wartość elementu będącego rozwiązaniem problemu Flawiusza, można także skorzystać z iteratora wskazującego pierwszy element listy:

*`lista.begin()`
 lub z bezparametrowej metody `front`, której wartością jest pierwszy element listy:
`lista.front()`

```
1. int main()
2. {
3.     int n, k;
4.     list<int> lista;
5.     list<int>::iterator it;
6.     cout<<"n = "; cin>>n;
7.     cout<<"k = "; cin>>k;
8.     for (int i=1;i<=n;i++) lista.push_back(i);
9.     it=lista.begin();
10.    for (int i=1;i<n;i++)
11.    {
12.        for (int j=1;j<k;j++)
13.        {
14.            it++;
15.            if (it==lista.end()) it=lista.begin();
16.        }
17.        cout<<"Krok "<<i<<": usuniety element ";
18.        cout<<*it<<endl;
19.        it=lista.erase(it);
20.        if (it==lista.end()) it=lista.begin();
21.    }
22.    cout<<"Ostatni element: "<<*it;
23.    return 0;
24. }
```

Metoda `push_back` **dla klasy `list`**

Dobra rada

Pamiętaj, że każda klasa udostępnia metody umożliwiające przetwarzanie danych. Nazwy metod mogą się powtarzać dla różnych klas, ale sposób korzystania z nich może być różny.

Metoda `erase` dla klasy **`list`**

Dobra rada

Więcej metod dostępnych dla klasy `list` znajdziesz w dodatku 5 na s. 426.

W linii 4 zadeklarowana jest zmienna `lista` typu `list`, która jest listą elementów typu całkowitego, a w linii 5 iterator `it` do przeglądania elementów tej listy. Pętla w linii 8 tworzy listę złożoną z liczb całkowitych od 1 do n . **Metoda `push_back`** powoduje dodanie elementu na końcu struktury danych – dodawany element jest jej parametrem (w tym przypadku ma on wartość i).

Wartością początkową iteratora `it` jest adres (wskaźnik) pierwszego elementu listy (linia 9). Zewnętrzna pętla (linie 10–21) usuwa z listy $n - 1$ elementów. Wewnętrzna pętla (linie 12–16) powoduje ustawienie się na elemencie do usunięcia – przesuwa iterator `it` o $k - 1$ pozycji. W linii 14 następuje przejście do następnego elementu. Jeśli takiego elementu nie ma (zmienna `it` przyjmie wartość zwracaną przez metodę `end`), to ponownie przechodzimy do pierwszego elementu listy (linia 15). W liniach 17–18 wypisywana jest wartość elementu do usunięcia, wskazywanego przez iterator `it`. **Metoda `erase`** powoduje usunięcie elementu listy wskazywanego przez parametr. Wartością tej metody jest iterator wskazujący na następny element za usuwanym (linia 19). W linii 20 ustawiamy się na początku listy, gdy usuwany element był jej ostatnim elementem.

Po zakończeniu wykonywania głównej pętli lista składa się z jednego elementu, którego wartość jest rozwiązaniem problemu. Wskazuje na niego iterator `it`. Instrukcja w linii 22 wypisuje wartość tego elementu.

Ćwiczenie 2

Napisz program symulujący problem Flawiusza zgodnie ze specyfikacją podaną na s. 50. Przetestuj działanie programu.

3.4. Sortowanie leksykograficzne

W słownikach i indeksach, z których korzystacie, hasła są ułożone według **porządku leksykograficznego**, czyli alfabetycznie. Aby uporządkować hasła w ten sposób, porównuje się znaki, z których są złożone, z pominięciem znaków spoza alfabetu (np. spacji, jeśli hasło jest dwuwyrazowe). Dla słów a , b słowo a wystąpi przed słowem b , gdy zachodzi jeden z dwóch przypadków.

1. Słowo a jest początkowym fragmentem słowa b . Na przykład słowo *kot* jest słowem wcześniejszym od słowa *kotwica*.
2. Na co najmniej jednej pozycji występuje znak różniący słowa a i b . Wówczas słowo a znajdzie się przed słowem b , jeśli na pierwszej różniącej je pozycji (od lewej) ma znak wcześniejszy, np. *kotlina* zapiszemy przed *kotwica* – decydują o tym litery na czwartej pozycji.

W ogólnym przypadku **sortowanie leksykograficzne** polega na porządkowaniu ciągów złożonych z określonych elementów. O tym, który z dwóch ciągów trzeba zapisać wcześniej, decyduje pierwsza pozycja od lewej różniąca te dwa ciągi. Jeśli taka pozycja nie istnieje, to jeden ciąg jest początkowym fragmentem drugiego.

Rozwiążemy problem sortowania leksykograficznego słów złożonych z liter pełnego polskiego alfabetu. W praktyce alfabet możemy zdefiniować dowolnie – wystarczy określić zbiór znaków go tworzących i kolejność ich występowania.

Sortowanie leksykograficzne słów

Do uporządkowania leksykograficznego słów wykorzystamy **sortowanie kubełkowe**. Polega ono na grupowaniu elementów mających wspólne cechy w tzw. kubełkach, a następnie przepisywaniu ich we właściwej kolejności. Wspólną cechą słów będzie taka sama litera na tej samej pozycji. Utworzymy po jednym kubełku dla każdej litery występującej w wyrazach. Kubełki będą ustawione w kolejności alfabetycznej. Założymy, że słowa są pamiętane w liście. Usuwamy wyrazy z listy i wkładamy do odpowiednich kubełków. Następnie usuwamy słowa z kolejnych kubełków i dopisujemy je na końcu listy. Jeśli w kubełku znajduje się kilka wyrazów, to dopisujemy je do listy w kolejności, w jakiej były wkładane do kubełka.

Litery słów będziemy przeglądać od prawej do lewej. Najpierw rozpatrzemy najdłuższe wyrazy – uporządkujemy je według liter na ostatniej pozycji. Następnie posortujemy słowa mające literę na pozycji o jeden mniejszej itd. – aż do uporządkowania słów względem pierwszej pozycji.

● **Porządek leksykograficzny**

Warto wiedzieć

Jeśli słowo a jest początkowym fragmentem słowa b , to taki fragment nazywamy prefiksem. Z kolei jeśli słowo a jest końcowym fragmentem słowa b , to taki fragment nazywamy sufiksem.

● **Sortowanie leksykograficzne**

● **Sortowanie kubełkowe**

Algorytm porządkujący słowa leksykograficznie

Uporządkujemy leksykograficznie listę słów: *wir, kra, kwiat, wiatr, świat, świt, kraj*. W tych słowach występuje osiem różnych liter: *a, i, j, k, r, ś, t, w*. Użyjemy zatem ośmiu kubeków reprezentujących te litery. Kubeczki muszą być ustawione w kolejności alfabetycznej. Najdłuższe wyrazy są pięcioliterowe, więc sortowanie rozpoczniemy, uwzględniając litery na piątej pozycji. Sposób postępowania ilustruje krok 1. W kolejnych krokach podobnie porządkujemy słowa według liter z wcześniejszych pozycji.

Lista słów na początku algorytmu:

wir kra kwiat wiatr świat świt kraj

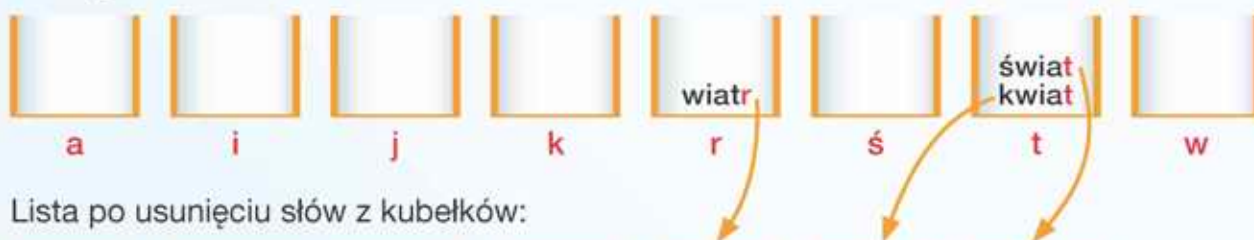
Krok 1

Sortujemy słowa według liter na piątej pozycji. Są trzy wyrazy pięcioliterowe: *kwiat, wiatr* i *świat* – usuwamy je kolejno z listy i każdy umieszczamy w kubku z literą odpowiadającą ostatniej literze wyrazu. Następnie usuwamy wyrazy z kolejnych kubków i dopisujemy je na końcu listy. Z kubka oznaczonego literą *t* usuwamy najpierw słowo, które znalazło się w nim jako pierwsze. Listę słów i kubki przeglądamy od lewej do prawej.

Lista po usunięciu słów mających literę na piątej pozycji:

wir kra świt kraj

Usunięte słowa umieszczone w kubkach:



Lista po usunięciu słów z kubków:

wir kra świt kraj wiatr kwiat świat

Krok 2

Porządkujemy słowa według liter znajdujących się na czwartej pozycji. Usuwamy z listy słowa mające literę na czwartej pozycji i umieszczamy je we właściwych kubkach.

Lista po usunięciu słów mających literę na czwartej pozycji:

wir kra

Usunięte słowa umieszczone w kubkach:



Lista po usunięciu słów z kubków:

wir kra kwiat świat kraj świt wiatr

Krok 3

Porządkujemy słowa według liter na trzeciej pozycji. Wszystkie wyrazy mają literę na trzeciej pozycji, więc po przeniesieniu słów do kubeków lista jest pusta. Słowa przyporządkowujemy do odpowiednich kubków, a potem umieszczamy je na liście we właściwej kolejności.

Słowa umieszczone w kubkach:

wiatr kraj kra	świt świat kwiat			wir			
a	i	j	k	r	ś	t	w

Lista po usunięciu słów z kubków:

kra	kraj	wiatr	kwiat	świat	świt	wir
-----	------	-------	-------	-------	------	-----

Krok 4

Porządkujemy słowa, uwzględniając litery na drugiej pozycji. Podobnie jak w poprzednim kroku, wszystkie wyrazy mają literę na drugiej pozycji, więc po przeniesieniu słów do kubków lista jest pusta.

Słowa umieszczone w kubkach:

	wir wiatr			kraj kra			świt świat kwiat
a	i	j	k	r	ś	t	w

Lista po usunięciu słów z kubków:

wiatr	wir	kra	kraj	kwiat	świat	świt
-------	-----	-----	------	-------	-------	------

Krok 5

Ustalamy kolejność słów na liście według liter na pierwszej pozycji. Jest to ostatni krok algorytmu – nie ma już więcej pozycji do rozpatrzenia.

Słowa umieszczone w kubkach:

			kwiat kraj kra		świt świat		wir wiatr
a	i	j	k	r	ś	t	w

Lista po usunięciu słów z kubków:

kra	kraj	kwiat	świat	świt	wiatr	wir
-----	------	-------	-------	------	-------	-----

Wynikiem działania algorytmu jest lista:

kra	kraj	kwiat	świat	świt	wiatr	wir
-----	------	-------	-------	------	-------	-----

Ćwiczenie 3

Korzystając z algorytmu sortowania kubełkowego, uporządkuj słowa: *krzesło, krzak, krem, kres, kraina, kret, kraksa, kredyt*.

Zapamiętaj

Sortowanie leksykograficzne polega na porządkowaniu ciągów złożonych z określonych elementów. W przypadku porządkowania słów elementami tymi są litery. O tym, który ciąg powinien być zapisany wcześniej, decyduje pierwsza pozycja (od lewej), na której w tych ciągach występują różne elementy. Jeśli taka pozycja nie istnieje, to krótszy ciąg zapisujemy przed dłuższym.

Implementacja algorytmu sortowania leksykograficznego

Sformułujemy najpierw specyfikację problemu leksykograficznego sortowania słów oraz zapiszemy omówiony algorytm w pseudokodzie. Słowa przechowamy w liście, a zawartość kubełków – w kolejkach.

Specyfikacja

Dane: alfabet – napis złożony z uporządkowanych znaków tworzących alfabet,
mdl – liczba całkowita określająca maksymalną długość słowa,
lista – lista słów o maksymalnej długości mdl, złożonych ze znaków napisu alfabet.

Wynik: lista – lista danych słów uporządkowanych leksykograficznie.

```
dla i ← mdl - 1, mdl - 2, ..., 0 wykonuj
    dopóki nie koniec listy wykonuj
        słowo ← aktualny element listy
        jeśli długość słowa > i to
            usuń aktualny element listy
            dodaj słowo do kolejki dla znaku słowo[i]
        dla j ← 0, 1, ..., długość alfabetu - 1 wykonuj
            dopóki nie pusta kolejka
                dla znaku alfabet[j] wykonuj
                    dopisz na końcu listy element
                    z początku kolejki
                    usuń element z początku kolejki
```

Dobra rada

Pamiętaj, że program uwzględniający polskie znaki diakrytyczne zadziała poprawnie, jeśli plik tekstowy wykorzystywany przez program oraz plik *cpp* z kodem źródłowym będą używały tej samej strony kodowej. W środowisku Code::Blocks pracującym w polskiej wersji systemu Windows użyj strony kodowej CP1250.

Do implementacji algorytmu wykorzystamy typ **list** oraz tablicę kolejek typu **queue**. Słowa do posortowania program wczyta z pliku tekstowego – w każdym wierszu pliku znajdzie się jedno słowo. Wynik sortowania program również zapisze w pliku tekstowym.

W programie zdefiniujemy dwie stałe. Jedna z nich będzie określała zestaw znaków tworzących alfabet, druga – liczbę znaków w alfabecie. Wykorzystamy pełny polski alfabet. Definicja tych stałych może wyglądać następująco:

```
const string alfabet="aąbcćdeęfghijklłmnńoópqrsśtuvwxyzźż";
const int N=35;
```

Kody źródłowe funkcji odczytującej słowa z pliku *słowa.txt* i budującej listę oraz funkcji zapisującej słowa do pliku *słownik.txt* mogą wyglądać tak jak poniżej. Podczas odczytywania słów wyliczamy też maksymalną długość słowa.

```
1. void BudujListe(list<string> &lista, int &mdl)
2. {
3.     ifstream we("słowa.txt");
4.     string s;
5.     mdl=0;
6.     while (we>>s)
7.     {
8.         lista.push_back(s);
9.         if (s.size()>mdl) mdl=s.size();
10.    }
11.    we.close();
12. }
13.
14. void ZapiszListe(list<string> lista)
15. {
16.     ofstream wy("słownik.txt");
17.     list<string>::iterator it;
18.     for (it=lista.begin();it!=lista.end();it++)
19.         wy<<*it<<endl;
20.     wy.close();
21. }
```

Fragment kodu źródłowego programu porządkującego słowa leksykograficznie – definicja funkcji wczytującej słowa z pliku i budującej listę oraz funkcji zapisującej słowa do pliku

Zwróć uwagę na parametry funkcji *BudujListe*. Obydwa są przekazywane przez referencję, ponieważ funkcja ma zwrócić zbudowaną listę oraz znalezionej maksymalną długość słowa.

W linii 6 w warunku pętli **while** do zmiennej *s* wczytywane jest słowo z pliku tekstowego. Jeśli nie uda się go wczytać (dojdziemy do końca pliku), instrukcja zwróci wartość **false**. W linii 8 dodajemy wczytane słowo na końcu listy, w następnej linii aktualizujemy dotychczas znalezionej maksymalną długość słowa, jeśli wczytane słowo było dłuższe.

Funkcja *ZapiszListe* przegląda całą listę za pomocą iteratora *it* i zapisuje wartość aktualnego elementu do pliku łącznie ze znakiem końca wiersza.

Oto kod źródłowy funkcji *SortujSłowa*, realizującej algorytm sortowania leksykograficznego zapisany wcześniej w pseudokodzie.

Fragment kodu
źródłowego programu
porządkującego słowa
leksykograficznie –
definicja funkcji
sortującej słowa

Dobra rada

Pamiętaj, że wartością metody `erase` z klasy `list` jest iterator wskazujący na element listy znajdujący się za elementem usuwanym.

```

1. void SortujSlova(list<string> &lista, int mdl)
2. {
3.     queue<string> Kubelki[N];
4.     list<string>::iterator it;
5.     int i, j;
6.     string s;
7.     for (i=mdl-1; i>=0; i--)
8.     {
9.         it=lista.begin();
10.        while (it!=lista.end())
11.        {
12.            s=*it;
13.            if (s.size()>i)
14.            {
15.                it=lista.erase(it);
16.                j=alfabet.find(s[i]);
17.                Kubelki[j].push(s);
18.            }
19.            else it++;
20.        }
21.        for (j=0; j<N; j++)
22.            while (!Kubelki[j].empty())
23.            {
24.                lista.push_back(Kubelki[j].front());
25.                Kubelki[j].pop();
26.            }
27.    }
28. }
```

W linii 3 deklarowana jest tablica N-elementowa, której elementami są kolejki reprezentujące kubelki. Wartość zmiennej sterującej główną pętlą `for` (linie 7–27) jest zmniejszana od wartości równej największemu indeksowi litery w słowie do zera. Pierwsza wewnętrzna pętla (linie 10–20) przegląda listę słów. W zmiennej pomocniczej `s` pamiętana jest aktualna wartość elementu listy (linia 12). Jeśli długość słowa (linia 13) jest dostatecznie duża (istnieje litera o indeksie `i`), słowo jest usuwane z listy (linia 15, iterator `it` wskazuje na następny element za usuniętym) i dodawane do odpowiedniej kolejki (linie 16–17). W przeciwnym przypadku (linia 19) słowo pozostaje na liście i przechodzi się do następnego słowa. Pętla w liniach 21–26 przegląda kolejno wszystkie kubelki. Wewnętrzna pętla (linie 22–26) przegląda pojedynczy kubełek, dopisuje jego elementy na końcu listy (linia 24) i usuwa je z kubelka (linia 25).

Ćwiczenie 4

Napisz program, który wczyta słowa z pliku otrzymanego od nauczyciela (np. *slova.txt*), następnie posortuje je w porządku leksykograficznym i zapisze w pliku tekstowym o nazwie *slovník.txt*.

Zwróć uwagę, że jeśli w pliku z danymi będą występowały takie same słowa, to w pliku wynikowym pojawią się w tej samej kolejności. Sortowanie, w którym elementy równe zachowują swoją kolejność po uporządkowaniu, nazywamy **sortowaniem stabilnym**.

• Sortowanie stabilne

3.5. Ocena złożoności obliczeniowej algorytmu rozwiązującego problem Flawiusza i algorytmu sortowania leksykograficznego

Czasowa złożoność obliczeniowa przedstawionej w temacie symulacji problemu Flawiusza wynosi $O(n \cdot k)$, gdzie n – liczba wszystkich elementów, k – liczba określająca, co który element jest usuwany.

Ostatni usuwany element można znaleźć sprawniej, w złożoności $O(n)$, bez użycia listy, korzystając z następującego wzoru rekurencyjnego:

$$f(n, k) = \begin{cases} 1 & \text{dla } n = 1 \\ ((f(n-1, k) + k - 1) \bmod n) + 1 & \text{dla } n > 1 \end{cases}$$

Gdybyśmy numerowali elementy od 0, zależność rekurencyjna byłaby następująca:

$$f(n, k) = \begin{cases} 0 & \text{dla } n = 0 \\ (f(n-1, k) + k) \bmod n & \text{dla } n > 0 \end{cases}$$

Po pierwszym usunięciu (k -tego elementu, który znajduje się na pozycji $k-1$) problem redukuje się do rozwiązania problemu dla $n-1$ elementów, z tym że kolejne odliczanie rozpoczynamy od elementu o numerze k (pierwszy element za usuniętym, dlatego dodajemy wartość k). Otrzymany wynik sprowadzamy do zakresu od 0 do $n-1$ (operacja **mod**).

Ponieważ w rzeczywistości numerujemy elementy od 1, najpierw odejmujemy 1, a po wykonaniu operacji **mod** dodajemy 1.

Dla przypadku, gdy $k = 2$, można obliczyć wynik w złożoności $O(\log n)$, korzystając ze wzoru:

$$f(n) = \begin{cases} 1 & \text{dla } n = 1 \\ 2 \cdot f(n \operatorname{div} 2) - 1 & \text{dla } n > 1, n - \text{liczba parzysta} \\ 2 \cdot f(n \operatorname{div} 2) + 1 & \text{dla } n > 1, n - \text{liczba nieparzysta} \end{cases}$$

Czasowa złożoność obliczeniowa omówionego algorytmu sortowania leksykograficznego słów zależy od długości listy, czyli liczby słów oraz ich długości. Oznaczmy długość listy przez n , a k niech będzie maksymalną długością słowa. Operacje wstawiania oraz usuwania elementów z listy i kolejek są wykonywane w **złożoności stałej** $O(1)$. Złożoność czasowa tego algorytmu wynosi więc $O(n \cdot k)$.

Warto wiedzieć

Dla $k = 2$ istnieje także wzór jawny:
 $f(n) = 2 \cdot (n - 2^{\lceil \log_2 n \rceil}) + 1$.
 Znaki $\lceil \cdot \rceil$ oznaczają zaokrąglenie w dół do liczb całkowitych, nazywane podłogą.

Złożoność stała,
 podręcznik Informatyka
 na czasie 2. Zakres
 rozszerzony,
 s. 167 

Podsumowanie

- Lista to dynamiczna struktura danych, w której dostęp do elementów jest sekwencyjny. Nowy element można wstawić w dowolnym miejscu, można też usunąć dowolny element z listy.
- Wyróżniamy listy jednokierunkowe i dwukierunkowe. Dodatkowo mogą one być listami cyklicznymi.
- Stos i kolejka są szczególnymi przypadkami listy.
- Problem Flawiusza polega na cyklicznym eliminowaniu elementu położonego co określoną liczbę pozycji, aż pozostanie jeden element.
- Do symulacji problemu Flawiusza można użyć listy cyklicznej.
- Sortowanie leksykograficzne polega na porządkowaniu ciągów złożonych z określonych elementów. O tym, który ciąg wystąpi wcześniej, decyduje pierwsza pozycja różniąca ciągi. Jeśli taka pozycja nie istnieje, to wcześniej znajdzie się krótszy ciąg.
- Przykładem sortowania leksykograficznego jest porządkowanie haseł w słownikach i skorowidzach (indeksach).
- Do sortowania leksykograficznego słów możemy wykorzystać listę oraz algorytm sortowania kubełkowego.
- Sortowanie kubełkowe polega na grupowaniu elementów o takich samych cechach w tzw. kubełkach, a następnie przepisywaniu elementów z kolejnych kubełków. Kolejność kubełków jest ściśle określona – zgodna z alfabetem. Wspólną cechą słów jest taka sama litera na danej pozycji. Litery słów przegląda się od prawej strony do lewej.

Zadania

- ★ **1** Napisz funkcję rekurencyjną rozwiązującą problem Flawiusza dla przypadku, gdy co drugi element jest usuwany, w złożoności czasowej $O(\log n)$, gdzie n – liczba elementów zbioru.
- ★ **2** Napisz funkcję rekurencyjną rozwiązującą problem Flawiusza dla ogólnego przypadku, gdy usuwany jest co k -ty element, w złożoności czasowej $O(n)$, gdzie n – liczba elementów zbioru.
- ★★ **3** Napisz program losujący tablicę n liczb całkowitych z zakresu od -9 do 9 i sortujący ją niemalejąco według wartości bezwzględnej liczb. Sortowanie powinno być stabilne, tzn. elementy równe co do wartości bezwzględnej powinny zachować kolejność występowania w posortowanym ciągu. Na przykład dla danych:
 $-6 \ 5 \ -1 \ 2 \ 1 \ -1 \ -5 \ 6 \ -2$
 wynikiem będzie ciąg: $-1 \ 1 \ -1 \ 2 \ -2 \ 5 \ -5 \ -6 \ 6$
- ★★ **4** Program sortujący wyrazy leksykograficznie, omówiony w tym temacie, zmodyfikuj tak, aby podczas odczytywania słów z pliku wyznaczał nie tylko maksymalną długość słowa, lecz także alfabet złożony wyłącznie z liter, które występują w co najmniej jednym słowie. Plik ze słowami do sortowania otrzymasz od nauczyciela (np. *słowa_alfabet.txt*).

- ★★ **5** Napisz program losujący tablicę n liczb całkowitych nieujemnych i sortujący ją niemalejąco według cyfry jedności, tzn. najpierw w posortowanym ciągu wystąpią liczby zakończone cyfrą 0, potem 1 itd. Sortowanie powinno być stabilne, tzn. elementy z taką samą ostatnią cyfrą powinny zachować kolejność występowania w posortowanym ciągu. Na przykład dla danych:
 62 59 12 24 19 17 52 60 22
 wynikiem będzie ciąg:
 60 62 12 52 22 24 17 59 19
- ★★★ **6** W pliku tekstowym otrzymanym od nauczyciela (np. *liczby2.txt*) znajduje się 10 ciągów liczbowych. Każdy z ciągów zapisany jest w oddzielnym wierszu, składa się z maksymalnie 10 liczb całkowitych oddzielonych spacjami i jest zakończony liczbą 0, która nie należy do ciągu. Napisz program sortujący ciągi leksykograficznie i zapisujący uporządkowane ciągi do pliku tekstowego *liczby2_wynik.txt*. Na przykład ciągi:
 62 59 12 20 19 17 52 60 22
 62 59 12 24 80
 62 59 12
 powinny być uporządkowane następująco:
 62 59 12
 62 59 12 20 19 17 52 60 22
 62 59 12 24 80
- ★★★ **7** W pliku tekstowym, który otrzymasz od nauczyciela (np. *hasla.txt*), znajduje się 10 haseł, każde w oddzielnym wierszu. Każde hasło składa się z jednego lub kilku słów (maksymalnie 5, złożonych z małych liter alfabetu łacińskiego) oddzielonych spacjami. Napisz program, który uporządkuje hasła leksykograficznie i zapisze wyniki do pliku *indeks.txt*. Jeśli dwa hasła mają to samo słowo na tej samej pozycji, to w kolejnych wystąpieniach powinno ono być zastąpione myślnikiem. Na przykład dla haseł:
 algorytm rekurencyjny
 algorytm iteracyjny
 algorytm naiwny
 algorytm obliczania silni iteracyjny
 algorytm obliczania silni rekurencyjny
 algorytm
 algorytm obliczania silni
 plik wynikowy powinien mieć postać:
 algorytm
 – iteracyjny
 – naiwny
 – obliczania silni
 – – – iteracyjny
 – – – rekurencyjny
 – rekurencyjny

4. Grafy. Znajdowanie najkrótszej drogi

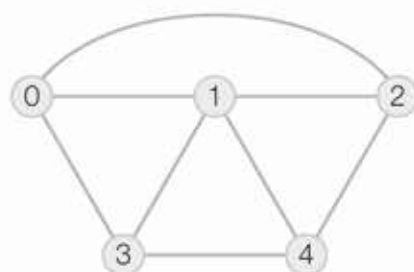
Zapewne zdarza ci się korzystać z map internetowych. Pozwalają one m.in. znaleźć najszybszą lub najkrótszą drogę między dwoma miejscami. Jakie algorytmy to umożliwiają i jakie struktury danych te algorytmy wykorzystują? W tym temacie odpowiemy na te pytania. Zajmiemy się strukturą danych nazwaną grafem oraz podstawowymi algorytmami operującymi na grafach.

Cele lekcji

- Dowiesz się, czym jest graf, i poznasz sposoby jego reprezentowania.
- Przejrzysz wszystkie wierzchołki grafu, stosując różne algorytmy.
- Znajdziesz najkrótszą drogę pomiędzy dwoma wierzchołkami grafu.

4.1. Czym jest graf?

Graf • **Graf** (ang. *graph*) to struktura danych składająca się z niepustego zbioru wierzchołków i zbioru połączeń między nimi, czyli krawędzi. Rysunek 4.1 przedstawia przykład grafu złożonego z pięciu wierzchołków (ponumerowanych od 0 do 4) oraz ośmiu krawędzi łączących pary wierzchołków: 0 i 1, 0 i 2, 0 i 3, 1 i 2, 1 i 3, 1 i 4, 2 i 4 oraz 3 i 4.



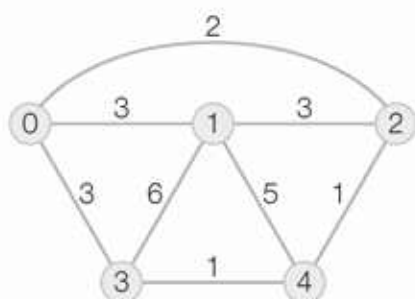
Rys. 4.1. Przykład grafu złożonego z pięciu wierzchołków i ośmiu krawędzi

Sąsiednie wierzchołki grafu • Wierzchołki połączone krawędzią nazywamy **sąsiednimi wierzchołkami grafu**. W odniesieniu do mapy wierzchołki możemy interpretować jako miejscowości, a krawędzie jako drogi pomiędzy miejscowościami.

Wierzchołek grafu może nie być połączony krawędzią z żadnym innym wierzchołkiem lub może być połączony krawędzią z samym sobą. Dwa wierzchołki mogą być też połączone więcej niż jedną krawędzią. W temacie zajmiemy się **grafami prostymi**, czyli takimi, w których dwa wierzchołki może łączyć co najwyżej jedna krawędź oraz z żadnego wierzchołka nie prowadzi krawędź do niego samego.

Waga krawędzi grafu • Krawędziom grafu mogą być przypisane wartości zwane **wagami**. W odniesieniu do mapy wagi mogą określać np. odległości pomiędzy miejscowościami lub czasy przejazdu. Rysunek 4.2 przedstawia przy-

Graf ważony • kład grafu z wagami krawędzi. Taki graf nazywamy **grafem ważonym**.

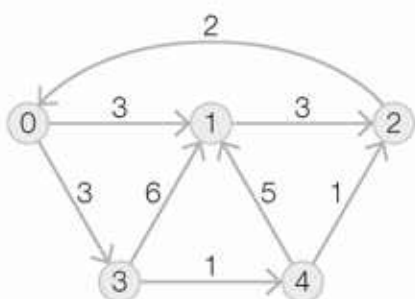


Rys. 4.2. Przykład grafu ważonego

Ćwiczenie 1

Narysuj na kartce papieru przykład grafu ważonego. Wykorzystaj część sieci dróg poprowadzonych między miejscowością, w której mieszkasz, a kilkoma sąsiednimi miejscowościami.

Graf nieskierowany to graf, w którym przejście wzdłuż krawędzi jest możliwe w obu kierunkach. Przykładami grafów nieskierowanych są grafy na rysunkach 4.1 i 4.2. **Graf skierowany** to graf, w którym po krawędzi można przejść w określonym kierunku. Rysunek 4.3 przedstawia przykład grafu skierowanego. Kierunek, w którym można przejść po krawędzi, oznaczony jest strzałką.



Rys. 4.3. Przykład grafu skierowanego

Graf, w którym istnieje droga z każdego wierzchołka do każdego, nazywamy **grafem spójnym**. Grafy przedstawione na rysunkach 4.1, 4.2 i 4.3 są spójne.

Zapamiętaj

Graf składa się z niepustego zbioru wierzchołków i zbioru krawędzi, czyli połączeń między wierzchołkami. W grafach nieskierowanych przejście między dwoma wierzchołkami połączonymi krawędzią jest możliwe w obu kierunkach. W grafach skierowanych między wierzchołkami można przechodzić tylko zgodnie z zaznaczonym kierunkiem. Jeśli krawędziom grafu przyporządkowane są wagi, to taki graf nazywamy grafem ważonym. Graf spójny to graf, w którym istnieje droga z każdego wierzchołka do każdego.

4.2. Reprezentacja grafu

Do reprezentacji grafu wykorzystuje się różne struktury danych i różne sposoby. Część z nich opiera się na rozwiązaniach, które już znasz. Pokażemy m.in., jak wykorzystać do reprezentacji grafu typ **vector** z biblioteki STL.

Macierz sąsiedztwa

Do przechowywania informacji o grafie można wykorzystać tablicę dwuwymiarową $n \times n$, gdzie n oznacza liczbę wierzchołków grafu. Element (i, j) tablicy informuje, czy istnieje krawędź od wierzchołka i do wierzchołka j . Gdy graf nie jest grafem ważonym, informacja, czy krawędź między wierzchołkami istnieje czy nie, może być informacją logiczną (wartości typu **bool**: **true** i **false** lub typu **int**: 0 i 1). Dla grafu ważonego z dodatnimi wagami elementem tablicy może być waga krawędzi oraz wartość 0, gdy krawędź nie istnieje. Jeśli graf jest nieskierowany, to wartości elementów (i, j) oraz (j, i) są takie same. Jeśli graf jest skierowany, a przejście między wierzchołkami i oraz j jest możliwe tylko w jednym kierunku, to wartość jednego z elementów (i, j) lub (j, i) będzie równa 0.

Opisaną powyżej reprezentację grafu nazywamy **macierzą sąsiedztwa**. Rysunek 4.4 przedstawia przykład reprezentacji grafu z rysunku 4.3 ze s. 63 w postaci macierzy sąsiedztwa.

	0	1	2	3	4
0	0	3	0	3	0
1	0	0	3	0	0
2	2	0	0	0	0
3	0	6	0	0	1
4	0	5	1	0	0

Rys. 4.4. Reprezentacja grafu z rysunku 4.3 w postaci macierzy sąsiedztwa

Macierz sąsiedztwa •

Warto wiedzieć

Z macierzy sąsiedztwa można odczytać wiele informacji dotyczących grafu, np. które wierzchołki sąsiadują z danym wierzchołkiem.

Ćwiczenie 2

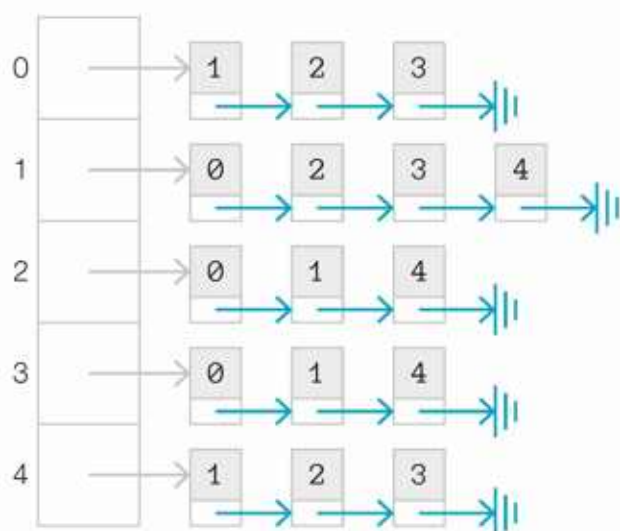
Zapisz w postaci macierzy sąsiedztwa reprezentacje grafów z rysunków 4.1 i 4.2 ze s. 62 i 63.

Na rysunku 4.4 widać, że wiele elementów tablicy jest niewykorzystanych (wypełnionych zerami, ponieważ krawędzie między danymi wierzchołkami nie istnieją). Złożoność pamięciowa i złożoność czasowa algorytmów operujących na takiej reprezentacji grafu wynosi co najmniej $O(n^2)$.

Listy sąsiedztwa

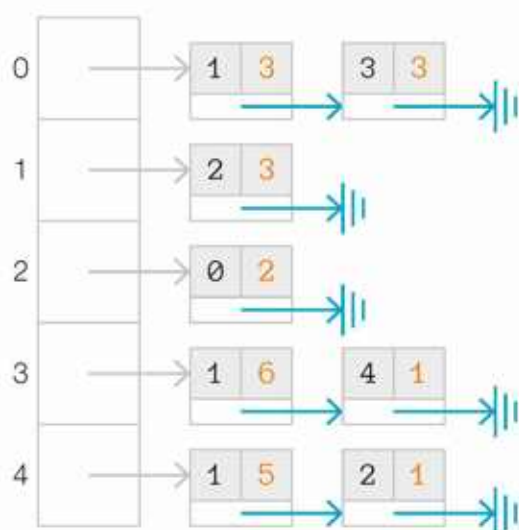
Listy sąsiedztwa • Inną formą reprezentacji grafu są **listy sąsiedztwa**. W tym przypadku dla każdego wierzchołka pamięta się listę sąsiadujących z nim wierzchołków (połączonych krawędzią z tym wierzchołkiem). Jeśli graf nie jest grafem ważonym, elementami danej listy będą numery sąsiednich wierzchołków.

Rysunek 4.5 przedstawia reprezentację grafu z rysunku 4.1 ze s. 62 w postaci list sąsiedztwa.



Rys. 4.5. Reprezentacja grafu z rysunku 4.1 w postaci list sąsiedztwa

W przypadku grafu ważonego dla każdego wierzchołka należy pamiętać dwie wartości: numer sąsiedniego wierzchołka oraz wagę krawędzi, która do niego prowadzi. Rysunek 4.6 przedstawia reprezentację grafu z rysunku 4.3 ze s. 63 w postaci list sąsiedztwa. Wagi krawędzi są zaznaczone na pomarańczowo.



Rys. 4.6. Reprezentacja grafu z rysunku 4.3 w postaci list sąsiedztwa

Pierwsze algorytmy związane z grafami, którymi będziemy się zajmować, będą dotyczyć grafu nieważonego. Dlatego omówimy najpierw możliwości reprezentacji takiego właśnie grafu w postaci list sąsiedztwa.

W języku C++ listy sąsiedztwa dla grafu nieważonego można przedstawić w postaci tablicy n -elementowej, gdzie n jest liczbą wierzchołków grafu. Każdy element tej tablicy odpowiada jednemu z wierzchołków i jest **listą** zawierającą numery wierzchołków sąsiadujących z tym wierzchołkiem. Przykład deklaracji tablicy `Graf`, służącej do przechowywania list sąsiedztwa dla grafu z pięcioma wierzchołkami, wygląda następująco:

```
const int N=5;
list<int> Graf[N];
```

Lista,
s. 47 [↗](#)

Iterator,
s. 51 

Przedstawimy jeszcze jedną możliwość deklaracji grafu w postaci list sąsiedztwa. Wykorzystuje ona typ **vector** z biblioteki STL. Do przeglądania elementów listy nie musimy wówczas wykorzystywać **iteratora**.

Tablica dynamiczna 

Typ **vector** jest w pewnym sensie odpowiednikiem **tablicy dynamicznej** (ang. *dynamic array*), czyli takiej, której rozmiar (liczbę elementów) można określać i zmieniać podczas działania programu. Do zmiennej typu **vector**, podobnie jak do zmiennej typu **list**, można wstawić w dowolne miejsce nowy element. Można też usunąć z niej dowolny element (zmieniając w ten sposób także rozmiar). Ogólna

Deklaracja zmiennej
typu **vector** 

deklaracja zmiennej typu **vector** jest następująca:

vector<typ elementów wektora> nazwa_wektora;

Żeby korzystać z typu **vector**, należy za pomocą dyrektywy **#include <vector>** dołączyć do programu bibliotekę **vector**.

Do elementu typu **vector** odwołujemy się tak samo jak do elementu w tablicy – podając w nawiasach kwadratowych indeks elementu.

Metoda **resize** dla klasy **vector** 

Rozmiar zmiennej typu **vector** można ustalić za pomocą **metody resize**, której parametrem jest liczba elementów. W programie można jej użyć wielokrotnie i w ten sposób zwiększać lub zmniejszać liczbę elementów. Metoda **resize** może także mieć drugi parametr, który określa wartość początkową elementów zmiennej typu **vector**. Aktualny rozmiar tej zmiennej zwraca bezparametrowa **metoda size**.

Metoda **size** dla klasy **vector** 

Poniższy fragment kodu źródłowego programu tworzy zmienną **Tab** typu **vector** o elementach całkowitych, następnie wczytuje z klawiatury rozmiar zmiennej i wypełnia ją losowymi liczbami.

Fragment kodu
źródłowego programu
tworzącego zmienną
typu **vector**, której
elementy są losowymi
liczbami 

```
1. vector<int> Tab;
2. int n;
3. cin>>n;
4. Tab.resize(n);
5. for (int i=0;i<Tab.size();i++)
6.     Tab[i]=rand()/100;
```

Dobra rada

Wstawianie elementów do struktury danych typu **vector** oraz usuwanie ich z tej struktury wymaga często przesunięcia w pamięci całej struktury danych (alokacji przydzielonej pamięci). Zwiększa to czas wykonywania tych operacji. Jeśli w programie, który piszesz, elementy będą często wstawiane bądź usuwane, lepiej użyj typu **list**.

Ćwiczenie 3

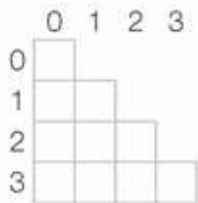
Napisz program, który utworzy zmienną typu **vector**, wczyta jej rozmiar z klawiatury, a następnie wypełni tę zmienną losowymi liczbami całkowitymi i ją wypisze.

Elementem zmiennej typu **vector** może być kolejna zmienna typu **vector**. Otrzymamy w ten sposób odpowiednik tablicy dwuwymiarowej, w której każdy wiersz może mieć różną liczbę elementów. Do poszczególnych elementów takiej zmiennej odwołujemy się jak do elementów tablicy dwuwymiarowej, podając w drugiej parze nawiasów kwadratowych drugi indeks.

Poniższy fragment kodu źródłowego programu dla $n = 4$ utworzy tablicę Tab przedstawioną na rysunku 4.7.

```
1. vector<vector<int> > Tab;
2. int n;
3. cin>>n;
4. Tab.resize(n);
5. for (int i=0;i<Tab.size();i++)
6.     Tab[i].resize(i+1);
```

Fragment kodu źródłowego programu tworzącego tablicę o różnej liczbie elementów w wierszach



Rys. 4.7. Przykład tablicy o różnej liczbie elementów w wierszach

Ćwiczenie 4

Napisz program, który utworzy tablicę taką jak na rysunku 4.7, wypełni ją losowymi liczbami całkowitymi i wypisze elementy tablicy.

Do reprezentacji grafu w postaci list sąsiedztwa można wykorzystać zmienną typu **vector**, której elementy są też typu **vector**. Oto deklaracja zmiennej Graf reprezentującej graf, w którym wierzchołki oznaczone są liczbami całkowitymi:

```
vector<vector<int> > Graf;
```

Można zdefiniować własną nazwę dla tak określonego typu. Jest to szczególnie przydatne, gdy zmienna reprezentująca graf będzie parametrem funkcji. Do **definiowania własnej nazwy typu zmiennych** służy słowo kluczowe **typedef**:

```
typedef<definicja typu> nazwa_typu;
```

Definicja typu tgraf reprezentującego graf jako listy sąsiedztwa oraz deklaracja zmiennej Graf tego typu przyjmą postać:

```
typedef vector<vector<int> > tgraf;
tgraf Graf;
```

Opis grafu wygodnie jest przygotować w pliku tekstowym. Pierwszy wiersz pliku będzie zawierał dwie liczby całkowite dodatnie określające odpowiednio liczbę wierzchołków grafu i liczbę krawędzi. Każdy kolejny wiersz będzie opisem jednej krawędzi w postaci numeru wierzchołka początkowego i numeru wierzchołka końcowego.

Rysunek 4.8 na s. 68 przedstawia przykład grafu skierowanego i jego opis w pliku tekstowym.

Warto wiedzieć

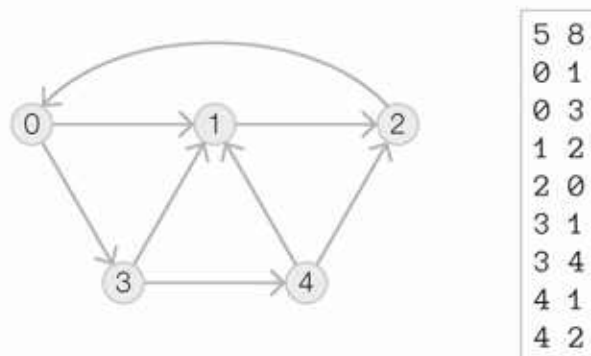
Elementy zmiennej typu **vector** można sortować za pomocą funkcji **sort** z biblioteki STL. Parametrami funkcji są wówczas wartości zwracane przez iteratory **begin** i **end**. Na przykład zapis funkcji sortującej wartości zmiennej Tab będzie wyglądał następująco:

```
sort(Tab.begin(),
     Tab.end());
```

Definicja własnej nazwy typu zmiennych

Dobra rada

Dwa znaki **>** zapisane obok siebie kompilator może uznać za operator wczytywania danych. Dlatego przy deklaracji zmiennych z wykorzystaniem szablonów typów z biblioteki STL oddziel te znaki spacją.



Rys. 4.8. Przykład grafu skierowanego i jego opis w pliku tekstowym

Kod źródłowy funkcji wczytującej opis grafu z pliku tekstowego i tworzącej zmienną `Graf` typu `tgraf` może być następujący:

Kod źródłowy funkcji
Czytaj, wczytującej
z pliku tekstowego opis
grafu nieważonego

```
1. void Czytaj(tgraf &Graf)
2. {
3.     int n, m, w1, w2;
4.     ifstream we("graf.txt");
5.     we >> n >> m;
6.     Graf.resize(n);
7.     for (int i=0; i<m; i++)
8.     {
9.         we >> w1 >> w2;
10.        Graf[w1].push_back(w2);
11.    }
12.    we.close();
13. }
```

W linii 5 z pliku tekstowego do zmiennych `n` i `m` są wczytywane informacje o liczbie wierzchołków i liczbie krawędzi grafu. Instrukcja w linii 6 tworzy graf o `n` wierzchołkach. W pętli w liniach 7–11 odczytywane są informacje o kolejnych krawędziach grafu (linia 9) i dodawane są one do struktury grafu (linia 10). W zmiennej `w1` zapisywany jest numer wierzchołka będącego początkiem krawędzi, a w zmiennej `w2` –

Metoda `push_back` dla
klasy `vector`

numer wierzchołka będącego końcem krawędzi. Metoda `push_back` dodaje nowy element na końcu zmiennej typu `vector`.



Zapamiętaj

Graf można reprezentować m.in. w postaci macierzy sąsiedztwa lub list sąsiedztwa. W przypadku macierzy sąsiedztwa informacje o grafie przechowuje się w tablicy dwuwymiarowej $n \times n$, gdzie n oznacza liczbę wierzchołków grafu. Element (i, j) tablicy informuje, czy istnieje krawędź od wierzchołka i do j . W reprezentacji grafu za pomocą list sąsiedztwa dla każdego wierzchołka pamięta się listę krawędzi z niego wychodzących. Do implementacji list sąsiedztwa można użyć typu `vector`.

4.3. Przeszukiwanie grafu w głąb

Zajmiemy się teraz problemem przejścia w usystematyzowany sposób wszystkich wierzchołków grafu. Założymy, że **graf jest skierowany i spójny**. Dla grafu nieskierowanego krawędzie w opisie występowałyby dwukrotnie. Oto specyfikacja problemu:

Graf skierowany,
graf spójny,
s. 63 [↗](#)

Specyfikacja

Dane: n – liczba wierzchołków grafu,

m – liczba krawędzi grafu,

Graf – reprezentacja spójnego grafu skierowanego w postaci list sąsiedztwa,

w_1 – numer wierzchołka, od którego zaczynamy przeglądanie grafu.

Wynik: numery kolejno odwiedzonych wierzchołków.

Algorytm przeglądania grafu, który teraz omówimy, nosi nazwę **przeszukiwania w głąb (DFS)** (od ang. *depth-first search*). Opiera się na podobnej zasadzie co **rekurencyjny algorytm znajdowania wyjścia z labiryntu**. Zapiszemy go także rekurencyjnie.

● Algorytm przeszukiwania
w głąb (DFS)

Wprowadzimy dodatkową strukturę danych, w której zapiszemy informację, czy dany wierzchołek został już odwiedzony. Będzie to tablica wartości logicznych **Odwiedzone** o rozmiarze zgodnym z liczbą wierzchołków grafu. Gdybyśmy nie zaznaczyli, które wierzchołki odwiedziliśmy, moglibyśmy wielokrotnie sprawdzać sąsiadów tego samego wierzchołka (wywoływać funkcję rekurencyjną dla tych samych parametrów, co prowadziłoby do rekurencji nieskończonej).

Rekurencyjny algorytm
znajdowania wyjścia
z labiryntu,
s. 31 [↗](#)

Oznaczmy wierzchołek początkowy jako odwiedzony, a następnie dla sąsiadów tego wierzchołka będziemy wywoływać funkcję rekurencyjną, tak jakby dany sąsiad wierzchołka był wierzchołkiem początkowym. Wywołanie rekurencyjne nastąpi pod warunkiem, że dany wierzchołek nie został jeszcze odwiedzony. Zapis algorytmu w pseudokodzie może być następujący:

```
dla  $i \leftarrow 0, 1, \dots, n - 1$  wykonuj  $\text{Odwiedzone}[i] \leftarrow \text{fałsz}$ 
funkcja DFS( $w_1$ )
    wypisz  $w_1$ 
     $\text{Odwiedzone}[w_1] \leftarrow \text{prawda}$ 
    dla  $i \leftarrow 0, 1, \dots, (\text{liczba krawędzi } w_1) - 1$  wykonuj
         $w_2 \leftarrow \text{Graf}[w_1][i]$ 
        jeśli nie  $\text{Odwiedzone}[w_2]$  to DFS( $w_2$ )
```

Zwróć uwagę, że tablica **Odwiedzone** (pierwsza linia pseudokodu) musi być inicjowana poza funkcją rekurencyjną. Zmienna pomocnicza w_2 przechowuje numer wierzchołka będącego końcem analizowanej krawędzi. Jej wartość jest parametrem wywołania rekurencyjnego. Kod źródłowy funkcji DFS, realizującej algorytm przedstawiony w pseudokodzie, oraz funkcji **main** jest następujący.

Fragment kodu
źródłowego programu
realizującego algorytm
przeszukiwania grafu
w głąb – funkcja DFS,
realizująca ten algorytm,
oraz funkcja main

```

1. void DFS(int w1, tgraf &Graf, vector<bool> &Odwiedzone)
2. {
3.     cout<<"Odwiedzony wierzcholek: "<<w1<<endl;
4.     Odwiedzone[w1]=true;
5.     for (int i=0;i<Graf[w1].size();i++)
6.     {
7.         int w2=Graf[w1][i];
8.         if (!Odwiedzone[w2]) DFS(w2,Graf,Odwiedzone);
9.     }
10. }
11.
12. int main()
13. {
14.     tgraf Graf;
15.     Czytaj(Graf);
16.     vector<bool> Odwiedzone;
17.     Odwiedzone.resize(Graf.size(),false);
18.     int w1;
19.     cout<<"Podaj numer wierzchołka początkowego: ";
20.     cin>>w1;
21.     DFS(w1,Graf,Odwiedzone);
22.     return 0;
23. }

```

Kod źródłowy funkcji
Czytaj, wczytującej
z pliku tekstowego opis
grafu nieważonego,
s. 68

Funkcja DFS nie zmienia wartości parametru Graf, ale jest on przekazywany przez referencję, aby nie była tworzona jego kopia przy wywołaniach rekurencyjnych. Parametr Odwiedzone musi być przekazany przez referencję, gdyż funkcja modyfikuje jego wartość. W linii 15 wywoływana jest funkcja Czytaj, wczytująca z pliku tekstowego opis grafu. W linii 16 deklarowana jest tablica Odwiedzone. Instrukcja w linii 17 tworzy tablicę Odwiedzone o rozmiarze zgodnym z liczbą wierzchołków grafu oraz nadaje jej elementom wartość początkową **false** (drugi parametr). W linii 20 wczytywany jest z klawiatury numer wierzchołka (zmienna w1), od którego chcemy rozpocząć przeglądanie grafu. W linii 21 wywołujemy funkcję DFS.

Rysunek 4.9 przedstawia przykład wykonania programu przeglądającego w głąb graf z rysunku 4.8 ze s. 68, zaczynając od wierzchołka numer 3.

```

Podaj numer wierzchołka początkowego: 3
Odwiedzony wierzcholek: 3
Odwiedzony wierzcholek: 1
Odwiedzony wierzcholek: 2
Odwiedzony wierzcholek: 0
Odwiedzony wierzcholek: 4

```

Rys. 4.9. Przykład wykonania programu przeglądającego w głąb graf z rysunku 4.8 od wierzchołka numer 3

Z wierzchołka oznaczonego liczbą 3 wychodzą dwie krawędzie, które prowadzą do wierzchołków 1 i 4. Najpierw został odwiedzony wierzchołek numer 1, potem jego sąsiedzi, a na końcu nastąpił powrót do wierzchołka początkowego i został odwiedzony wierzchołek numer 4.

Ćwiczenie 5

Napisz program, który wczyta opis spójnego grafu skierowanego z pliku tekstowego otrzymanego od nauczyciela (np. *graf.txt*), a następnie przejrzy wierzchołki grafu, wykorzystując algorytm przeglądania w głąb, i wypisze kolejno odwiedzone wierzchołki.

Zapamiętaj

Algorytm przeszukiwania w głąb przegląda wszystkie wierzchołki grafu spójnego. Jeśli istnieje nieodwiedzony wierzchołek sąsiadujący z aktualnie rozpatrywanym wierzchołkiem, to algorytm przechodzi do niego i ten wierzchołek staje się aktualnie rozpatrywanym. Jeśli taki sąsiedni wierzchołek nie istnieje, algorytm cofa się do poprzednio rozpatrywanego i powtarza czynności dla kolejnego sąsiedniego wierzchołka. Algorytm można zapisać rekurencyjnie.

4.4. Przeszukiwanie grafu wszerz

Omówimy teraz **algorytm przeszukiwania wszerz** (BFS, od ang. *breadth-first search*). Jest on podobny do **iteracyjnego algorytmu znajdującego wyjście z labiryntu** z wykorzystaniem kolejki.

Wierzchołek, od którego zaczniemy analizę, zostanie wstawiony do kolejki i oznaczony w tablicy *Odwiedzone* jako odwiedzony. Następnie, dopóki kolejka nie będzie pusta, pobieramy wierzchołek z początku kolejki i analizujemy wierzchołki z nim sąsiadujące. Jeśli sąsiedni wierzchołek nie został wcześniej odwiedzony, to jest dopisywany na końcu kolejki i oznaczany jako odwiedzony. Zapis algorytmu w pseudokodzie może być następujący:

```
dla i ← 0, 1, ..., n - 1 wykonuj Odwiedzone[i] ← fałsz
wstaw w1 do kolejki
Odwiedzone[w1] ← prawda
dopóki nie pusta kolejka wykonuj
    w1 ← numer wierzchołka z początku kolejki
    usuń wierzchołek z kolejki
    wypisz w1
    dla i ← 0, 1, ..., (liczba krawędzi z w1) - 1 wykonuj
        w2 ← Graf[w1][i]
        jeśli nie Odwiedzone[w2] to
            wstaw w2 do kolejki
            Odwiedzone[w2] ← prawda
```

● Algorytm przeszukiwania wszerz (BFS)

Iteracyjny algorytm znajdowania wyjścia z labiryntu, s. 40 [↗](#)

Kod źródłowy funkcji BFS, realizującej algorytm przeszukiwania grafu wszerz, może wyglądać następująco.

Kod źródłowy funkcji
realizującej algorytm
przeszukiwania
grafu wszerz

```

1. void BFS(int w1, tgraf Graf)
2. {
3.     int w2;
4.     vector<bool> Odwiedzone;
5.     Odwiedzone.resize(Graf.size(), false);
6.     queue<int> wierzch;
7.     wierzch.push(w1); Odwiedzone[w1]=true;
8.     while (!wierzch.empty())
9.     {
10.        w1=wierzch.front(); wierzch.pop();
11.        cout<<"Odwiedzony wierzcholek: "<<w1<<endl;
12.        for (int i=0; i<Graf[w1].size(); i++)
13.        {
14.            w2=Graf[w1][i];
15.            if (!Odwiedzone[w2])
16.            {
17.                wierzch.push(w2); Odwiedzone[w2]=true;
18.            }
19.        }
20.    }
21. }
```

Dobra rada

Pamiętaj, że aby korzystać ze zmiennych typu `queue`, do programu musisz dołączyć bibliotekę `queue`.

W linii 4 jest deklarowana tablica `Odwiedzone`, a w linii 5 jest ona tworzona. Ponieważ funkcja `BFS` nie jest rekurencyjna, tablica `Odwiedzone` może być w tej funkcji zmienną lokalną. W linii 6 deklarowana jest kolejka `wierzch`, w której będą pamiętane numery wierzchołków do analizy. W linii 7 wierzchołek, od którego rozpoczniemy przeglądanie grafu, jest wstawiany do kolejki i oznaczany w tablicy `Odwiedzone` jako odwiedzony. Pętla w liniach 8–20 przegląda kolejkę, wykonując następujące operacje:

1. Pobiera pierwszy element kolejki i go usuwa. Numer wierzchołka pobranego z kolejki jest pamiętany w zmiennej `w1` (linia 10).
2. Wypisuje numer analizowanego wierzchołka (linia 11).
3. Rozpatruje sąsiednie wierzchołki (pętla w liniach 12–19). Najpierw pobiera numer sąsiedniego wierzchołka do zmiennej `w2` (linia 14). Jeśli wierzchołek nie został jeszcze odwiedzony (linia 15), to wstawia jego numer do kolejki i oznacza go jako odwiedzony (linia 17).

W programie realizującym przeglądanie grafu wszerz pozostałe funkcje (funkcja `Czytaj`, funkcja `main`) i deklaracje (poza tablicą `Odwiedzone`) wyglądają podobnie jak w programie przeglądającym graf w głąb.

Rysunek 4.10 przedstawia przykład wykonania programu przeglądającego wszerz graf z rysunku 4.8 ze s. 68, zaczynając od wierzchołka numer 3. Z tego wierzchołka wychodzą dwie krawędzie. Najpierw zostały odwiedzony wierzchołki numer 1 i 4, potem ich sąsiedzi.

```

Podaj numer wierzchołka początkowego: 3
Odwiedzony wierzchołek: 3
Odwiedzony wierzchołek: 1
Odwiedzony wierzchołek: 4
Odwiedzony wierzchołek: 2
Odwiedzony wierzchołek: 0

```

Rys. 4.10. Przykład wykonania programu przeglądającego wszerz graf z rysunku 4.8 od wierzchołka numer 3

Warto wiedzieć

Złożoność pamięciowa i czasowa algorytmów przeglądania grafu wszerz i w głąb wynosi $O(n + m)$, gdzie n określa liczbę wierzchołków, a m – liczbę krawędzi.

Ćwiczenie 6

Napisz program wczytujący opis grafu z pliku tekstowego otrzymanego od nauczyciela (np. *graf.txt*) i przeglądający wierzchołki grafu wszerz. Program powinien wypisać kolejno odwiedzane wierzchołki.

Zapamiętaj

Algorytm przeszukiwania wszerz przegląda wszystkie wierzchołki grafu spójnego. Najpierw odwiedza wszystkie wierzchołki sąsiadujące z danym wierzchołkiem, potem wszystkie nieodwiedzone jeszcze wierzchołki sąsiadujące z wierzchołkami sąsiednimi itd.

4.5. Algorytm Dijkstry

Rozważmy problem znajdowania drogi pomiędzy dwoma wierzchołkami w grafie ważonym, tak aby suma wag krawędzi, po których przejdziemy, była jak najmniejsza. Problem odpowiada znalezieniu najkrótszej drogi, jeśli wagi krawędzi będą odległościami między wierzchołkami, albo najszybszej drogi, jeśli wagi będą oznaczały czas przejścia. Łączną długość drogi lub łączny czas połączenia (sumę wag krawędzi) nazwiemy kosztem drogi, a drogę o najmniejszym koszcie – najkrótszą.

Rozwiązanie problemu dla grafu ważonego o nieujemnych wagach krawędzi umożliwia **algorytm Dijkstry**. Wyznacza on najmniejsze koszty dojścia z wybranego wierzchołka grafu do pozostałych wierzchołków oraz wskazuje, przez które wierzchołki prowadzą drogi o najmniejszych kosztach. Na razie skupimy się na sposobie wyznaczania najmniejszych kosztów dojścia.

Najpierw algorytm przypisuje wierzchołkom grafu początkowe koszty dojścia do nich: wierzchołkowi, z którego wychodzimy – koszt 0, pozostałym – wartość ∞ (nieskończoność). W każdym kroku z wierzchołków jeszcze nieodwiedzonych wybiera ten, który ma aktualnie najmniejszy koszt dojścia i oznacza go jako odwiedzony. Sprawdza koszty dojścia z tego wierzchołka do nieodwiedzonych wierzchołków sąsiadujących. Jeśli są mniejsze niż znalezione do tej pory, aktualizuje je.

• Algorytm Dijkstry

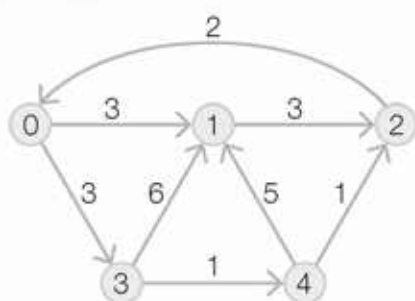
Warto wiedzieć

Algorytm Dijkstry opiera się na metodzie zachłannej, która polega na realizowaniu przez algorytm najlepszego wyboru w danym kroku rozwiązywania problemu. O metodzie tej można przeczytać w podręczniku *Informatyka na czasie 2. Zakres rozszerzony* (s. 255).

Algorytm Dijkstry – wyznaczanie najmniejszych kosztów dojścia do wierzchołków grafu

Prześledzimy działanie algorytmu dla skierowanego grafu ważonego o nieujemnych wagach krawędzi, w którym wierzchołki są oznaczone liczbami całkowitymi od 0 do 4. Wyznamy najmniejsze koszty dojścia z wierzchołka 0 do pozostałych wierzchołków. W każdym kroku na grafie kolorem niebieskim zaznaczony jest aktualnie rozpatrywany wierzchołek (oznaczany w danym kroku jako odwiedzony) wraz z krawędziami z niego wychodzącymi, a zielonym – wierzchołki odwiedzone w poprzednich krokach. Niebieski kolor w tabeli wskazuje zmianę w stosunku do poprzedniego kroku.

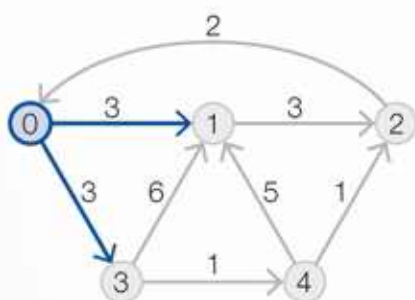
Graf oraz tabela z kosztami dojścia do poszczególnych wierzchołków na początku algorytmu wyglądają następująco:



Numer wierzchołka	0	1	2	3	4
Koszt dotarcia do wierzchołka	0	∞	∞	∞	∞

Krok 1

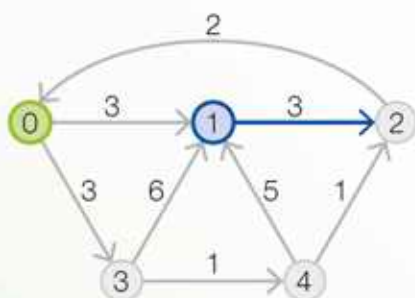
Żaden wierzchołek nie został jeszcze odwiedzony. Najniższy koszt ma wierzchołek 0, zatem od niego zaczynamy poszukiwanie najkrótszych dróg i oznaczamy go jako odwiedzony. Wychodzą z niego dwie krawędzie. Prowadzą one do wierzchołków o numerach 1 i 3, obie mają wagę 3. W tabeli zmieniamy koszty dotarcia do wierzchołków 1 i 3.



Numer wierzchołka	0	1	2	3	4
Koszt dotarcia do wierzchołka	0	3	∞	3	∞

Krok 2

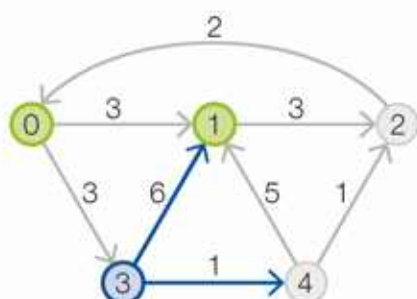
Najmniejszy koszt z nieodwiedzonych wierzchołków (1, 2, 3, 4) mają wierzchołki 1 i 3. Wybieramy dowolny z nich, np. wierzchołek 1, i oznaczamy go jako odwiedzony. Wychodzi z niego krawędź do wierzchołka 2. Koszt dojścia do wierzchołka 2 wynosi 6 i jest sumą kosztu dojścia do wierzchołka 1 oraz wagi krawędzi prowadzącej od wierzchołka 1 do 2 ($3 + 3$). Aktualizujemy w tabeli koszt dojścia do wierzchołka 2.



Numer wierzchołka	0	1	2	3	4
Koszt dotarcia do wierzchołka	0	3	6	3	∞

Krok 3

Spośród nieodwiedzonych wierzchołków (2, 3, 4) wybieramy ten o najmniejszym koszcie dotarcia – wierzchołek 3 – i oznaczamy go jako odwiedzony. Można z niego dojść do wierzchołków 1 oraz 4. Wierzchołek 1 został już odwiedzony, więc go nie rozpatrujemy. Koszt dotarcia do wierzchołka 4 przez wierzchołek 3 jest równy 4 ($3 + 1$). Zapisujemy tę wartość w tabeli.

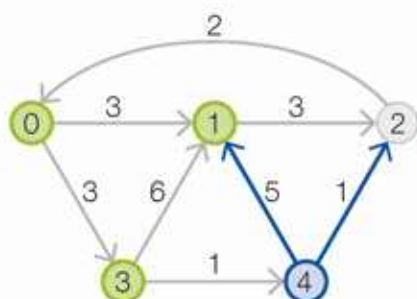


Numer wierzchołka	0	1	2	3	4
Koszt dotarcia do wierzchołka	0	3	6	3	4

0	1	2	3	4
0	3	6	3	4

Krok 4

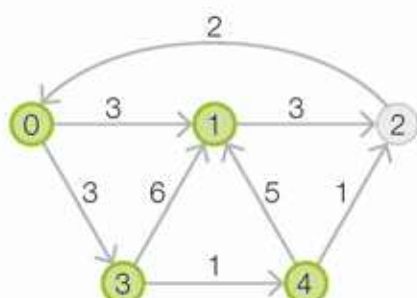
Z nieodwiedzonych wierzchołków (2 i 4) wybieramy ten o najmniejszym koszcie dotarcia – wierzchołek 4 – i oznaczamy go jako odwiedzony. Można z niego dojść do wierzchołków 1 i 2. Wierzchołek 1 został już odwiedzony, więc go nie rozpatrujemy. Koszt dotarcia do wierzchołka 2 jest równy 5 ($4 + 1$). Jest mniejszy od dotychczas znalezionej (6), dlatego zmieniamy wartość w tabeli.



Numer wierzchołka	0	1	2	3	4
Koszt dotarcia do wierzchołka	0	3	5	3	4

0	1	2	3	4
0	3	5	3	4

Algorytm zakończył działanie, ponieważ pozostał jeden nieodwiedzony wierzchołek (2), z którego nie może już wychodzić żadna krawędź do nieodwiedzonego wierzchołka.



Wynikiem działania algorytmu są koszty dotarcia z wierzchołka 0 do pozostałych wierzchołków przedstawione w poniższej tabeli.

Numer wierzchołka	0	1	2	3	4
Koszt dotarcia do wierzchołka	0	3	5	3	4

0	1	2	3	4
0	3	5	3	4

Specyfikacja problemu znajdowania najmniejszych kosztów dojścia od danego wierzchołka do pozostałych wierzchołków grafu wygląda następująco:

Specyfikacja

Dane: n – liczba wierzchołków grafu,

m – liczba krawędzi grafu,

Graf – reprezentacja skierowanego grafu ważonego o nieujemnych wagach krawędzi w postaci list sąsiedztwa,

pocz – numer wierzchołka początkowego.

Wynik: $\text{Koszt}[n]$ – tablica najmniejszych kosztów dojścia z wierzchołka pocz do pozostałych wierzchołków.

Na początku dla każdego wierzchołka koszt dojścia ustawimy na nieskończoność (w programie będzie to duża wartość przekraczająca możliwy koszt drogi). Dla wierzchołka pocz koszt wynosi 0. Algorytm polega na przeglądaniu wierzchołków. Za każdym razem wybieramy wierzchołek jeszcze nieodwiedzony o najmniejszym dotychczas znalezionym koszcie dojścia i oznaczamy go jako odwiedzony. Następnie rozpatrujemy krawędzie wychodzące z tego wierzchołka i aktualizujemy koszty dotarcia do sąsiednich nieodwiedzonych jeszcze wierzchołków, o ile są niższe.

Oto zapis algorytmu w pseudokodzie:

```
dla i ← 0, 1, ..., n - 1 wykonuj
    Odwiedzone[i] ← fałsz
    Koszt[i] ← ∞
Koszt[pocz] ← 0
dla i ← 0, 1, ..., n - 2 wykonuj
    w1 ← numer wierzchołka nieodwiedzonego
        o najmniejszym koszcie
    Odwiedzone[w1] ← prawda
    dla j ← 0, 1, ..., (liczba krawędzi z w1) - 1 wykonuj
        w2 ← Graf[w1][j]
        waga ← waga krawędzi od w1 do w2
        jeśli nie Odwiedzone[w2] oraz
            Koszt[w1] + waga < Koszt[w2] to
                Koszt[w2] ← Koszt[w1] + waga
```

Implementując powyższy algorytm, musimy uwzględnić wagę krawędzi. Krawędź będzie więc reprezentowana przez dwie liczby: numer wierzchołka i wagę krawędzi. Wygodnie będzie zdefiniować krawędź jako strukturę. Wówczas graf w postaci list sąsiedztwa możemy reprezentować jako zmienną typu **vector** o elementach typu **vector**, które przechowują informacje o krawędziach.

Definicja struktury przechowującej informację o krawędzi oraz definicja grafu wykorzystująca tę strukturę mogą wyglądać następująco.

Warto wiedzieć

Planszę z labiryntem z tematu 2 można reprezentować w postaci grafu. Pola, na których można stanąć, odpowiadają wierzchołkom. Jeśli na dwóch sąsiednich polach można stanąć, to otrzymujemy krawędź pomiędzy wierzchołkami. Odległość (waga krawędzi) pomiędzy dwoma sąsiednimi wierzchołkami jest dla każdej krawędzi taka sama (jednostkowa).

```

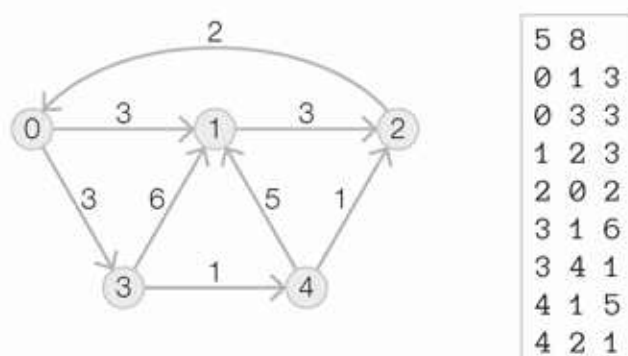
1. struct krawedz
2. {
3.     int w2;    // numer wierzchołka będącego
4.               // końcem krawędzi
5.     int waga;  // waga krawędzi
6. };
7.
8. typedef vector<vector<krawedz> > tgraf;

```

Definicja struktury przechowującej informacje o krawędziach skierowanego grafu ważonego oraz definicja grafu wykorzystująca tę strukturę

Opis skierowanego grafu ważonego w pliku tekstowym będzie wyglądał podobnie jak opis grafu, w którym krawędziom nie przypisano wag (rys. 4.8, s. 68). W każdym wierszu reprezentującym krawędź dodamy trzecią liczbę określającą wagę.

Przykład skierowanego grafu ważonego oraz jego opis w pliku tekstowym przedstawia rysunek 4.11.



Rys. 4.11. Przykład skierowanego grafu ważonego i jego opis w pliku tekstowym

Kod źródłowy zmodyfikowanej funkcji Czytaj, która wczyta dane o grafie ważonym z pliku tekstowego, oraz kod źródłowy funkcji realizującej algorytm Dijkstry są takie jak na s. 78.

A to ciekawe

Mosty królewieckie

Czy można narysować kopertę bez odrywania ołówka od kartki tak, aby nie rysować dwa razy tego samego odcinka? Ta i wiele podobnych zagadek sprowadza się do problemów związanych z grafami. Jednym z nich jest słynny problem mostów królewieckich.

W osiemnastowiecznym Królewcu (dzisiejszy Kaliningrad)

było aż siedem mostów łączących różne części miasta (zdjęcie obok przedstawia jeden z mostów w 1920 r.). Pytanie brzmiało: czy można przejść przez wszystkie mosty tak, aby każdy odwiedzić tylko raz i wrócić do miejsca, z którego się wyruszyło? W 1736 r. szwajcarski matematyk Leonhard Euler wykazał, że jest to niemożliwe. Jego praca dała początek teorii grafów – działowi matematyki zajmującemu się badaniem ich własności.



Kod źródłowy funkcji
Czytaj, wczytującej
z pliku tekstowego opis
grafu ważonego

```

1. void Czytaj(tgraf &Graf)
2. {
3.     int n, m, w1;
4.     krawedz kraw;
5.     ifstream we("graf_1.txt");
6.     we>>n>>m;
7.     Graf.resize(n);
8.     for (int i=0;i<m;i++)
9.     {
10.         we>>w1>>kraw.w2>>kraw.waga;
11.         Graf[w1].push_back(kraw);
12.     }
13.     we.close();
14. }

```

Kod źródłowy funkcji
realizującej algorytm
Dijkstry

```

1. void Dijkstra(tgraf Graf, int pocz, vector<int> &Koszt)
2. {
3.     krawedz kraw;
4.     int i, j, k, w1;
5.     vector<bool> Odwiedzone;
6.     Odwiedzone.resize(Graf.size(),false);
7.     Koszt[pocz]=0;
8.     for (i=0;i<Graf.size()-1;i++)
9.     {
10.         k=0; while (Odwiedzone[k]) k++; w1=k;
11.         for (j=k+1;j<Graf.size();j++)
12.             if (!Odwiedzone[j] && Koszt[j]<Koszt[w1]) w1=j;
13.         Odwiedzone[w1]=true;
14.         for (j=0;j<Graf[w1].size();j++)
15.         {
16.             kraw=Graf[w1][j];
17.             if (!Odwiedzone[kraw.w2] &&
18.                 Koszt[w1]+kraw.waga<Koszt[kraw.w2])
19.                 Koszt[kraw.w2]=Koszt[w1]+kraw.waga;
20.         }
21.     }
22. }

```

Pętla w liniach 8–21 przegląda wierzchołki grafu. Jednym z kluczowych elementów algorytmu jest szukanie nieodwiedzonego wierzchołka o najmniejszym dotychczas znalezionym koszcie. Wykonują to instrukcje w liniach 10–12. Najpierw znajdowany jest nieodwiedzony wierzchołek o najmniejszym numerze (linia 10) – wartość początkowa zmiennej w1. Następnie w pętli przeglądamy tablicę kosztów w poszukiwaniu nieodwiedzonego wierzchołka o mniejszym koszcie (linie 11–12). Pętla w liniach 14–20 przegląda krawędzie wychodzące ze znalezionej wierzchołka i modyfikuje koszt jego nieodwiedzonych sąsiadów, jeśli przejście po danej krawędzi stanowi krótszą drogę.

Kod źródłowy funkcji main może być następujący:

```

1. int main()
2. {
3.     tgraf Graf;
4.     Czytaj(Graf);
5.     vector<int> Koszt;
6.     Koszt.resize(Graf.size(),1000);
7.     int pocz;
8.     cout<<"Numer wierzchołka początkowego: ";
9.     cin>>pocz;
10.    Dijkstra(Graf,pocz,Koszt);
11.    cout<<"Koszt dojścia z wierzchołka "<<pocz;
12.    cout<<" do wierzchołka:"<<endl;
13.    for (int i=0;i<Graf.size();i++)
14.        if (i!=pocz) cout<<i<<" : "<<Koszt[i]<<endl;
15.    return 0;
16. }
```

❶ Fragment kodu źródłowego programu obliczającego koszty najkrótszych dróg od danego wierzchołka do pozostałych wierzchołków grafu – funkcja main

Zwróć uwagę na linię 6. Tworzona jest w niej tablica kosztów. Każdemu elementowi tej tablicy przypisana jest wartość 1000, czyli wartość wielokrotnie większa od kosztu potencjalnej drogi. Pętla w liniach 13–14 wypisuje wartości tablicy Koszt z wyjątkiem kosztu wierzchołka początkowego, czyli najmniejsze koszty dojścia do poszczególnych wierzchołków. Jeśli droga dojścia do jakiegoś wierzchołka nie istnieje, to zostanie wypisana wartość początkowa 1000.

Rysunek 4.12 przedstawia efekt wykonania programu dla grafu z rysunku 4.11 ze s. 77, jeśli wierzchołkiem początkowym jest wierzchołek o numerze 0.

```

Numer wierzchołka początkowego: 0
Koszt dojścia z wierzchołka 0 do wierzchołka:
1: 3
2: 5
3: 3
4: 4
```

Rys. 4.12. Efekt wykonania programu dla grafu z rysunku 4.11 i wierzchołka początkowego o numerze 0

Ćwiczenie 7

Napisz program wczytujący opis grafu ważonego z pliku tekstowego, który otrzymasz od nauczyciela (np. *graf_1.txt*), i realizujący algorytm, który znajduje najniższe koszty dojścia z wierzchołka początkowego do pozostałych wierzchołków.

Złożoność obliczeniowa przedstawionej implementacji algorytmu znajdowania najmniejszych kosztów jest kwadratowa względem liczby wierzchołków. Przeglądamy $n - 1$ wierzchołków, a wyszukanie wierzchołka o najmniejszym koszcie odbywa się liniowo w pętli przeglądającej wszystkie wierzchołki. Złożoność obliczeniową można zredukować do liniowo-logarytmicznej, usprawniając metodę wyszukania kolejnego wierzchołka do rozpatrzenia. Wymaga to użycia **kolejki priorytetowej**. Jest to kolejka, w której elementy są uporządkowane według określonego kryterium. Koszt wstawienia elementu do takiej kolejki (a także koszt usunięcia elementu) jest logarytmiczny. Implementację tej wersji algorytmu pomijamy.

Kolejka priorytetowa

Pozostaje jeszcze zmodyfikować algorytm tak, aby oprócz znajdowania najmniejszych kosztów dojścia z wierzchołka początkowego do pozostałych wskazywał drogę do wybranego wierzchołka końcowego. Oto zmodyfikowana specyfikacja problemu:

Specyfikacja

Dane: n – liczba wierzchołków grafu,

m – liczba krawędzi grafu,

Graf – reprezentacja skierowanego grafu ważonego o nieujemnych wagach krawędzi w postaci list sąsiedztwa,

pocz – numer wierzchołka początkowego,

kon – numer wierzchołka końcowego.

Wynik: Koszt[n] – tablica najmniejszych kosztów dojścia z wierzchołka pocz do pozostałych wierzchołków, numery wierzchołków, przez które należy przejść od pocz do kon z kosztem Koszt[kon].

Zapis zmodyfikowanego algorytmu w pseudokodzie jest następujący:

```
dla i ← 0, 1, ..., n - 1 wykonuj
    Odwiedzone[i] ← fałsz
    Koszt[i] ← ∞
    Poprz[i] ← -1
Koszt[pocz] ← 0
dla i ← 0, 1, ..., n - 2 wykonuj
    w1 ← numer wierzchołka nieodwiedzonego
        o najmniejszym koszcie
    Odwiedzone[w1] ← prawda
    dla j ← 0, 1, ..., (liczba krawędzi z w1) - 1 wykonuj
        w2 ← Graf[w1][j]
        waga ← waga krawędzi od w1 do w2
        jeśli nie Odwiedzone[w2] oraz
            Koszt[w1] + waga < Koszt[w2] to
            Koszt[w2] ← Koszt[w1] + waga
            Poprz[w2] ← w1
```

W pseudokodzie użyliśmy dodatkowej tablicy o nazwie `Poprz`, w której dla każdego wierzchołka jest pamiętany numer wierzchołka bezpośrednio go poprzedzającego na drodze o najniższym koszcie. Elementom tablicy `Poprz` przypisujemy początkowe wartości -1 . Oznaczają one, że na początku działania algorytmu wierzchołki nie mają wierzchołków poprzedzających. Podczas aktualizacji elementu tablicy `Koszt` aktualizujemy także wartość w tablicy `Poprz`.

Na podstawie informacji z tablicy `Poprz`, korzystając ze **stosu**, możemy wypisać drogę o najniższym koszcie od wierzchołka początkowego do końcowego.

Stos,
s. 11 [↗](#)

```
w ← kon
włóż w na stos
dopóki w ≠ pocz wykonuj
    w ← Poprz[w]
    włóż w na stos
dopóki nie pusty stos wykonuj
    wypisz wierzchołek stosu
    usuń element ze stosu
```

Kod źródłowy zmodyfikowanej funkcji `main` programu wyznaczającego drogę o najmniejszym koszcie może być następujący:

```
1. int main()
2. {
3.     tgraf Graf;
4.     Czytaj(Graf);
5.     vector<int> Koszt;
6.     Koszt.resize(Graf.size(),1000);
7.     vector<int> Poprz;
8.     Poprz.resize(Graf.size(),-1);
9.     int pocz, kon;
10.    cout<<"Numer wierzcholka początkowego: ";
11.    cin>>pocz;
12.    cout<<"Numer wierzcholka końcowego: ";
13.    cin>>kon;
14.    Dijkstra(Graf,pocz,Koszt,Poprz);
15.    cout<<"Koszt dojscia: "<<Koszt[kon]<<endl<<"Droga: ";
16.    WypiszDroge(pocz,kon,Poprz);
17.    return 0;
18. }
```

Fragment kodu
źródłowego programu
wyznaczającego
drogę o najmniejszym
koszcie między dwoma
wierzchołkami – funkcja
`main`

W linii 8 tworzona i inicjowana jest tablica `Poprz`. Zwróć uwagę na wywołanie funkcji `Dijkstra` (linia 14). Ma ona dodatkowy parametr – tablicę `Poprz`. Funkcja określa jej wartości, które są potem wykorzystywane w funkcji `WypiszDroge` (linia 16), realizującej algorytm przedstawiony w pseudokodzie. Dlatego parametr ten w funkcji `Dijkstra` przekazywany jest przez referencję, tak jak tablica `Koszt`.

Ćwiczenie 8

Napisz program, który wczyta opis skierowanego grafu ważonego z pliku tekstowego otrzymanego od nauczyciela (np. *graf_1.txt*), a następnie znajdzie i wypisze drogę o najmniejszym koszcie łączącą dwa podane wierzchołki.

Rysunek 4.13 przedstawia przykład wykonania programu dla grafu z rysunku 4.11 ze s. 77. Najmniejszy koszt dojścia z wierzchołka o numerze 0 do wierzchołka o numerze 2 wynosi 5. Droga o takim koszcie prowadzi przez wierzchołki 0, 3, 4, 2.

```
Numer wierzchołka początkowego: 0
Numer wierzchołka końcowego: 2
Koszt dojścia: 5
Droga: 0-->3-->4-->2
```

Rys. 4.13. Efekt wykonania programu dla grafu z rysunku 4.11 i wierzchołków o numerach 0 i 2

Zapamiętaj

Algorytm Dijkstry wyznacza najmniejsze koszty dojścia z wybranego wierzchołka grafu do pozostałych wierzchołków oraz wskazuje, przez które wierzchołki prowadzą drogi o najmniejszych kosztach. Można go stosować dla grafu ważonego o nieujemnych wagach krawędzi. Przeglądając wierzchołki grafu, wybieramy nieodwiedzony jeszcze wierzchołek grafu o najmniejszym dotychczas znalezionym koszcie dojścia i rozpatrujemy sąsiadujące z nim nieodwiedzone wierzchołki.

A to ciekawe

Jak powstał algorytm Dijkstry

Nazwa algorytmu pochodzi od nazwiska jego twórcy – holenderskiego informatyka Edsgera Dijkstry. Podczas pracy w Centrum Matematycznym w Amsterdamie zajmował się on m.in. programowaniem maszyny obliczeniowej ARMAC. Aby zademonstrować jej możliwości podczas prezentacji w 1956 r., postanowił posłużyć się rozwiązaniem problemu, który zrozumieją także osoby niezwiązane z matematyką. Dlatego napisał program znajdujący najkrótszą trasę między dwoma miastami w Holandii, korzystając z mapy drogowej, na której wybrał 64 miasta. W jednym z wywiadów Dijkstra powiedział, że wymyślenie algorytmu zajęło mu ok. 20 minut, a na jego pomysł wpadł, kiedy wraz z narzeczoną odpoczywał po zakupach w kawiarni przy kawie.



Podsumowanie

- Graf składa się z niepustego zbioru wierzchołków i zbioru krawędzi, czyli połączeń między wierzchołkami.
- Istnieją różne rodzaje grafów. W grafie nieskierowanym po krawędzi można przejść w obu kierunkach. W grafie skierowanym kierunek przejścia po krawędzi jest określony.
- Krawędziom mogą być przypisane wagi, określające np. odległość lub czas przejścia pomiędzy wierzchołkami. Taki graf nazywamy ważonym.
- Jeśli w grafie dowolne dwa wierzchołki łączy co najwyżej jedna krawędź oraz z żadnego wierzchołka nie prowadzi krawędź do niego samego, to taki graf jest grafem prostym.
- Graf, w którym istnieje droga z każdego wierzchołka do każdego, to graf spójny.
- Graf można reprezentować m.in. w postaci macierzy sąsiedztwa lub list sąsiedztwa.
- Do reprezentacji grafu można wykorzystać zmienną typu **vector** z biblioteki STL.
- Typ **vector** łączy możliwości tablicy dynamicznej i listy. Tak jak w tablicy dynamicznej w trakcie działania programu można określać i zmieniać rozmiar zmiennej typu **vector**. Podobnie jak w przypadku list nowy element można wstawić w dowolne miejsce tej zmiennej, można także usunąć dowolny element.
- Jednym z podstawowych problemów dotyczących grafów jest przejście wierzchołków grafu w usystematyzowany sposób. Wyróżniamy dwie podstawowe metody przeglądania grafu: w głąb (DFS) oraz wszerz (BFS).
- Algorytm Dijkstry umożliwia znalezienie dróg o najmniejszych kosztach prowadzących od wybranego wierzchołka grafu ważonego o nieujemnych wagach krawędzi do wszystkich pozostałych wierzchołków grafu, do których można dojść.
- W każdym kroku algorytm Dijkstry wybiera nieodwiedzony wierzchołek o najmniejszym dotychczas znalezionym koszcie dojścia i rozpatruje sąsiadujące z nim nieodwiedzone wierzchołki.

Zadania

- ★ **1** Napisz program, który wczyta opis skierowanego grafu nieważonego z pliku tekstowego otrzymanego od nauczyciela (np. *graf_2.txt*) oraz numer wierzchołka początkowego z klawiatury. Liczby w pierwszym wierszu pliku określają liczbę wierzchołków i liczbę krawędzi grafu, każdy kolejny wiersz zawiera numery wierzchołków będących początkiem i końcem krawędzi. Program ma sprawdzić, czy da się dojść z tego wierzchołka do pozostałych wierzchołków grafu.
- ★ **2** Napisz program, który wczyta opis skierowanego grafu nieważonego z pliku tekstowego otrzymanego od nauczyciela (np. *graf_2.txt*) oraz numery wierzchołków początkowego i końcowego z klawiatury. Liczby w pierwszym wierszu pliku określają liczbę wierzchołków i liczbę krawędzi grafu, każdy kolejny wiersz zawiera numery wierzchołków będących początkiem i końcem krawędzi. Program powinien sprawdzić, czy istnieje droga z wierzchołka początkowego do końcowego.

- ★ **3** Napisz program, który wczyta opis skierowanego grafu nieważonego z pliku tekstowego otrzymanego od nauczyciela (np. *graf_2.txt*) oraz numer wierzchołka z klawiatury. Liczby w pierwszym wierszu pliku odpowiadają liczbie wierzchołków i liczbie krawędzi grafu. Każdy kolejny wiersz zawiera numery wierzchołków będących początkiem i końcem krawędzi. Program ma sprawdzić, czy istnieje cykl zawierający ten wierzchołek.
 Uwaga: Z cyklem w grafie mamy do czynienia, gdy istnieje ścieżka przejścia, w której wierzchołek początkowy i końcowy jest ten sam. Na przykład w grafie na rysunku 4.3 ze s. 63 jest cykl $0 \rightarrow 1 \rightarrow 2 \rightarrow 0$.
- ★★ **4** Napisz program, który wczyta opis skierowanego grafu nieważonego z pliku tekstowego otrzymanego od nauczyciela (np. *graf_2.txt*) i sprawdzi, czy w grafie istnieje cykl. Liczby w pierwszym wierszu pliku odpowiadają liczbie wierzchołków i liczbie krawędzi grafu. Każdy kolejny wiersz pliku zawiera numery wierzchołków będących początkiem i końcem krawędzi.
 Uwaga: Z cyklem w grafie mamy do czynienia, gdy istnieje ścieżka przejścia, w której wierzchołek początkowy i końcowy jest ten sam. Na przykład w grafie na rysunku 4.3 ze s. 63 jest cykl $0 \rightarrow 1 \rightarrow 2 \rightarrow 0$.
- ★★ **5** Napisz program, który wczyta opis skierowanego grafu nieważonego z pliku tekstowego otrzymanego od nauczyciela (np. *graf_2.txt*) i sprawdzi, czy graf jest spójny. Liczby w pierwszym wierszu pliku określają liczbę wierzchołków i liczbę krawędzi grafu, każdy kolejny wiersz zawiera numery wierzchołków będących początkiem i końcem krawędzi.
- ★★ **6** Napisz program, który wczyta opis skierowanego grafu nieważonego z pliku tekstowego otrzymanego od nauczyciela (np. *graf_2.txt*) i przejrzy wierzchołki grafu w głąb bez użycia rekurencji, z wykorzystaniem stosu. Liczby w pierwszym wierszu pliku odpowiadają liczbie wierzchołków i liczbie krawędzi grafu. Każdy kolejny wiersz pliku zawiera numery wierzchołków będących początkiem i końcem krawędzi.
- ★★★ **7** Napisz program, który wczyta opis skierowanego grafu ważonego z pliku tekstowego otrzymanego od nauczyciela (np. *graf_3.txt*) oraz numery wierzchołków początkowego i końcowego z klawiatury. W pierwszym wierszu pliku znajdują się dwie liczby: pierwsza odpowiada liczbie wierzchołków grafu, a druga liczbie krawędzi. Każdy następny wiersz zawiera cztery liczby odpowiadające kolejno: numerowi wierzchołka będącego początkiem krawędzi, numerowi wierzchołka będącego końcem krawędzi, odległości między wierzchołkami, czasowi przejścia między wierzchołkami. Program ma znaleźć najkrótszą oraz najszybszą drogę pomiędzy wczytanymi wierzchołkami.
- ★★★ **8** Napisz program, który wczyta opis skierowanego grafu ważonego z pliku tekstowego otrzymanego od nauczyciela (np. *graf_4.txt*) oraz numery wierzchołków początkowego i końcowego z klawiatury. Liczby w pierwszym wierszu pliku odpowiadają liczbie wierzchołków i liczbie krawędzi grafu. Każdy kolejny wiersz pliku zawiera: numer wierzchołka będącego początkiem krawędzi, numer wierzchołka będącego końcem krawędzi, wagę krawędzi. Program ma znaleźć najkrótszą drogę pomiędzy wczytanymi wierzchołkami w czasie $O(n \cdot \log n)$.
 Wskazówka: Wykorzystaj szablon typu `priority_queue` z biblioteki STL. Typ ten reprezentuje kolejkę priorytetową.



Sposób na zadania

Zadanie 1

W pliku, który otrzymasz od nauczyciela (np. *WUZ1_zad1_dane_liczby.txt*), w kolejnych 10 wierszach jest zapisanych po 20 liczb całkowitych z zakresu od 0 do 100 000 oddzielonych spacjami. Wśród tych 200 liczb są liczby zakończone każdą z cyfr od 0 do 9. Oto zawartość dwóch początkowych wierszy pliku:

```
820 21292 12637 322 21013 17178 16445 14757 9026 5161 25709 17961 14093 14598
12379 8458 8103 29640 426 2723
```

```
820 21292 12637 322 6736 6851 12166 8597 8781 9130 15482 25861 30047 15757 29632
30114 20605 29853 10139 14390
```

Napisz program lub programy rozwiązujące poniższe zadania. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.1 (0–3)

Utwórz plik wynikowy (np. *WUZ1_zad1_1_dane_liczby_wynik.txt*), w którym będzie 10 wierszy składających się z liczb oddzielonych spacjami. W poszczególnych wierszach znajdą się liczby z pliku z danymi, przy czym w pierwszym wierszu będą liczby zakończone cyfrą 0, w drugim – cyfrą 1 itd. Liczby w danym wierszu pliku wynikowego powinny wystąpić w tej samej kolejności co w pliku z danymi. Na przykład dla dwóch początkowych wierszy pliku z danymi wynik będzie następujący:

```
820 29640 820 9130 14390
5161 17961 6851 8781 25861
21292 322 21292 322 15482 29632
21013 14093 8103 2723 29853
30114
16445 20605
9026 426 6736 12166
12637 14757 12637 8597 30047 15757
17178 14598 8458
25709 12379 10139
```

Rozwiązanie

Tagi: tablica, dynamiczna struktura danych, kolejka

Sposób I. Wielokrotny odczyt danych z pliku bez użycia złożonych struktur danych

Krok 1

Liczby z pliku z danymi należy podzielić na 10 grup. O przynależności do grupy decyduje ostatnia cyfra liczby, czyli reszta z dzielenia liczby przez 10. Przejrzemy cały plik z danymi 10 razy. Za każdym razem zapiszemy do pliku wynikowego w oddzielnym wierszu liczby należące do jednej grupy.

Krok 2

Liczby z pliku z danymi odczytamy za pomocą zmiennej plikowej we typu **ifstream**. Wyniki zapiszemy z wykorzystaniem zmiennej plikowej wy typu **ofstream**.

Krok 3

Dla każdej grupy liczb będziemy otwierać i zamykać plik z danymi – otworzymy go za pomocą metody `open`, zamkniemy przy użyciu metody `close`. Po wypisaniu liczby dodamy spację, a po wypisaniu grupy liczb – znacznik końca wiersza `endl`. Na koniec zamkniemy plik wynikowy.

Odpowiedź: Kod źródłowy programu rozwiązującego to zadanie może być następujący:

```

1. #include <fstream>
2.
3. using namespace std;
4.
5. int main()
6. {
7.     int x;
8.     ifstream we;
9.     ofstream wy("WUZ1_zad1_1_dane_liczby_wynik.txt");
10.    for (int i=0;i<10;i++)
11.    {
12.        we.open("WUZ1_zad1_dane_liczby.txt");
13.        for (int j=0;j<200;j++)
14.        {
15.            we>>x;
16.            if (x%10==i)
17.                wy<<x<<" ";
18.        }
19.        wy<<endl;
20.        we.close();
21.    }
22.    wy.close();
23.    return 0;
24. }
```

Sposób II. Z wykorzystaniem tablicy, której elementami są liczby całkowite

Krok 1

Po otwarciu pliku z danymi odczytamy kolejne liczby i zapiszemy je w tablicy. Dzięki temu plik wejściowy przejrzymy tylko raz. Do przechowania liczb możemy wykorzystać np. tablicę dwuwymiarową o 10 wierszach i 20 kolumnach lub jednowymiarową o 200 elementach. Użyjemy tablicy jednowymiarowej.

Krok 2

Liczby zapisane w tablicy należy podzielić na 10 grup. O przynależności do grupy decyduje ostatnia cyfra liczby, czyli reszta z dzielenia liczby przez 10. Przejrzymy tablicę 10 razy, za każdym razem zapiszemy do pliku wynikowego liczby należące do jednej grupy.

Krok 3

Podczas wypisywania liczb z grupy po każdej liczbie dodamy spację, a po wypisaniu wszystkich liczb z grupy – znacznik końca wiersza `endl`.

Odpowiedź: Kod źródłowy programu rozwiązującego to zadanie może być następujący:

```

1. #include <fstream>
2.
3. using namespace std;
4.
5. int main()
6. {
7.     int Tab[200];
8.     ifstream we("WUZ1_zad1_dane_liczby.txt");
9.     for (int i=0;i<200;i++)
10.         we>>Tab[i];
11.     we.close();
12.     ofstream wy("WUZ1_zad1_1_dane_liczby_wynik.txt");
13.     for (int i=0;i<10;i++)
14.     {
15.         for (int j=0;j<200;j++)
16.             if (Tab[j]%10==i)
17.                 wy<<Tab[j]<<" ";
18.         wy<<endl;
19.     }
20.     wy.close();
21.     return 0;
22. }
```

Sposób III. Z wykorzystaniem tablicy, której elementami są kolejki złożone z liczb całkowitych

Krok 1

Liczby z pliku z danymi należy podzielić na 10 grup. O przynależności do grupy decyduje ostatnia cyfra liczby, czyli reszta z dzielenia liczby przez 10. Ponieważ liczby w grupie mają zachować kolejność występowania z pliku wejściowego, do ich pamiętania możemy wykorzystać strukturę danych o nazwie kolejka. Utworzymy 10-elementową tablicę, której poszczególne elementy to kolejki przechowujące liczby z poszczególnych grup (zaczynając od grupy liczb zakończonych cyfrą 0, a kończąc na grupie liczb zakończonych cyfrą 9).

Krok 2

Plik z danymi przejrzymy tylko raz. Podczas odczytywania danych z pliku każdą liczbę dołączymy do właściwej kolejki. Liczbę zakończoną cyfrą 0 dołączymy do kolejki pamiętanej w tablicy pod indeksem 0, liczbę zakończoną cyfrą 1 – do kolejki pamiętanej pod indeksem 1 itd.

Krok 3

Przejrzymy wszystkie kolejki, zaczynając od kolejki pamiętającej liczby zakończone cyfrą 0, a kończąc na kolejce pamiętającej liczby zakończone cyfrą 9. Będziemy usuwać liczby z kolejek i zapisywać je w pliku wynikowym. Po wypisaniu każdej liczby dodamy spację, a po przejrzeniu wszystkich elementów z danej kolejki – znacznik końca wiersza **endl**.

Odpowiedź: Kod źródłowy programu rozwiązującego to zadanie może być następujący:

```

1. #include <fstream>
2. #include <queue>
3.
4. using namespace std;
5.
6. int main()
7. {
8.     int x;
9.     queue<int> Tab[10];
10.    ifstream we("WUZ1_zad1_dane_liczby.txt");
11.    for (int i=0;i<200;i++)
12.    {
13.        we>>x;
14.        Tab[x%10].push(x);
15.    }
16.    we.close();
17.    ofstream wy("WUZ1_zad1_1_dane_liczby_wynik.txt");
18.    for (int i=0;i<10;i++)
19.    {
20.        while (!Tab[i].empty())
21.        {
22.            wy<<Tab[i].front()<<" ";
23.            Tab[i].pop();
24.        }
25.        wy<<endl;
26.    }
27.    wy.close();
28.    return 0;
29. }

```

Zadanie 1.2 (0–3)

Utwórz plik wynikowy (np. *WUZ1_zad1_2_dane_liczby_wynik.txt*), w którym będzie 10 wierszy składających się z liczb oddzielonych spacjami. W poszczególnych wierszach znajdą się liczby z pliku z danymi, przy czym w pierwszym wierszu będą liczby zakończone cyfrą 0, w drugim – cyfrą 1 itd. Liczby w danym wierszu pliku wynikowego powinny wystąpić w odwrotnej kolejności niż w pliku z danymi. Na przykład dla dwóch początkowych wierszy pliku z danymi wynik będzie następujący:

```

14390 9130 820 29649 820
25861 8781 6851 17961 5161
29632 15482 322 21292 322 21292
29853 2723 8103 14093 21013
30114
20605 16445
12166 6736 426 9026
15757 30047 8597 12637 14757 12637
8458 14598 17178
10139 12379 25709

```

Rozwiązanie

Tagi: tablica, dynamiczna struktura danych, stos

Sposób I. Z wykorzystaniem tablicy, której elementami są liczby całkowite

Krok 1

Aby plik z danymi przejrzeć tylko raz, po jego otwarciu odczytamy kolejne liczby i zapiszemy je w tablicy. Do przechowania liczb możemy wykorzystać np. tablicę dwuwymiarową o 10 wierszach i 20 kolumnach lub jednowymiarową o 200 elementach. Użyjemy tablicy jednowymiarowej.

Krok 2

Liczby pamiętane w tablicy należy podzielić na 10 grup. O przynależności do grupy decyduje ostatnia cyfra liczby, czyli reszta z dzielenia liczby przez 10. Liczby w pliku wynikowym w danej grupie mają wystąpić w odwrotnej kolejności niż w pliku z danymi, dlatego przejrzymy tablicę 10 razy od ostatniego elementu do pierwszego i za każdym razem zapiszemy do pliku wynikowego liczby należące do jednej grupy.

Krok 3

Po wypisaniu każdej liczby dodamy spację, a po wypisaniu wszystkich liczb z danej grupy – znacznik końca wiersza `endl`.

Odpowiedź: Kod źródłowy programu rozwiązującego to zadanie może być następujący:

```

1. #include <fstream>
2.
3. using namespace std;
4.
5. int main()
6. {
7.     int Tab[200];
8.     ifstream we("WUZ1_zad1_dane_liczby.txt");
9.     for (int i=0;i<200;i++)
10.         we>>Tab[i];
11.     we.close();
12.     ofstream wy("WUZ1_zad1_2_dane_liczby_wynik.txt");
13.     for (int i=0;i<10;i++)
14.     {
15.         for (int j=199;j>=0;j--)
16.             if (Tab[j]%10==i)
17.                 wy<<Tab[j]<<" ";
18.         wy<<endl;
19.     }
20.     wy.close();
21.     return 0;
22. }
```


Sposób II. Z wykorzystaniem tablicy, której każdy element jest stosem przechowującym liczby całkowite

Krok 1

Liczby z pliku z danymi należy podzielić na 10 grup. O przynależności do grupy decyduje ostatnia cyfra liczby, czyli reszta z dzielenia liczby przez 10. Ponieważ liczby w danej grupie mają wystąpić w odwrotnej kolejności niż w pliku wejściowym, do pamiętania liczb z grupy wykorzystamy strukturę danych o nazwie `stos`. Utworzymy 10-elementową tablicę, w której każdy z elementów będzie stosem pamiętającym liczby z jednej grupy (zaczynając od grupy liczb zakończonych cyfrą 0, a kończąc na grupie liczb zakończonych cyfrą 9).

Krok 2

Podczas odczytywania danych z pliku wejściowego każdą liczbę włożymy na właściwy stos. Dzięki temu nie będziemy musieli wielokrotnie przeglądać danych.

Krok 3

Przejrzymy wszystkie stosy, zaczynając od stosu pamiętającego liczby zakończone cyfrą 0, a kończąc na stosie pamiętającym liczby zakończone cyfrą 9. Będziemy zdejmować liczby ze stosów i zapisywać je w pliku wynikowym. Po wypisaniu każdej liczby dodamy spację, a po zdjęciu wszystkich liczb z danego stosu – znacznik końca wiersza `endl`.

Odpowiedź: Kod źródłowy programu rozwiązującego to zadanie może być następujący:

```

1. #include <fstream>
2. #include <stack>
3.
4. using namespace std;
5.
6. int main()
7. {
8.     int x;
9.     stack<int> Tab[10];
10.    ifstream we("WUZ1_zad1_dane_liczby.txt");
11.    for (int i=0; i<200; i++)
12.    {
13.        we>>x; Tab[x%10].push(x);
14.    }
15.    we.close();
16.    ofstream wy("WUZ1_zad1_2_dane_liczby_wynik.txt");
17.    for (int i=0; i<10; i++)
18.    {
19.        while (!Tab[i].empty())
20.        {
21.            wy<<Tab[i].top()<<" ";
22.            Tab[i].pop();
23.        }
24.        wy<<endl;
25.    }
26.    wy.close();
27.    return 0;
28. }
```

Zadanie 1.3 (0–4)

Każdy wiersz pliku z danymi to ciąg liczbowy złożony z 20 wyrazów. Posortuj te ciągi leksykograficznie i zapisz uporządkowane ciągi w pliku wynikowym (np. *WUZ1_zad1_3_dane_liczby_wynik.txt*), każdy w oddzielnym wierszu. Z dwóch ciągów wcześniej powinien być zapisany ten, którego pierwszy wyraz różniący ciągi (patrzac od lewej) jest mniejszą liczbą. Na przykład z dwóch początkowych ciągów liczbowych z pliku z danymi wcześniejszy będzie ciąg drugi. Pierwszy wyraz różniący ciągi występuje na piątej pozycji, w pierwszym ciągu jest równy 21013, a w drugim 6736.

Rozwiązanie

Tagi: tablica, algorytmy sortowania, sortowanie leksykograficzne

Sposób I. Z wykorzystaniem pomocniczej tablicy pamiętającej kolejność ciągów

Krok 1

Po otwarciu pliku z danymi odczytamy kolejne liczby i zapiszemy je w tablicy dwuwymiarowej o 10 wierszach i 20 kolumnach. W każdym wierszu tablicy zapamiętamy jeden ciąg liczbowy. Dodatkowo zadeklarujemy tablicę jednowymiarową o 10 elementach, która będzie pamiętać kolejność ciągów. Wartości początkowe tej tablicy to kolejne liczby całkowite nieujemne, a początkowa kolejność ciągów jest zgodna z kolejnością ciągów w pliku z danymi.

Krok 2

Zdefiniujemy funkcję porównującą dwa ciągi. Jej parametrami będą dwie tablice jednowymiarowe o 20 elementach. Funkcja zwróci wartość **true**, gdy pierwszy ciąg powinien wystąpić wcześniej lub gdy oba ciągi są takie same. W przeciwnym przypadku funkcja zwróci wartość **false**.

Krok 3

Będziemy sortować elementy z pomocniczej tablicy pamiętającej kolejność ciągów. Można wykorzystać dowolną metodę sortowania. My zastosujemy sortowanie przez wybieranie. O ewentualnej zamianie wartości elementów tablicy pomocniczej decyduje wynik porównania dwóch ciągów pamiętanych w tablicy dwuwymiarowej. Numery wierszy, w których są pamiętane porównywane ciągi, są określone przez elementy tablicy pamiętającej kolejność ciągów.

Krok 4

Zapisujemy do pliku wynikowego ciągi we właściwej kolejności. Przeglądamy tablicę pamiętającą kolejność ciągów i wypisujemy elementy z odpowiednich wierszy tablicy dwuwymiarowej – numer wiersza tablicy dwuwymiarowej odpowiada elementowi z tablicy pamiętającej kolejność ciągów.

Odpowiedź: Kod źródłowy programu rozwiązującego to zadanie może być następujący:

```
1. #include <fstream>
2. using namespace std;
3.
4. const int N=10, M=20;
5.
6. bool Wczesniejszy(int Tab1[], int Tab2[])
7. {
8.     int i=0;
9.     while (i<M && Tab1[i]==Tab2[i]) i++;
10.    if (i<M) return (Tab1[i]<Tab2[i]);
11.    else return true;
12. }
```



```

13.
14. int main()
15. {
16.     int Tab[N][M], W[N];
17.     for (int i=0; i<N; i++)
18.         W[i]=i;
19.     ifstream we("WUZ1_zad1_dane_liczby.txt");
20.     for (int i=0; i<N; i++)
21.         for (int j=0; j<M; j++)
22.             we>>Tab[i][j];
23.     we.close();
24.     for (int i=0; i<N-1; i++)
25.     {
26.         int k=i;
27.         for (int j=i+1; j<N; j++)
28.             if (Wczesniejszy(Tab[W[j]], Tab[W[k]]))
29.                 k=j;
30.         swap(W[i], W[k]);
31.     }
32.     ofstream wy("WUZ1_zad1_3_dane_liczby_wynik.txt");
33.     for (int i=0; i<N; i++)
34.     {
35.         for (int j=0; j<M; j++)
36.             wy<<Tab[W[i]][j]<<" ";
37.         wy<<endl;
38.     }
39.     wy.close();
40.     return 0;
41. }

```

Sposób II. Z wykorzystaniem funkcji `sort` z biblioteki STL

Krok 1

Do pamiętania ciągów wykorzystamy typ `vector` z biblioteki STL. Utworzymy zmienną typu `vector` o 10 elementach. Każdy element tej zmiennej również będzie typu `vector` i będzie reprezentować jeden ciąg liczbowy. Odczytamy dane z pliku i zapamiętamy je w tak utworzonej strukturze danych.

Krok 2

Zdefiniujemy pomocniczą funkcję porównującą dwa ciągi. Jej parametry będą typu `vector` – każdy z nich będzie odpowiadał jednemu ciągowi liczbowemu. Wartością funkcji będzie wartość logiczna `true`, gdy pierwszy ciąg powinien wystąpić wcześniej lub gdy oba ciągi są takie same. W przeciwnym przypadku funkcja zwróci wartość `false`.

Krok 3

Do uporządkowania ciągów we właściwej kolejności użyjemy funkcji `sort` z biblioteki STL. Jej parametrami będą wartości metod `begin` i `end` struktury danych pamiętającej ciągi oraz nazwa funkcji porównującej dwa ciągi.

Krok 4

Przeglądamy wartości zmiennej typu **vector** pamiętającej posortowane leksykograficznie ciągi i wypisujemy je do pliku wynikowego.

Odpowiedź: Kod źródłowy programu rozwiązującego to zadanie może być następujący:

```

1. #include <fstream>
2. #include <vector>
3. #include <algorithm>
4.
5. using namespace std;
6.
7. const int N=10, M=20;
8.
9. bool Wczesniejszy(vector<int> Tab1, vector<int> Tab2)
10. {
11.     int i=0;
12.     while (i<M && Tab1[i]==Tab2[i])
13.         i++;
14.     if (i<M)
15.         return (Tab1[i]<Tab2[i]);
16.     else return true;
17. }
18.
19. int main()
20. {
21.     vector<vector<int> > Tab;
22.     Tab.resize(N);
23.     for (int i=0;i<N;i++)
24.         Tab[i].resize(M);
25.     ifstream we("WUZ1_zad1_dane_liczby.txt");
26.     for (int i=0;i<N;i++)
27.         for (int j=0;j<M;j++)
28.             we>>Tab[i][j];
29.     we.close();
30.     sort(Tab.begin(),Tab.end(),Wczesniejszy);
31.     ofstream wy("WUZ1_zad1_3_dane_liczby_wynik.txt");
32.     for (int i=0;i<N;i++)
33.     {
34.         for (int j=0;j<M;j++)
35.             wy<<Tab[i][j]<<" ";
36.         wy<<endl;
37.     }
38.     wy.close();
39.     return 0;
40. }

```


Trening

■ Zadanie 1

W pliku, który otrzymasz od nauczyciela (np. *WUZ1_zad1_ciagi.txt*), zapisanych jest 100 liczb całkowitych z zakresu od 0 do 100 000, każda w oddzielnym wierszu. Ostatnia liczba w pliku jest równa 0. W pliku nie występują bezpośrednio jedna po drugiej dwie liczby 0. Wyrazy z pierwszych wierszy pliku aż do liczby 0 tworzą ciąg liczbowy – liczba 0 nie należy do tego ciągu, pełni jedynie funkcję znacznika końca ciągu. Kolejne ciągi składają się z liczb znajdujących się między liczbami 0. Pierwszych 14 wierszy z danymi wygląda następująco:

```
29660
3662
6457
2340
26892
21832
23534
0
3153
5304
14659
10064
19277
0
```

Napisz program lub programy rozwiązujące poniższe zadania. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.1 (0–2) ROZWIĄŻ NA KOMPETERZE

Policz liczbę ciągów występujących w pliku z danymi oraz długość najkrótszego i najdłuższego ciągu. Otrzymane wyniki zapisz w pliku tekstowym (np. *WUZ1_zad1_1_ciagi_wynik.txt*). Na przykład dla pierwszych 14 liczb z pliku z danymi wyniki są następujące:

Liczba ciagow: 2, dlugosc najkrotszego ciagu: 5, dlugosc najdluzszego ciagu: 7

Zadanie 1.2 (0–3) ROZWIĄŻ NA KOMPETERZE

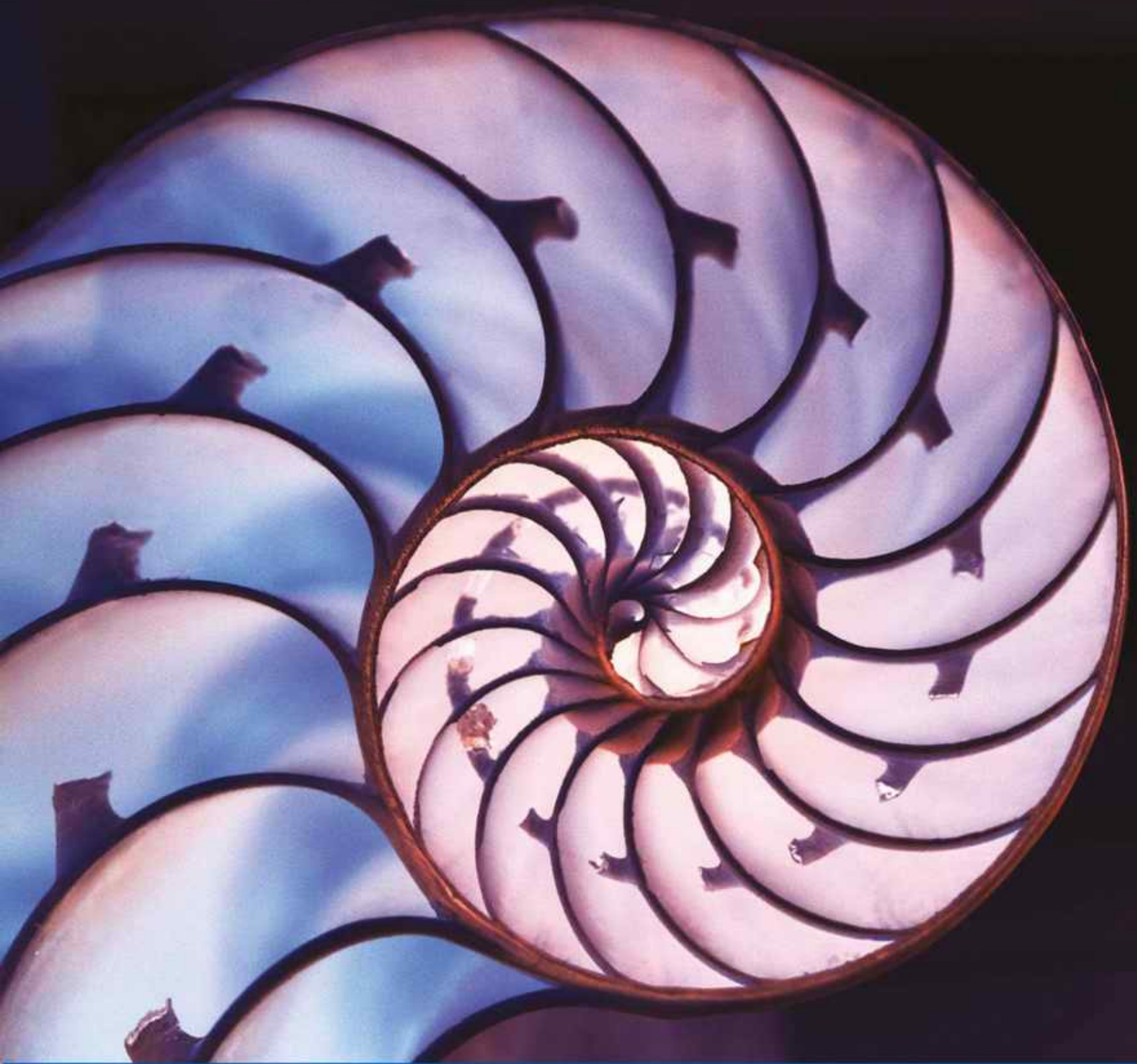
Utwórz plik wynikowy (np. *WUZ1_zad1_2_ciagi_wynik.txt*), w którym elementy każdego ciągu będą zapisane w oddzielnym wierszu (oddzielone spacjami) i posortowane niemalejąco. Na przykład dla pierwszych 14 liczb z pliku z danymi wynikiem są ciągi:

```
2340 3662 6457 21832 23534 26892 29660
3153 5304 10064 14659 19277
```

Zadanie 1.3 (0–4) ROZWIĄŻ NA KOMPETERZE

Utwórz plik wynikowy (np. *WUZ1_zad1_3_ciagi_wynik.txt*), w którym kolejne elementy każdego ciągu będą zapisane w oddzielnym wierszu, a liczby będą oddzielone spacjami. Ciągi w pliku wynikowym mają być uporządkowane od najkrótszego do najdłuższego. Na przykład dla pierwszych 14 liczb z pliku z danymi wynikiem są ciągi:

```
3153 5304 14659 10064 19277
29660 3662 6457 2340 26892 21832 23534
```



2 Algorytmy numeryczne

- 5. Reprezentacja liczb rzeczywistych w komputerze
- 6. Błędy w obliczeniach
- 7. Obliczanie wartości wielomianu
- 8. Metody obliczeń przybliżonych
- 9. Algorytmy badające własności geometryczne
- 10. Fraktale

Wiesz, umiesz, zdasz

Algorytmika i programowanie

```
10 LET A=112
20 LET B=88
30 LET R=A-B*INT (A/B)
40 IF R=0 THEN GOTO 80
50 LET M=N
60 LET N=R
70 GOTO 30
80 PRINT N_
```

W 1964 r. Amerykanie Thomas E. Kurtz i John Kemeny z Dartmouth College opracowali język programowania **BASIC** (od ang. *Beginner's All-purpose Symbolic Instruction Code*). Stał się on popularny w latach 80. XX w. wraz z rozwojem mikrokomputerów, które miały wbudowany w pamięć stałą translator języka. Oryginalne rozwiązanie zastosowano w mikrokomputerach ZX Spectrum – do każdego klawisza na klawiaturze były przypisane komendy języka, co miało przyspieszyć pisanie programów. Na składni jednego z dialektów języka BASIC opiera się używany współcześnie język Visual Basic.

W 1967 r. Seymour Papert, pochodzący z RPA, opracował na podstawie języka LISP język **Logo**. Był to pierwszy język edukacyjny do nauki programowania przeznaczony dla dzieci i młodzieży. Mimo że wówczas komputery były dostępne tylko dla wąskiej grupy specjalistów, już wtedy myślano o nauce programowania w szkołach. Cechą charakterystyczną języka Logo była tzw. grafika żółwia. W Polsce wykorzystywane były m.in. Logo Komeniusz i Imagine Logo (polska nazwa Logomocja), opracowane na Uniwersytecie Komeńskiego w Bratysławie przez zespół pod kierunkiem prof. Ivana Kalaša.



1964 r.

1967 r.

1967 r.

1969 r.



W 1967 r. naukowcy Ole-Johan Dahl, Bjørn Myrhaug i Kristen Nygaard stworzyli w Norweskim Ośrodku Obliczeniowym język do symulacji komputerowych **Simula 67**. Był to pierwszy język programowania, który wprowadzał zasady programowania obiektowego, pojęcia klasy i obiektu, dziedziczenie i metody wirtualne. Miał bardzo duży wpływ na rozwój współczesnych języków programowania (np. języka C++). W 2001 r. Ole-Johan Dahl i Kristen Nygaard za współtworzenie koncepcji języków obiektowych otrzymali nagrodę Turinga.

20 lipca 1969 r. osiągnięto cel **misji Apollo 11** i człowiek pierwszy raz stanął na Księżycu. Statkiem kosmicznym sterowały dwa komputery AGC (od ang. *Apollo Guidance Computer*). Amerykanka Margaret Hamilton kierowała zespołem tworzącym dla nich oprogramowanie. Wagę tego wyczynu zobrazowali pracownicy Google'a, którzy policzyli, że komputery AGC były ok. 100 tys. razy wolniejsze niż dobry laptop z 2012 r. Mimo to misja zakończyła się sukcesem.





W 1970 r. szwajcarski informatyk Niklaus Wirth stworzył język ogólnego przeznaczenia o nazwie **Pascal**. W latach 80. i 90. XX w. dzięki powstaniu bardzo sprawnych kompilatorów tego języka stał się on liderem wśród języków programowania zarówno w środowiskach akademickich, jak i w zastosowaniu praktycznym. Jednym z popularniejszych kompilatorów był Turbo Pascal. Język Pascal został wyparty przez języki C++, Python i Java. Współcześnie wykorzystywanym środowiskiem programistycznym wywodzącym się z Pascala jest Delphi.

W 1972 r. firma Hewlett-Packard wprowadziła na rynek pierwszy **kalkulator naukowy HP35**. Kalkulatory znane były już wcześniej, ale wykonywały jedynie podstawowe działania arytmetyczne, a inżynierowie do obliczeń używali np. suwaków logarytmicznych. Kalkulator HP35 nie ograniczał się do czterech podstawowych działań, lecz umożliwiał także znajdowanie wartości funkcji trygonometrycznych i logarytmicznych oraz potęgowanie i pierwiastkowanie. Kalkulator wykorzystywał odwrotną notację polską.



1970 r.

1972 r.

1972 r.

1977 r.



W 1972 r. amerykański informatyk Dennis M. Ritchie opracował **język C**. Prace nad nim rozpoczęły się w 1969 r. Ritchie chciał stworzyć język do pisania oprogramowania systemowego. W języku C powstało jądro systemu operacyjnego UNIX. Zyskał jednak ogromną popularność jako język ogólnego przeznaczenia, a jego rozbudowa doprowadziła do powstania języka C++. W 1983 r. Dennis Ritchie został uhonorowany nagrodą Turinga.

W 1977 r. trzech amerykańskich profesorów z MIT, Ron Rivest, Adi Shamir i Leonard Adleman, opublikowało **algorytm RSA**. Jest to jeden z pierwszych i obecnie najpopularniejszych asymetrycznych algorytmów kryptograficznych. Jego nazwa pochodzi od pierwszych liter nazwisk twórców. Algorytm stanowił przełom w szyfrowaniu informacji przesyłanych w sieciach komputerowych. Jest powszechnie stosowany w bankowości elektronicznej, sklepach internetowych i innych usługach sieciowych wymagających szyfrowania.



5. Reprezentacja liczb rzeczywistych w komputerze

Rozwiązaliśmy już wiele problemów i poznaliśmy różne techniki algorytmiczne. Dotychczas posługiwaliśmy się jednak tylko liczbami całkowitymi oraz znakami, które w komputerze są reprezentowane przez liczby całkowite. W tym temacie dowiesz się, jak komputery radzą sobie z liczbami rzeczywistymi.

Cele lekcji

- Znajdziesz rozwinięcie binarne ułamka właściwego.
- Poznasz reprezentację stało- i zmiennoprzecinkową liczb rzeczywistych.
- Dowiesz się, czym jest normalizacja reprezentacji zmiennoprzecinkowej.
- Wykonasz działania arytmetyczne na liczbach zmiennoprzecinkowych.

5.1. Rozwinięcie binarne ułamka właściwego

W części drugiej podręcznika, przy okazji omawiania **systemu dwójkowego**, znajdowaliśmy reprezentację binarną liczby całkowitej dodatniej zapisanej w systemie dziesiętnym. Teraz wyznaczmy rozwinięcie binarne ułamka zapisanego w systemie dziesiętnym.

Założmy, że mamy dodatni ułamek właściwy. Na początku mnożymy wyjściowy ułamek przez 2. Część całkowita wyniku (może nią być tylko 1 lub 0) odpowiada pierwszej cyfrze szukanego rozwinięcia binarnego. Część ułamkową wyniku wykorzystamy do wyznaczenia kolejnej cyfry rozwinięcia binarnego – postępujemy z nią w ten sam sposób co z wyjściowym ułamkiem. Aby znaleźć następne cyfry, czynności powtarzamy. Działanie algorytmu na przykładzie ułamka $\frac{5}{8}$ przedstawia tabela 5.1. Dla tego ułamka algorytm kończy działanie, gdy licznik części ułamkowej jest równy 0.

Wykonywane działanie	Część całkowita wyniku	Część ułamkowa wyniku	Zapis binarny
–	0	$\frac{5}{8}$	0
$\frac{5}{8} \cdot 2$	1	$\frac{2}{8}$	0,1
$\frac{2}{8} \cdot 2$	0	$\frac{4}{8}$	0,10
$\frac{4}{8} \cdot 2$	1	$\frac{0}{8}$	0,101

Tabela 5.1. Kolejne etapy wyznaczania rozwinięcia binarnego ułamka $\frac{5}{8}$

System dwójkowy,
podręcznik
Informatyka na czasie 2.
Zakres rozszerzony,
s. 28–29

Dobra rada

Pamiętaj, że w algorytmie zamiany liczby całkowitej dodatniej z postaci dziesiętnej na binarną wykorzystuje się dzielenie przez 2. Reszta z dzielenia danej liczby przez 2 jest jedną z szukanych cyfr binarnych, a część całkowita stanowi podstawę wyznaczania w ten sam sposób kolejnej cyfry binarnej. Algorytm wyznacza cyfry binarne od ostatniej do pierwszej, a kończy działanie, gdy część całkowita jest równa zero.

Jednak nie każdy ułamek właściwy, który ma skończone rozwinięcie dziesiętne, ma też skończone rozwinięcie binarne. Przykładem jest ułamek $\frac{1}{10}$. Tabela 5.2 ilustruje wyznaczanie jego rozwinięcia binarnego.

Wykonywane działanie	Część całkowita wyniku	Część ułamkowa wyniku	Zapis binarny
–	0	$\frac{1}{10}$	0
$\frac{1}{10} \cdot 2$	0	$\frac{2}{10}$	0,0
$\frac{2}{10} \cdot 2$	0	$\frac{4}{10}$	0,00
$\frac{4}{10} \cdot 2$	0	$\frac{8}{10}$	0,000
$\frac{8}{10} \cdot 2$	1	$\frac{6}{10}$	0,0001
$\frac{6}{10} \cdot 2$	1	$\frac{2}{10}$	0,00011
$\frac{2}{10} \cdot 2$	0	$\frac{4}{10}$	0,000110
$\frac{4}{10} \cdot 2$	0	$\frac{8}{10}$	0,0001100
$\frac{8}{10} \cdot 2$	1	$\frac{6}{10}$	0,00011001
$\frac{6}{10} \cdot 2$	1	$\frac{2}{10}$	0,000110011
$\frac{2}{10} \cdot 2$	0	$\frac{4}{10}$	0,0001100110
...	...	...	0,0(0011)

Tabela 5.2. Poszukiwanie rozwinięcia binarnego ułamka $\frac{1}{10}$

Jak widać, licznik części ułamkowej nigdy nie będzie równy 0. Otrzymywane części całkowite i ułamkowe od pewnego miejsca się powtarzają – zaznaczone są w tabeli 5.2 różnymi odcieniami zieleni (po 4 kolejne wiersze). Wynikiem jest zatem ułamek okresowy.

Ćwiczenie 1

Znajdź rozwinięcie binarne ułamków $\frac{13}{16}$ i $\frac{2}{5}$.

Zastanówmy się, dla jakich ułamków otrzymamy rozwinięcie binarne skończone, a dla jakich – nieskończone okresowe. Weźmy liczbę binarną o skończonym rozwinięciu, np. $0,1011_2$, i znajdziemy jej reprezentację w systemie dziesiętnym. Aby to zrobić, postępujemy podobnie jak w przypadku liczb całkowitych – mnożymy cyfry binarne części ułamkowej liczby przez odpowiednie potęgi podstawy systemu, czyli potęgi liczby 2. W przypadku zamiany części ułamkowej wykładniki potęg będą ujemne:

$$0,1011_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = \frac{1}{2} + \frac{0}{4} + \frac{1}{8} + \frac{1}{16} = \frac{11}{16}$$

 **Dobra rada**

Pamiętaj, że ułamek właściwy (jako liczba wymierna) w systemie dziesiętnym ma rozwinięcie dziesiętne skończone albo nieskończone okresowe.

 **Warto wiedzieć**

W systemie dziesiętnym skończone rozwinięcie dziesiętne mają te ułamki, w których w postaci nieskracalnej mianownik jest iloczynem potęgi liczby 2 i potęgi liczby 5.

Na podstawie tego przykładu można zauważyć, że skończone rozwinięcie binarne mają ułamki właściwe nieskracalne, których mianownikiem jest potęga liczby 2. Pozostałe ułamki zwykłe, także te o skończonym rozwinięciu dziesiętnym, mają okresowe rozwinięcie binarne.

Ćwiczenie 2

Znajdź reprezentację liczby $0,00011_2$ w systemie dziesiętnym.

Dobra rada

Jeśli chcesz znaleźć reprezentację binarną liczby, która w systemie dziesiętnym składa się z części całkowitej różnej od zera oraz części ułamkowej (np. liczby 1,25), musisz zamieniać oddzielnie część całkowitą i część ułamkową. Do zamiany części całkowitej wykorzystaj sposób omówiony w podręczniku *Informatyka na czasie 2. Zakres rozszerzony* (s. 28–29, 33–35).

Omówimy algorytmy wyznaczające rozwinięcia binarne ułamków właściwych nieskracalnych zapisanych w systemie dziesiętnym. Najpierw rozważymy przypadek, gdy mianownik ułamka jest potęgą liczby 2, a więc rozwinięcie binarne ułamka będzie skończone. Oto specyfikacja problemu oraz zapis algorytmu w pseudokodzie:

Specyfikacja

Dane: *licz*, *mian* – liczby całkowite dodatnie reprezentujące licznik i mianownik ułamka, $\text{licz} < \text{mian}$, $\text{mian} = 2^n$, gdzie n – liczba całkowita dodatnia, $\text{NWD}(\text{licz}, \text{mian}) = 1$.

Wynik: *s* – napis reprezentujący rozwinięcie binarne ułamka.

```
s ← "0,"
dopóki licz > 0 wykonuj
    licz ← licz * 2
    jeśli licz div mian = 1 to s ← s + "1"
    w przeciwnym przypadku s ← s + "0"
    licz ← licz mod mian
```

Wartością początkową wyniku *s* jest napis „0,”. Mnożymy licznik przez 2. Jeśli część całkowita z dzielenia otrzymanego wyniku przez mianownik jest równa 1, to na końcu wyniku dopisujemy cyfrę 1. W przeciwnym przypadku dopisujemy cyfrę 0. Nowa wartość licznika to reszta z dzielenia licznika przez mianownik. Działania powtarzamy, dopóki licznik jest większy od 0.

Ćwiczenie 3

Napisz program znajdujący rozwinięcie binarne nieskracalnego ułamka właściwego, którego mianownik jest potęgą liczby 2.

Trudniejszym problemem jest wyznaczenie okresu rozwinięcia binarnego ułamków nieskracalnych, których mianownik nie jest potęgą liczby 2. Należy wówczas pamiętać znajdowane wartości części całkowitej i licznika części ułamkowej (mianownika nie musimy pamiętać, ponieważ jest cały czas taki sam – ułamka nie skracamy).

Ponieważ nie wiemy, jak długi będzie okres, powinniśmy użyć dynamicznej struktury danych, np. **zmiennej typu `vector`**. Po każdym mnożeniu przez 2 należy sprawdzić, czy część całkowita i licznik już wystąpiły. Jeśli tak, kończymy obliczenia. Okres rozpoczyna się od miejsca, w którym aktualnie policzone wartości części całkowitej i licznika wystąpiły po raz pierwszy. Jeśli obliczone części całkowita i licznik jeszcze nie wystąpiły, to ich wartości dopisujemy na końcu struktury danych przechowującej te wartości.

Oto specyfikacja problemu znajdowania rozwinięcia binarnego ułamka właściwego nieskracalnego, którego mianownik nie jest potęgą liczby 2:

Deklaracja zmiennej typu `vector`,
s. 66 

 **Dobra rada**

Zamiast typu `vector` możesz użyć typu `list`.

Specyfikacja

Dane: `licz`, `mian` – liczby całkowite dodatnie reprezentujące licznik i mianownik ułamka, $\text{licz} < \text{mian}$, $\text{mian} \neq 2^n$, gdzie n – liczba całkowita dodatnia, $\text{NWD}(\text{licz}, \text{mian}) = 1$.

Wynik: `s` – napis reprezentujący rozwinięcie binarne ułamka.

Przyjmijmy, że elementami zmiennej `Tab` są pary liczb całkowitych reprezentujących odpowiednio część całkowitą i licznik części ułamkowej. Niech `m` oznacza aktualną liczbę elementów zmiennej `Tab`. W zmiennej `calk` będzie pamiętana część całkowita. Oto zapis algorytmu w pseudokodzie:

```
s ← "0,"
jest_okres ← fałsz
dopóki nie jest_okres wykonuj
    licz ← licz * 2
    calk ← licz div mian
    licz ← licz mod mian
    i ← 0
    dopóki i < m oraz nie jest_okres wykonuj
        jeśli Tab[i].calk=calk oraz Tab[i].licz=licz to
            jest_okres ← prawda
        w przeciwnym wypadku
            i ← i + 1
    jeśli nie jest_okres to
        dodaj nowy element na końcu Tab
        Tab[m-1].calk ← calk
        Tab[m-1].licz ← licz
dla j ← 0, 1, ..., m - 1 wykonuj
    jeśli j = i to s ← s + "("
    jeśli Tab[j].calk = 1 to s ← s + "1"
    w przeciwnym przypadku s ← s + "0"
s ← s + ")"
```

Znalezienie okresu sygnalizuje zmienna `jest_okres`. Gdy okres zostanie znaleziony (za jego poszukiwanie odpowiada wewnętrzna pętla **dopóki**), wówczas zmienna `i` zapamięta miejsce rozpoczęcia okresu.

Budując napis reprezentujący rozwinięcie binarne ułamka, przeglądamy wszystkie elementy zmiennej `Tab`. Bezpośrednio przed elementem o indeksie `i` dopisujemy nawias otwierający, który wskazuje rozpoczęcie okresu. Na końcu dopisujemy nawias zamykający.

W programie realizującym omówiony algorytm części całkowite i liczniki będziemy pamiętać w zmiennej typu `vector`. Każdy element tej zmiennej będzie przechowywał parę liczb: część całkowitą i licznik wyliczone w jednym kroku algorytmu. Do przechowania takiej pary wykorzystamy strukturę o nazwie `wymierna`. Oto jej definicja:

```
struct wymierna
{
    int calk;
    int licz;
};
```

Kod źródłowy funkcji `main` może być następujący:

Fragment kodu
źródłowego programu
znajdującego
rozwinięcie binarne
ułamka właściwego
nieskracalnego, którego
mianownik nie jest
potęgą liczby 2 –
funkcja `main`

```
1. int main()
2. {
3.     int licz, mian, i, j;
4.     cout<<"Licznik: "; cin>>licz;
5.     cout<<"Mianownik: "; cin>>mian;
6.     string s="0,";
7.     bool jest_okres=false;
8.     wymierna w;
9.     vector<wymierna> Tab;
10.    while (!jest_okres)
11.    {
12.        licz=licz*2;
13.        w.calk=licz/mian;
14.        licz=licz%mian;
15.        w.licz=licz;
16.        i=0;
17.        while (i<Tab.size() && !jest_okres)
18.            if (Tab[i].calk==w.calk && Tab[i].licz==w.licz)
19.                jest_okres=true;
20.            else i++;
21.        if (!jest_okres) Tab.push_back(w);
22.    }
23.    for (j=0;j<Tab.size();j++)
24.    {
25.        if (j==i) s=s+'(';
26.        if (Tab[j].calk==1) s=s+'1';
27.        else s=s+'0';
28.    }
29.    s=s+')';
30.    cout<<"Rozwinięcie binarne: "<<s;
31.    return 0;
32. }
```

W linii 9 zadeklarowana jest zmienna `Tab` typu **vector**, składająca się z elementów typu wymierna. Pętla w liniach 10–22 poszukuje okresu rozwinięcia binarnego. W zmiennej w typu wymierna pamiętane są policzone ostatnio część całkowita i licznik części ułamkowej (linie 13–15). Pętla w liniach 17–20 sprawdza, czy ta para wartości już wystąpiła. Jeśli tak, to w zmiennej `i` zapamiętywany jest indeks elementu wektora, w którym zapisane jest pierwsze wystąpienie tej pary. Jeśli para jeszcze nie wystąpiła, to wartość zmiennej `w` dodawana jest jako nowy element na końcu wektora (linia 21). Pętla w liniach 23–28 przegląda wektor i zapisuje w zmiennej `s` rozwinięcie binarne ułamka. Dla elementu o indeksie `i` (początek okresu) dodaje nawias otwierający (linia 25). W linii 29 dodawany jest nawias zamykający (kończący okres).

Rysunek 5.1 przedstawia efekt wykonania programu dla ułamka $\frac{1}{10}$.

```
Licznik: 1
Mianownik: 10
Rozwinięcie binarne: 0,0(0011)
```

Rys. 5.1. Efekt wykonania programu dla ułamka $\frac{1}{10}$

Ćwiczenie 4

Napisz program znajdujący rozwinięcie binarne ułamka właściwego nieskracalnego, którego mianownik nie jest potęgą liczby 2.

Zapamiętaj

Skończone rozwinięcie binarne mają tylko ułamki właściwe nieskracalne, których mianownikiem jest potęga liczby 2. Pozostałe ułamki zwykłe, także te o skończonym rozwinięciu dziesiętnym, mają okresowe rozwinięcie binarne.

5.2. Reprezentacja stało- i zmiennoprzecinkowa liczb rzeczywistych

Liczb rzeczywiste, podobnie jak liczby całkowite, są pamiętane w komputerze w postaci binarnej. Na ich reprezentację przeznacza się skończoną liczbę bajtów. Ponieważ wiele liczb wymiernych oraz liczby niewymierne mają nieskończone rozwinięcie binarne części ułamkowej, są pamiętane przez komputer w przybliżeniu. Konsekwencje tego można zaobserwować po uruchomieniu programu ze s. 104, który dodaje sto tysięcy liczb 0,1 (liczba 0,1 ma nieskończone rozwinięcie binarne). Wykorzystany w programie typ **float** poznasz w dalszej części tematu. Rysunek 5.2 na s. 104 przedstawia efekt wykonania programu.



Dobra rada

Dynamiczne struktury danych stosuj wtedy, gdy nie znasz liczby elementów, które program ma pamiętać.

Kod źródłowy programu  dodającego 100 000 liczb 0,1 oraz wskazującego różnicę między otrzymanym a poprawnym wynikiem

```
1. #include <iostream>
2. using namespace std;
3.
4. const int N=100000;
5.
6. int main()
7. {
8.     float liczba=0.1, suma=0;
9.     for (int i=0;i<N;i++) suma+=liczba;
10.    cout<<"Wynik dodawania: "<<suma<<endl;
11.    cout<<"Poprawny wynik: "<<N*liczba<<endl;
12.    cout<<"Bład: "<<N*liczba-suma<<endl;
13.    return 0;
14. }
```

```
Wynik dodawania: 9998.56
Poprawny wynik: 10000
Bład: 1.44351
```

Rys. 5.2. Efekt wykonania programu dodającego 100 000 liczb 0,1 oraz wskazującego różnicę między otrzymanym a poprawnym wynikiem

Reprezentacja stałoprzecinkowa (stałopozycyjna)

Reprezentacja  stałoprzecinkowa (stałopozycyjna) liczb

W **reprezentacji stałoprzecinkowej (stałopozycyjnej)** liczb przeznacza się określoną liczbę bitów na część całkowitą liczby i na jej część ułamkową. Części te traktuje się niezależnie. Położenie przecinka oddzielającego część całkowitą od ułamkowej jest stałe.

Tabela 5.3 przedstawia zaokrąglenia wartości $0,0(0011)_2$, czyli ułamka $\frac{1}{10}$ w systemie dziesiętnym, w zależności od liczby bitów przeznaczonych na reprezentację części ułamkowej.

Warto wiedzieć

Ponieważ część całkowita zawiera informację o znaku liczby, zapisuje się ją w kodzie U2.

Liczba bitów przeznaczonych na część ułamkową	Zaokrąglona wartość binarna	Reprezentacja w systemie dziesiętnym zaokrąglonej wartości binarnej
1	0,0	0
2	0,00	0
3	0,001	0,125
4	0,0010	0,125
5	0,00011	0,09375
6	0,000110	0,09375
7	0,0001101	0,1015625
8	0,00011010	0,1015625

Tabela 5.3. Zaokrąglenia liczby $0,0(0011)_2$ w zależności od liczby bitów przeznaczonych na część ułamkową oraz wartości dziesiętne zaokrągleń

Zaokrąglenie części ułamkowej odbywa się w następujący sposób: jeśli pierwszą niemieszczącą się cyfrą jest zero, to odrzucane są wszystkie niemieszczące się cyfry bez zmiany wartości zaokrąglenia. W przeciwnym przypadku do ostatniej pozycji zaokrąglenia dodaje się 1.

Ćwiczenie 5

Znajdź reprezentację w systemie dziesiętnym zaokrąglonej wartości binarnej:

- liczby $\frac{1}{10}$, jeśli na część ułamkową przeznaczono 9 bitów,
- liczby $\frac{1}{5}$, jeśli na część ułamkową przeznaczono 8 bitów,
- liczby $\frac{2}{7}$, jeśli na część ułamkową przeznaczono 7 bitów.

Przypuśćmy, że na część całkowitą i ułamkową przeznaczono po 1 bajcie (8 bitów) pamięci. Wówczas liczba 100,1 zapisana w systemie dziesiętnym będzie miała reprezentację stałoprzecinkową taką jak na rysunku 5.3.



Rys. 5.3. Reprezentacja stałoprzecinkowa liczby 100,1 w systemie dwójkowym, jeśli na część całkowitą i ułamkową przeznaczono po 1 bajcie

Reprezentacja stałoprzecinkowa liczb jest mało praktyczna, zwłaszcza w programach, w których będą stosowane liczby z szerokiego zakresu (od bardzo dużych po bardzo małe). Na przykład w astronomii używa się często bardzo dużych liczb, a dokładność części ułamkowej nie ma większego znaczenia. Z kolei w mikrobiologii czy fizyce kwantowej ważna jest duża precyzja części ułamkowej. Użycie reprezentacji stałoprzecinkowej do zapisu liczb z tak dużego zakresu wymaga wielu bajtów pamięci, które często nie byłyby wykorzystane. Dlatego przedstawimy bardziej praktyczny sposób reprezentacji liczb rzeczywistych w komputerze.

Warto wiedzieć

W arytmetyce liczb stałoprzecinkowych obowiązują prawa łączności i przemienności dodawania.

Zapamiętaj

W reprezentacji stałoprzecinkowej (stałopozycyjnej) liczb oznacza się określoną liczbę bitów na część całkowitą i część ułamkową liczby. Część całkowitą oraz część ułamkową traktuje się niezależnie. Reprezentacja stałoprzecinkowa jest mało praktyczna w przypadku zapisywania liczb z szerokiego zakresu.

Reprezentacja zmiennoprzecinkowa (zmiennopozycyjna)

Reprezentacja zmiennoprzecinkowa (zmiennopozycyjna) liczb W reprezentacji zmiennoprzecinkowej (zmiennopozycyjnej) liczb przecinek oddzielający część całkowitą od ułamkowej nie ma stałej pozycji. Liczbę x w reprezentacji zmiennoprzecinkowej przedstawia wzór:

Mantysa $x = m \cdot p^c$, gdzie m – **mantysa**,

p – podstawa systemu, w którym zapisana jest liczba,
 c – **cecha**, wyrażona liczbą całkowitą.

Cecha

Stosując powyższy wzór, daną liczbę można przedstawić w różny sposób. Na przykład w systemie dziesiętnym liczbę 123,456 można zapisać m.in. jako $0,123456 \cdot 10^3$ lub $1,23456 \cdot 10^2$. Aby liczba miała jednoznaczny reprezentację zmiennoprzecinkową, na mantysę narzuca się pewne warunki. Powinna być ona liczbą z przedziału $[1; p)$, gdzie p jest podstawą systemu liczbowego. Mówimy, że liczba zapisana z taką

Postać znormalizowana mantysą jest w **postaci znormalizowanej**. Liczba 123,456 zapisana w tej postaci wygląda następująco: $1,23456 \cdot 10^2$. Mantysa jest równa 1,23456, a cecha 2. Zwróć uwagę, że w postaci znormalizowanej mantysa jest zawsze dodatnia, więc oddzielnie trzeba pamiętać znak liczby. Nie da się zapisać w postaci znormalizowanej liczby 0.

Warto wiedzieć

Przyjmuje się, że liczba 0 jest reprezentowana przez mantysę równą 0.

Warto wiedzieć

Reprezentacja zmiennoprzecinkowa nazywana jest też notacją wykładniczą (naukową). Wykorzystuje się ją często w różnych dziedzinach. Na przykład stałą grawitacji zapisuje się w postaci: $G = 6,67259 \cdot 10^{-11}$.

W opisanym przykładzie widać, że przesunięcie przecinka w mantysie wpływa na wartość cechy. Na przykład przesunięcie przecinka w prawo o jedną pozycję skutkuje zmniejszeniem cechy o 1, z kolei przesunięcie przecinka w lewo o jedną pozycję – zwiększeniem cechy o 1.

Zajmiemy się teraz reprezentacją zmiennoprzecinkową liczb binarnych w postaci znormalizowanej. W przypadku tych liczb nie trzeba pamiętać części całkowitej mantysy, ponieważ jest ona zawsze równa 1. Liczbę binarną można reprezentować jako znak liczby oraz parę liczb: mantysę i cechę. Liczbę można więc wyrazić wzorem:

$x = (-1)^{\text{znak}} \cdot \text{mantysa} \cdot 2^{\text{cecha}}$, gdzie **znak** – liczba 0 dla liczby dodatniej i 1 dla liczby ujemnej.

A to ciekawe

Jak zaokrąglenie liczby może zmienić skład parlamentu?

Konsekwencje zaokrąglania liczb mogą mieć duży wpływ na życie obywateli. 5 kwietnia 1992 r. podczas wyborów do parlamentu w Szlezwiku-Holsztynie, jednym z krajów związkowych Niemiec, o mały włos nie doszło do poważnej pomyłki. Aby wejść do parlamentu, partia musiała zdobyć minimum 5% głosów. Po podliczeniu głosów wydawało się, że jedno ze startujących ugrupowań uzyskało dokładnie tyle. Okazało się jednak, że naprawdę było to 4,97% głosów, a program komputerowy zaokrąglił wynik do jednego miejsca po przecinku, czyli 5,0%. Na szczęście błąd wykryto na czas i odpowiednio przeliczono mandaty do parlamentu.

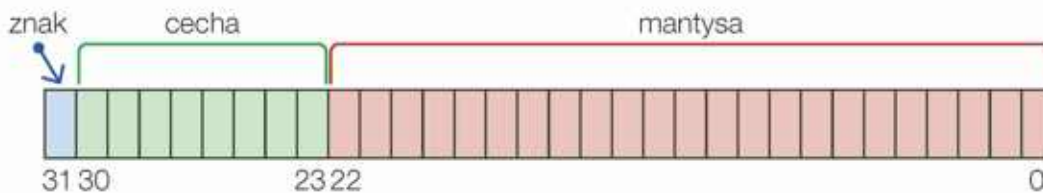


W języku C++ do pamiętania liczb zmiennoprzecinkowych najczęściej używa się dwóch typów zmiennych:

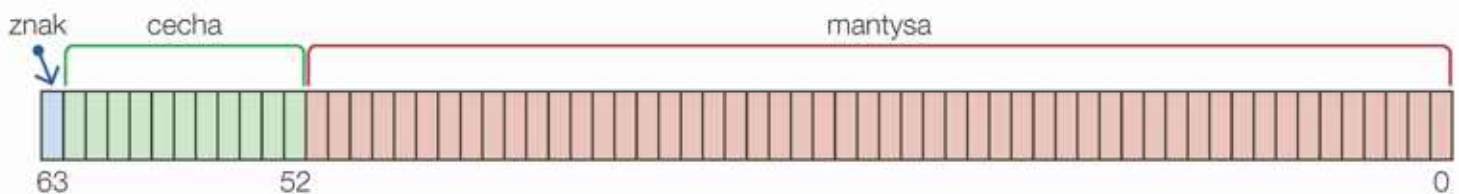
- ▶ typu **float**, wykorzystującego 4 bajty (32 bity) pamięci z przeznaczeniem: 1 bitu na znak, 8 bitów na cechę, 23 bitów na mantysę,
- ▶ typu **double**, wykorzystującego 8 bajtów (64 bity) pamięci z przeznaczeniem: 1 bitu na znak, 11 bitów na cechę, 52 bitów na mantysę.

Liczyby zmiennoprzecinkowe zapisane na 32 bitach nazywane są **liczbami pojedynczej precyzji**, natomiast zapisane na 64 bitach – **liczbami podwójnej precyzji**. Rysunki 5.4 i 5.5 przedstawiają schematy reprezentacji liczb zmiennoprzecinkowych pojedynczej i podwójnej precyzji.

● **Liczba pojedynczej precyzji, liczba podwójnej precyzji**



Rys. 5.4. Reprezentacja liczby zmiennoprzecinkowej pojedynczej precyzji



Rys. 5.5. Reprezentacja liczby zmiennoprzecinkowej podwójnej precyzji

O precyzji, czyli dokładności (liczbie cyfr znaczących), decyduje liczba bitów przeznaczonych na mantysę. O zakresie decyduje liczba bitów przeznaczonych na cechę. Tabela 5.4 przedstawia zakresy oraz precyzję liczb typu **float** oraz **double**.

Warto wiedzieć

Sposób reprezentacji liczb zmiennoprzecinkowych pojedynczej i podwójnej precyzji opisuje standard IEEE 754.

Nazwa typu	Liczba bajtów	Przeznaczenie i zakres
float	4	Liczyby rzeczywiste z zakresu od $1,17549 \cdot 10^{-38}$ do $3,40282 \cdot 10^{38}$, w których liczba cyfr znaczących (dokładność) jest nie większa niż 7
double	8	Liczyby rzeczywiste z zakresu od $2,22507 \cdot 10^{-308}$ do $1,79769 \cdot 10^{308}$, w których liczba cyfr znaczących (dokładność) jest nie większa niż 15

Tabela 5.4. Zakresy i dokładność typów zmiennoprzecinkowych

Cecha jest pamiętana przez komputer w tzw. **kodzie z nadmiarem**. Dla liczb pojedynczej precyzji nadmiar wynosi 127. Oznacza to, że aby obliczyć, ile jest równa cecha, od pamiętanej wartości cechy należy odjąć 127. Kod z nadmiarem umożliwia dowolne przesunięcie przedziału reprezentowanych wartości. W tym przypadku reprezentowany jest przedział $[-127; 128]$. Skrajne wartości są zarezerwowane, m.in. na reprezentację zera. Dla liczb podwójnej precyzji nadmiar wynosi 1023.

● **Kod z nadmiarem**

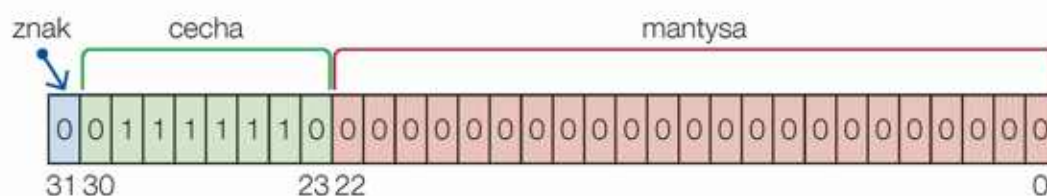
Dobra rada

Pamiętaj, że zero w znormalizowanej postaci nie ma swojej reprezentacji.

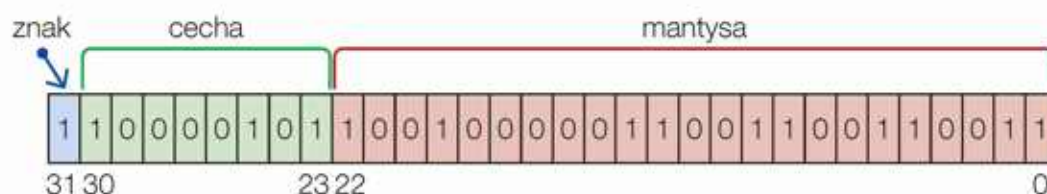
Warto wiedzieć

Pierwszym komputerem wykorzystującym binarne liczby zmiennoprzecinkowe był komputer Z1. Skonstruował go w 1938 r. niemiecki inżynier Konrad Zuse.

Rysunki 5.6 i 5.7 przedstawiają reprezentacje liczb 0,5 i –100,1 (w systemie dziesiętnym) jako liczb zmiennoprzecinkowych pojedynczej precyzji (typ **float**).



Rys. 5.6. Reprezentacja binarna liczby 0,5 jako liczby zmiennoprzecinkowej pojedynczej precyzji (typ **float**)



Rys. 5.7. Reprezentacja binarna liczby –100,1 jako liczby zmiennoprzecinkowej pojedynczej precyzji (typ **float**)

Warto wiedzieć

W reprezentacji binarnej liczby w postaci znormalizowanej pierwsza cyfra mantysy (jej część całkowita) jest zawsze równa 1, dlatego nie trzeba jej pamiętać. Dzięki temu na reprezentację mantysy można przeznaczyć jeden bit więcej.

Dla liczby 0,5 pamiętana wartość cechy wynosi 126, po odjęciu 127 otrzymujemy cechę –1. Pamiętana mantysa to 0, ale trzeba dodać do niej 1. Otrzymujemy wartość 1,0, której przecinek należy przesunąć o 1 pozycję w lewo (cecha –1). Liczba $0,1_2$ to liczba 0,5 w systemie dziesiętnym.

W przypadku liczby dziesiętnej –100,1 pamiętana cecha to 133, po odjęciu 127 otrzymujemy cechę 6. Po dodaniu 1 do mantysy otrzymujemy mantysę 1,100100000110011..., a po przesunięciu w prawo przecinka o 6 pozycji – liczbę binarną: 1100100,000110011... Część całkowita 1100100 ma wartość dziesiętną 100, a część ułamkowa to zaokrąglone rozwinięcie ułamka dziesiętnego 0,1. Bit znaku jest równy 1, więc liczba jest ujemna.

Warto wiedzieć

Komputer może być wyposażony w koprocesor arytmetyczny, który wspomaga procesor w wykonywaniu działań na liczbach zmiennoprzecinkowych. Ma to znaczenie szczególnie w programach, które wykonują dużo takich obliczeń, np. w grach komputerowych wyznaczających współrzędne obiektów w przestrzeni czy programach do tworzenia modeli 2D i 3D.

Ćwiczenie 6

Podaj reprezentację liczby zapisanej w systemie dziesiętnym jako liczba zmiennoprzecinkowa pojedynczej precyzji.

- 10,25
- 0,2

Zapamiętaj

Liczby rzeczywiste są reprezentowane w komputerze najczęściej w postaci znormalizowanej zmiennoprzecinkowej. Pamiętane są: znak liczby, jej mantysa i cecha. Liczbę binarną można wyrazić wzorem:

$$x = (-1)^{\text{znak}} \cdot \text{mantysa} \cdot 2^{\text{cecha}}$$

5.3. Działania arytmetyczne na liczbach zmiennoprzecinkowych

Wyjaśnimy najpierw, jak wykonać mnożenie i dzielenie liczb zmiennoprzecinkowych. Następnie zajmiemy się dodawaniem i odejmowaniem takich liczb.

Mnożenie i dzielenie liczb zmiennoprzecinkowych

Żeby pomnożyć dwie liczby zmiennoprzecinkowe, należy ustalić znak wyniku, pomnożyć mantysy i dodać cechy. Jeśli w otrzymanym wyniku mantysa nie jest z przedziału $[1; p)$, gdzie p – podstawa systemu liczbowego, należy przekształcić wynik tak, aby był w postaci znormalizowanej.

Postać znormalizowana,
s. 106 

Mnożenie przykładowych liczb $9,8765 \cdot 10^3$ i $-4,321 \cdot 10^2$ zapisanych w systemie dziesiętnym wykonalibyśmy następująco:

1. Ustalamy znak wyniku: znaki liczb są przeciwne, więc wynik będzie ujemny.
2. Mnożymy mantysy: $9,8765 \cdot 4,321 = 42,6763565$.
3. Dodajemy cechy: $3 + 2 = 5$.
4. Sprawdzamy, czy mantysa jest z przedziału $[1; 10)$. Ponieważ nie jest, przekształcamy otrzymany wynik ($-42,6763565 \cdot 10^5$) tak, aby mantysa należała do wskazanego przedziału. W tym celu trzeba przesunąć przecinek o jedną pozycję w lewo, co skutkuje zwiększeniem cechy o 1. Otrzymujemy zatem wynik: $-4,26763565 \cdot 10^6$.

Przy dzieleniu liczb zmiennoprzecinkowych ustala się znak wyniku, dzieli mantysy i odejmuje cechy. Jeśli otrzymany wynik nie jest w postaci znormalizowanej, należy go przekształcić do takiej postaci.

Dodawanie i odejmowanie liczb zmiennoprzecinkowych

Liczby zmiennoprzecinkowe można dodawać i odejmować tylko wtedy, gdy mają takie same cechy. Jeśli dodawane (odejmowane) liczby mają równe cechy, wystarczy dodać (odjąć) mantysy, a następnie w razie potrzeby znormalizować wynik. Jeśli cechy liczb są różne, to liczbę, która ma mniejszą cechę, przekształcamy tak, aby miała cechę taką jak druga liczba.

Na przykład aby dodać liczby $9,8765 \cdot 10^3$ oraz $4,321 \cdot 10^2$ zapisane w systemie dziesiętnym, przekształcamy drugą liczbę tak, aby miała cechę równą 3. Będzie miała ona wówczas postać $0,4321 \cdot 10^3$. Zatem: $9,8765 \cdot 10^3 + 0,4321 \cdot 10^3 = 10,3086 \cdot 10^3$. Po znormalizowaniu otrzymujemy wynik $1,03086 \cdot 10^4$.

Warto wiedzieć

W arytmetyce liczb zmiennoprzecinkowych wynik może zależeć od kolejności wykonywanych działań, np. dla liczb a, b i c : wynik $(a + b) + c$ może być różny od wyniku $a + (b + c)$.

Ćwiczenie 7

Oblicz:

- $5,123 \cdot 10^{-1} \cdot 4,987 \cdot 10^2$,
- $5,123 \cdot 10^{-1} + 4,987 \cdot 10^2$.

Podsumowanie

- Skończone rozwinięcie binarne mają tylko te ułamki właściwe, których mianownikiem w postaci nieskracalnej jest potęga liczby 2.
- Ułamki właściwe, które w postaci nieskracalnej mają mianownik niebędący potęgą liczby 2, w systemie binarnym są ułamkami okresowymi.
- W algorytmie wyznaczającym rozwinięcie binarne ułamka właściwego mnoży się ten ułamek przez 2: część całkowita wyniku jest pierwszą cyfrą rozwinięcia binarnego, a część ułamkową mnoży się przez 2, aby wyznaczyć kolejną cyfrę rozwinięcia binarnego. Czynności powtarza się do momentu uzyskania licznika części ułamkowej równego 0 lub znalezienia okresu.
- Liczby rzeczywiste można reprezentować w postaci stałoprzecinkowej (stałopozycyjnej) oraz zmiennoprzecinkowej (zmiennopozycyjnej).
- W reprezentacji stałoprzecinkowej przeznacza się stałą liczbę bitów na część całkowitą oraz na część ułamkową. Przecinek oddzielający część całkowitą od ułamkowej ma stałą pozycję.
- W reprezentacji zmiennoprzecinkowej liczbę zapisuje się w postaci $x = m \cdot p^c$, gdzie m – mantysa, p – podstawa systemu, w którym zapisana jest liczba, c – cecha, wyrażona liczbą całkowitą. Przecinek nie ma stałej pozycji w reprezentacji liczby.
- Postać zmiennoprzecinkową liczby nazywamy znormalizowaną, jeśli mantysa jest liczbą z przedziału $[1; p)$, gdzie p – podstawa systemu.
- Zapis zmiennoprzecinkowy jest bardziej uniwersalny od zapisu stałoprzecinkowego, ponieważ pozwala pamiętać liczby z wykorzystaniem mniejszej liczby bajtów.
- Liczby rzeczywiste w komputerze pamiętane są najczęściej jako liczby zmiennoprzecinkowe.
- Aby pomnożyć liczby zmiennoprzecinkowe, ustalamy znak wyniku, mnożymy mantysy i dodajemy cechy. Jeśli otrzymany wynik nie jest w postaci znormalizowanej, trzeba przekształcić go do tej postaci.
- Przy dzieleniu liczb zmiennoprzecinkowych ustala się znak wyniku, dzieli mantysy i odejmuje cechy. Jeśli otrzymany wynik nie jest w postaci znormalizowanej, należy go przekształcić do takiej postaci.
- Żeby dodać (odjąć) liczby zapisane w postaci zmiennoprzecinkowej, wystarczy dodać (odjąć) mantysy, pod warunkiem że cechy są takie same. Jeśli cechy nie są równe, to jedną z liczb przekształca się tak, aby obie cechy były takie same.

Zadania

- ★ **1** Znajdź rozwinięcie binarne ułamka $\frac{1}{3}$ zapisanego w systemie dziesiętnym.
- ★ **2** Znajdź reprezentację w systemie trójkowym ułamka $\frac{1}{3}$ zapisanego w systemie dziesiętnym.
- ★ **3** Znajdź binarną reprezentację stałoprzecinkową liczby 123,375 zapisanej w systemie dziesiętnym. Przeznacz 1 bajt na część całkowitą oraz 1 bajt na część ułamkową.

- ★★ **4** Znajdź binarną reprezentację liczby 100,1 zapisanej w systemie dziesiętnym jako liczba zmiennoprzecinkowa podwójnej precyzji (typ `double`).
- ★★ **5** Napisz program znajdujący rozwinięcie binarne ułamka właściwego wczytanego w systemie dziesiętnym, który w postaci ułamka zwykłego nieskracalnego nie ma w mianowniku potęgi liczby 2. Wykorzystaj typ `list`.
- ★★ **6** Napisz program znajdujący rozwinięcie binarne ułamka właściwego wczytanego w systemie dziesiętnym. Program powinien znajdować zarówno rozwinięcia skończone, jak i okresowe.
- ★★ **7** Napisz program znajdujący reprezentację stałoprzecinkową liczby z przedziału $[0; 255]$ zapisanej w systemie dziesiętnym w naturalnym kodzie binarnym. Program powinien wykorzystywać 1 bajt na część całkowitą i 1 bajt na część ułamkową. Wczytaj do programu niezależnie część całkowitą i część ułamkową liczby.
- ★★★ **8** Napisz program symulujący dodawanie nieujemnych liczb binarnych w reprezentacji zmiennopozycyjnej, w której na cechę przeznaczony jest 1 bajt i na mantysę także 1 bajt. Cechy zapisane są w kodzie z nadmiarem 127. Dane do programu stanowią dwa napisy złożone z 16 znaków 0 i 1, w każdym napisie co najmniej jeden bit cechy jest równy 0. Wynikiem jest także napis złożony z 16 znaków 0 i 1, reprezentujący obliczoną sumę.

6. Błędy w obliczeniach

Komputery potrafią płać figle. Zdarza się, że choć wykonują obliczenia matematycznie poprawne, zwracają błędne wyniki. Warto znać przyczyny powstawania tych błędów, aby np. móc projektować systemy odporne na wynikające z nich niebezpieczeństwa. Dlatego w tym temacie poznasz podstawowe zagadnienia dotyczące błędów numerycznych.

Cele lekcji

- Poznasz różne przyczyny i rodzaje błędów w obliczeniach komputerowych.
- Dowiesz się, czym się różni błąd bezwzględny od względnego.
- Zrozumiesz, czym się charakteryzują algorytmy niestabilny i stabilny.
- Znajdziesz pierwiastki równania kwadratowego, wykorzystując zarówno algorytm niestabilny, jak i algorytm stabilny.

6.1. Źródła błędów w obliczeniach

Wiesz już, że liczby rzeczywiste są pamiętane w komputerze na skończonej liczbie bitów. Skutkuje to zaokrągleniem reprezentacji binarnych wielu z nich, np. liczb wymiernych mających nieskończone rozwinięcie binarne i liczb niewymiernych. Takie reprezentacje binarne są więc

Błąd reprezentacji • obarczone pewnym błędem. Nazywamy go **błędem reprezentacji**.

Mnożenie i dzielenie liczb
zmiennoprzecinkowych,
s. 109 

Kiedy wykonujemy działania arytmetyczne na liczbach reprezentowanych na skończonej liczbie bitów, do zapamiętania wyniku często potrzebujemy większej liczby bitów. Na przykład podczas **mnożenia dwóch liczb zmiennoprzecinkowych** mnożymy ich mantysy. Jeśli mantysy są reprezentowane na n bitach, to na wynik mnożenia warto byłoby przeznaczyć $2n$ bitów. Jednak wynik musimy reprezentować także na n bitach, dlatego trzeba go zaokrąglić. Powstaje błąd, który

Błąd zaokrąglenia • nazywamy **błędem zaokrąglenia**.

Konsekwencje błędów reprezentacji i zaokrąglenia można zaobserwować po wykonaniu programu dodającego liczby 0,1, dopóki wynik jest różny od 1. Oto fragment kodu źródłowego tego programu:

Fragment kodu •
źródłowego programu
dodawającego liczby 0,1,
dopóki wynik
jest różny od 1

Warto wiedzieć

W przykładach używamy liczb pojedynczej precyzji (typ `float`), aby łatwiej pokazać błędy w obliczeniach.

```
1. int main()
2. {
3.     float x=0;
4.     while (x!=1)
5.     {
6.         x=x+0.1;
7.         cout<<x<<endl;
8.     }
9.     return 0;
10. }
```

Program powinien zakończyć działanie po wykonaniu dziesięciu instrukcji $x = x + 0.1$, czyli osiągnięciu przez zmienną x wartości 1. Niestety program będzie działał w nieskończoność – nigdy nie osiągnie dokładnej wartości 1, ponieważ liczba 0,1 ma nieskończone rozwinięcie binarne (równe $0,0(0011)_2$) i jest pamiętana przez komputer w przybliżeniu. Konsekwencje przybliżania liczb mających nieskończone rozwinięcie binarne obserwowaliśmy już w programie dodającym 100 000 liczb 0,1, przedstawionym w poprzednim temacie.

Zmodyfikujmy program tak, aby zakończył działanie po przekroczeniu przez zmienną x wartości 1, a po każdym wyliczeniu wartości $x + 0.1$ wypisał wynik z dziesięcioma cyframi części ułamekowej. Oto fragment kodu źródłowego zmodyfikowanego programu:

```
1. int main()
2. {
3.     float x=0;
4.     cout<<setprecision(10);
5.     while (x<1)
6.     {
7.         x=x+0.1;
8.         cout<<x<<endl;
9.     }
10.    return 0;
11. }
```

W linii 4 wykorzystaliśmy funkcję `setprecision`, pozwalającą określić, ile pozycji części ułamekowej zostanie wyświetlonych. Wyniki zwracane przez program będą zaokrąglane. Jeśli cyfry od pewnego miejsca do końca wyniku są zerami, to nie zostaną wypisane. Funkcja `setprecision` jest zdefiniowana w bibliotece `iomanip`, dlatego w programie należy użyć dyrektywy `#include <iomanip>`. Rysunek 6.1 przedstawia efekt wykonania powyższego programu.

Ponieważ wartości rzeczywiste wyliczane przez program są zaokrąglane, nie należy ich porównywać. Najczęściej w programach wprowadza się stałą, która informuje o **dokładności obliczeń**. Kiedy błąd jest mniejszy od tej stałej, kończymy obliczenia. Otrzymujemy rozwiązanie przybliżone obciążone pewnym błędem – tzw. **błędem przybliżenia**.

W programie przedstawionym powyżej moglibyśmy wprowadzić stałą, np. o nazwie EPS. Jej definicja wyglądałaby następująco.

Kod źródłowy programu dodającego 100 000 liczb 0,1 oraz wskazującego różnicę między otrzymanym a poprawnym wynikiem, s. 104 [↗](#)

Fragment kodu źródłowego programu dodającego liczby 0,1, do momentu, gdy wynik przekroczy wartość 1

```
0.1000000015
0.200000003
0.3000000119
0.400000006
0.5
0.6000000238
0.7000000477
0.8000000715
0.9000000954
1.000000119
```

Rys. 6.1. Kolejne wyniki zwracane przez program dodający liczby 0,1 do momentu, gdy wynik przekroczy wartość 1

Funkcja `setprecision`

Warto wiedzieć

Funkcja `setprecision` (podobnie jak poznana wcześniej funkcja `setw`) jest przykładem manipulatora strumienia. Manipulatory strumieni są dostępne m.in. w bibliotekach `iostream` i `iomanip`.

Dokładność obliczeń

Błąd przybliżenia

 Warto wiedzieć

Nazwa stałej EPS
pochodzi od greckiej litery
epsilon (ϵ).

```
const float EPS=0.01;
```

Wówczas warunek pętli określimy w następujący sposób:

```
while (abs(x-1)>EPS)
```

Ponieważ dodajemy za każdym razem wartość 0,1, wystarczy nam **Funkcja abs** • dokładność obliczeń równa 0,01. **Funkcja abs** z biblioteki `cmath` zwraca wartość bezwzględną liczby zmiennoprzecinkowej. W naszym przypadku parametrem funkcji jest różnica między otrzymaną wartością zmiennej `x` a oczekiwanym wynikiem 1.

Ćwiczenie 1

Napisz program, który będzie dodawał wartość 0.1 do zmiennej `x`, aż otrzyma wartość 1 z dokładnością określoną przez stałą. Wartość początkowa zmiennej `x` wynosi 0. Program powinien wypisywać wartości pośrednie dodawania z dziesięcioma cyframi po kropce dziesiętnej.

Przy obliczaniu pewnych wartości na komputerze trzeba ograniczyć liczbę wykonywanych operacji. Na przykład wartość funkcji sinus dla zmiennej rzeczywistej x można obliczyć ze wzoru:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \dots$$

Według wzoru działania wykonywane są w nieskończoność. Jednak algorytm obliczający wartość funkcji sinus jest skończony – wykonuje skończoną liczbę operacji (pomija od pewnego momentu liczby o małych wartościach, niemających dużego wpływu na wynik). Stanowi to kolejne źródło błędów w obliczeniach. Błąd tego typu to

Błąd obcięcia • **błąd obcięcia**.

Niektóre działania wykonywane przez komputer na określonych danych powodują kumulację błędów. Może tak się stać np. podczas odejmowania bliskich sobie liczb, dzielenia dużej liczby przez bardzo małą lub odejmowania bardzo małej liczby od dużej. Błędy mogą kumulować się także, gdy wykonujemy wiele działań arytmetycznych. Czasami kumulacja błędów może prowadzić do bardzo niedokładnych wyników pomimo małych błędów danych wejściowych.

 Zapamiętaj

Istnieją różne źródła błędów w obliczeniach komputerowych. Jednym z nich jest błąd reprezentacji, wynikający z zaokrąglania reprezentacji binarnej liczb zmiennoprzecinkowych. Działania arytmetyczne na takich liczbach powodują błąd zaokrąglenia wyniku. Często też stosuje się algorytmy, które zwracają wynik przybliżony. Taka wartość jest obciążona błędem przybliżenia lub obcięcia.

 Warto wiedzieć

Do obliczenia przybliżonej wartości dowolnej funkcji trygonometrycznej wystarczy cztery podstawowe działania arytmetyczne.

6.2. Błąd bezwzględny i względny

Błąd bezwzględny informuje, o ile wartość dokładna różni się od otrzymanej przybliżonej wartości. Oblicza się go ze wzoru:

$|x - x_p|$, gdzie x – wartość dokładna, a x_p – wartość przybliżona

Na przykład jeśli liczbę dziesiętną 100,5 zaokrąglimy do wartości całkowitej, to błąd bezwzględny tego przybliżenia wynosi 0,5. Taki sam błąd bezwzględny otrzymamy, gdy zaokrąglimy liczbę 0,5 do wartości całkowitej, choć w tym przypadku wartość zaokrąglona jest dwa razy większa od dokładnej.

Lepszym oszacowaniem wielkości błędu jest **błąd względny**. Określa on, o jaką część wartości dokładnej różni się wartość przybliżona. Błąd względny obliczamy ze wzoru:

$\left| \frac{x - x_p}{x} \right|$, gdzie x – wartość dokładna, x_p – wartość przybliżona

Czasami wartość błędu względnego wyraża się w procentach. Wówczas obliczamy ją ze wzoru:

$\left| \frac{x - x_p}{x} \right| \cdot 100\%$, gdzie x – wartość dokładna, x_p – wartość przybliżona

W przypadku zaokrąglenia liczby 100,5 do wartości całkowitej błąd względny wynosi niecałe 0,5%, a w przypadku zaokrąglenia liczby 0,5 do wartości całkowitej – aż 100%.

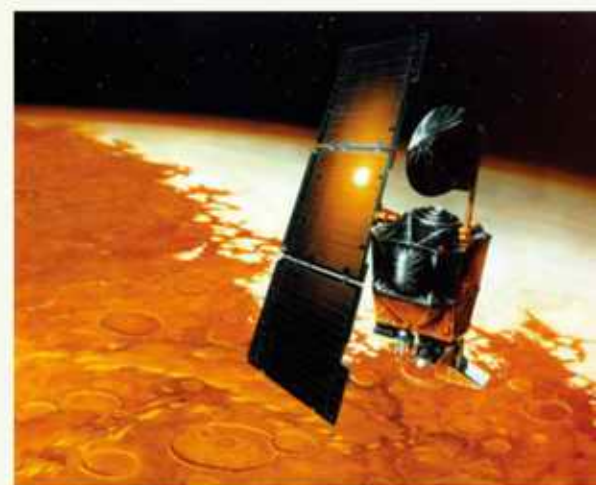
6.3. Algorytmy znajdowania pierwiastków równania kwadratowego

Napiszemy programy znajdujące rozwiązanie równania kwadratowego $ax^2 + bx + c = 0$, gdzie $a \neq 0$, czyli programy wyznaczające pierwiastki równania. Rozpatrzmy dwa algorytmy, które dla tych samych danych dadzą różne wyniki, wynikające z błędów reprezentacji danych.

A to ciekawe

Niespodziewany finał podróży

11 grudnia 1998 r. w kierunku Marsa wystartowała sonda NASA – Mars Climate Orbiter – której zadaniem było badanie Czerwonej Planety. 23 września 1999 r., po blisko 300 dniach podróży, sonda dotarła do Marsa. Jednak w czasie wchodzenia na orbitę utracono z nią kontakt. Okazało się, że Mars Climate Orbiter znalazł się ok. 100 km za blisko powierzchni planety i prawdopodobnie uległ zniszczeniu w jej atmosferze. Dochodzenie wykazało, że główną przyczyną niepowodzenia był błąd ludzki. Polegał on na używaniu do sterowania sondą danych wyrażonych w jednostkach z różnych systemów miar: anglosaskiego i metrycznego.



Znajdowanie pierwiastków równania kwadratowego – algorytm 1

Tradycyjny algorytm, nazywany czasem algorytmem z deltą, znasz z lekcji matematyki. Zgodnie z nim najpierw należy policzyć wyróżnik równania kwadratowego, oznaczany literą Δ , a następnie wyliczyć

Pierwiastki równania kwadratowego

to rozwiązania równania kwadratowego $ax^2 + bx + c = 0$, gdzie $a \neq 0$. Liczba pierwiastków zależy od wartości wyróżnika (którą obliczamy ze wzoru: $\Delta = b^2 - 4ac$)

- ▶ jeśli $\Delta > 0$, to równanie ma dwa pierwiastki:

$$x_1 = \frac{-b - \sqrt{\Delta}}{2a}, x_2 = \frac{-b + \sqrt{\Delta}}{2a}$$
- ▶ jeśli $\Delta = 0$, to równanie ma jeden pierwiastek:

$$x_1 = x_2 = \frac{-b}{2a}$$
- ▶ jeśli $\Delta < 0$, to równanie nie ma pierwiastków.

Sformułujmy specyfikację problemu znajdowania pierwiastków równania kwadratowego oraz zapiszmy w pseudokodzie algorytm tradycyjny z deltą, wyznaczający pierwiastki równania.

Specyfikacja

Dane: a, b, c – liczby rzeczywiste będące współczynnikami trójmianu kwadratowego $ax^2 + bx + c$, gdzie $a \neq 0$.

Wynik: x_1, x_2 – liczby rzeczywiste będące pierwiastkami równania kwadratowego $ax^2 + bx + c = 0$, gdzie $a \neq 0$, lub komunikat „Brak pierwiastków”.

```
delta ← b * b - 4 * a * c
jeśli delta < 0 to wypisz "Brak pierwiastków"
w przeciwnym przypadku
    pdelta ← pierwiastek(delta)
    x1 ← (-b - pdelta)/(2 * a)
    x2 ← (-b + pdelta)/(2 * a)
    wypisz x1, x2
```

Fragment kodu źródłowego programu realizującego powyższy algorytm wygląda następująco:

Fragment kodu źródłowego programu wyznaczającego pierwiastki równania kwadratowego – algorytm 1

```
1. int main()
2. {
3.     float a, b, c, delta, pdelta, x1, x2;
4.     cout<<"a = "; cin>>a;
5.     cout<<"b = "; cin>>b;
6.     cout<<"c = "; cin>>c;
7.     delta=b*b-4*a*c;
8.     if (delta<0) cout<<"Brak pierwiastkow";
9.     else
10.    {
11.        pdelta=sqrt(delta);
12.        x1=(-b-pdelta)/(2*a);
13.        x2=(-b+pdelta)/(2*a);
14.        cout<<"x1 = "<<x1<<endl;
15.        cout<<"x2 = "<<x2;
16.    }
17.    return 0;
18. }
```

W linii 11 korzystamy z **funkcji** `sqrt`, obliczającej pierwiastek kwadratowy liczby. Funkcja ta znajduje się w bibliotece `cmath`, więc należy pamiętać o dodaniu w programie linii dołączającej tę bibliotekę – `#include <cmath>`.

Rysunek 6.2 przedstawia wynik działania programu dla równania $x^2 + 10\,000x + 1 = 0$.

```
a = 1
b = 10000
c = 1
x1 = -10000
x2 = 0
```

Rys. 6.2. Wynik działania programu dla równania $x^2 + 10\,000x + 1 = 0$

Sprawdźmy, czy wybrany pierwiastek równania, np. `x2`, jest rzeczywiście równy wyliczonej przez program wartości – liczbie `0`. Po podstawieniu wartości pierwiastka do równania otrzymamy sprzeczność: jeden równa się zero. Zatem liczba `0` nie jest poszukiwanym pierwiastkiem.

Ćwiczenie 2

Napisz program znajdujący pierwiastki równania kwadratowego z wykorzystaniem przedstawionego w temacie algorytmu z deltą. Sprawdź działanie programu dla różnych danych.

W przykładzie na rysunku 6.2 danymi są liczby całkowite, ale w programie są one reprezentowane jako liczby zmiennoprzecinkowe (typ `float`). Program wykonał tylko kilka działań arytmetycznych, a otrzymał wynik z dużym **błędem względnym**. Jego przyczyną jest odejmowanie liczb. W naszym przypadku wartość iloczynu `4*a*c` jest bardzo mała względem wartości `b*b`, więc wartość delta jest w przybliżeniu równa wartości `b*b`. Powoduje to zaokrąglenie wartości pierwiastka `x2` do `0`.

Błąd względny,
s. 115 [🔗](#)

Przedstawiony algorytm jest przykładem **algorytmu niestabilnego**. **Algorytm niestabilny** Charakteryzuje się on tym, że dla pewnych danych, które mogą być obarczone małym błędem względnym, powoduje duży błąd względny w wyniku.

Ćwiczenie 3

Podaj przykład wartości a , b i c , innych niż na rysunku 6.2, dla których program znajdujący pierwiastki równania kwadratowego $ax^2 + bx + c = 0$, gdzie $a \neq 0$, za pomocą algorytmu tradycyjnego z deltą zachowuje się niestabilnie.

Znajdowanie pierwiastków równania kwadratowego – algorytm 2

Skonstruujemy teraz algorytm znajdujący rozwiązanie równania kwadratowego, w którym otrzymane wyniki będą się nieznacznie różniły od oczekiwanych. Jeśli algorytm dla danych, które mogą być obarczone niewielkimi błędami, zawsze zwraca wyniki nieznacznie tylko zaburzo-

Algorytm stabilny  ne, to nazywamy go **algorytmem stabilnym**.

W algorytmie jeden z pierwiastków policzymy z wykorzystaniem jednego ze wzorów na pierwiastek kwadratowy równania. Wybierzemy ten wzór, w którym nie wystąpi odejmowanie liczb prawie równych co do wartości. Drugi pierwiastek wyznaczmy, korzystając ze wzoru Viète'a na iloczyn pierwiastków: $x_1 \cdot x_2 = \frac{c}{a}$.

Oto zapis algorytmu w pseudokodzie:

```
delta ← b * b - 4 * a * c
jeśli delta < 0 to wypisz "Brak pierwiastków"
w przeciwnym przypadku
    pdelta ← pierwiastek(delta)
    jeśli b > 0 to
        x1 ← (-b - pdelta)/(2 * a)
    w przeciwnym przypadku
        x1 ← (-b + pdelta)/(2 * a)
    jeśli x1 ≠ 0 to x2 ← c/(a * x1)
    w przeciwnym przypadku x2 ← 0
    wypisz x1, x2
```

Dla współczynnika $b > 0$ wybieramy wzór na pierwiastek, w którym odejmujemy wartość pdelta (liczby $-b$ i $-\text{pdelta}$ są niedodatnie), w przeciwnym przypadku – wzór na pierwiastek, w którym dodawana jest wartość pdelta (liczby $-b$ i $+\text{pdelta}$ są nieujemne). Zabezpieczamy się w ten sposób przed sytuacją, w której odejmiemy od siebie dwie liczby o prawie równych wartościach. Drugi pierwiastek obliczamy ze wzoru Viète'a pod warunkiem, że pierwszy jest różny od zera. Jeśli $x_1 = 0$, to współczynnik c jest równy 0, a w konsekwencji wartość pdelta jest równa $|b|$. Podczas obliczania x_1 w liczniku otrzymujemy wartość $2b$ lub $-2b$, więc współczynnik b ma także wartość 0. Równanie ma postać $ax^2 = 0$, dlatego gdy $x_1 = 0$, wystarczy przyjąć $x_2 = 0$.

Fragment kodu źródłowego programu realizującego algorytm zapisany powyżej w pseudokodzie może wyglądać następująco:

Fragment kodu 
źródłowego programu
wyznaczającego
pierwiastki równania
kwadratowego –
algorytm 2

```
1. const float EPS=0.0000001;
2.
3. int main()
4. {
5.     float a, b, c, delta, pdelta, x1, x2;
6.     cout<<"a = "; cin>>a;
7.     cout<<"b = "; cin>>b;
8.     cout<<"c = "; cin>>c;
```

```

9.     delta=b*b-4*a*c;
10.    if (delta<0) cout<<"Brak pierwiastkow";
11.    else
12.    {
13.        pdelta=sqrt(delta);
14.        if (b>0)
15.            x1=(-b-pdelta)/(2*a);
16.        else
17.            x1=(-b+pdelta)/(2*a);
18.        if (abs(x1)>EPS) x2=c/(a*x1);
19.        else x2=0;
20.        cout<<"x1 = "<<x1<<endl;
21.        cout<<"x2 = "<<x2;
22.    }
23.    return 0;
24. }

```

Zwróć uwagę na zapis w linii 18. Wartość zmiennej x_1 porównujemy z zerem z dokładnością do stałej EPS. Jej wartość (linia 1) jest bardzo mała – na poziomie dokładności liczb pojedynczej precyzji.

Rysunek 6.3 przedstawia wynik działania programu z wykorzystaniem wzoru Viète'a dla równania $x^2 + 10\,000x + 1 = 0$.

Zwróć uwagę, że pierwiastek x_1 także jest policzony w przybliżeniu. Błąd przybliżenia jest jednak bardzo mały. Dokładne wartości pierwiastków równania $x^2 + 10\,000x + 1 = 0$ są następujące:

$$x_1 = -5000 - \sqrt{24999999}$$

$$x_2 = -5000 + \sqrt{24999999}$$

```

a = 1
b = 10000
c = 1
x1 = -10000
x2 = -0.0001

```

Rys. 6.3. Wynik działania programu z wykorzystaniem wzoru Viète'a dla równania $x^2 + 10\,000x + 1 = 0$



Dobra rada

Kiedy porównujesz wartości rzeczywiste, nie rób tego wprost, za pomocą równości lub nierówności. Korzystaj z relacji mniejszości lub większości z pewną dokładnością.

Ćwiczenie 4

Napisz program znajdujący pierwiastki równania kwadratowego z wykorzystaniem wzoru Viète'a na iloczyn pierwiastków. Sprawdź działanie programu dla różnych danych.

Zapamiętaj

Algorytm stabilny to algorytm, który dla danych obarczonych niewielkimi błędami względnymi zwróci wynik obarczony niewielkim błędem względnym. Algorytm niestabilny to algorytm, który dla danych obarczonych niewielkimi błędami względnymi może zwrócić wynik z dużym błędem względnym.

Podsumowanie

- Błąd reprezentacji powstaje w wyniku zaokrąglenia liczby zmiennoprzecinkowej.
- Błąd zaokrąglenia powstaje wskutek zaokrąglenia wyniku działań arytmetycznych na liczbach zmiennoprzecinkowych.
- Błąd obcięcia powstaje w wyniku wykonania skończonej liczby iteracji podczas obliczania wartości, które są wyrażone np. sumą nieskończonej wielu wyrazów.
- Błąd przybliżenia powstaje, gdy algorytm znajduje rozwiązanie problemu z pewną dokładnością.
- Błąd bezwzględny to wartość bezwzględna różnicy pomiędzy wartością dokładną i wartością przybliżoną. Informuje on, o ile wartość przybliżona różni się od dokładnej.
- Błąd względny określa, o jaką część wartości dokładnej różni się wartość przybliżona. Błąd względny można wyrazić w procentach.
- Algorytm stabilny to algorytm, który dla danych obarczonych niewielkimi błędami względnymi zwraca wynik obarczony niewielkim błędem względnym.
- Algorytm niestabilny to algorytm, który dla danych obarczonych niewielkimi błędami względnymi może zwrócić wynik z dużym błędem względnym.
- Przykładem algorytmu niestabilnego jest tzw. algorytm z deltą, obliczający pierwiastki równania kwadratowego.
- Pierwiastki równania kwadratowego można wyznaczyć algorytmem stabilnym z wykorzystaniem wzoru Viète'a na iloczyn pierwiastków równania.

Zadania

- ★ **1** Oblicz błąd względny reprezentacji liczby dziesiętnej 0,1, jeśli na część ułamkową przeznaczonych jest 8 bitów.
Wskazówka: Możesz skorzystać z tabeli 5.3 na s. 104.
- ★ **2** Zmodyfikuj wybrany program znajdujący pierwiastki równania kwadratowego tak, aby rozpoznawał sytuację, gdy wyróżnik równania kwadratowego jest równy 0. Pamiętaj, aby nie porównywać wartości rzeczywistych wprost, ale z pewną dokładnością.
- ★★ **3** W programie znajdującym pierwiastki równania kwadratowego tradycyjnym algorytmem z deltą zamień typ pojedynczej precyzji (**float**) na typ podwójnej precyzji (**double**). Sprawdź działanie programu dla równania $x^2 + 10\,000x + 1 = 0$. Podaj przykład danych, dla których program zachowa się niestabilnie.
- ★★ **4** Napisz program, który obliczy wartość $\sin 1$, korzystając ze wzoru:

$$\sin 1 = 1 - \frac{1}{3!} + \frac{1}{5!} - \frac{1}{7!} + \frac{1}{9!} - \frac{1}{11!} + \dots$$

Obliczenia wykonaj z dokładnością do 4 miejsc po kropce dziesiętnej.

- ★★ **5** Napisz program, który obliczy wartość $\cos 1$, korzystając ze wzoru:

$$\cos 1 = 1 - \frac{1}{2!} + \frac{1}{4!} - \frac{1}{6!} + \frac{1}{8!} - \frac{1}{10!} + \dots$$

Obliczenia wykonaj z dokładnością do 4 miejsc po kropce dziesiętnej.

- ★★★ **6** Napisz program, który obliczy wartość $\sin x$, korzystając ze wzoru:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \dots$$

dla $0 \leq x \leq 6,28$. Obliczenia wykonaj z dokładnością do 4 miejsc po kropce dziesiętnej.

- ★★★ **7** Napisz program, który obliczy wartość $\cos x$, korzystając ze wzoru:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \dots$$

dla $0 \leq x \leq 6,28$. Obliczenia wykonaj z dokładnością do 4 miejsc po kropce dziesiętnej.

- ★★★ **8** Napisz program rozwiązujący układ równań liniowych

$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$

metodą wyznaczników. Podaj przykład danych, dla których program zachowa się niestabilnie.

7. Obliczanie wartości wielomianu

Pojęcie wielomianu znasz z lekcji matematyki. Wielomianów używa się m.in. do opisu zależności między wielkościami fizycznymi w modelowaniu różnych zjawisk. Mają zastosowanie także w informatyce. Za ich pomocą można wyznaczyć przybliżoną wartość funkcji, gdy dokładna wartość jest trudna do znalezienia (np. w przypadku funkcji trygonometrycznej). W tym temacie zajmiemy się obliczaniem wartości wielomianu. Wykorzystamy do tego różne algorytmy: algorytm naiwny oraz algorytm, który wykona mniej działań arytmetycznych.

Cele lekcji

- Obliczysz wartość wielomianu algorytmem naiwnym.
- Wyznaczysz wartość wielomianu algorytmem optymalnym – wykorzystującym schemat Hornera.
- Dowiesz się, w których z omawianych do tej pory problemów użyliśmy schematu Hornera.

7.1. Sformułowanie problemu obliczania wartości wielomianu

Wielomian • Przypomnijmy, że **wielomianem** stopnia n zmiennej rzeczywistej x nazywamy funkcję postaci:

$$W(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x + a_0$$

gdzie $a_n \neq 0$, n – liczba naturalna dodatnia, $a_n, a_{n-1}, \dots, a_2, a_1, a_0$ – współczynniki wielomianu będące liczbami rzeczywistymi. Współczynnik a_0 nazywamy wyrazem wolnym.

W temacie nie będziemy się zajmować wielomianem zerowym, czyli funkcją $W(x) = 0$. Oto specyfikacja problemu obliczania wartości wielomianu dla danego argumentu:

Specyfikacja

Dane: n – liczba całkowita dodatnia oznaczająca stopień wielomianu, $A[0..n]$ – tablica liczb rzeczywistych będących współczynnikami wielomianu, $A[n] \neq 0$, element $A[0]$ odpowiada współczynnikowi a_0 , $A[1]$ – współczynnikowi a_1 itd.,
 x – liczba rzeczywista.

Wynik: y – wartość wielomianu o współczynnikach z tablicy A dla argumentu x .

7.2. Obliczanie wartości wielomianu algorytmem naiwnym

Wartość wielomianu dla danego argumentu można policzyć, podstawiając ten argument bezpośrednio do wzoru wielomianu. Taki algorytm jest algorytmem naiwnym. Sumowanie wyrazów wielomianu warto rozpocząć od wyrazu wolnego i wykorzystywać policzoną aktualnie potęgę x do wyliczenia kolejnej ($x^n = x^{n-1} \cdot x$ dla $n > 1$). Oto zapis algorytmu w pseudokodzie:

```
y ← A[0]
z ← 1
dla i ← 1, 2, ..., n wykonuj
    z ← z * x
    y ← y + A[i] * z
```

Wartością początkową wyniku (zmienną y) jest wartość wyrazu wolnego ($A[0]$). Kolejne potęgi wartości zmiennej x będziemy pamiętać w zmiennej pomocniczej z . Na początku przyjmuje ona wartość 1 (czyli x^0). Na potrzeby naszych obliczeń przyjmujemy, że $0^0 = 1$.

Chcemy obliczać wartości wielomianów różnego stopnia, a więc liczba współczynników będzie się zmieniać. Dlatego współczynniki zapamiętamy w zmiennej typu **vector**.

Kod źródłowy funkcji Czytaj, wczytującej stopień wielomianu oraz jego współczynniki, może wyglądać następująco:

```
1. void Czytaj(vector<float> &A)
2. {
3.     int n;
4.     cout<<"Stopien wielomianu: ";
5.     cin>>n;
6.     A.resize(n+1);
7.     for (int i=n;i>=0;i--)
8.     {
9.         cout<<"a"<<i<<" = ";
10.        cin>>A[i];
11.    }
12. }
```

Parametr A typu **vector**, pamiętający współczynniki wielomianu, przekazywany jest przez referencję (linia 1), ponieważ funkcja określa jego wartość. W linii 5 wczytywany jest stopień wielomianu. Następnie w linii 6 rozmiar parametru A jest ustalany na $n+1$ (ponieważ parametr A przechowuje współczynniki od a_0 do a_n). Pętla w liniach 7–11 wczytuje współczynniki wielomianu, zaczynając od współczynnika a_n przy najwyższej potędze x (element $A[n]$).

Kod źródłowy funkcji realizującej algorytm naiwny obliczania wartości wielomianu może być następujący.

Dobra rada

Algorytm naiwny to algorytm wykorzystujący najbardziej intuicyjną metodę. Pamiętaj, że takie algorytmy są często mało efektywne.

Dobra rada

Pamiętaj, że korzystanie z typu **vector** jest możliwe po dołączeniu do programu biblioteki **vector** (dyrektywa `#include <vector>`).

Kod źródłowy funkcji wczytującej stopień oraz współczynniki wielomianu

Dobra rada

Nie musisz przekazywać na zewnątrz funkcji stopnia wielomianu (rozmiaru parametru typu **vector**), ponieważ jest on dostępny dzięki metodzie **size**.

Kod źródłowy funkcji
obliczającej wartość
wielomianu algorytmem
naiwnym

```

1. float W(vector<float> A, float x)
2. {
3.     float y=A[0], z=1;
4.     for (int i=1;i<A.size();i++)
5.     {
6.         z=z*x;
7.         y=y+A[i]*z;
8.     }
9.     return y;
10. }

```

W linii 3 deklarowane są dwie zmienne y i z : pierwszą z nich wykorzystamy do sumowania wyrazów wielomianu, drugą – do obliczania potęg x . Zmiennym y i z przy deklaracji nadawane są wartości początkowe. Zwróć uwagę na zakres zmiennej i , sterującej pętlą **for** (linia 4). Zmienia się on od 1 (jako pierwszy w pętli będzie wykorzystany współczynnik a_1) do wartości $A.size()-1$ (rozmiar zmiennej typu **vector** wynosi $n+1$, więc współczynnik a_n ma indeks $A.size()-1$). W linii 6 wartość zmiennej z mnożona jest przez wartość zmiennej x , wskutek czego otrzymujemy wartość x^i . W linii 7 do wartości zmiennej y dodawany jest i -ty wyraz wielomianu.

Rysunek 7.1 przedstawia efekt wywołania programu dla wielomianu $x^3 + 2x^2 + 3x + 4$ i argumentu $x = 2$.

```

Stopień wielomianu: 3
a3 = 1
a2 = 2
a1 = 3
a0 = 4
x = 2
W(2) = 26

```

Rys. 7.1. Efekt wywołania programu dla wielomianu $x^3 + 2x^2 + 3x + 4$ oraz argumentu $x = 2$

Ćwiczenie 1

Napisz program, który obliczy wartość wielomianu algorytmem naiwnym zgodnie ze specyfikacją podaną na s. 122. Wykorzystaj przedstawione w temacie funkcje **Czytaj** i **W**.

Omówiony algorytm naiwny obliczania wartości wielomianu wykonuje $2n$ operacji mnożenia oraz n operacji dodawania. Warto się zastanowić, czy nie można przekształcić wzoru ogólnego funkcji wielomianowej, aby wykonywać mniej działań arytmetycznych.

7.3. Obliczanie wartości wielomianu za pomocą schematu Hornera

Liczbę operacji arytmetycznych podczas wyznaczania wartości wielomianu możemy zmniejszyć, jeśli wykorzystamy schemat Hornera. Jego działanie wyjaśnimy na przykładzie wielomianu stopnia 3. Z trzech pierwszych wyrazów wyłączmy x przed nawias.

$$a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0 = x \cdot (a_3 \cdot x^2 + a_2 \cdot x + a_1) + a_0$$

W nawiasie otrzymujemy wielomian stopnia o 1 mniejszego. Dla tego wielomianu (w tym przypadku trójmianu kwadratowego) postępujemy w ten sam sposób – wyłączamy x przed nawias.

$$x \cdot (a_3 \cdot x^2 + a_2 \cdot x + a_1) + a_0 = x \cdot (x \cdot (a_3 \cdot x + a_2) + a_1) + a_0$$

Obliczając wartość wielomianu z tego wzoru, wykonamy 3 mnożenia i 3 dodawania. Gdybyśmy skorzystali ze wzoru funkcji wprost, wykonalibyśmy także 3 dodawania, ale 6 operacji mnożenia.

Podobnie będziemy postępować dla wielomianu stopnia n : wielokrotnie wyłączamy x przed nawias, aż do otrzymania w najbardziej wewnętrznych nawiasach dwumianu $a_n \cdot x + a_{n-1}$. Dla wielomianu stopnia n wykonamy n operacji mnożenia (zamiast $2n$). Jest to minimalna liczba operacji mnożenia, jaką trzeba wykonać, aby obliczyć wartość wielomianu. Algorytm wykorzystujący ten wzór jest więc algorytmem optymalnym.

Dla wielomianu stopnia n **schemat Hornera** można zapisać wzorem: **• Schemat Hornera**

$$W(x) = x \cdot (x \cdot \dots \cdot (a_n \cdot x + a_{n-1}) + \dots + a_1) + a_0$$

Algorytm obliczający wartość wielomianu z wykorzystaniem schematu Hornera można zapisać następująco:

```
y ← A[n]
dla i ← n - 1, n - 2, ..., 0 wykonuj
    y ← x * y + A[i]
```

Obliczenia rozpoczynamy od współczynnika przy najwyższej potędze (wartość początkowa zmiennej y). W pętli mnożymy dotychczasową wartość zmiennej y przez x i dodajemy kolejny współczynnik. Kod źródłowy funkcji realizującej ten algorytm znajduje się na s. 126.

A to ciekawe

Horner i początki animacji

Nazwa „schemat Hornera” pochodzi od nazwiska angielskiego matematyka Williama G. Hornera (1786–1837). Znany jest on nie tylko z dokonań matematycznych – interesował się też m.in. optyką. W 1834 r. skonstruował zootrop (ang. *zoetrope*) – urządzenie służące do uzyskiwania ruchomego obrazu. Ma ono postać obrotowego bębna, wewnątrz którego znajduje się papierowa taśma z sekwencją rysunków przedstawiających kolejne etapy ruchu. Po wprowadzeniu bębna w ruch przez pionowe szczeliny można zaobserwować ruchomy obraz. Zootrop upowszechnił się jako zabawka w latach 60. XIX w.



Dobra rada

Pamiętaj, że algorytm optymalny to algorytm wykonujący najmniejszą możliwą liczbę operacji charakterystycznych dla danego problemu. Nie istnieje rozwiązanie tego problemu, które wykorzystuje mniejszą liczbę operacji.

Kod źródłowy
funkcji obliczającej
wartość wielomianu
z wykorzystaniem
schematu Hornera

```
1. float W(vector<float> A, float x)
2. {
3.     int n=A.size()-1;
4.     float y=A[n];
5.     for (int i=n-1;i>=0;i--) y=x*y+A[i];
6.     return y;
7. }
```

Kod źródłowy funkcji
wczytującej stopień
oraz współczynniki
wielomianu,
s. 123 [🔗](#)

Funkcja wczytująca stopień wielomianu oraz jego współczynniki będzie wyglądała tak samo jak w przypadku programu wyznaczającego wartość wielomianu algorytmem naiwnym.

Ćwiczenie 2

Napisz program, który obliczy wartość wielomianu z wykorzystaniem schematu Hornera, zgodnie ze specyfikacją podaną na s. 122.

Zapamiętaj

Stosując schemat Hornera, wielokrotnie wyłącza się argument przed nawias we wzorze ogólnym funkcji wielomianowej, aż w najbardziej wewnętrznych nawiasach otrzymamy wielomian pierwszego stopnia. Z powstałego wzoru można obliczyć wartość wielomianu stopnia n , wykonując n operacji mnożenia i n operacji dodawania. Jest to algorytm optymalny.

7.4. Inne zastosowania schematu Hornera

Algorytmy zamiany
liczby dwójkowej na
dziesiętną, podręcznik
Informatyka na czasie 2.
Zakres rozszerzony,
s. 50–52 [🔗](#)

Zamiana liczby
w systemie
pozycyjnym o wybranej
podstawie na liczbę
dziesiętną, podręcznik
Informatyka na czasie 2.
Zakres rozszerzony,
s. 55–56 [🔗](#)

Schemat Hornera wykorzystywaliśmy podczas zamiany liczby dwójkowej na dziesiętną oraz podczas zamiany liczby w systemie pozycyjnym o wybranej podstawie na liczbę dziesiętną. W tych algorytmach rolę współczynników wielomianu pełnią cyfry reprezentacji liczby w systemie o podstawie p . Stopień wielomianu określa liczba cyfr (stopień jest o jeden mniejszy od liczby cyfr), a podstawa systemu jest argumentem, dla którego obliczamy wartość wielomianu.

Wartość dziesiętną liczby w systemie pozycyjnym o podstawie p zapisanej za pomocą n cyfr, gdzie c_i oznacza i -tą cyfrę liczby przegłądanej od prawej do lewej, $0 \leq i < n$, obliczymy ze wzoru:

$$c_{n-1} \cdot p^{n-1} + c_{n-2} \cdot p^{n-2} + \dots + c_1 \cdot p + c_0$$

Podany wzór to sposób obliczania wartości wielomianu o współczynnikach $c_{n-1}, \dots, c_0$ dla argumentu p .

Rozważamy wielomian $W(x)$ stopnia $n - 1$:

$$W(x) = c_{n-1} \cdot x^{n-1} + \dots + c_2 \cdot x^2 + c_1 \cdot x + c_0$$

Obliczmy wartość $W(p)$, korzystając ze schematu Hornera:

$$W(p) = p \cdot (p \cdot \dots \cdot (c_{n-1} \cdot p + c_{n-2}) + \dots + c_1) + c_0$$

Tego sposobu używaliśmy do obliczania wartości dziesiętnej liczby zapisanej w systemie pozycyjnym o podstawie innej niż 10.

Schemat Hornera stosuje się także w algorytmie obliczającym potęgę liczby. W wersji iteracyjnej **algorytmu szybkiego podnoszenia do potęgi** wykorzystywaliśmy zapis wykładnika potęgi w systemie binarnym. Dokładniej – przeglądaliśmy cyfry binarne wykładnika w kolejności od prawej do lewej, a więc w kolejności, w jakiej są one uzyskiwane podczas zamiany liczby dziesiętnej na binarną.

Jeśli znamy zapis wykładnika w postaci binarnej, możemy także zapisać algorytm szybkiego podnoszenia do potęgi, który przegląda cyfry binarne wykładnika od lewej do prawej, czyli od cyfry najbardziej znaczącej. Korzystamy wtedy wprost ze schematu Hornera. Na przykład:

$$\begin{aligned} x^{13} &= x^{1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0} = x^{2 \cdot (1 \cdot 2^2 + 1 \cdot 2 + 0) + 1} = x^{2 \cdot (2 \cdot (1 \cdot 2 + 1) + 0) + 1} = \\ &= (x^{2 \cdot (1 \cdot 2 + 1) + 0})^2 \cdot x = ((x^{1 \cdot 2 + 1})^2) \cdot x = ((x^2 \cdot x)^2) \cdot x \end{aligned}$$

Algorytm wykorzystujący ten wzór można zapisać w pseudokodzie tak jak poniżej. Zakładamy, że napis s jest reprezentacją binarną wykładnika potęgi. Wartość zmiennej y jest policzoną potęgą liczby x .

```
y ← 1
dla i ← 0, 1, ..., długość(s) – 1 wykonuj
    y ← y * y
    jeśli s[i]='1' to y ← y * x
```

W każdym kroku algorytmu podnosimy aktualną wartość wyniku do kwadratu (wartością początkową jest 1, czyli x^0). Jeśli w zapisie binarnym wykładnika występuje jedynka, to dodatkowo mnożymy wynik przez podstawę potęgi.

Zwróć uwagę, że przedstawiony algorytm jest iteracyjną wersją **rekurencyjnego algorytmu szybkiego podnoszenia do potęgi**. W rekurencyjnej wersji algorytmu reprezentacja binarna wykładnika była pamiętana niejawnie na stosie.

Ćwiczenie 3

Napisz program, który wczyta podstawę potęgi oraz napis reprezentujący wykładnik potęgi w systemie binarnym, a następnie wypisze wartość potęgi. Zastosuj algorytm szybkiego podnoszenia do potęgi z wykorzystaniem schematu Hornera.

Zapamiętaj

Schemat Hornera pozwala ograniczać liczbę operacji wykonywanych przez algorytmy. Wykorzystuje się go np. w algorytmach zamiany reprezentacji liczby w systemie pozycyjnym o podstawie innej niż 10 na reprezentację w systemie dziesiętnym, a także w algorytmie szybkiego podnoszenia do potęgi.

Algorytm szybkiego podnoszenia do potęgi, podręcznik *Informatyka na czasie 2. Zakres rozszerzony*, s. 57–58 [↗](#)

Dobra rada

Jeśli wykładnik potęgi jest reprezentowany w systemie dziesiętnym, zastosuj algorytm przeglądający cyfry binarne od prawej do lewej, ponieważ wówczas nie musisz pamiętać reprezentacji binarnej wykładnika potęgi.

Rekurencyjny algorytm szybkiego podnoszenia do potęgi, podręcznik *Informatyka na czasie 2. Zakres rozszerzony*, s. 241–244 [↗](#)

Podsumowanie

- Wartość wielomianu można obliczyć algorytmem naiwnym, który wykonuje $2n$ operacji mnożenia i n operacji dodawania, gdzie n oznacza stopień wielomianu.
- Schemat Hornera polega na wielokrotnym wyłączaniu argumentu przed nawias we wzorze ogólnym funkcji wielomianowej, aż w najbardziej wewnętrznych nawiasach otrzymamy wielomian pierwszego stopnia.
- Obliczając wartość wielomianu stopnia n z wykorzystaniem schematu Hornera, wykonujemy n operacji mnożenia i n operacji dodawania. Algorytm ten jest optymalny.
- Schemat Hornera stosuje się m.in. do obliczania wartości dziesiętnej liczb zapisanych w systemie o innej podstawie oraz do szybkiego podnoszenia do potęgi.

Zadania

- 1 Napisz program, który wczyta z klawiatury dwa wielomiany, a następnie wyznaczy wielomian będący ich sumą i go wypisze.
- 2 Napisz program, który z pliku tekstowego przekazanego ci przez nauczyciela (np. *wielomian.txt*) wczyta stopień i współczynniki wielomianu, a następnie dla argumentu wczytanego z klawiatury obliczy wartość wielomianu, korzystając ze schematu Hornera. Stopień wielomianu znajduje się w pierwszym wierszu pliku tekstowego, w kolejnych wierszach podane są współczynniki wielomianu.
- 3 Napisz program, który wczyta z klawiatury podstawę systemu pozycyjnego z zakresu od 2 do 9, liczbę cyfr liczby oraz cyfry tej liczby i obliczy jej wartość dziesiętną, korzystając ze schematu Hornera.
- 4 Napisz program obliczający wartość wielomianu zgodnie ze specyfikacją przedstawioną na s. 122, z wykorzystaniem schematu Hornera. Funkcję realizującą schemat Hornera zapisz rekurencyjnie.
- 5 Napisz program, który wczyta z klawiatury dwa wielomiany, a następnie wyznaczy wielomian będący ich iloczynem i go wypisze.
Wskazówka: Stopień wielomianu będącego iloczynem wielomianów to suma stopni wielomianów składowych.
- 6 Napisz program, który wczyta z klawiatury wielomian oraz liczbę całkowitą c i obliczy iloraz wielomianu przez dwumian $x - c$. Zastosuj schemat Hornera.
- 7 Napisz program obliczający przybliżoną wartość $\sin x$ ze wzoru:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \dots$$
 dla $0 \leq x \leq 6,28$. Obliczenia zakończ po zsumowaniu sześciu pierwszych wyrazów. Wykorzystaj schemat Hornera, a wartości silni zapamiętaj w tablicy.

8. Metody obliczeń przybliżonych

W wielu sytuacjach wymagających zastosowania obliczeń łatwo jest otrzymać dokładny wynik – np. w wyborach parlamentarnych, w których zliczany jest każdy oddany głos. Istnieją jednak problemy, których nie potrafimy rozwiązać za pomocą znanych algorytmów znajdujących dokładne wyniki lub algorytmy takie są czasochłonne. Można wówczas wykorzystać metody, które dają wyniki przybliżone.

Cele lekcji

- Znajdziesz miejsce zerowe funkcji metodą bisekcji.
- Obliczysz pierwiastek kwadratowy z liczby metodą bisekcji i metodą Newtona–Raphsona.
- Poznasz metodę prostokątów i metodę trapezów, wykorzystywane do obliczania pól obszarów zamkniętych.
- Zastosujesz metodę Monte Carlo do policzenia przybliżonej wartości liczby π oraz do symulacji ruchów Browna.

W poprzednich tematach zajmowaliśmy się m.in. **znajdowaniem pierwiastków równania kwadratowego** oraz **obliczaniem wartości wielomianu**. Choć przedstawione tam algorytmy umożliwiają znalezienie dokładnych wyników, to programy realizujące te algorytmy podają rozwiązania obarczone błędami. Wynikają one jednak z reprezentacji liczb w komputerze, a nie z samych algorytmów. Teraz zajmemy się problemami i metodami ich rozwiązania, w których przybliżony wynik jest przede wszystkim skutkiem zastosowanego algorytmu.

Algorytmy znajdowania pierwiastków równania kwadratowego, s. 115–119 [↗](#)

Obliczanie wartości wielomianu, s. 122–126 [↗](#)

8.1. Znajdowanie miejsc zerowych funkcji

Jedną z metod znajdowania miejsc zerowych funkcji, czyli argumentów, dla których funkcja przyjmuje wartość 0, jest **metoda bisekcji** (połowienia). Postępuje się w niej podobnie jak w **algorytmie przeszukiwania binarnego** – w kolejnych krokach zmniejszamy o połowę zakres przeszukiwanych danych.

W ogólnym przypadku w metodzie bisekcji konstruujemy ciąg, którego wyrazami są przybliżenia poszukiwanej wartości, a granicą tego ciągu jest poszukiwana wartość. Kolejne wyrazy ciągu to środki coraz mniejszych przedziałów, które przeszukujemy. Zakres poszukiwań zawężamy o połowę, zastępując jeden z końców przedziału środkiem przedziału. Kończymy, gdy wartość kolejnego wyrazu ciągu jest równa szukanej wartości lub gdy długość przeszukiwanego przedziału jest nie większa od przyjętej dokładności obliczeń (wynik jest wówczas przybliżony – jest nim ostatnio wyznaczony środek przedziału).

Metoda bisekcji (połowienia)

Przeszukiwanie binarne, podręcznik *Informatyka na czasie 2. Zakres rozszerzony*, s. 154 [↗](#)

Aby można było zastosować metodę bisekcji do poszukiwania miejsca zerowego funkcji f w przedziale $[a; b]$, muszą być spełnione warunki:

- ▶ funkcja f jest określona i ciągła w przedziale $[a; b]$,
- ▶ wartości funkcji dla końców przedziału $[a; b]$ mają różne znaki, czyli $f(a) \cdot f(b) < 0$.

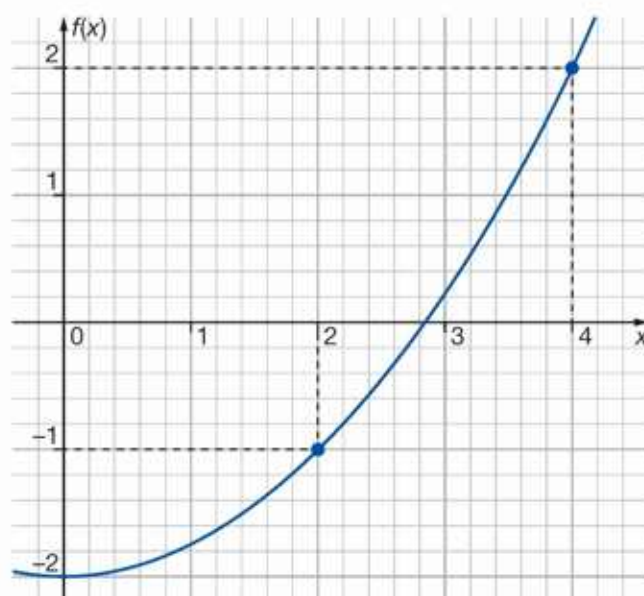
Warto wiedzieć

Istnieją inne metody przybliżonego znajdowania miejsc zerowych funkcji, np. metoda siecznych.

Zauważ, że jeśli funkcja jest ciągła w przedziale $[a; b]$ i wartości tej funkcji dla końców przedziału mają różne znaki, musi istnieć przynajmniej jedno miejsce zerowe. Takiej pewności nie mielibyśmy, gdyby wartości funkcji dla końców przedziału miały jednakowe znaki.

Metodą bisekcji w danym przedziale będziemy szukać jednego miejsca zerowego. Żeby znaleźć inne miejsca zerowe, należałoby wielokrotnie stosować algorytm dla różnych przedziałów.

Rysunek 8.1 przedstawia fragment wykresu funkcji $f = \frac{1}{4}x^2 - 2$. Jest ona określona i ciągła w przedziale $[2; 4]$. Ponadto wartości funkcji dla końców przedziału mają różne znaki. Można zatem zastosować metodę bisekcji do poszukiwania miejsca zerowego tej funkcji w przedziale $[2; 4]$.



Rys. 8.1. Fragment wykresu funkcji $f = \frac{1}{4}x^2 - 2$, określonej i ciągłej w przedziale $[2; 4]$, $f(2) \cdot f(4) < 0$

Na początku poszukiwania miejsca zerowego musimy ustalić dokładność obliczeń. W kolejnych krokach najpierw sprawdzamy, czy wartość funkcji dla środka przedziału jest równa 0 – jeśli tak jest, znaleźliśmy miejsce zerowe i algorytm kończy działanie. W przeciwnym przypadku dzielimy przedział na pół, zastępując jeden z jego końców wyznaczonym środkiem (wybieramy połowę spełniającą warunki metody bisekcji). Jeśli długość przedziału jest większa od przyjętej dokładności obliczeń, ponawiamy poszukiwanie miejsca zerowego w tym przedziale, jeśli nie – algorytm kończy działanie, a za miejsce zerowe przyjmujemy ostatnio wyznaczony środek przedziału (wynik jest przybliżony).

Tabela 8.1 zawiera informacje o przedziałach i wyliczanych wartościach w sześciu początkowych krokach algorytmu dla funkcji z rysunku 8.1.

Krok algorytmu	Rozpatrywany przedział	Wartość funkcji dla lewego końca przedziału	Wartość funkcji dla prawego końca przedziału	Środek przedziału	Wartość funkcji dla środka przedziału
1	[2; 4]	-1	2	3	0,25
2	[2; 3]	-1	0,25	2,5	-0,4375
3	[2,5; 3]	-0,4375	0,25	2,75	-0,109375
4	[2,75; 3]	-0,109375	0,25	2,875	0,06640625
5	[2,75; 2,875]	-0,109375	0,06640625	2,8125	-0,0224609375
6	[2,8125; 2,875]	-0,0224609375	0,06640625	2,84375	0,021728515625

Tabela 8.1. Informacje o przedziałach oraz wyliczanych wartościach w sześciu pierwszych krokach algorytmu poszukującego miejsca zerowego funkcji $f = 0,25x^2 - 2$ w przedziale [2; 4] metodą bisekcji

Gdybyśmy za dokładność obliczeń przyjęli 0,01, otrzymalibyśmy miejsce zerowe równe 2,82812. Dla dokładności 0,00001 wyznaczonym miejscem zerowym byłaby liczba 2,82843. Dokładną wartością miejsca zerowego jest $2\sqrt{2}$, czyli w przybliżeniu 2,828427.

Oto specyfikacja problemu poszukiwania miejsca zerowego funkcji w danym przedziale:

Specyfikacja

Dane: a, b – liczby rzeczywiste, $a < b$,

$f(x)$ – funkcja określona i ciągła w przedziale $[a; b]$, $f(a)$ i $f(b)$ mają różne znaki, czyli $f(a) \cdot f(b) < 0$,

eps – dokładność obliczeń.

Wynik: x_0 – miejsce zerowe funkcji $f(x)$ w przedziale $[a; b]$ znalezione z dokładnością eps .

Najpierw obliczymy wartość $x_0 = (a + b)/2$. Jeśli $f(x_0) = 0$, to szukanym miejscem zerowym jest x_0 . W przeciwnym przypadku poszukujemy dalej miejsca zerowego: w przedziale $[a; x_0]$, gdy $f(a) \cdot f(x_0) < 0$, albo w przedziale $[x_0; b]$, gdy $f(x_0) \cdot f(b) < 0$. Obliczenia kończymy, gdy otrzymamy $f(x_0) = 0$ lub długość przedziału jest nie większa od dokładności obliczeń eps . Oto zapis algorytmu w pseudokodzie:

```

 $x_0 \leftarrow (a + b)/2$ 
dopóki  $f(x_0) \neq 0$  oraz  $b - a > \text{eps}$  wykonuj
    jeśli  $f(a) * f(x_0) < 0$  to  $b \leftarrow x_0$ 
    w przeciwnym przypadku  $a \leftarrow x_0$ 
     $x_0 \leftarrow (a + b)/2$ 

```

Implementując powyższy algorytm, założymy, że mamy zdefiniowane stałą EPS, informującą o dokładności obliczeń, oraz funkcję f , znajdującą wartość funkcji dla parametru x .

Dobra rada

Przy ustalaniu dokładności obliczeń w programie znajdującym miejsce zerowe funkcji pamiętaj o ograniczeniach komputera – przechowuje on liczby z określoną precyzją. Ponadto zbyt duża liczba iteracji w algorytmie może znacznie spowolnić działanie programu.

Definicja stałej oraz nagłówki funkcji mogą wyglądać następująco:

```
const float EPS=0.00001;
float f(float x);
```

Oto kod źródłowy funkcji `main` programu znajdującego miejsce zerowe funkcji `f` metodą bisekcji:

Fragment kodu
źródłowego programu
znajdującego miejsce
zerowe funkcji `f` metodą
bisekcji – funkcja `main`

Dobra rada

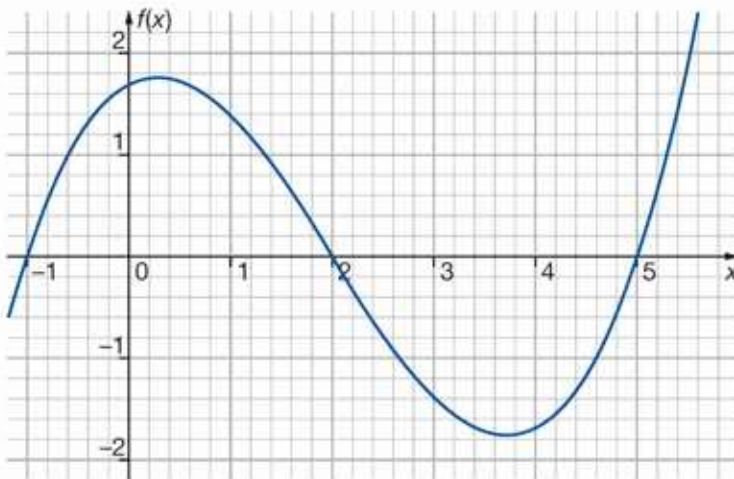
Pamiętaj, że z funkcji `abs` możesz korzystać w programie, jeśli dołączysz bibliotekę `cmath`.

```
1. int main()
2. {
3.     float a, b, x0, fa, fb, fx0;
4.     cout<<"a = "; cin>>a;
5.     cout<<"b = "; cin>>b;
6.     fa=f(a); fb=f(b);
7.     if (fa*fb>=0)
8.     {
9.         cout<<"Wartosci funkcji dla koncow przedzialu ";
10.        cout<<"nie maja roznych znakow"<<endl;
11.        return 0;
12.    }
13.    x0=(a+b)/2; fx0=f(x0);
14.    while (abs(fx0)>EPS && b-a>EPS)
15.    {
16.        if (fa*fx0<0) b=x0;
17.        else
18.        {
19.            a=x0; fa=fx0;
20.        }
21.        x0=(a+b)/2; fx0=f(x0);
22.    }
23.    cout<<"x0 = "<<x0;
24.    return 0;
25. }
```

W linii 3 oprócz zmiennych `a`, `b`, `x0` są zadeklarowane zmienne `fa`, `fb` i `fx0`, służące do zapamiętania wartości funkcji `f` dla parametrów odpowiednio `a`, `b`, `x0`. Dzięki temu nie musimy parokrotnie obliczać wartości funkcji dla tych samych parametrów. W liniach 4–5 wczytywane są końce przedziału, a w linii 6 obliczane są wartości funkcji dla tych końców. Dodatkowo w liniach 7–12 sprawdzamy, czy wartości funkcji dla końców przedziału mają różne znaki. Jeśli znaki nie są różne, to program kończy działanie – metodą bisekcji nie znajdziemy miejsca zerowego w tym przedziale (co nie oznacza, że go nie ma).

Linie 13–22 realizują algorytm zapisany w pseudokodzie. Zwróć uwagę na linię 14 – nie porównujemy w niej wartości `fx0` z zerem, lecz ze stałą `EPS`. Należy tak zrobić, ponieważ zmienna `fx0` jest reprezentowana jako liczba zmiennopozycyjna, czyli wartość pamiętana przez komputer jest zaokrąglana. Instrukcja z linii 23 wypisuje otrzymane miejsce zerowe.

Rysunek 8.2 przedstawia wykres funkcji $f(x) = \frac{1}{6}x^3 - x^2 + \frac{1}{2}x + \frac{5}{3}$ oraz efekty wykonania programu dla tej funkcji i różnych przedziałów.



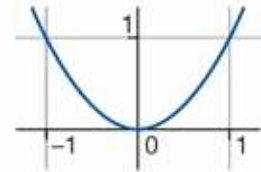
$a = -3$
 $b = 0$
 $x_0 = -1$

$a = 0$
 $b = 3$
 $x_0 = 2$

$a = 3$
 $b = 10$
 $x_0 = 5$

Warto wiedzieć

Istnieją funkcje ciągłe, które mają miejsce zerowe, ale nie mają wartości o różnych znakach dla końców jakiegokolwiek przedziału. Dlatego metodą bisekcji nie znajdziemy ich miejsca zerowego. Taką funkcją jest np. $f(x) = x^2$.



Rys. 8.2. Wykres funkcji $f(x) = \frac{1}{6}x^3 - x^2 + \frac{1}{2}x + \frac{5}{3}$ oraz efekty wykonania programu dla przedziałów $[-3; 0]$, $[0; 3]$ i $[3; 10]$

Oto definicja funkcji f dla przykładu z rysunku 8.2:

```
1. float f(float x)
2. {
3.     return 1.0/6*x*x*x-x*x+0.5*x+5.0/3;
4. }
```

Definicja funkcji obliczającej wartość funkcji

$f(x) = \frac{1}{6}x^3 - x^2 + \frac{1}{2}x + \frac{5}{3}$
dla danego parametru

Ćwiczenie 1

Napisz program znajdujący miejsce zerowe funkcji metodą bisekcji.

Zapamiętaj

Metoda bisekcji (połowienia) umożliwia znalezienie miejsca zerowego funkcji określonej i ciągłej w przedziale $[a; b]$, jeśli wartości funkcji dla końców przedziału mają różne znaki, czyli $f(a) \cdot f(b) < 0$. Metoda polega na zawężaniu w każdym kroku przedziału poszukiwań o połowę do momentu, gdy środek przedziału jest miejscem zerowym lub długość przedziału jest nie większa od dokładności obliczeń (wówczas wynikiem jest ostatnio wyznaczony środek przedziału). Zawężając przedział, wybieramy tę połowę, dla której zachowany jest warunek $f(a) \cdot f(b) < 0$.



Dobra rada

Pamiętaj, że dla argumentów całkowitych znak / oznacza część całkowitą z dzielenia. Jeśli chcesz, żeby program potraktował znak / jako dzielenie z częścią ułamkową, przynajmniej jeden z argumentów zapisz z użyciem kropki. Na przykład dla $5.0/3$ otrzymasz wynik rzeczywisty, a dla $5/3$ – całkowity.

8.2. Obliczanie pierwiastka kwadratowego

Pierwiastek kwadratowy z nieujemnej liczby rzeczywistej obliczaliśmy za pomocą funkcji `sqrt`. Pokażemy, jak to zrobić bez użycia tej funkcji.

Funkcja `sqrt`,
s. 117 [↗](#)

Obliczanie pierwiastka kwadratowego metodą bisekcji

Na początku będziemy poszukiwać pierwiastka kwadratowego z liczby rzeczywistej większej od 1. Oto specyfikacja problemu:

Specyfikacja

Dane: c – liczba rzeczywista, $c > 1$,
eps – dokładność obliczeń.

Wynik: pkw – pierwiastek kwadratowy z liczby c wyliczony z dokładnością eps.

Metoda bisekcji
(połowienia),
s. 129 

Poszukiwanie pierwiastka kwadratowego z $c > 1$ **metodą bisekcji** sprowadza się do poszukiwania miejsca zerowego funkcji $f(x) = x^2 - c$ w przedziale $[1; c]$ – w tym przedziale jest ono równe $\sqrt{c}$.

Oto algorytm znajdowania pierwiastka kwadratowego z liczby rzeczywistej $c > 1$ metodą bisekcji, zapisany w pseudokodzie:

```
a ← 1
b ← c
dopóki b - a > eps wykonuj
    pkw ← (a + b)/2
    jeśli pkw * pkw < c to a ← pkw
    w przeciwnym przypadku b ← pkw
pkw ← (a + b)/2
```

Najpierw obliczamy środek przedziału (pkw). Jeśli jego kwadrat jest mniejszy od c , poszukiwania kontynuujemy w lewej połowie przedziału, w przeciwnym przypadku – w prawej. Obliczenia kończymy, gdy długość przedziału jest mniejsza od dokładności obliczeń eps lub jej równa.

Ćwiczenie 2

Napisz program znajdujący przybliżoną wartość pierwiastka kwadratowego z liczby rzeczywistej większej od 1 metodą bisekcji.

Obliczanie pierwiastka kwadratowego metodą Newtona–Raphsona

Zauważ, że $\sqrt{c}$, gdzie c jest dodatnią liczbą rzeczywistą, odpowiada długości boku kwadratu o polu c . Poszukiwanie takiego kwadratu można rozpocząć od dowolnego prostokąta o bokach a i b oraz polu c . Potem zmieniamy boki prostokąta w następujący sposób: bok a zastępujemy średnią arytmetyczną boków z poprzedniego kroku, bok b wyliczamy, dzieląc pole prostokąta przez bok a . Działania powtarzamy do momentu, gdy różnica długości boków prostokąta będzie nie większa od założonej dokładności obliczeń (boki będą takiej samej lub prawie takiej samej długości). Opisany sposób postępowania nazywa się **metodą Newtona–Raphsona**. Jej działanie pokażemy na przykładzie.

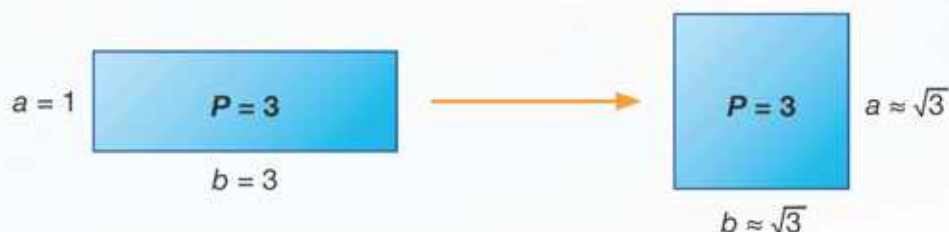
Metoda
Newtona–Raphsona 

Warto wiedzieć

Metodą bisekcji można także znaleźć pierwiastek kwadratowy z dodatniej liczby rzeczywistej $0 < c < 1$. Należy wówczas poszukiwać miejsca zerowego funkcji $f(x) = x^2 - c$ w przedziale $[c; 1]$.

Algorytm obliczania pierwiastka kwadratowego metodą Newtona–Raphsona

Znajdziemy przybliżoną wartość $\sqrt{3}$. Odpowiada ona długości boku kwadratu o polu 3. Poszukiwanie kwadratu o takim polu można rozpocząć od dowolnego prostokąta o iloczynie długości boków równym 3. My zaczniemy od prostokąta o bokach $a = 1$ i $b = 3$. Przyjmiemy, że kolejne przybliżenia $\sqrt{3}$ to długości boku a wyznaczone w poszczególnych krokach. Wartość pierwiastka znajdziemy z dokładnością do 0,00001, zatem przybliżenia wyznaczamy, dopóki $|a - b| > 0,00001$. Wyjściowy prostokąt oraz wynik, do którego dążymy, ilustruje poniższy rysunek.



Krok 1

Zmieniamy długości boków prostokąta tak, aby długość boku a była średnią arytmetyczną długości boków wyjściowego prostokąta, czyli:

$$a = \frac{(1 + 3)}{2} = 2. \text{ Zatem } b = \frac{3}{2} = 1,5.$$

$|a - b| > 0,00001$, więc przechodzimy do kolejnego kroku.

$$a = 2$$

$$b = 1,5$$

Krok 2

Obliczamy długości boków a i b :

$$a = \frac{(2 + 1,5)}{2} = 1,75$$

$$b = \frac{3}{1,75} \approx 1,71429$$

$|a - b| > 0,00001$, więc szukamy dalej.

$$a = 1,75$$

$$b \approx 1,71429$$

Krok 3

Obliczamy długości boków a i b :

$$a \approx \frac{(1,75 + 1,71429)}{2} \approx 1,73215$$

$$b \approx \frac{3}{1,73215} \approx 1,73195$$

$|a - b| > 0,00001$, więc przechodzimy do następnego kroku.

$$a \approx 1,73215$$

$$b \approx 1,73195$$

Krok 4

Obliczamy długości boków a i b :

$$a \approx \frac{(1,73215 + 1,73195)}{2} \approx 1,73205$$

$$b \approx \frac{3}{1,73205} \approx 1,73205$$

$|a - b| < 0,00001$, zatem algorytm kończy działanie.

$$a \approx 1,73205$$

$$b \approx 1,73205$$

Wynikiem działania algorytmu jest przybliżona wartość $\sqrt{3}$ równa 1,73205.

Warto wiedzieć

Inny ciąg kolejnych przybliżeń, zbieżny do poszukiwanego pierwiastka kwadratowego, tworzą kolejne wartości boku b prostokąta.

Warto wiedzieć

Metodę Newtona–Raphsona można wykorzystać także do znalezienia miejsca zerowego funkcji.

Warto wiedzieć

W podanym algorytmie wyznaczania pierwiastka kwadratowego metodą Newtona–Raphsona pierwszym wyrazem ciągu (wartością początkową zmiennej a) może być dowolna liczba dodatnia. Wówczas wartością początkową zmiennej b będzie c/a .

Wyznaczane w poszczególnych krokach długości boku a tworzą ciąg złożony z kolejnych przybliżeń $\sqrt{c}$, a granicą tego ciągu jest $\sqrt{c}$. Jeśli pierwszym wyrazem ciągu będzie 1 – długość boku a pierwszego prostokąta – to n -ty wyraz tego ciągu, gdzie n jest liczbą całkowitą dodatnią, można wyrazić wzorem rekurencyjnym:

$$a_n = \begin{cases} 1 & \text{dla } n = 1 \\ \frac{a_{n-1} + \frac{c}{a_{n-1}}}{2} & \text{dla } n > 1 \end{cases}$$

Oto zmodyfikowana specyfikacja problemu znajdowania przybliżonej wartości pierwiastka kwadratowego oraz zapisany w pseudokodzie algorytm znajdujący wartość pierwiastka kwadratowego metodą Newtona–Raphsona. Wartości zmiennych a i b w pseudokodzie odpowiadają długościom boków prostokąta w danym kroku.

Specyfikacja

Dane: c – liczba rzeczywista, $c > 0$,
eps – dokładność obliczeń.

Wynik: pkw – pierwiastek kwadratowy z liczby c wyliczony z dokładnością eps.

$a \leftarrow 1$

$b \leftarrow c$

dopóki $|b - a| > \text{eps}$ **wykonuj**

$a \leftarrow (a + b)/2$

$b \leftarrow c/a$

pkw $\leftarrow (a + b)/2$

Ćwiczenie 3

Napisz program znajdujący przybliżoną wartość pierwiastka kwadratowego z dodatniej liczby rzeczywistej metodą Newtona–Raphsona.

Zapamiętaj

Metoda Newtona–Raphsona służy do znajdowania pierwiastka kwadratowego z dodatniej liczby rzeczywistej c i polega na obliczeniu przybliżonej długości boku kwadratu o polu c z wykorzystaniem prostokątów o tym samym polu. Za wartości początkowe długości boków prostokąta można przyjąć dowolne liczby, których iloczyn jest równy c , np. 1 i c . Długości boków w każdym kolejnym kroku to średnia arytmetyczna dotychczasowych długości oraz iloraz wartości c i policzonej średniej. Obliczenia kończymy, gdy długości boków prostokąta różnią się o nie więcej niż dokładność obliczeń, a więc otrzymaliśmy w przybliżeniu kwadrat.

Porównanie metod bisekcji i Newtona–Raphsona

Zastanówmy się, za pomocą której z dwóch omówionych metod szybciej otrzymamy przybliżenie wartości pierwiastka kwadratowego. Łatwo zauważyć, że szybsza jest metoda Newtona–Raphsona, ponieważ w jednym kroku zmieniamy obydwa końce przedziału poszukiwań (długości boków prostokąta). W przypadku metody bisekcji zmienialiśmy w jednym kroku tylko jeden koniec. Ilustruje to dobrze implementacja obu algorytmów w arkuszu kalkulacyjnym (rys. 8.3).

Warto wiedzieć

Przybliżenie pierwiastka trzeciego stopnia z dodatniej liczby rzeczywistej c odpowiada długości krawędzi sześcianu o objętości c . Długość tej krawędzi można wyznaczyć, wykorzystując prostopadłościany o tej samej objętości. Za długości krawędzi wyjściowego prostopadłościanu można przyjąć np. liczby 1, 1 i c .

1	c = 15		eps = 0,001	
2	a	b	b - a	pkw
3	1	15	14,0000	8,0000
4	1	8	7,0000	4,5000
5	1	4,5	3,5000	2,7500
6	2,75	4,5	1,7500	3,6250
7	3,625	4,5	0,8750	4,0625
8	3,625	4,0625	0,4375	3,8438
9	3,84375	4,0625	0,2188	3,9531
10	3,84375	3,953125	0,1094	3,8984
11	3,84375	3,898438	0,0547	3,8711
12	3,871094	3,898438	0,0273	3,8848
13	3,871094	3,884766	0,0137	3,8779
14	3,871094	3,87793	0,0068	3,8745
15	3,871094	3,874512	0,0034	3,8728
16	3,872803	3,874512	0,0017	3,8737
17	3,872803	3,873657	0,0009	3,8732

1	c = 15		eps = 0,001	
2	a	b	b - a	pkw
3	1	15	14,0000	8,0000
4	8	1,875	6,1250	4,9375
5	4,9375	3,037975	1,8995	3,9877
6	3,987737	3,761532	0,2262	3,8746
7	3,874634	3,871333	0,0033	3,8730
8	3,872984	3,872983	0,0000	3,8730

Rys. 8.3. Implementacja w arkuszu kalkulacyjnym algorytmów znajdowania pierwiastka kwadratowego metodą bisekcji (z lewej) oraz metodą Newtona–Raphsona (z prawej)

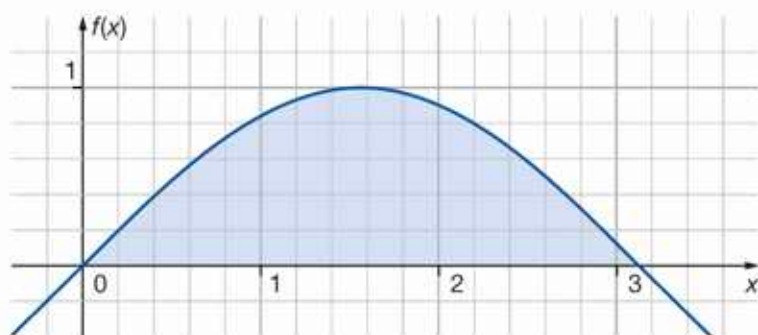
Ćwiczenie 4

Przygotuj w arkuszu kalkulacyjnym implementację algorytmów obliczania przybliżonej wartości pierwiastka kwadratowego:

- metodą bisekcji,
- metodą Newtona–Raphsona.

8.3. Obliczanie pól obszarów zamkniętych

Umiesz zapewne policzyć pola wybranych obszarów zamkniętych ograniczonych odcinkami, takich jak trójkąty, prostokąty czy trapezy. Jak policzyć pole obszaru zamkniętego ograniczonego wybraną krzywą i osią układu współrzędnych? Weźmy np. obszar ograniczony przez wykres funkcji $f(x) = \sin x$ i oś OX w przedziale od 0 do π . Na rysunku 8.4 na s. 138 obszar ten jest zaznaczony niebieskim kolorem.



Rys. 8.4. Obszar zamknięty ograniczony wykresem funkcji $f(x) = \sin x$ i osią OX w przedziale $[0; \pi]$

Ogólnie problem możemy sformułować następująco: chcemy obliczyć przybliżone pole obszaru zamkniętego ograniczonego prostymi $x = a$, $x = b$, osią OX oraz funkcją $f(x)$, która jest ciągła i ma nieujemne wartości dla argumentów z przedziału $[a; b]$. Niebieski obszar z rysunku 8.4 jest ograniczony przez proste $x = 0$, $x = \pi$, oś OX oraz funkcję $f(x) = \sin x$. Oto specyfikacja problemu:

Specyfikacja

Dane: a, b – liczby rzeczywiste, $a < b$,
 $f(x)$ – funkcja ciągła o nieujemnych wartościach dla argumentów z przedziału $[a; b]$.

Wynik: s – przybliżona wartość pola obszaru zamkniętego ograniczonego prostymi $x = a$, $x = b$ oraz funkcją $f(x)$ i osią OX .

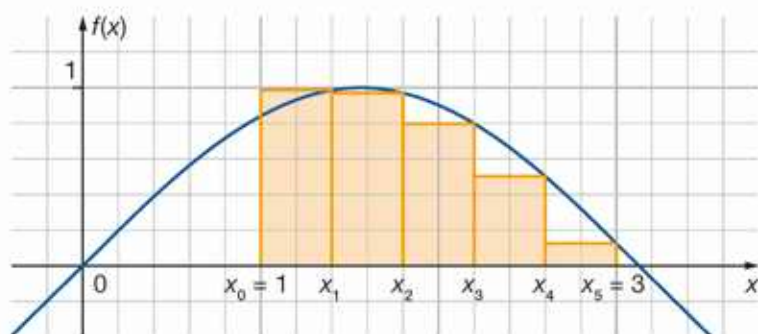
Metoda prostokątów

Jedną z metod jest podzielenie przedziału $[a; b]$ na n części tej samej długości, czyli ciąg przedziałów $[x_0, x_1]$, $[x_1, x_2]$, ..., $[x_{n-1}, x_n]$, $x_0 = a$ i $x_n = b$. Następnie oblicza się pola prostokątów o wierzchołkach w punktach o współrzędnych $(x_{i-1}, 0)$, $(x_i, 0)$, $(x_{i-1}, f(x_i))$, $(x_i, f(x_i))$, gdzie $i = 1, \dots, n$, i sumuje te pola. Otrzymamy w ten sposób przybliżoną wartość pola obszaru zamkniętego. Metodę tę nazywamy

Metoda prostokątów • **metodą prostokątów**. Rysunek 8.5 ilustruje ją dla funkcji $f(x) = \sin x$ oraz przedziału $[1; 3]$ (prostych $x = 1$ i $x = 3$) i dla $n = 5$. Prostokąty, których pola są sumowane, zaznaczono pomarańczowym kolorem.

Warto wiedzieć

W przypadku górnych wierzchołków prostokątów drugie współrzędne mogłyby być też równe $f(x_{i-1})$. Na pewno jednak muszą być takie same.



Rys. 8.5. Szukanie metodą prostokątów przybliżenia pola obszaru ograniczonego funkcją $f(x) = \sin x$ i osią OX w przedziale $[1; 3]$ dla $n = 5$

Niektóre prostokąty będą obejmować obszar z nadmiarem (wykraczać poza wykres funkcji), inne – obszar z niedomiarem. Im większa będzie wartość n , tym niedomiar lub nadmiar będzie mniejszy, a przybliżenie wartości pola dokładniejsze. Oto zapis algorytmu w pseudokodzie:

```
dx ← (b - a)/n
x ← a
s ← 0
dla i ← 1, 2, ..., n wykonuj
    x ← x + dx
    s ← s + dx * f(x)
```

Zmienna dx pamięta długość przedziału, czyli szerokość prostokąta. W zmiennej s zapiszemy sumę pól prostokątów. Pole pojedynczego prostokąta to iloczyn jego szerokości (dx) i wysokości (wartość funkcji dla argumentu x). Kod źródłowy funkcji `main` programu realizującego opisany algorytm jest następujący:

```
1. int main()
2. {
3.     float a, b, x, dx, s=0;
4.     int n;
5.     cout<<"a = "; cin>>a;
6.     cout<<"b = "; cin>>b;
7.     cout<<"n = "; cin>>n;
8.     dx=(b-a)/n;
9.     x=a;
10.    for (int i=1;i<=n;i++)
11.    {
12.        x=x+dx;
13.        s=s+dx*f(x);
14.    }
15.    cout<<"Pole obszaru: "<<s;
16.    return 0;
17. }
```

❶ Fragment kodu źródłowego programu obliczającego metodą prostokątów przybliżoną wartość pola obszaru ograniczonego funkcją $f(x)$ i osią OX w przedziale $[a; b]$ – funkcja `main`

Rysunek 8.6 przedstawia efekty wykonania programu obliczającego metodą prostokątów pole obszaru ograniczonego przez funkcję $f(x) = \sin x$ oraz oś OX w przedziale $[0; \pi]$, odpowiednio dla $n = 10$ oraz $n = 100$. Dokładna wartość pola tego obszaru wynosi 2. Widać, że dla $n = 100$ otrzymaliśmy dokładniejsze przybliżenie.

```
a = 0
b = 3.14
n = 10
Pole obszaru: 1.98379
```

```
a = 0
b = 3.14
n = 100
Pole obszaru: 1.99986
```

👍 Dobra rada

Jeśli chcesz obliczyć metodą prostokątów pole obszaru znajdującego się pod osią OX , wystarczy, że dla wartości funkcji zastosujesz wartość bezwzględną.

Rys. 8.6. Przybliżenia pola obszaru ograniczonego funkcją $f(x) = \sin x$ i osią OX w przedziale $[0; \pi]$ dla $n = 10$ i $n = 100$ wyliczone metodą prostokątów

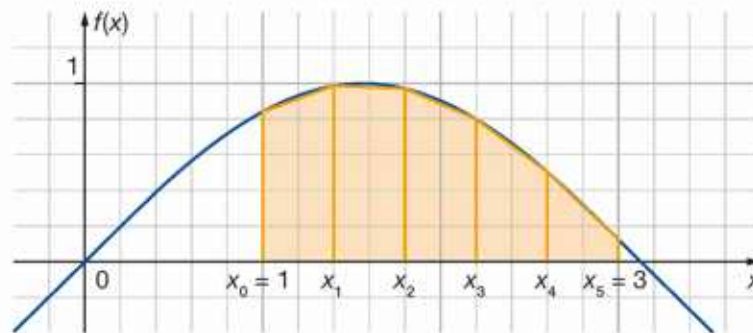
Ćwiczenie 5

Napisz program obliczający metodą prostokątów przybliżoną wartość pola obszaru zamkniętego, zgodnie ze specyfikacją na s. 138.

Metoda trapezów

Metoda trapezów •
Metoda prostokątów,
s. 138

Podczas obliczania przybliżonej wartości pola obszaru zamkniętego **metodą trapezów** postępuje się bardzo podobnie jak w wypadku **metody prostokątów**. Zamiast prostokątów do przybliżania pola używa się trapezów. Ilustruje to rysunek 8.7.



Rys. 8.7. Szukanie metodą trapezów przybliżenia pola obszaru ograniczonego funkcją $f(x) = \sin x$ i osią OX w przedziale $[1; 3]$ dla $n = 5$

Pole pojedynczego trapezu możemy obliczyć, przyjmując za wysokość poziomy odcinek pomiędzy punktami x_{i-1} i x_i o długości dx , a za podstawy – odcinki o długościach $f(x_{i-1})$ i $f(x_i)$, łączące odpowiednio punkty o współrzędnych $(x_{i-1}, 0)$ z $(x_{i-1}, f(x_{i-1}))$ oraz $(x_i, 0)$ z $(x_i, f(x_i))$, gdzie $i = 1, \dots, n$. Pole pojedynczego trapezu wyraża się więc wzorem:

$$P = \frac{1}{2} \cdot (f(x_{i-1}) + f(x_i)) \cdot (x_i - x_{i-1}) = \frac{1}{2} \cdot (f(x_{i-1}) + f(x_i)) \cdot dx$$

Oto zapis algorytmu w pseudokodzie:

```
dx ← (b - a)/n
x ← a
s ← 0
f1 ← f(x)
dla i ← 1, 2, ..., n wykonuj
    x ← x + dx
    f2 ← f(x)
    s ← s + dx * (f1 + f2)/2
    f1 ← f2
```

Wprowadziliśmy dwie dodatkowe zmienne $f1$ i $f2$, w których są pamiętane wartości funkcji dla końców rozpatrywanego przedziału $[x_{i-1}; x_i]$, gdzie $i = 1, \dots, n$.

Fragment kodu źródłowego funkcji `main` programu, który realizuje algorytm obliczania przybliżonej wartości pola obszaru zamkniętego metodą trapezów, jest następujący.

```

1.     dx=(b-a)/n;
2.     x=a;
3.     f1=f(x);
4.     for (int i=1;i<=n;i++)
5.     {
6.         x=x+dx;
7.         f2=f(x);
8.         s=s+dx*(f1+f2)/2;
9.         f1=f2;
10.    }

```

Fragment kodu źródłowego funkcji `main` programu, który oblicza metodą trapezów przybliżoną wartość obszaru ograniczonego funkcją $f(x)$ i osią OX w przedziale $[a; b]$

Ćwiczenie 6

Napisz program obliczający metodą trapezów przybliżoną wartość pola obszaru zamkniętego, zgodnie ze specyfikacją na s. 138.

Rysunek 8.8 przedstawia porównanie metody prostokątów i metody trapezów na przykładzie funkcji $f(x) = \sin x$ w przedziale $[0; \frac{\pi}{2}]$ i $n = 10$. W tym przedziale funkcja jest rosnąca, więc wszystkie prostokąty zawyżają wartość pola. Metoda trapezów w tym przypadku daje zdecydowanie lepsze przybliżenie.

```

a = 0
b = 1.57
n = 10
Pole obszaru: 1.07565

```

```

a = 0
b = 1.57
n = 10
Pole obszaru: 0.99715

```

Rys. 8.8. Przybliżenie pola obszaru zamkniętego ograniczonego funkcją $f(x) = \sin x$ i osią OX w przedziale $[0; \frac{\pi}{2}]$ dla $n = 10$ wyliczone metodą prostokątów (z lewej) i metodą trapezów (z prawej)

Zapamiętaj

Metoda prostokątów, służąca do obliczania pola obszarów zamkniętych ograniczonych wykresem funkcji w określonym przedziale, polega na wyznaczaniu przybliżonej wartości tego pola za pomocą sumy pól odpowiednio dobranych prostokątów. Natomiast metoda trapezów korzysta w podobny sposób z odpowiednio dobranych trapezów.

8.4. Znajdowanie przybliżenia liczby π

Do poszukiwania przybliżonej wartości liczby π można wykorzystać **metodę prostokątów** lub **metodę trapezów**. Można też zastosować metodę Monte Carlo.

Metoda prostokątów,
s. 138 [🔗](#)

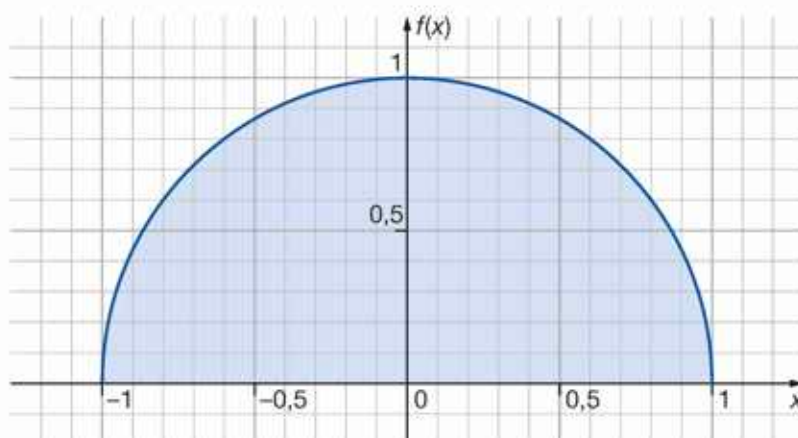
Metoda trapezów,
s. 140 [🔗](#)

Dobra rada

Liczba π to iloraz długości okręgu do długości jego średnicy.

Przybliżenie liczby π – metoda prostokątów i metoda trapezów

Równanie okręgu o środku w punkcie $(0, 0)$ i promieniu r to: $x^2 + y^2 = r^2$. Przyjmijmy, że $r = 1$. Zauważ, że pole obszaru ograniczonego okręgiem o równaniu $x^2 + y^2 = 1$ jest równe liczbie π . Aby obliczyć to pole, najpierw trzeba policzyć pole obszaru ograniczonego krzywą $f(x) = \sqrt{1 - x^2}$ w przedziale $[-1; 1]$ i znajdującego się nad osią OX (rys. 8.9). Następnie otrzymany wynik należy pomnożyć przez 2.



Rys. 8.9. Obszar ograniczony funkcją $f(x) = \sqrt{1 - x^2}$ w przedziale $[-1; 1]$ znajdujący się nad osią OX

Ćwiczenie 7

Napisz program obliczający przybliżoną wartość liczby π . Wykorzystaj:

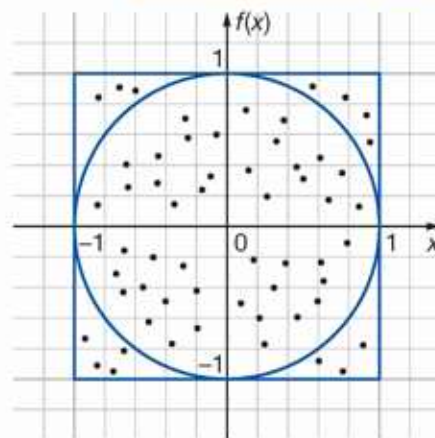
- metodę prostokątów,
- metodę trapezów.

Przybliżenie liczby π – metoda Monte Carlo

Czy wykorzystując liczby losowe, można obliczać przybliżenia różnych wartości lub symulować pewne zjawiska? Okazuje się, że tak. Metoda

Metoda Monte Carlo używająca w tym celu liczb losowych nosi nazwę **metody Monte Carlo**.

Przeprowadźmy następujące doświadczenie. Wylosujmy n punktów w obrębie kwadratu o środku w początku układu współrzędnych i długości boku $a = 2$ (a więc punktów o współrzędnych z zakresu od -1 do 1). Policzymy, ile z wylosowanych punktów znajdzie się w kole o środku w punkcie $(0, 0)$ i promieniu $r = 1$, czyli w kole wpisanym w kwadrat. Liczbę punktów w kole oznaczmy przez n_0 . Przykładowy wynik doświadczenia ilustruje rysunek 8.10.



Rys. 8.10. Przykładowy wynik losowania punktów w obrębie kwadratu o środku w punkcie $(0, 0)$ i długości boku 2

Dobra rada

Do wylosowania współrzędnych punktów możesz wykorzystać arkusz kalkulacyjny.

Iloraz pola koła do pola kwadratu jest w przybliżeniu równy ilorazowi liczby punktów w kole do liczby wszystkich punktów:

$$\frac{\pi \cdot r^2}{a^2} \approx \frac{n_0}{n}$$

gdzie r – promień koła, a – długość boku kwadratu, n – liczba wszystkich punktów w kwadracie, n_0 – liczba punktów w kole.

W opisanym doświadczeniu $a = 2r$, czyli $a^2 = 4r^2$. Otrzymujemy:

$$\frac{\pi \cdot r^2}{4 \cdot r^2} = \frac{\pi}{4} \approx \frac{n_0}{n}, \text{ zatem } \pi \approx \frac{4 \cdot n_0}{n}$$

Daną, która wpływa na dokładność przybliżenia liczby π , jest liczba punktów losowanych w obrębie kwadratu. Im będzie ona większa, tym dokładniejsze przybliżenie otrzymamy. Oto specyfikacja problemu:

Specyfikacja

Dane: n – liczba całkowita dodatnia określająca liczbę losowań punktów o współrzędnych (x, y) , gdzie x i y są liczbami rzeczywistymi z przedziału $[-1; 1]$.

Wynik: π – przybliżona wartość liczby π .

Oto zapis algorytmu znajdującego przybliżenie liczby π metodą Monte Carlo (n_0 odpowiada liczbie punktów w kole) oraz kod źródłowy funkcji `main` realizującej ten algorytm:

```
n0 ← 0
dla i ← 0, 1, ..., n - 1 wykonuj
    x ← losowa liczba rzeczywista z przedziału [-1; 1]
    y ← losowa liczba rzeczywista z przedziału [-1; 1]
    jeśli x * x + y * y ≤ 1 to n0 ← n0 + 1
pi ← 4 * n0/n
```

```
1. int main()
2. {
3.     int n, n0=0;
4.     float x, y;
5.     cout<<"Liczba punktow: "; cin>>n;
6.     srand(time(NULL));
7.     for (int i=0;i<n;i++)
8.     {
9.         x=-1+2*(float(rand())/RAND_MAX);
10.        y=-1+2*(float(rand())/RAND_MAX);
11.        if (x*x+y*y<=1) n0++;
12.    }
13.    cout<<"Przyblizenie pi: "<<4*(float(n0)/n);
14.    return 0;
15. }
```

Warto wiedzieć

Metodę Monte Carlo można zastosować do obliczania pola obszarów zamkniętych.

● **Fragment kodu** źródłowego programu znajdującego przybliżoną wartość liczby π metodą Monte Carlo – funkcja `main`

Dobra rada

Pamiętaj, że korzystanie z funkcji `srand` i `rand` wymaga dołączenia biblioteki `cstdlib`, a korzystanie z funkcji `time` – biblioteki `ctime`.

Warto wiedzieć

Liczby losowane przez komputer nazywa się często pseudolosowymi. Są bowiem generowane według ustalonych algorytmów, czyli tak naprawdę nie są losowane.

Stała `RAND_MAX` •

Warto wiedzieć

Standard języka C++ nie określa dokładnej wartości stałej `RAND_MAX`, gwarantuje jednak, że nie jest mniejsza niż $2^{15} - 1 = 32\,767$. Wartość stałej `RAND_MAX` zależy od kompilatora i bibliotek.

Dobra rada

Dokładniejsze przybliżenie liczby π uzyskasz, gdy wielokrotnie wywołasz program i policzysz średnią arytmetyczną otrzymanych wyników.

Instrukcja w linii 6 inicjuje generator liczb losowych i uzależnia losowanie liczb od czasu wskazywanego przez zegar komputera (dzięki temu po każdym uruchomieniu programu otrzymamy różne wyniki).

Pętla w liniach 7–12 losuje n razy współrzędne punktu (linie 9–10) i bada, czy wylosowany punkt należy do koła o promieniu 1 oraz środka w początku układu współrzędnych (linia 11). Jeśli tak, powiększamy wartość zmiennej `n0`. Funkcja `rand` losuje liczbę całkowitą z zakresu od 0 do `RAND_MAX`. Predefiniowana stała `RAND_MAX` określa maksymalną wartość, jaką może zwrócić funkcja `rand`. Po konwersji wylosowanej liczby całkowitej na rzeczywistą (instrukcja `float(rand())`) i podzieleniu jej przez stałą `RAND_MAX` otrzymujemy liczbę rzeczywistą z przedziału $[0; 1]$. Po pomnożeniu przez 2 i odjęciu 1 otrzymujemy liczbę z przedziału $[-1; 1]$. W linii 13 wypisywana jest przybliżona wartość liczby π obliczona ze wzoru podanego na s. 143. Ponieważ dzielimy dwie wartości całkowite, należy dokonać konwersji na typ rzeczywisty.

Ćwiczenie 8

Napisz program obliczający przybliżoną wartość liczby π metodą Monte Carlo.

Rysunek 8.11 przedstawia przykład wykonania programu obliczającego przybliżoną wartość liczby π metodą Monte Carlo dla $n = 1\,000\,000$. W tym przypadku otrzymaliśmy dokładność jedynie do trzech miejsc części ułamkowej. Dla tej samej wartości n możemy otrzymać różne przybliżenia liczby π .

Liczba punktów: 1000000
Przybliżenie pi: 3.14162

Rys. 8.11. Przykład wykonania programu obliczającego przybliżoną wartość liczby π metodą Monte Carlo dla 1 000 000 wylosowanych punktów

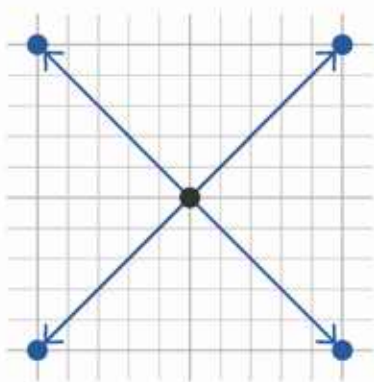
A to ciekawe**Przybliżenia liczby π**

Już w starożytności odkryto, że stosunek obwodu koła do jego średnicy jest pewną stałą. Obecnie znamy ją jako liczbę π i wiemy, że jest to liczba niewymierna. Od tysiącleci stosuje się różne jej przybliżenia, np. Archimedes używał do tego ułamka $\frac{22}{7}$. Dzięki rozwojowi komputerów udaje się wyznaczać kolejne cyfry rozwinięcia dziesiętnego liczby π . W 1949 r. wyliczono 2038 cyfr po przecinku, a w 2020 r. – 50 bilionów. Często dla uproszczenia obliczeń przyjmuje się $\pi \approx 3,14$. Może to przypadek, ale stosunek sumy długości dwóch dowolnych boków podstawy piramidy Cheopsa do wysokości piramidy wynosi około 3,14.

8.5. Symulacja ruchów Browna

Ruchy Browna to chaotyczny ruch cząstek substancji stałej zawieszonych w cieczy lub gazie. Ponieważ są to ruchy przypadkowe, do ich symulacji można wykorzystać liczby losowe i **metodę Monte Carlo**.

Przyjmijmy model, w którym cząstka przemieszcza się na płaszczyźnie w jednym z czterech kierunków. W jednostce czasu zmienia swoje położenie o jednostkową odległość wzdłuż każdej z osi (w lewo lub prawo wzdłuż osi OX oraz do góry lub do dołu wzdłuż osi OY). Możliwe przesunięcia cząstki pokazuje rysunek 8.12. Czarnym kolorem zaznaczona jest aktualna pozycja cząstki, a niebieskim – jej możliwe położenia po wykonaniu ruchu w jednostce czasu.



Rys. 8.12. Możliwe położenia cząstki po wykonaniu ruchu w jednostce czasu zgodnie z przedstawionym modelem

Specyfikację symulacji ruchów Browna możemy zapisać w następujący sposób:

Specyfikacja

Dane: n – liczba całkowita dodatnia, liczba ruchów w symulacji odpowiadająca liczbie jednostek czasu.

Wynik: x, y – położenie cząstki po n ruchach.

Algorytm w pseudokodzie możemy zapisać następująco. Przyjmujemy, że punktem początkowym cząstki jest punkt $(0, 0)$.

```

 $x \leftarrow 0$ 
 $y \leftarrow 0$ 
dla  $i \leftarrow 1, 2, \dots, n$  wykonuj
     $dx \leftarrow$  liczba losowa  $-1$  lub  $1$ 
     $dy \leftarrow$  liczba losowa  $-1$  lub  $1$ 
     $x \leftarrow x + dx$ 
     $y \leftarrow y + dy$ 

```

Symulację zgodnie z przedstawionym modelem można łatwo przeprowadzić w arkuszu kalkulacyjnym i zobrazować wykresem punktowym. Rysunek 8.13 na s. 146 przedstawia przykład takiej symulacji.

Ruchy Browna

Metoda Monte Carlo,
s. 142

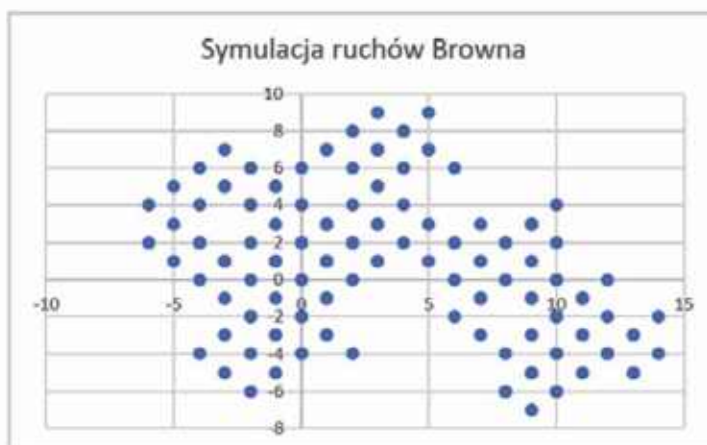
Warto wiedzieć

Nazwa „ruchy Browna” pochodzi od nazwiska szkockiego biologa Johna Browna. W 1827 r. obserwował on przez mikroskop przemieszczanie się cząstek pyłku kwiatowego w wodzie i zauważył, że są one w bezustannym, chaotycznym ruchu.

Warto wiedzieć

Zjawisko ruchów Browna badali niezależnie Albert Einstein oraz polski fizyk Marian Smoluchowski. Na początku XX w. opisali oni matematyczny model zjawiska oraz podali przyczynę ruchów – odbijanie się cząstek substancji stałej od cząsteczek cieczy lub gazu.

1	x	y	Zmiana lewo/prawo	Zmiana górze/dół
2	0	0	1	-1
3	1	-1	1	1
4	2	0	-1	1
5	1	1	-1	1
6	0	2	-1	1
7	-1	3	-1	-1
8	-2	2	1	-1
9	-1	1	1	-1
10	0	0	-1	1
197	1	3	-1	1
198	0	4	1	-1
199	1	3	-1	-1
200	0	2	-1	-1
201	-1	1		



Rys. 8.13. Przykład symulacji ruchów Browna dla 200 ruchów cząstki (zaznaczone punkty oznaczają miejsca, w których znalazła się cząstka)

Dobra rada

Jeśli przy otwartym arkuszu kalkulacyjnym naciśniesz klawisz F9, komputer przeliczy ponownie wszystkie zapisane w nim formuły. W arkuszu symulującym ruchy Browna za pomocą tego klawisza ponownie wylosujesz liczby.

Ćwiczenie 9

Przygotuj w arkuszu kalkulacyjnym symulację ruchów Browna zgodnie z modelem omówionym w tym temacie. Przedstaw ruch cząstki na wykresie punktowym. Przeprowadź symulację wielokrotnie.

Zapamiętaj

Metoda Monte Carlo wykorzystuje liczby losowe do modelowania różnych zjawisk oraz obliczania przybliżeń pewnych wartości. Metodą tą można np. obliczyć przybliżoną wartość liczby π oraz symulować ruchy Browna.

A to ciekawe

Skąd pochodzi nazwa metody Monte Carlo?

Za jednego z twórców metody Monte Carlo uważa się polskiego matematyka Stanisława Ulama (1909–1984). Nadał on tej metodzie nazwę, która prawdopodobnie pochodzi od nazwy dzielnicy Monako – Monte Carlo. Niektórym kojarzy się ona z kasynami, gdzie wielu powierza swoje szczęście losowi. Stanisław Ulam w książce *Przygody matematyka* tak wspominał powstanie metody: „Zauważyłem, że znacznie praktyczniejszym sposobem oceniania prawdopodobieństwa ułożenia pasjansa jest wykładanie kart, czyli eksperymentowanie z tym procesem i po prostu zapisywanie procentu wygranych, niż próba obliczenia wszystkich możliwości kombinatorycznych, których liczba rośnie wykładniczo i jest tak wielka, że pominiawszy najprostsze przypadki, jej oszacowanie jest niemożliwe”.

Podsumowanie

- Metoda bisekcji (połowienia) polega na zmniejszaniu zakresu przeszukiwanych danych o połowę do znalezienia rozwiązania lub do momentu, gdy długość przedziału poszukiwań jest nie większa niż dokładność obliczeń (wówczas wynikiem jest ostatnio wyznaczony środek przedziału).
- Metodę bisekcji można zastosować do znajdowania miejsca zerowego funkcji $f(x)$ w przedziale $[a; b]$. Aby można było to zrobić, funkcja musi być określona i ciągła w przedziale $[a; b]$ oraz $f(a) \cdot f(b) < 0$.
- Metodę bisekcji można zastosować także do poszukiwania pierwiastka kwadratowego z liczby rzeczywistej c większej od 1. Problem ten sprowadza się do znalezienia miejsca zerowego funkcji $f(x) = x^2 - c$ w przedziale $[1; c]$.
- Metoda Newtona–Raphsona polega na szukaniu przybliżonej długości boku kwadratu o polu x za pomocą prostokątów o tym samym polu.
- Metodą Newtona–Raphsona można znaleźć przybliżoną wartość pierwiastka kwadratowego z dodatniej liczby rzeczywistej x . Za wartości początkowe przedziału poszukiwań można przyjąć 1 i x . Przedział zawężamy, zamieniając jedną z wartości na średnią arytmetyczną końców przedziału z poprzedniego kroku, a drugą – na iloraz wartości x i policzonej średniej.
- Metoda prostokątów służy do obliczania pola obszarów zamkniętych ograniczonych wykresem funkcji i osią układu współrzędnych w określonym przedziale. Polega na znajdowaniu przybliżonej wartości tego pola jako sumy pól odpowiednio dobranych prostokątów.
- Metoda trapezów wykorzystywana jest do obliczania pola obszarów zamkniętych ograniczonych wykresem funkcji i osią układu współrzędnych w określonym przedziale. Polega na znajdowaniu przybliżonej wartości tego pola jako sumy pól odpowiednio dobranych trapezów.
- Metoda Monte Carlo wykorzystuje liczby losowe do modelowania różnych zjawisk oraz wyznaczania niektórych przybliżonych wartości.
- Metodą Monte Carlo można obliczyć przybliżoną wartość liczby π oraz symulować ruchy Browna.

Zadania

- ★ **1** Wykorzystując program znajdujący miejsce zerowe funkcji metodą bisekcji omówiony w tym temacie, wyznacz kilka miejsc zerowych funkcji $f(x) = x \cdot \sin x$ w przedziale $[-7; 7]$. Znajdź w tym przedziale inne miejsce zerowe, którego metodą bisekcji nie da się wyznaczyć.
- ★ **2** Napisz program, który wyznaczy metodą bisekcji pierwiastek kwadratowy z dodatniej liczby rzeczywistej $0 < c < 1$.
- ★ **3** Omówione w temacie programy obliczające pola obszarów zamkniętych metodą prostokątów i metodą trapezów zmodyfikuj tak, aby obliczały prawidłowo pola także dla ujemnych wartości funkcji ograniczającej obszar.

- ★ **4** Oblicz metodą trapezów pole obszaru zamkniętego ograniczonego osią OX oraz funkcją $f(x) = x \cdot \sin x$ w przedziale $[0; \pi]$.
- ★ **5** Napisz program znajdujący przybliżenie pierwiastka kwadratowego metodą Newtona–Raphsona z wykorzystaniem funkcji rekurencyjnej.
- ★★ **6** Przygotuj arkusz kalkulacyjny umożliwiający poszukiwanie miejsc zerowych funkcji $f(x) = x^3 - 2x^2 - 5x + 6$ metodą bisekcji.
- ★★ **7** Napisz program znajdujący jedno miejsce zerowe wielomianu stopnia n . Stopień wielomianu, jego współczynniki oraz przedział, w którym będziesz szukać miejsca zerowego, wczytaj z klawiatury. Do obliczania wartości wielomianu wykorzystaj schemat Hornera.
- ★★ **8** Napisz program znajdujący pierwiastek trzeciego stopnia z dodatniej liczby rzeczywistej.
- ★★ **9** Oblicz metodą trapezów pole obszaru zamkniętego ograniczonego osią OX oraz funkcją $f(x) = x \cdot \sin x$ w przedziale $[-2\pi; 2\pi]$.
- ★★★ **10** Przygotuj symulację ruchów Browna w arkuszu kalkulacyjnym. Przyjmij model, w którym cząstka pokonuje w jednostce czasu odległość równą 1 w dowolnym kierunku. Zilustruj symulację wykresem punktowym.
- ★★★ **11** Oblicz pole obszaru zamkniętego ograniczonego osią OX oraz funkcją $f(x) = -4x^2 + 4x$ w przedziale $[0; 1]$. Zastosuj metodę Monte Carlo.
- ★★★ **12** Napisz program znajdujący pierwiastek n -tego stopnia z dodatniej liczby rzeczywistej, gdzie $n \geq 2$.

9. Algorytmy badające własności geometryczne

Podczas projektowania gier komputerowych czy programów użytkowych czasami trzeba określić wzajemne położenie obiektów. Wykorzystuje się w tym celu pojęcia i zależności matematyczne. W tym temacie poznasz algorytmy badające własności geometryczne obiektów na płaszczyźnie.

Cele lekcji

- Zbadasz położenie punktów względem prostej i odcinka.
- Sprawdzisz, czy dwa odcinki się przecinają.
- Zbadasz przynależność punktu do wielokąta wypukłego.

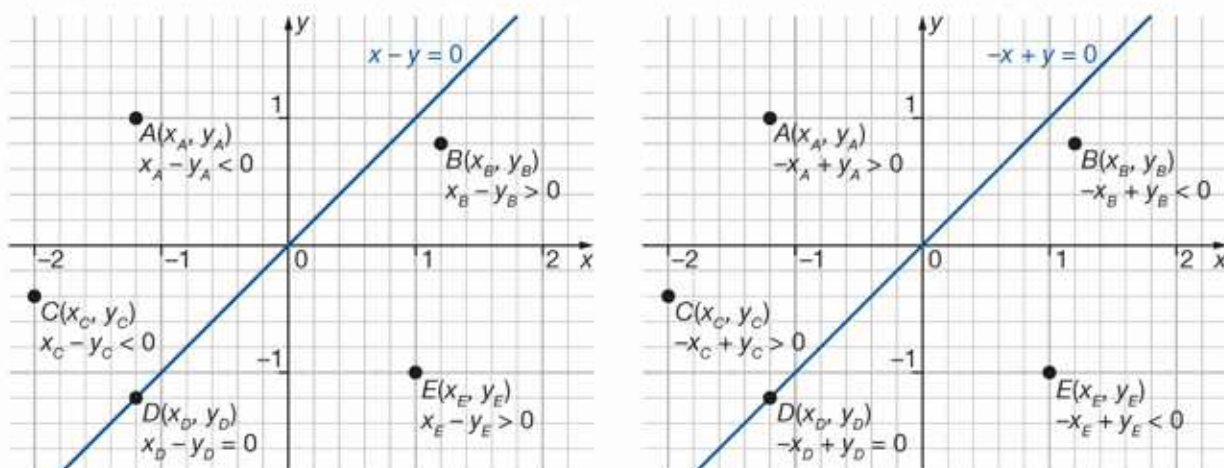
9.1. Położenie punktów względem prostej i odcinka

Do badania położenia punktów wykorzystamy m.in. zagadnienia omawiane na lekcjach matematyki.

Położenie punktów względem prostej

Zapewne znasz **równanie ogólne prostej**: $Ax + By + C = 0$, gdzie $A \neq 0$ lub $B \neq 0$. Na podstawie tego równania można łatwo określić **położenie punktu względem prostej**. Wszystkie punkty $P(x_p, y_p)$ spełniające nierówność $Ax_p + By_p + C > 0$ leżą po jednej stronie prostej, a punkty spełniające nierówność $Ax_p + By_p + C < 0$ – po drugiej stronie prostej. Jeśli $Ax_p + By_p + C = 0$, to punkt $P(x_p, y_p)$ leży na prostej.

Prostą można opisać różnymi równaniami ogólnymi. Rysunek 9.1 przedstawia przykładową prostą i położenie kilku punktów względem niej oraz informacje o wynikach otrzymanych po podstawieniu współrzędnych punktów do lewej strony równań ogólnych.



Rys. 9.1. Położenie punktów względem prostej $y = x$ (w postaci ogólnej np. $x - y = 0$ lub $-x + y = 0$)

Warto wiedzieć

Za pomocą równania kierunkowego ($y = ax + b$) nie można opisać prostej prostopadłej do osi OX . Natomiast w postaci ogólnej można zapisać równanie każdej prostej.

Warto wiedzieć

Jedną prostą można opisać nieskończenie wieloma równaniami ogólnymi – aby otrzymywać kolejne równania, wystarczy mnożyć obie strony równania przez liczby różne od zera. Natomiast każda prosta nieprostopadła do osi OX jest opisana dokładnie jednym równaniem kierunkowym.

Znak wartości otrzymanej po podstawieniu współrzędnych punktu do lewej strony równania ogólnego zależy od postaci równania. Dla punktów leżących po tej samej stronie prostej w przypadku jednego równania otrzymamy wartości dodatnie, a w przypadku innego ujemne.

Zbadamy, czy dwa punkty leżą po tej samej stronie prostej. Specyfikacja problemu jest następująca:

Specyfikacja

Dane: A, B, C – liczby rzeczywiste, współczynniki równania ogólnego prostej, $A \neq 0$ lub $B \neq 0$,

$xP1, yP1, xP2, yP2$ – liczby rzeczywiste, współrzędne punktów P_1 i P_2 .

Wynik: wartość logiczna **prawda**, gdy punkty P_1 i P_2 leżą po tej samej stronie prostej, lub **fałsz** – w przeciwnym przypadku.

Oto zapis algorytmu w pseudokodzie w postaci funkcji:

```
funkcja PktPoTejSamStr(A,B,C,xP1,yP1,xP2,yP2)
    jeśli  $(A*xP1+B*yP1+C)*(A*xP2+B*yP2+C) > 0$  to
        zwróć prawda i zakończ
    w przeciwnym przypadku
        zwróć fałsz i zakończ
```

Jeśli znaki wyrażeń $A*xP1+B*yP1+C$ oraz $A*xP2+B*yP2+C$ są takie same, czyli ich iloczyn jest liczbą dodatnią, to punkty leżą po tej samej stronie prostej. Zwróć uwagę, że jeśli któryś z punktów leży na prostej, czyli wartość wyrażenia dla tego punktu jest równa 0, to punkty nie leżą po tej samej stronie i wartością funkcji jest **fałsz**.

Przy implementacji powyższego algorytmu współrzędne punktu można pamiętać w **strukturze**. Oto jej definicja:

```
struct punkt
{
    float x, y;
};
```

W programie danymi będą współrzędne dwóch punktów. Dlatego warto zdefiniować pomocniczą funkcję `CzytajPkt`, którą wykorzystamy dwukrotnie: najpierw do wczytania współrzędnych pierwszego punktu, a potem do wczytania współrzędnych drugiego punktu. Kod źródłowy tej funkcji może być następujący:

Struktura, podręcznik
Informatyka na czasie 2.
Zakres rozszerzony,
s. 109 

Kod źródłowy 
funkcji wczytującej
do programu
współrzędne punktu

```
1. void CzytajPkt(string info, punkt &P)
2. {
3.     cout<<info<<endl;
4.     cout<<"x = "; cin>>P.x;
5.     cout<<"y = "; cin>>P.y;
6. }
```

Funkcja `CzytajPkt` ma dwa parametry: info typu `string` – napis wyświetlany na ekranie z informacją o tym, którego punktu współrzędne wczytujemy, oraz parametr `P` typu punkt przekazywany przez referencję – współrzędne wczytywanego punktu.

Oto kod źródłowy funkcji `PktPoTejSamStr`, sprawdzającej, czy dwa dane punkty leżą po tej samej stronie prostej:

```
1. bool PktPoTejSamStr(float A, float B, float C,
2.                    punkt P1, punkt P2)
3. {
4.     return ((A*P1.x+B*P1.y+C)*(A*P2.x+B*P2.y+C)>0);
5. }
```

❶ Kod źródłowy funkcji sprawdzającej, czy dwa punkty leżą po tej samej stronie prostej – z użyciem równania ogólnego prostej

Zwróć uwagę na linię 4. Nie trzeba używać instrukcji warunkowej – wartością funkcji jest wartość logiczna wyrażenia.

Kod źródłowy funkcji `main` programu badającego, czy dwa punkty leżą po tej samej stronie prostej, może być następujący:

```
1. int main()
2. {
3.     float A, B, C;
4.     punkt P1, P2;
5.     cout<<"Podaj wartosci A, B i C ";
6.     cout<<"rownania ogolnego prostej. "<<endl;
7.     cout<<"A = "; cin>>A;
8.     cout<<"B = "; cin>>B;
9.     cout<<"C = "; cin>>C;
10.    CzytajPkt("Podaj wspolrzedne punktu P1. ",P1);
11.    CzytajPkt("Podaj wspolrzedne punktu P2. ",P2);
12.    if (PktPoTejSamStr(A,B,C,P1,P2))
13.        cout<<"Punkty leza po tej samej stronie prostej.";
14.    else
15.        cout<<"Punkty nie leza po tej samej stronie prostej.";
16.    return 0;
17. }
```

❷ Fragment kodu źródłowego programu badającego, czy dwa punkty leżą po tej samej stronie prostej – funkcja `main`

W liniach 7–9 wczytujemy dane prostej, a w liniach 10–11 za pomocą funkcji `CzytajPkt` – współrzędne punktów P_1 i P_2 . Instrukcja warunkowa w liniach 12–15 wypisuje odpowiedni komentarz w zależności od wartości funkcji `PktPoTejSamStr`.

Ćwiczenie 1

Napisz program sprawdzający, czy dwa punkty leżą po tej samej stronie prostej, zgodnie ze specyfikacją podaną na s. 150. Wykorzystaj omówione w temacie funkcje `CzytajPkt` oraz `PktPoTejSamStr`.

Zapamiętaj

Do badania położenia punktu względem prostej można wykorzystać równanie ogólne prostej: $Ax + By + C = 0$, gdzie $A \neq 0$ lub $B \neq 0$. Wartość wyrażenia $Ax + By + C$ dla punktów (x, y) znajdujących się po tej samej stronie prostej ma taki sam znak. Jeśli punkt (x, y) leży na prostej, to wartość wyrażenia $Ax + By + C$ jest równa 0.

Przynależność punktu do odcinka

Odcinek to fragment prostej ograniczony dwoma punktami leżącymi na tej prostej. Jeśli dany punkt należy do odcinka, to należy też do prostej przechodzącej przez końce tego odcinka.

Równanie prostej przechodzącej przez dwa punkty

Równanie prostej przechodzącej przez dwa punkty $A(x_A, y_A)$ oraz $B(x_B, y_B)$ ma postać:

$$(y - y_A)(x_B - x_A) - (y_B - y_A)(x - x_A) = 0$$

Do opisu prostej przechodzącej przez dwa punkty wykorzystuje się też **macierz**. Można ją zilustrować jako tablicę przechowującą określone elementy, np. liczby. W informatyce zwykle stosuje się macierze odpowiadające tablicom dwuwymiarowym. Rysunek 9.2 przedstawia przykład macierzy o trzech wierszach i trzech kolumnach. Liczba kolumn i liczba wierszy w macierzy może być różna.

Macierz

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Rys. 9.2. Przykład macierzy o trzech wierszach i trzech kolumnach, przechowującej liczby całkowite

Jeśli macierz ma taką samą liczbę wierszy i kolumn, to nazywa się ją **macierzą kwadratową**. Takiej macierzy można przyporządkować **Wyznacznik macierzy** liczbę nazywaną **wyznacznikiem macierzy**. Oznacza się go np. skrótem $\det$ (od ang. *determinant* – wyznacznik). Wyznacznik macierzy z rysunku 9.2 zapiszemy tak jak na rysunku 9.3.

Innym sposobem oznaczania wyznacznika jest umieszczenie elementów macierzy między pionowymi kreskami (rys. 9.4).

Warto wiedzieć

Macierze są wykorzystywane w grafice komputerowej. Za ich pomocą można opisać m.in. przekształcenia obiektów na płaszczyźnie, a także w przestrzeni trójwymiarowej, np. obroty.

$$\det \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Rys. 9.3. Oznaczenie wyznacznika macierzy z rysunku 9.2 – sposób 1

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix}$$

Rys. 9.4. Oznaczenie wyznacznika macierzy z rysunku 9.2 – sposób 2

W tym temacie będziemy wykorzystywać macierze kwadratowe o trzech wierszach i trzech kolumnach. Dla macierzy o takiej liczbie wierszy i kolumn wyznacznik najwygodniej policzyć, korzystając z **reguły Sarrusa**. Przy obliczaniu wyznacznika wystarczy przepisać pod spodem dwa pierwsze wiersze. Iloczyn elementów znajdujących się na rysunku 9.5 na pomarańczowych liniach zapisujemy ze znakiem plus, a iloczyn elementów na niebieskich liniach – ze znakiem minus.

Reguła Sarrusa

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} = 1 \cdot 5 \cdot 9 + 4 \cdot 8 \cdot 3 + 7 \cdot 2 \cdot 6 - 7 \cdot 5 \cdot 3 - 1 \cdot 8 \cdot 6 - 4 \cdot 2 \cdot 9 = 45 + 96 + 84 - 105 - 48 - 72 = 0$$

Rys. 9.5. Ilustracja reguły Sarrusa

Warto wiedzieć

Regułę Sarrusa można stosować tylko do obliczania wyznacznika macierzy o trzech wierszach i trzech kolumnach.

Równanie prostej przechodzącej przez dwa punkty można zapisać z wykorzystaniem wyznacznika macierzy. Dla prostej przechodzącej przez punkty $A(x_A, y_A)$ oraz $B(x_B, y_B)$ wygląda ono następująco:

$$\begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x & y & 1 \end{vmatrix} = 0$$

Obliczmy wyznacznik znajdujący się po lewej stronie równania:

$$\begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x & y & 1 \end{vmatrix} = x_A \cdot y_B \cdot 1 + x_B \cdot y \cdot 1 + x \cdot y_A \cdot 1 - x \cdot y_B \cdot 1 - x_A \cdot y \cdot 1 - x_B \cdot y_A \cdot 1 = x_A \cdot y_B + x_B \cdot y + x \cdot y_A - x \cdot y_B - x_A \cdot y - x_B \cdot y_A$$

Po przekształceniu lewej strony równania prostej z poprzedniej strony otrzymujemy wyliczony powyżej wyznacznik:

$$\begin{aligned} (y - y_A)(x_B - x_A) - (y_B - y_A)(x - x_A) &= \\ = x_B \cdot y - x_A \cdot y - x_B \cdot y_A + x_A \cdot y_A - x \cdot y_B + x_A \cdot y_B + x \cdot y_A - x_A \cdot y_A &= \\ = x_A \cdot y_B + x_B \cdot y + x \cdot y_A - x \cdot y_B - x_A \cdot y - x_B \cdot y_A \end{aligned}$$

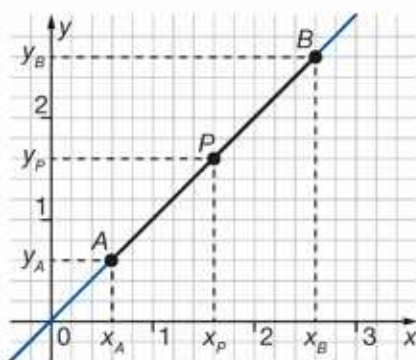
Podczas badania **przynależności punktu do odcinka**, np. punktu $P(x_P, y_P)$ do odcinka AB o końcach w punktach $A(x_A, y_A)$ i $B(x_B, y_B)$, sprawdzimy najpierw, czy należy on do prostej przechodzącej przez punkty A i B . W tym celu do równania prostej wykorzystującego wyznacznik macierzy podstawiamy w miejsce x i y współrzędne punktu P . Jeśli wyznacznik będzie równy 0, to punkty A, B, P są współliniowe. Wyznacznik macierzy dla punktów A, B, P oznaczamy $\det(A, B, P)$:

$$\det(A, B, P) = \begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_P & y_P & 1 \end{vmatrix}$$

Jeśli punkt P należy do prostej, to trzeba jeszcze sprawdzić, czy punkt ten znajduje się pomiędzy punktami A i B . Gdy tak jest, prawdziwe są poniższe nierówności (min oznacza minimum, a max – maksimum).

$$\begin{aligned} \min(x_A, x_B) &\leq x_P \leq \max(x_A, x_B) \\ \min(y_A, y_B) &\leq y_P \leq \max(y_A, y_B) \end{aligned}$$

Zależność między współrzędnymi punktu P a współrzędnymi końców odcinka AB ilustruje rysunek 9.6.



Rys. 9.6. Zależność między współrzędnymi współliniowych punktów A, B, P , gdzie P – punkt należący do odcinka AB

Równanie prostej przechodzącej przez dwa punkty – zapis z wykorzystaniem wyznacznika macierzy

Przynależność punktu do odcinka

Oto specyfikacja problemu sprawdzania, czy punkt należy do odcinka:

Specyfikacja

Dane: x_A, y_A, x_B, y_B – liczby rzeczywiste, współrzędne różnych punktów A i B tworzących odcinek AB ,

x_P, y_P – liczby rzeczywiste, współrzędne punktu P .

Wynik: wartość logiczna **prawda**, jeżeli punkt P należy do odcinka AB , lub **fałsz** – w przeciwnym przypadku.

Algorytm sprawdzający, czy punkt należy do odcinka, można zapisać w pseudokodzie w postaci funkcji tak jak poniżej. Zakładamy, że mamy do dyspozycji funkcję `det`, obliczającą wyznacznik wykorzystywany w równaniu prostej dla trzech danych punktów, oraz funkcje `min` i `max`, wyznaczające odpowiednio minimum i maksimum z dwóch liczb.

```
funkcja PktWOdcinku( $x_A, y_A, x_B, y_B, x_P, y_P$ )
    jeśli  $\text{det}(x_A, y_A, x_B, y_B, x_P, y_P) \neq 0$  to
        zwróć fałsz i zakończ
    jeśli  $x_P \geq \min(x_A, x_B)$  oraz  $x_P \leq \max(x_A, x_B)$  oraz
         $y_P \geq \min(y_A, y_B)$  oraz  $y_P \leq \max(y_A, y_B)$  to
        zwróć prawda i zakończ
    w przeciwnym przypadku
        zwróć fałsz i zakończ
```

Oto kod źródłowy funkcji `det` oraz funkcji `PktWOdcinku`:

Kod źródłowy
funkcji obliczającej
wyznacznik macierzy
dla współrzędnych
trzech punktów oraz
funkcji sprawdzającej,
czy punkt P należy do
odcinka AB

```
1. float det(float x1, float y1,
2.           float x2, float y2,
3.           float x3, float y3)
4. {
5.     return (y3-y1)*(x2-x1)-(y2-y1)*(x3-x1);
6. }
7.
8. bool PktWOdcinku(punkt A, punkt B, punkt P)
9. {
10.    float w=det(A.x,A.y,B.x,B.y,P.x,P.y);
11.    if (abs(w)>EPS) return false;
12.    return (P.x>=min(A.x,B.x) && P.x<=max(A.x,B.x) &&
13.           P.y>=min(A.y,B.y) && P.y<=max(A.y,B.y));
14. }
```

Równanie prostej
przechodzącej przez
dwa punkty,
s. 152 [↗](#)

W funkcji `det` skorzystaliśmy z **równania prostej przechodzącej przez dwa punkty** bez wyznacznika macierzy, ponieważ w tym przypadku program wykona mniej działań mnożenia. Aby poprawić czytelność zapisu funkcji `PktWOdcinku`, zapamiętaliśmy wyznacznik w zmiennej pomocniczej w (linia 10). Zwróć uwagę, że nie porównujemy wyznacznika wprost z zerem, tylko z dokładnością do stałej `EPS`

ze względu na przybliżoną reprezentację liczb rzeczywistych (linia 11). Kiedy badamy tylko znak wyrażenia, nie musimy używać stałej EPS. W liniach 12–13 wykorzystujemy predefiniowane funkcje `min` i `max`, których wartościami są odpowiednio mniejszy lub większy element z dwóch parametrów.

Rysunek 9.7 przedstawia przykład wykonania programu sprawdzającego, czy punkt należy do odcinka.

```
Podaj współrzędne punktu A.
x = 0
y = 0
Podaj współrzędne punktu B.
x = 2
y = 2
Podaj współrzędne punktu P.
x = 1
y = 1
Punkt P należy do odcinka AB.
```

Rys. 9.7. Przykład wykonania programu sprawdzającego, czy punkt należy do odcinka

Dobra rada

Jeśli chcesz porównywać wartości z dokładnością do stałej, pamiętaj, aby ją zdefiniować, np.:

```
const float EPS=0.000001;
```

Dobra rada

W funkcjach `min` i `max` możesz dodatkowo określić trzeci parametr funkcji, który będzie nazwą funkcji logicznej wskazującej sposób porównania dwóch pierwszych parametrów.

Ćwiczenie 2

Napisz program sprawdzający, czy punkt należy do odcinka, zgodnie ze specyfikacją podaną na s. 154. Wykorzystaj omówione w temacie funkcje `CzytajPkt`, `det` oraz `PktWOdcinku`.

Zapamiętaj

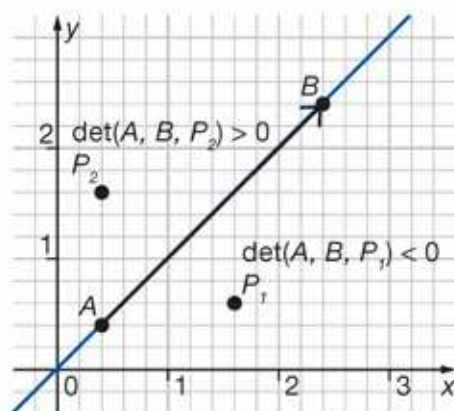
Do badania przynależności punktu do odcinka można wykorzystać równanie prostej przechodzącej przez dwa punkty. Badany punkt musi należeć do prostej, na której leży odcinek, a jego współrzędne muszą się zawierać pomiędzy współrzędnymi końców odcinka.

Położenie punktów względem prostej zawierającej wektor

Rozważmy wektor $\overrightarrow{AB}$ zawarty w prostej. W równaniu prostej przechodzącej przez punkty A i B , zapisanym z wykorzystaniem wyznacznika macierzy, kolejność punktów w dwóch pierwszych wierszach wyznacznika ma znaczenie. Współrzędne punktu A z pierwszego wiersza to współrzędne początku wektora $\overrightarrow{AB}$, a współrzędne punktu B z drugiego wiersza to współrzędne końca wektora.

Na podstawie wyznacznika można określić położenie punktu P względem prostej zawierającej wektor $\overrightarrow{AB}$. Jeśli $\det(A, B, P) > 0$, to punkt P leży po lewej stronie prostej, a jeśli $\det(A, B, P) < 0$, to po prawej stronie. Ilustruje to rysunek 9.8 na s. 156.

Wektor to uporządkowana para punktów – jeden z tych punktów jest początkiem, a drugi końcem wektora. Każdy wektor ma kierunek, zwrot i wartość.

Rys. 9.8. Położenie punktów P_1 i P_2 względem prostej zawierającej wektor $\overline{AB}$

Kod źródłowy funkcji sprawdzającej, czy dwa punkty leżą po tej samej stronie prostej – z użyciem równania ogólnego prostej, s. 151 [🔗](#)

Wcześniej w zapisie funkcji `PktPoTejSamStr`, sprawdzającej, czy dwa punkty leżą po tej samej stronie prostej, zastosowaliśmy równanie ogólne prostej. Teraz zapiszemy kod źródłowy tej funkcji, korzystając z równania prostej przechodzącej przez dwa punkty.

Kod źródłowy funkcji sprawdzającej, czy dwa punkty leżą po tej samej stronie prostej – z wykorzystaniem równania prostej przechodzącej przez dwa punkty

```
1. bool PktPoTejSamStr(punkt A, punkt B,
2.                     punkt P1, punkt P2)
3. {
4.     float w1, w2;
5.     w1=det(A.x,A.y,B.x,B.y,P1.x,P1.y);
6.     w2=det(A.x,A.y,B.x,B.y,P2.x,P2.y);
7.     return (w1*w2>0);
8. }
```

Ćwiczenie 3

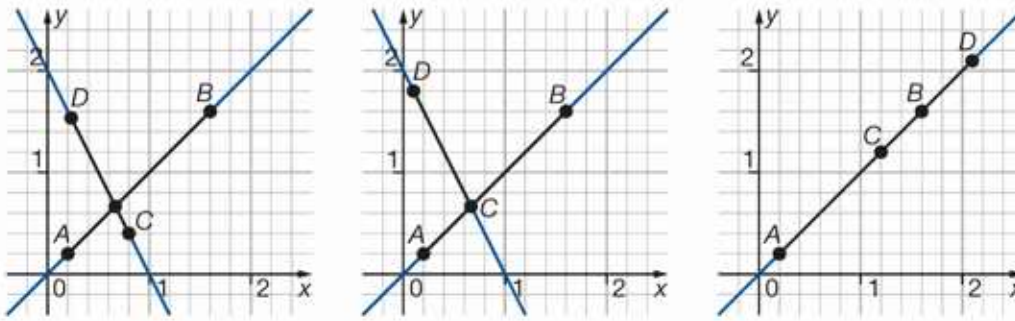
Napisz program sprawdzający, czy dwa punkty P_1 i P_2 leżą po tej samej stronie prostej przechodzącej przez dwa dane punkty A i B .

9.2. Przecinanie się dwóch odcinków

Dwa odcinki AB i CD się przecinają, jeśli mają co najmniej jeden punkt wspólny. Możemy wyróżnić następujące sytuacje.

1. Punkty A i B leżą po przeciwnych stronach prostej przechodzącej przez punkty C i D oraz punkty C i D leżą po przeciwnych stronach prostej przechodzącej przez punkty A i B . Wówczas istnieje dokładnie jeden punkt wspólny.
2. Początek lub koniec jednego odcinka należy do drugiego odcinka. Wówczas odcinki mają jeden punkt wspólny, jeśli leżą na prostych przecinających się, lub więcej niż jeden punkt wspólny – gdy leżą na tej samej prostej.

Możliwe wzajemne położenia przecinających się odcinków AB oraz CD pokazuje rysunek 9.9.



Rys. 9.9. Wzajemne położenia przecinających się odcinków AB oraz CD

Żeby rozpatrzyć pierwszy z wymienionych przypadków, wystarczy policzyć cztery wyznaczniki z równania prostej przechodzącej przez dwa punkty – odpowiednio $\det(A, B, C)$, $\det(A, B, D)$, $\det(C, D, A)$ oraz $\det(C, D, B)$. Ponieważ punkty mają leżeć po przeciwnych stronach prostej, odpowiednie wyznaczniki muszą mieć przeciwne znaki. Wystarczy więc sprawdzić, czy iloczyny $\det(A, B, C) \cdot \det(A, B, D)$ oraz $\det(C, D, A) \cdot \det(C, D, B)$ są ujemne.

W drugim z wymienionych przypadków co najmniej jeden z wyznaczników rozpatrywanych w pierwszym przypadku będzie równy zero. Należy dodatkowo sprawdzić, czy punkt, dla którego wyznacznik jest równy zero, należy do drugiego odcinka.

Oto specyfikacja problemu sprawdzenia, czy dwa odcinki się przecinają, oraz zapis algorytmu w postaci funkcji w pseudokodzie:

Specyfikacja

Dane: $x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D$ – liczby rzeczywiste, współrzędne punktów A, B, C i D tworzących odcinki AB i CD .

Wynik: wartość logiczna **prawda**, gdy odcinki AB i CD się przecinają, lub **fałsz** – w przeciwnym przypadku.

```
funkcja PrzecOdcinki( $x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D$ )
   $w_1 \leftarrow \det(x_A, y_A, x_B, y_B, x_C, y_C)$ 
   $w_2 \leftarrow \det(x_A, y_A, x_B, y_B, x_D, y_D)$ 
   $w_3 \leftarrow \det(x_C, y_C, x_D, y_D, x_A, y_A)$ 
   $w_4 \leftarrow \det(x_C, y_C, x_D, y_D, x_B, y_B)$ 
  jeśli  $w_1 * w_2 < 0$  oraz  $w_3 * w_4 < 0$  to
    zwróć prawda i zakończ
  jeśli PktWOdcinku( $x_A, y_A, x_B, y_B, x_C, y_C$ ) to
    zwróć prawda i zakończ
  jeśli PktWOdcinku( $x_A, y_A, x_B, y_B, x_D, y_D$ ) to
    zwróć prawda i zakończ
  jeśli PktWOdcinku( $x_C, y_C, x_D, y_D, x_A, y_A$ ) to
    zwróć prawda i zakończ
  jeśli PktWOdcinku( $x_C, y_C, x_D, y_D, x_B, y_B$ ) to
    zwróć prawda i zakończ
  zwróć fałsz i zakończ
```


W zapisie funkcji `PrzecOdcinki` zastosowaliśmy dwie funkcje omówione w tym temacie: funkcję `det`, obliczającą wyznacznik, oraz funkcję `PktWOdcinku`, sprawdzającą przynależność punktu do odcinka. Zwróć uwagę, że funkcję `PktWOdcinku` na potrzeby tego problemu można uprościć, wykorzystując obliczone wyznaczniki.

Kod źródłowy funkcji `PrzecOdcinki` jest następujący:

Kod źródłowy funkcji sprawdzającej, czy dwa odcinki się przecinają

```
1. bool PrzecOdcinki(punkt A, punkt B, punkt C, punkt D)
2. {
3.     float w1, w2, w3, w4;
4.     w1=det(A.x,A.y,B.x,B.y,C.x,C.y);
5.     w2=det(A.x,A.y,B.x,B.y,D.x,D.y);
6.     w3=det(C.x,C.y,D.x,D.y,A.x,A.y);
7.     w4=det(C.x,C.y,D.x,D.y,B.x,B.y);
8.     if (w1*w2<0 && w3*w4<0) return true;
9.     if (PktWOdcinku(A,B,C)) return true;
10.    if (PktWOdcinku(A,B,D)) return true;
11.    if (PktWOdcinku(C,D,A)) return true;
12.    if (PktWOdcinku(C,D,B)) return true;
13.    return false;
14. }
```

Rysunek 9.10 przedstawia przykład wykonania programu sprawdzającego, czy dwa odcinki się przecinają.

```
Podaj współrzędne końców odcinka AB.
Współrzędne punktu A.
x = 0
y = 0
Współrzędne punktu B.
x = 2
y = 2
Podaj współrzędne końców odcinka CD.
Współrzędne punktu C.
x = 0
y = 2
Współrzędne punktu D.
x = 2
y = 0
Odcinki się przecinają.
```

Rys. 9.10. Przykład wykonania programu sprawdzającego, czy dwa odcinki się przecinają

Ćwiczenie 4

Napisz program sprawdzający, czy dwa odcinki się przecinają, zgodnie ze specyfikacją podaną na s. 157.

Zapamiętaj

Żeby zbadać, czy odcinki AB i CD się przecinają, wystarczy sprawdzić, czy punkty A i B leżą po przeciwnych stronach prostej przechodzącej przez punkty C i D oraz czy punkty C i D leżą po przeciwnych stronach prostej przechodzącej przez punkty A i B . Dodatkowo należy rozpatrzyć przypadki, gdy koniec jednego odcinka należy do drugiego odcinka.

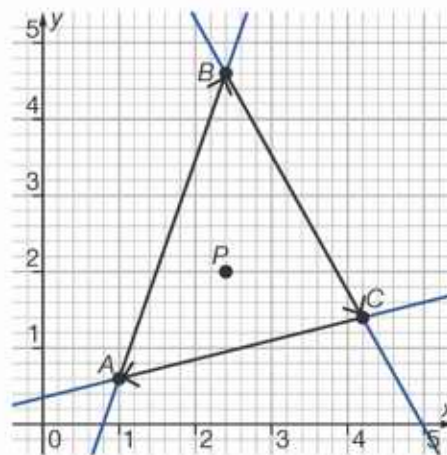
9.3. Przynależność punktu do wielokąta wypukłego

Najpierw sprawdzimy, czy punkt należy do trójkąta. Następnie zbadamy, czy punkt należy do dowolnego **wielokąta wypukłego**.

Przynależność punktu do trójkąta

Boki trójkąta możemy potraktować jako **wektory**. Koniec jednego wektora będzie początkiem następnego. Jeśli punkt P należy do trójkąta i nie leży na żadnym z jego boków, to musi znajdować się po tej samej stronie każdego z wektorów. Ilustruje to rysunek 9.11: punkt P znajduje się po prawej stronie wektorów $\overrightarrow{AB}$, $\overrightarrow{BC}$ i $\overrightarrow{CA}$. Wówczas znaki wyznaczników z równania prostej przechodzącej przez dwa punkty, odpowiednio $\det(A, B, P)$, $\det(B, C, P)$, $\det(C, A, P)$, są takie same.

Jeśli punkt będzie leżał na którymś z boków trójkąta (może być też jego wierzchołkiem), wtedy co najmniej jeden z wyznaczników będzie równy zero.



Rys. 9.11. Ilustracja przynależności punktu P do trójkąta ABC , gdy punkt P nie leży na żadnym z boków trójkąta

• **Wielokąt wypukły** to wielokąt, w którym każdy z kątów ma miarę mniejszą niż 180° .

Wektor,
s. 155 

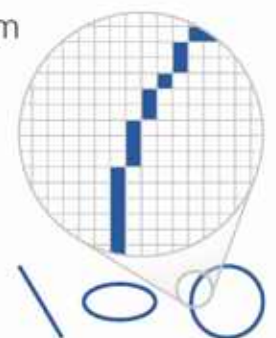
Dobra rada

Jeśli każdy odcinek o końcach należących do danego wielokąta zawiera się w wielokącie, to wielokąt ten jest wypukły. Zwróć uwagę, że każdy trójkąt jest wielokątem wypukłym.

A to ciekawe

Jak na ekranie komputera rysowane są odcinki, okręgi, elipsy?

W grafice rastrowej obraz jest siatką pikseli. Aby dany kształt był jak najwierniejszym odwzorowaniem oczekiwanego, trzeba wybrać te piksele, które muszą zostać pokolorowane. W 1962 r. amerykański informatyk Jack Bresenham opracował algorytm, który umożliwia rysowanie odcinka na urządzeniach rastrowych. Wykorzystuje on operacje tylko na liczbach całkowitych, co jest ważne przy implementacjach algorytmu (nie wymaga to czasochłonnych operacji na liczbach rzeczywistych). Algorytm Bresenhama przystosowano później do rysowania innych kształtów, np. okręgu czy elipsy.



Oto specyfikacja problemu przynależności punktu do trójkąta:

Specyfikacja

Dane: $x_A, y_A, x_B, y_B, x_C, y_C$ – liczby rzeczywiste, współrzędne nie-współliniowych punktów A, B, C tworzących trójkąt ABC ,
 x_P, y_P – liczby rzeczywiste, współrzędne punktu P .

Wynik: wartość logiczna **prawda**, jeżeli punkt P należy do trójkąta ABC , lub **fałsz** – w przeciwnym przypadku.

Funkcję `PktWTrojkanie`, realizującą opisany algorytm badania przynależności punktu do trójkąta, można zapisać w pseudokodzie następująco:

```
funkcja PktWTrojkanie( $x_A, y_A, x_B, y_B, x_C, y_C, x_P, y_P$ )
     $w_1 \leftarrow \det(x_A, y_A, x_B, y_B, x_P, y_P)$ 
     $w_2 \leftarrow \det(x_B, y_B, x_C, y_C, x_P, y_P)$ 
    jeśli  $w_1 * w_2 < 0$  to zwróć fałsz i zakończ
     $w_2 \leftarrow \det(x_C, y_C, x_A, y_A, x_P, y_P)$ 
    jeśli  $w_1 * w_2 < 0$  to zwróć fałsz i zakończ
    zwróć prawda i zakończ
```

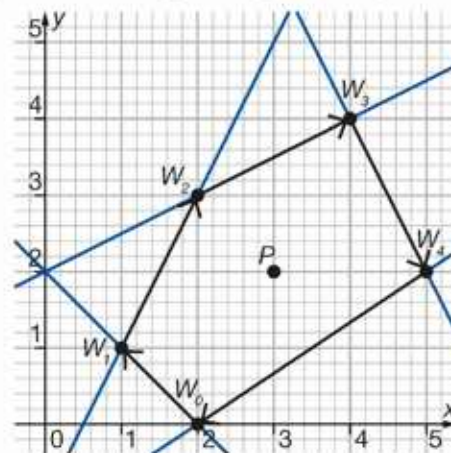
Obliczamy wyznaczniki w_1 i w_2 , określające położenie punktu P względem prostych wyznaczonych przez boki AB i BC . Jeśli iloczyn wyznaczników jest ujemny, to punkt P nie należy do trójkąta. W przeciwnym przypadku obliczamy nową wartość zmiennej w_2 dla trzeciego wyznacznika (dla prostej przechodzącej przez punkty C oraz A) i ponownie badamy znak iloczynu wyznaczników w_1 i w_2 .

Ćwiczenie 5

Napisz program sprawdzający, czy punkt P należy do trójkąta ABC , zgodnie ze specyfikacją podaną powyżej.

Przynależność punktu do dowolnego wielokąta wypukłego

Omówiony algorytm badania przynależności punktu do trójkąta można zastosować dla dowolnego wielokąta wypukłego. Załóżmy, że punkty $W_i(x_i, y_i)$, gdzie $i = 0, \dots, n-1$, są wierzchołkami wielokąta wypukłego o n bokach. Odcinki $W_i W_{i+1}$ dla $i = 0, \dots, n-2$ oraz $W_{n-1} W_0$ tworzą kolejne boki wielokąta. Należy sprawdzić, czy punkt P leży po tej samej stronie wszystkich prostych wyznaczonych przez wektory $\overrightarrow{W_i W_{i+1}}$ i wektor $\overrightarrow{W_{n-1} W_0}$. Ilustruje to rysunek 9.12.



Rys. 9.12. Ilustracja przynależności punktu P do wielokąta wypukłego, gdy punkt P nie leży na żadnym z jego boków

Specyfikacja problemu badania przynależności punktu do wielokąta wypukłego może być następująca:

Specyfikacja

Dane: n – liczba wierzchołków wielokąta wypukłego, $n \geq 3$,
 $W[0..n-1]$ – tablica przechowująca współrzędne punktów będących kolejnymi wierzchołkami wielokąta wypukłego, każdy element tablicy pamięta informację o jednym punkcie, żadne trzy punkty nie są współliniowe, a współrzędne punktu to dwie liczby rzeczywiste (x_i, y_i) , $i = 0, \dots, n-1$,
 xP, yP – liczby rzeczywiste, współrzędne punktu P .

Wynik: wartość logiczna **prawda**, gdy punkt P należy do wielokąta wypukłego o wierzchołkach z tablicy W , lub **fałsz** – w przeciwnym przypadku.

Oto funkcja `PktWWielWyp`, realizująca opisany algorytm badania przynależności punktu do wielokąta wypukłego, zapisana w pseudokodzie:

```
funkcja PktWWielWyp( $n, W, xP, yP$ )
     $w1 \leftarrow \det(W[0].x, W[0].y, W[1].x, W[1].y, xP, yP)$ 
    dla  $i \leftarrow 1, \dots, n-1$  wykonuj
         $w2 \leftarrow \det(W[i].x, W[i].y, W[(i+1) \bmod n].x,$ 
             $W[(i+1) \bmod n].y, xP, yP)$ 
        jeśli  $w1 * w2 < 0$  to zwróć fałsz i zakończ
    zwróć prawda i zakończ
```

Obliczamy najpierw wyznacznik określający położenie punktu P względem prostej wyznaczonej przez dwa pierwsze wierzchołki z tablicy W (zmienna $w1$). Następnie w pętli obliczamy wyznaczniki określające położenie punktu P względem prostych wyznaczonych przez końce kolejnych boków (zmienna $w2$). Zwróć uwagę, że indeks drugiego wierzchołka dzielimy z resztą przez n . Dzięki temu dla $i = n-1$ otrzymamy wartość $(i+1) \bmod n = 0$, czyli indeks końcowego wierzchołka ostatniego boku (czyli $W[0]$). Jeśli iloczyn $w1*w2$ dla którejś z wartości $w2$ będzie ujemny, to punkt P nie należy do wielokąta.

Dane opisujące wielokąt wygodnie jest przygotować w pliku tekstowym. Pierwszy wiersz może zawierać jedną liczbę całkowitą n określającą liczbę wierzchołków, następne n wierszy – po dwie liczby rzeczywiste będące współrzędnymi kolejnych wierzchołków wielokąta. Na przykład opis wielokąta wypukłego z rysunku 9.12 będzie taki jak na rysunku 9.13.

5
2 0
1 1
2 3
4 4
5 2

Rys. 9.13. Opis pięciokąta z rysunku 9.12 w pliku tekstowym

Oto kod źródłowy funkcji `CzytajWiel`, wczytującej opis wielokąta z pliku tekstowego:

Kod źródłowy funkcji wczytującej z pliku tekstowego informację o liczbie wierzchołków wielokąta wypukłego oraz współrzędne tych wierzchołków

```
1. void CzytajWiel(vector<punkt> &W)
2. {
3.     int i, n;
4.     ifstream we("wielwyp.txt");
5.     we>>n; W.resize(n);
6.     for (i=0;i<n;i++) we>>W[i].x>>W[i].y;
7.     we.close();
8. }
```

Parametr `W` funkcji `CzytajWiel` jest przekazywany przez referencję. Jest on typu `vector`. Każdy jego element jest strukturą typu `punkt`, która pamięta współrzędne jednego wierzchołka. W linii 4 otwierany jest plik `wielwyp.txt` z opisem wielokąta, w linii 5 wczytywana jest liczba wierzchołków i ustalany jest rozmiar parametru `W`. Pętla w linii 6 wczytuje współrzędne kolejnych punktów.

Kod źródłowy funkcji `PktWWielWyp`, sprawdzającej przynależność punktu do wielokąta wypukłego, jest następujący:

Kod źródłowy funkcji sprawdzającej, czy punkt należy do wielokąta wypukłego

```
1. bool PktWWielWyp(vector<punkt> W, punkt P)
2. {
3.     int n=W.size();
4.     float w1, w2;
5.     w1=det(W[0].x,W[0].y,W[1].x,W[1].y,P.x,P.y);
6.     for (int i=1;i<n;i++)
7.     {
8.         w2=det(W[i].x,W[i].y,W[(i+1)%n].x,W[(i+1)%n].y,P.x,P.y);
9.         if (w1*w2<0) return false;
10.    }
11.    return true;
12. }
```

Ćwiczenie 6

Napisz program sprawdzający, czy punkt P należy do wielokąta, zgodnie ze specyfikacją podaną na s. 161. Opis wielokąta wczytaj z pliku tekstowego, który otrzymasz od nauczyciela (np. `wielwyp.txt`).

Zapamiętaj

Aby określić przynależność punktu P do wielokąta wypukłego, wystarczy sprawdzić położenie punktu P względem prostych wyznaczonych przez wektory tworzące boki wielokąta. Koniec jednego wektora jest początkiem następnego.

Podsumowanie

- Położenie punktów względem prostej można określić, korzystając z równania ogólnej prostej $Ax + By + C = 0$. Współrzędne punktu podstawiamy do lewej strony równania. Jeśli punkt należy do prostej, to wartość wyrażenia jest równa 0. Dla punktów znajdujących się po tej samej stronie prostej znaki otrzymanych wyników są takie same.
- Żeby zbadać, czy punkt P należy do odcinka AB , trzeba sprawdzić, czy należy on do prostej przechodzącej przez punkty A i B oraz czy współrzędne punktu P zawierają się pomiędzy współrzędnymi punktów A i B .
- Równanie prostej przechodzącej przez dwa punkty można zapisać z wykorzystaniem wyznacznika macierzy. Dla punktu P leżącego po prawej stronie prostej zawierającej wektor $\overrightarrow{AB}$ wartość $\det(A, B, P)$ jest ujemna, a dla punktu leżącego po lewej stronie prostej – dodatnia. Dla punktów leżących na prostej wyznacznik ma wartość 0.
- Żeby zbadać przecinanie się odcinków AB i CD , wystarczy sprawdzić, czy punkty A i B leżą po przeciwnych stronach prostej przechodzącej przez punkty C i D oraz czy punkty C i D leżą po przeciwnych stronach prostej przechodzącej przez punkty A i B . Dodatkowo należy rozpatrzyć przypadki, gdy koniec jednego odcinka należy do drugiego odcinka.
- Badając przynależność punktu do wielokąta wypukłego, sprawdzamy położenie tego punktu względem prostych wyznaczonych przez wektory tworzące boki wielokąta. Koniec jednego wektora jest początkiem następnego. Pierwszy wierzchołek wielokąta jest początkiem wektora wyznaczającego pierwszy bok i równocześnie końcem wektora tworzącego ostatni bok.
- Przynależność punktu do trójkąta badamy tak samo jak przynależność do dowolnego wielokąta wypukłego. Każdy trójkąt jest wielokątem wypukłym.

Zadania

- ★ **1** Napisz program obliczający pole trójkąta ABC , którego wierzchołkami są niewspółliniowe punkty $A(x_A, y_A)$, $B(x_B, y_B)$ i $C(x_C, y_C)$.

Wskazówka: Możesz skorzystać z poniższego wzoru na pole trójkąta ABC .

$$P_{ABC} = \frac{1}{2} \cdot \left| (x_B - x_A) \cdot (y_C - y_A) - (x_C - x_A) \cdot (y_B - y_A) \right|$$

- ★ **2** Punktem kratowym nazywamy punkt o współrzędnych całkowitych. Wczytaj z klawiatury współrzędne dwóch różnych punktów kratowych A i B o wartościach każdej ze współrzędnych z zakresu od 1 do 9. Oblicz liczbę punktów kratowych znajdujących się po każdej ze stron prostej przechodzącej przez punkty A i B w obrębie kwadratu, którego wierzchołki znajdują się w punktach o współrzędnych $(0, 0)$, $(0, 10)$, $(10, 10)$ i $(10, 0)$. Na przykład dla punktów $A(1, 1)$ i $B(2, 2)$ po obu stronach prostej przechodzącej przez te punkty znajduje się po 55 punktów kratowych spełniających warunki zadania.

- ★ **3** Napisz program sprawdzający położenie punktu P względem prostej zawierającej wektor $\overrightarrow{AB}$, gdzie punkt A leży poniżej punktu B . Dane punktów A , B i P wczytaj z klawiatury. Program powinien wypisać jeden z komunikatów: punkt leży na prostej poniżej punktu A , punkt należy do odcinka AB , punkt leży na prostej powyżej punktu B , punkt nie należy do prostej przechodzącej przez punkty A i B .
- ★★ **4** Punktem kratowym nazywamy punkt o współrzędnych całkowitych. Napisz program, który obliczy liczbę punktów kratowych należących do trójkąta ABC , którego wierzchołki mają współrzędne całkowite. Załóż, że boki trójkąta nie zawierają punktów kratowych poza wierzchołkami trójkąta.
- ★★ **5** Napisz program obliczający pole wielokąta wypukłego. Dane wczytaj z pliku tekstowego, który otrzymasz od nauczyciela (np. *wielokat_1.txt*). W pliku w pierwszym wierszu znajduje się jedna liczba całkowita dodatnia $n \geq 3$, określająca liczbę wierzchołków wielokąta. W następnych n wierszach pliku znajdują się po dwie liczby rzeczywiste oddzielone spacją, określające współrzędne kolejnych wierzchołków w kartezjańskim układzie współrzędnych.
- ★★★ **6** Napisz program, który sprawdzi przynależność punktu P do wielokąta wklęsłego. Dane wielokąta wczytaj z pliku tekstowego, który otrzymasz od nauczyciela (np. *wielokat_2.txt*). W pliku w pierwszym wierszu znajduje się jedna liczba całkowita dodatnia $n \geq 3$, określająca liczbę wierzchołków wielokąta. W następnych n wierszach pliku znajdują się po dwie liczby rzeczywiste oddzielone spacją, określające współrzędne kolejnych wierzchołków w kartezjańskim układzie współrzędnych. Współrzędne punktu P wczytaj z klawiatury.
- ★★★ **7** Napisz program znajdujący tzw. wypukłą otoczkę zbioru punktów. Wypukła otoczka to podzbiór zbioru punktów tworzący wielokąt wypukły zawierający wszystkie punkty zbioru. Dane do programu wczytaj z pliku tekstowego, który otrzymasz od nauczyciela (np. *punkty_1.txt*). W pliku w pierwszym wierszu znajduje się jedna liczba całkowita dodatnia $n \geq 3$, określająca liczbę punktów, w następnych n wierszach pliku znajdują się po dwie liczby rzeczywiste oddzielone spacją, określające współrzędne kolejnych punktów w kartezjańskim układzie współrzędnych. Wynik, czyli współrzędne punktów tworzących wypukłą otoczkę, zapisz w pliku tekstowym (np. *punkty_1_wynik.txt*) – w każdym wierszu pliku współrzędne jednego punktu oddzielone spacją.

10. Fraktale

W przyrodzie można zaobserwować wiele obiektów, których fragmenty są podobne do całości. Podobieństwo znajdziemy np. w liściu paproci, kalafiorze czy drzewach. Dzięki rozwojowi technik komputerowych rysowanie przybliżeń takich obiektów jest ułatwione. W tym temacie zajmiemy się algorytmami, które to umożliwiają, i narysujemy przybliżenia wybranych figur.

Cele lekcji

- Dowiesz się, czym jest fraktal.
- Poznasz zbiór Cantora.
- Narysujesz krzywą Kocha oraz płatek Kocha.
- Dowiesz się, czym jest drzewo binarne, i utworzysz przykład takiej figury.
- Narysujesz dywan Sierpińskiego.
- Zastosujesz rekurencję oraz język JavaScript z biblioteką p5.js.
- Poznasz metodę IFS, służącą do generowania fraktali.

Fraktal to figura, którą cechuje m.in. samopodobieństwo, czyli fragmen- **Fraktal**
ty tej figury są podobne do całości. Pierwsze znane opisy takich figur
pochodzą z XIX w., choć wówczas nie używano jeszcze pojęcia „fraktal”.

10.1. Zbiór Cantora

Zbiór Cantora jest podzbiorem odcinka jednostkowego (o długości 1), **Zbiór Cantora**
który można opisać w następujący sposób:

- ▶ zbiór Cantora stopnia 0 to odcinek jednostkowy,
- ▶ zbiór Cantora stopnia 1 otrzymujemy z odcinka jednostkowego: dzielimy go na trzy równe części i usuwamy część środkową,
- ▶ zbiór Cantora wyższego stopnia tworzymy ze zbioru Cantora stopnia o jeden mniejszego: dzielimy każdy jego odcinek na trzy równe części i usuwamy części środkowe.

Proces dzielenia odcinków tworzących zbiór Cantora i usuwania z nich części środkowych jest nieskończony, więc będziemy generować jedynie zbiory Cantora określonego stopnia. Rysunek 10.1 ilustruje zbiory Cantora stopni od 0 do 3.



Rys. 10.1. Zbiory Cantora stopni od 0 do 3 (w danej linii przedstawiony jest zbiór jednego stopnia)

Warto wiedzieć

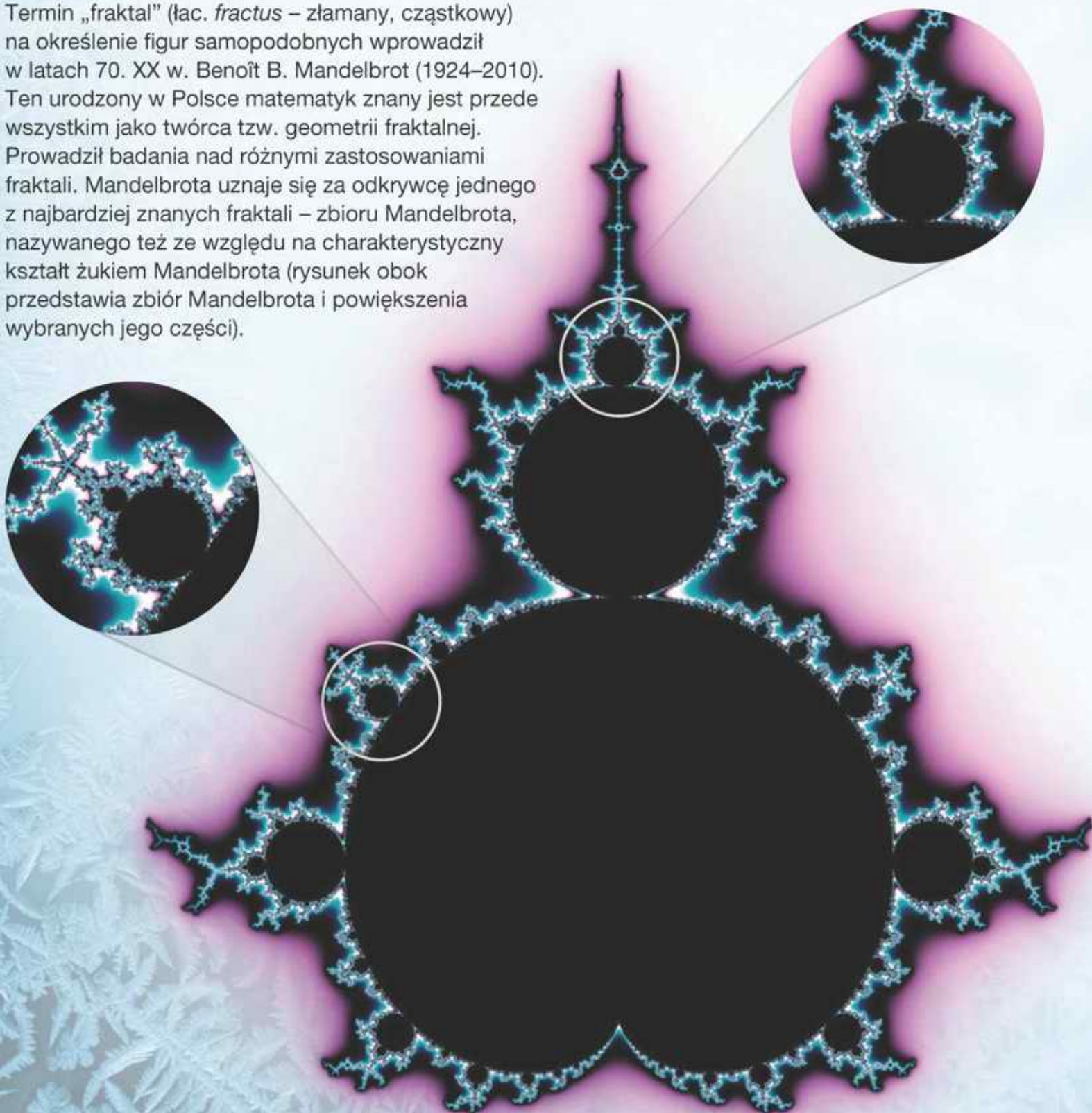
Zbiór Cantora odkrył w 1875 r. irlandzki matematyk Henry J.S. Smith, opisał go natomiast niemiecki matematyk Georg F.L.P. Cantor w 1883 r. Prace Cantora są kluczowe dla podstaw współczesnej matematyki.

Co warto wiedzieć o fraktalach

Badania pokazują, że obiekty cechujące się samopodobieństwem są odbierane przez ludzi pozytywnie i określane jako ładne. Być może jest tak dlatego, że żyjemy w świecie, w którym bez trudu można wskazać przykłady takich obiektów. Obecnie z własności fraktali korzysta się w różnych dziedzinach życia, np. przy produkcji urządzeń czy analizowaniu zjawisk przyrodniczych.

Geometria fraktalna

Termin „fraktal” (łac. *fractus* – złamany, cząstkowy) na określenie figur samopodobnych wprowadził w latach 70. XX w. Benoît B. Mandelbrot (1924–2010). Ten urodzony w Polsce matematyk znany jest przede wszystkim jako twórca tzw. geometrii fraktalnej. Prowadził badania nad różnymi zastosowaniami fraktali. Mandelbrota uznaje się za odkrywcę jednego z najbardziej znanych fraktali – zbioru Mandelbrota, nazywanego też ze względu na charakterystyczny kształt żukiem Mandelbrota (rysunek obok przedstawia zbiór Mandelbrota i powiększenia wybranych jego części).



Fraktale w przyrodzie

W naturze rzadko spotykamy obiekty o regularnych kształtach, ale w wielu łatwo wskazać cechę samopodobieństwa. Niektóre struktury fraktalne można dostrzec gołym okiem, inne zauważymy dopiero na zdjęciach wykonanych z dużej odległości (góry, kaniony, delty rzek, linie brzegowe, chmury) lub po znacznym powiększeniu (płatki śniegu, bakterie, struktura białek).



Kalafior
romanesco



Kultura bakterii
Paenibacillus



Fragment Himalajów (zdjęcie satelitarne)

Zastosowania fraktali

Fraktale wykorzystuje się w produkcji filmów i gier, np. do generowania krajobrazu, tła, ognia. Tworzy się wówczas małe fragmenty obrazów, a następnie komponuje z nich całość. Własność samopodobieństwa ma zastosowanie przy projektowaniu anten. Użycie struktury fraktalnej pozwala znacznie zmniejszyć rozmiar anteny bez pogorszenia jej parametrów. Z własności fraktali korzysta się także podczas analizowania różnych procesów i przy kompresji danych, a wzory fraktalne stosują projektanci tkanin oraz malarze. Można je dostrzec m.in. na płótnach Jacksona Pollocka.

Analizy fraktalnej używa się m.in. przy prognozowaniu notowań giełdowych i kursów walut



W telefonach komórkowych stosuje się anteny o strukturze fraktalnej



Algorytmy fraktalne wykorzystano do wygenerowania lawy w słynnej scenie walki na miecze w części III *Gwiezdnych wojen*

Algorytm rysowania zbioru Cantora stopnia n

Zbiór Cantora,
s. 165 

Podczas rysowania **zbioru Cantora** stopnia n za odcinek jednostkowy (o długości 1) przyjmujemy odcinek o długości d . Nazwijmy go odcinkiem bazowym. Specyfikacja problemu może być następująca:

Specyfikacja

Dane: n – liczba całkowita nieujemna, stopień zbioru Cantora,
 x, y – liczby rzeczywiste, współrzędne punktu będącego początkiem odcinka bazowego,
 d – długość odcinka bazowego.

Wynik: rysunek zbioru Cantora stopnia n .

Funkcję rekurencyjną Cantor, która narysuje zbiór Cantora stopnia n , można zapisać w pseudokodzie tak jak poniżej. Funkcja Odcinek rysuje odcinek wyznaczony przez dwa punkty o współrzędnych, które są parametrami funkcji.

```
funkcja Cantor( $n, x, y, d$ )
    jeśli  $n = 0$  to
        Odcinek( $x, y, x+d, y$ )
    w przeciwnym przypadku
        Cantor( $n-1, x, y, d/3$ )
        Cantor( $n-1, x+2*d/3, y, d/3$ )
```

Jeśli stopień zbioru Cantora jest równy 0, to rysujemy odcinek o końcach w punktach (x, y) i $(x+d, y)$. W przeciwnym przypadku wywołujemy rekurencyjnie funkcję Cantor dla dwóch odcinków: pierwszej i trzeciej części dzielonego odcinka.

Rysowanie zbioru Cantora stopnia n z wykorzystaniem języka JavaScript

Biblioteka p5.js  Do rysowania fraktali wykorzystamy język JavaScript z **biblioteką p5.js**. Ułatwia ona m.in. tworzenie aplikacji graficznych, multimedialnych i stron internetowych.

Funkcję Cantor, realizującą przedstawiony wyżej algorytm, możemy zapisać w języku JavaScript z użyciem biblioteki p5.js następująco:

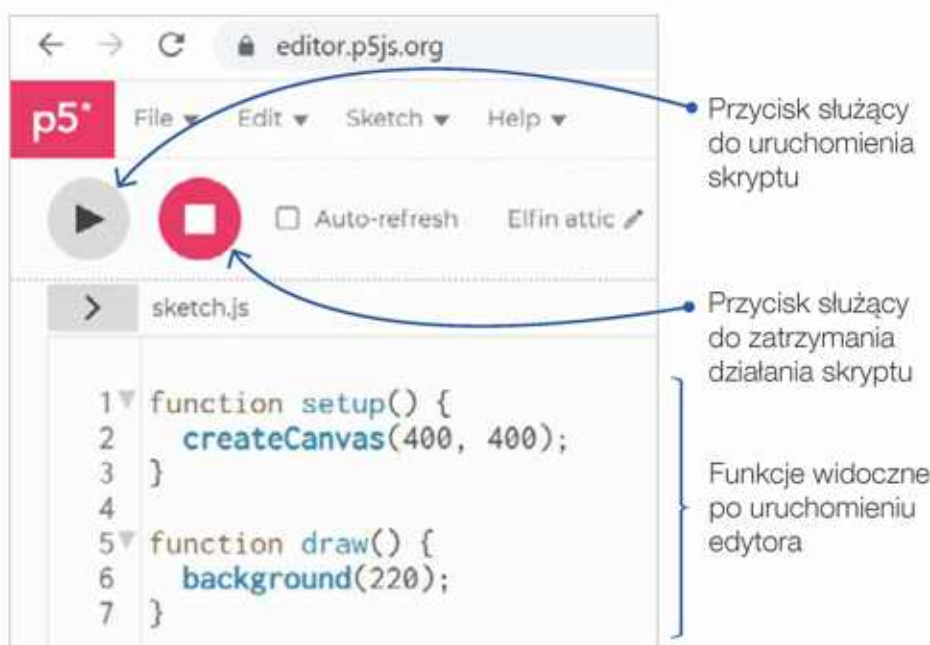
Funkcja rysująca zbiór 
Cantora stopnia n

```
1. function Cantor( $n, x, y, d$ ) {
2.   if ( $n==0$ ) {
3.     line( $x, y, x+d, y$ );
4.   }
5.   else {
6.     Cantor( $n-1, x, y, d/3$ );
7.     Cantor( $n-1, x+2*d/3, y, d/3$ );
8.   }
9. }
```

Definicję funkcji rozpoczynamy od słowa kluczowego `function`. Zwróć uwagę, że **podstawowe elementy składni języka JavaScript** są podobne do składni języka C++. W JavaScript, w odróżnieniu od języka C++, nie deklarujemy typu funkcji oraz typu parametrów. Użyta w linii 3 **funkcja line** rysuje odcinek między punktami, których współrzędne są parametrami funkcji. Funkcja **line** jest elementem biblioteki p5.js.

Aby sprawdzić działanie funkcji **Cantor**, można skorzystać z edytora dostępnego na stronie <https://p5js.org> – należy wybrać opcję **Editor**. W edytorze oprócz definiowania skryptów można je od razu uruchomić.

Po przejściu do edytora zobaczymy dwie predefiniowane **funkcje**: **setup** i **draw**. Musi je zawierać każdy projekt – tutaj nazywany szkicem (ang. *sketch*). Po uruchomieniu skryptu funkcja **setup** wykona się tylko raz, natomiast funkcja **draw** jest domyślnie wywoływana 60 razy na sekundę. Umożliwia to tworzenie animacji. Fragment okna edytora ilustruje rysunek 10.2.



Rys. 10.2. Fragment okna edytora na stronie <https://p5js.org> po uruchomieniu

Podstawowe elementy składni języka JavaScript, podręcznik Informatyka na czasie 1, Zakres rozszerzony, s. 281–284 ↗

• **Funkcja line**

• **Funkcja setup, funkcja draw**

Warto wiedzieć

Częstotliwość wykonywania funkcji **draw** można zmienić za pomocą funkcji **frameRate**.

Dobra rada

Na stronie <https://p5js.org/> (opcja **Reference**) znajdziesz pomoc dotyczącą funkcji zdefiniowanych w bibliotece p5.js.

Funkcja createCanvas tworzy kanwę (ang. *canvas* – płótno), czyli obszar rysowania – będzie on widoczny obok skryptu. Parametry funkcji określają szerokość i wysokość kanwy w pikselach. Kanwa ma zdefiniowany następująco układ współrzędnych: punktowi (0, 0) odpowiada punkt w lewym górnym rogu, wartości osi pionowej rosną w dół, a osi poziomej – w prawo.

Funkcja background służy do ustalenia koloru tła. Jeśli ma jeden parametr, to określa on odcień szarości. Parametr ten jest z zakresu od 0 do 255.

Definicję funkcji **Cantor** zapiszemy pod funkcjami **setup** i **draw**.

Aby narysować zbiór Cantora danego stopnia, musimy zmodyfikować funkcje **setup** i **draw**. Ich zapis może wyglądać następująco.

Fragment skryptu
rysującego zbiór
Cantora stopnia 3 –
przykład definicji funkcji
setup i draw

👍 Dobra rada

Długość rysowanego odcinka maleje trzykrotnie wraz z powiększeniem stopnia zbioru Cantora o 1. Dlatego stopień zbioru i długość odcinka bazowego dobieraj tak, aby długość rysowanego odcinka nie była mniejsza od 1 (1 piksele).

```
1. function setup() {
2.   createCanvas(600, 400);
3. }
4.
5. function draw() {
6.   background(220);
7.   Cantor(3,10,200,580);
8.   noLoop();
9. }
```

W funkcji `setup` zmieniliśmy szerokość kanwy na 600 – dzięki temu mamy wystarczająco dużo miejsca na rysunek. W funkcji `draw` wywołujemy funkcję `Cantor` z konkretnymi wartościami parametrów. W naszym przykładzie stopień zbioru Cantora jest równy 3, a odcinek bazowy ma początek w punkcie (10, 200) i długość 580. W linii 8 wywołana jest funkcja `noLoop`, zatrzymująca automatyczne powtarzanie funkcji `draw`. Jeśli tworzy się rysunek statyczny (nie animację), wywołanie funkcji `noLoop` powinno być ostatnią instrukcją funkcji `draw`. Okno edytora z efektem działania skryptu ilustruje rysunek 10.3.

Funkcja `noLoop`

• wywoływana jest funkcja `noLoop`, zatrzymująca automatyczne powtarzanie funkcji `draw`. Jeśli tworzy się rysunek statyczny (nie animację), wywołanie funkcji `noLoop` powinno być ostatnią instrukcją funkcji `draw`. Okno edytora z efektem działania skryptu ilustruje rysunek 10.3.



Rys. 10.3. Fragment okna edytora na stronie <https://p5js.org> z wywołaniem funkcji `Cantor` dla zbioru Cantora stopnia 3

👍 Dobra rada

Po założeniu konta na stronie <https://p5js.org> możesz zapisywać na nim swoje prace. Jeśli wolisz pracować bez rejestracji, kod źródłowy możesz skopiować do wybranego edytora tekstu, np. Notepad++.

Ćwiczenie 1

W edytorze dostępnym na stronie <https://p5js.org> utwórz skrypt, dzięki któremu narysujesz zbiór Cantora stopnia 3. Następnie zmodyfikuj skrypt tak, aby narysować zbiór Cantora stopnia 6.

Funkcję `draw` można zmodyfikować tak, aby wyświetlić jeden pod drugim zbiory Cantora kolejnych stopni. Wystarczy wówczas wywołać funkcję `Cantor` w pętli. Oto zmodyfikowana funkcja `draw`.

```

1. function draw() {
2.   background(220);
3.   for (n=0;n<7;n++){
4.     Cantor(n,10,(n+1)*50,580);
5.   }
6.   noLoop();
7. }

```

Fragment skryptu rysującego zbiory Cantora stopni od 0 do 6 – funkcja draw

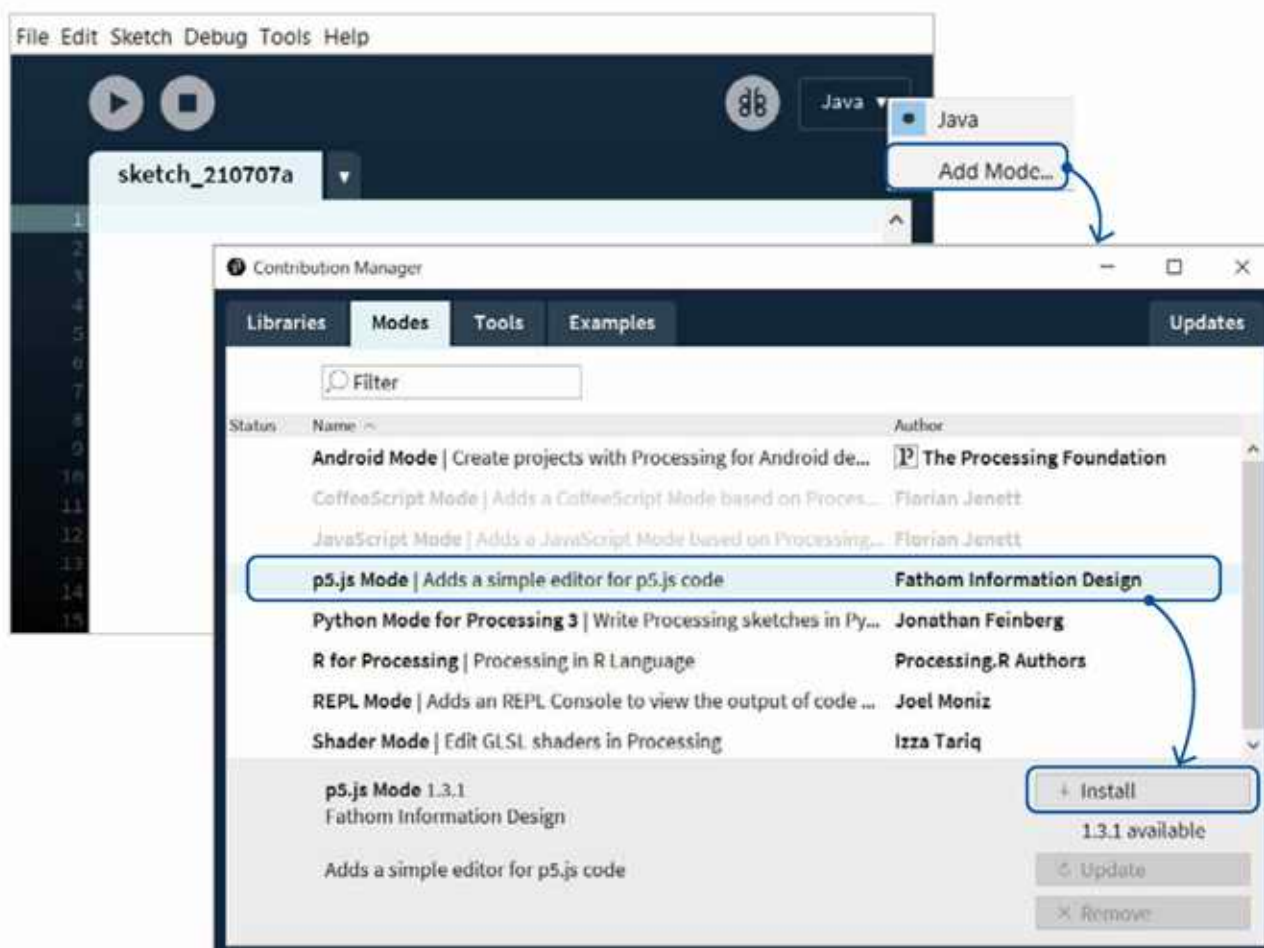
Warto wiedzieć

Processing jest środowiskiem programistycznym umożliwiającym łatwe tworzenie aplikacji multimedialnych. Można pracować m.in. w trybie Javy (domyślnie), Pythona, p5.js czy Androida. Środowisko można pobrać ze strony www.processing.org. Dostępne są wersje dla różnych systemów operacyjnych.

Ćwiczenie 2

W edytorze na stronie <https://p5js.org> utwórz skrypt, dzięki któremu narysujesz na jednej kanwie zbiory Cantora stopni od 0 do 6.

Skrypty w języku JavaScript z wykorzystaniem biblioteki p5.js można też tworzyć i uruchamiać offline. W tym celu wystarczy pobrać środowisko programistyczne **Processing** (nie trzeba go instalować), a następnie dołączyć bibliotekę p5.js. Aby ją dodać, po uruchomieniu środowiska należy w prawym górnym rogu okna rozwinąć listę trybów, wybrać opcję **Add Mode...**, a następnie na liście proponowanych trybów zaznaczyć **p5.js Mode** i kliknąć przycisk **Install** (rys. 10.4).



Rys. 10.4. Instalacja trybu p5.js w środowisku Processing

Dobra rada

Jeśli napiszesz skrypt z wykorzystaniem biblioteki p5.js i chcesz sprawdzić jego działanie w przeglądarce internetowej, ale bez uruchamiania środowiska Processing, musisz zmodyfikować plik *index.html*. Linie zawierającą informację o dołączeniu skryptu *libraries/p5.min.js* należy uzupełnić o atrybut `async="true"`.

Warto wiedzieć

Sumę długości odcinków usuwanych z odcinka jednostkowego przy wyznaczaniu zbioru Cantora można obliczyć następująco:

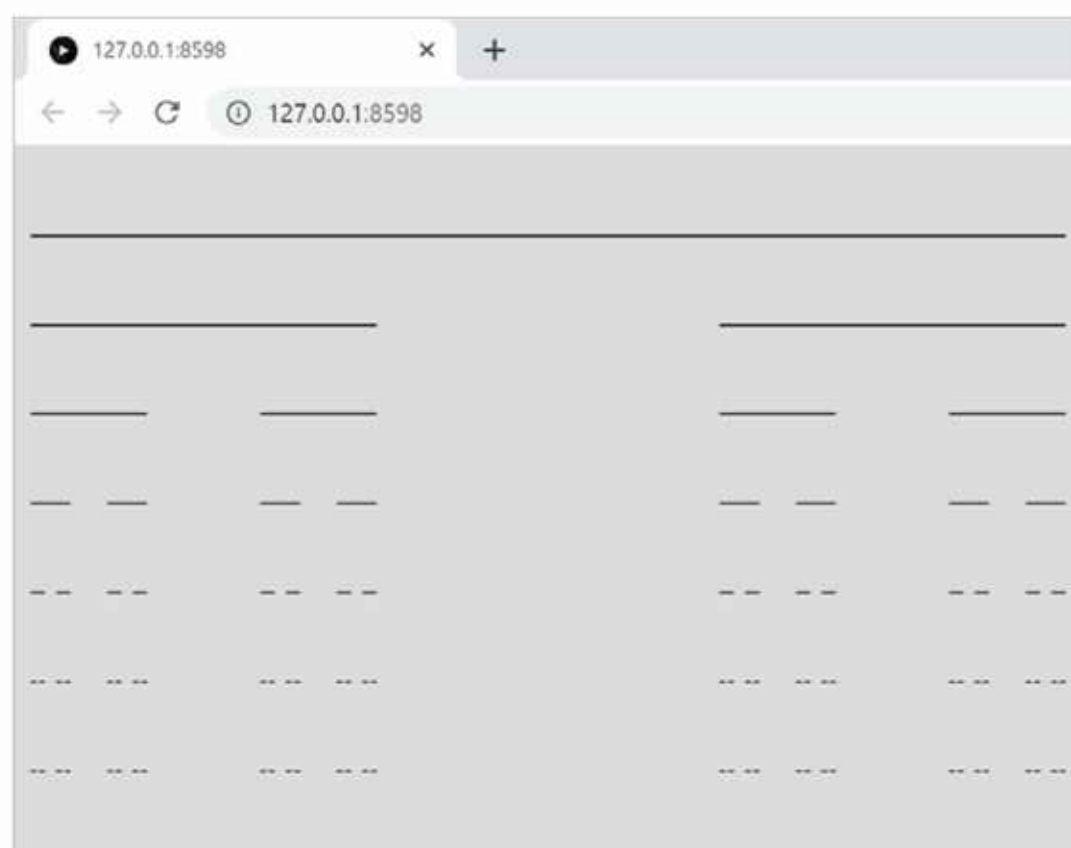
$$\begin{aligned}
 &1 \cdot \frac{1}{3} + 2 \cdot \frac{1}{9} + 4 \cdot \frac{1}{27} + \dots = \\
 &= \frac{1}{3} \cdot \sum_{n=1}^{\infty} \left(\frac{2}{3}\right)^{n-1} = \\
 &= \frac{1}{3} \cdot \frac{1}{1 - \frac{2}{3}} = 1
 \end{aligned}$$

Zatem część odcinka, która pozostaje, ma długość równą 0. Zbiór Cantora składa się z nieskończonej liczby punktów, nie zawiera żadnego odcinka.

Skrypty w środowisku Processing tworzy się tak samo jak w edytorze na stronie <https://p5js.org>. Oprócz skryptu generowany jest automatycznie plik *index.html*. Efekt działania skryptu wyświetli się w domyślnej przeglądarce internetowej. W tym celu tworzony jest lokalny serwer WWW. Rysunek 10.5 pokazuje okno środowiska Processing z fragmentem skryptu rysującego zbiory Cantora stopni od 0 do 6, a rysunek 10.6 – efekt uruchomienia skryptu w przeglądarce internetowej.



Rys. 10.5. Okno środowiska Processing z fragmentem skryptu rysującego zbiory Cantora stopni od 0 do 6



Rys. 10.6. Efekt uruchomienia skryptu rysującego zbiory Cantora stopni od 0 do 6

Zapamiętaj

Fraktal to figura, którą cechuje m.in. samopodobieństwo – mniejsze jej fragmenty są podobne do całości. Przykładem fraktala jest zbiór Cantora – podzbiór odcinka jednostkowego. Aby otrzymać ten zbiór, dzielimy odcinek jednostkowy na trzy równe części i usuwamy środkową. Następnie każdy z odcinków dzielimy na trzy równe części i usuwamy części środkowe. Proces dzielenia każdego z otrzymanych odcinków na trzy równe części i usuwania części środkowych jest nieskończony.

10.2. Płatek Kocha

Kolejnym przykładem fraktala, którym się zajmiemy, będzie tzw. płatek Kocha. Ponieważ jest to figura złożona z trzech krzywych Kocha, opowiemy najpierw, jak wygląda ta krzywa, i narysujemy jej przybliżenie.

Krzywa Kocha jest fraktalem, który można opisać następująco:

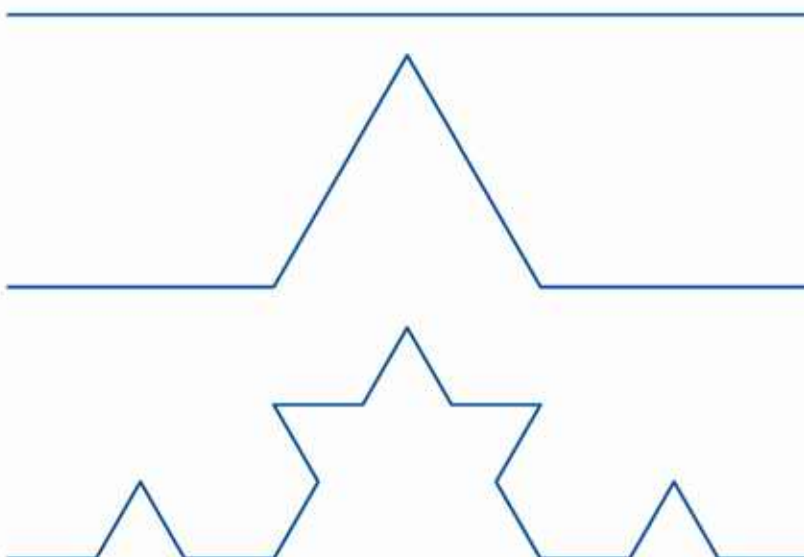
Krzywa Kocha

- ▶ krzywa Kocha stopnia 0 to odcinek określonej długości,
- ▶ krzywą Kocha stopnia 1 otrzymujemy z odcinka bazowego: dzielimy go na trzy równe części i zastępujemy środkową część fragmentem trójkąta równobocznego o długości boku takiej jak usuwana część odcinka – zostawiamy dwa boki trójkąta,
- ▶ krzywą Kocha wyższego stopnia tworzymy z krzywej Kocha stopnia o jeden mniejszego: dzielimy każdy jej odcinek na trzy równe części i zastępujemy środkową część fragmentem trójkąta równobocznego.

Proces dzielenia odcinków tworzących daną krzywą i zastępowania środkowych części fragmentem trójkąta równobocznego jest nieskończony, więc będziemy generować jedynie krzywe określonego stopnia. Krzywa Kocha stopnia n składa się czterech krzywych stopnia $n - 1$ dla $n > 0$. Krzywe Kocha stopni od 0 do 2 ilustruje rysunek 10.7.

Warto wiedzieć

W środowiskach języka C++ także można tworzyć programy z wykorzystaniem grafiki. Należy wtedy doinstalować jedną z bibliotek graficznych, np. **allegro**. Pisanie programów z użyciem tego typu bibliotek nie jest jednak intuicyjne i wymaga doświadczenia w posługiwaniu się wybraną biblioteką oraz środowiskiem.



Rys. 10.7. Krzywe Kocha stopni 0, 1 i 2 (w danej linii przedstawiona jest krzywa jednego stopnia)

Oto specyfikacja problemu rysowania krzywej Kocha stopnia n :

Specyfikacja

Dane: n – liczba całkowita nieujemna, stopień krzywej Kocha,
 x, y – liczby rzeczywiste, współrzędne punktu będącego początkiem krzywej,

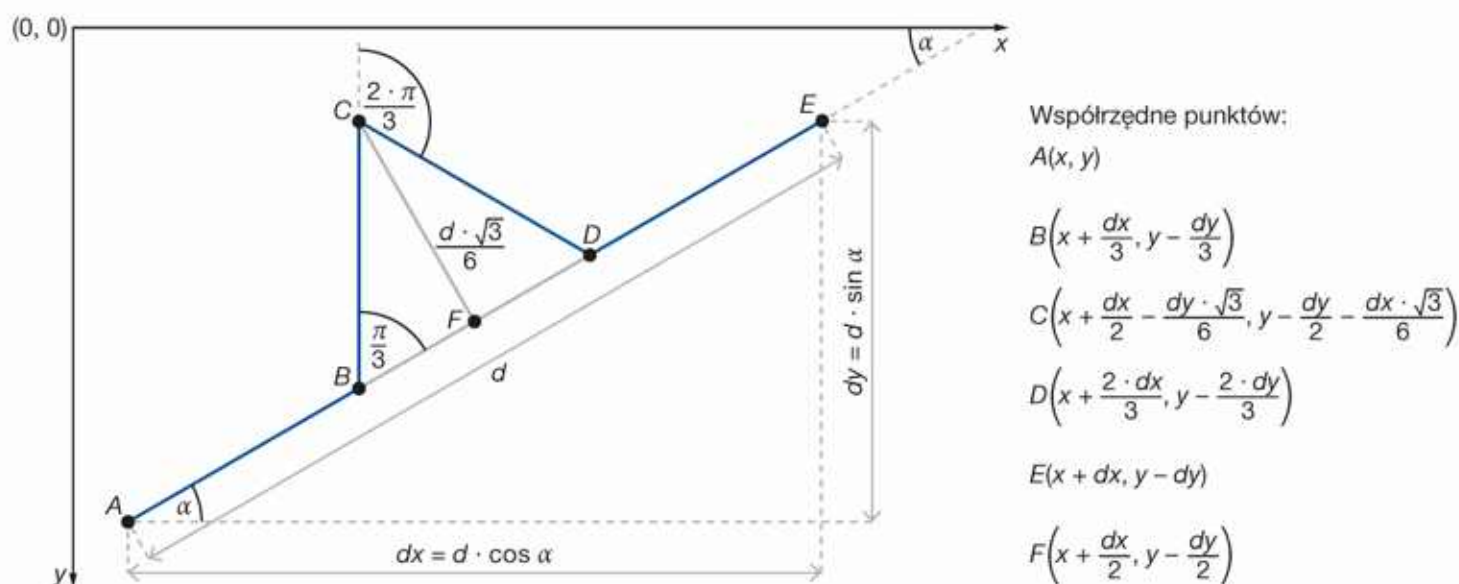
d – liczba rzeczywista, długość odcinka bazowego.

Wynik: rysunek krzywej Kocha stopnia n .

Warto wiedzieć

W językach programowania argumenty funkcji trygonometrycznych zazwyczaj są wyrażane w radianach. Kątowi pełnemu (360°) odpowiada miara łukowa 2π radianów.

Funkcja rekurencyjna rysująca krzywą Kocha danego stopnia będzie miała budowę podobną do funkcji tworzącej zbiór Cantora. Dla stopnia 0 narysuje ona odcinek o określonej długości. Dla stopnia większego od 0 wywoła cztery razy rekurencyjnie funkcję dla stopnia o jeden mniejszego. W tym przypadku kłopot może sprawić wyznaczenie współrzędnych punktów będących końcami krzywych stopnia o 1 mniejszego. W przypadku zbioru Cantora wszystkie odcinki były poziome, teraz krzywe mogą być nachylone pod pewnym kątem do osi poziomej. W obliczeniu współrzędnych ich końców pomoże rysunek 10.8.



Rys. 10.8. Sposób obliczania współrzędnych końców krzywych stopnia o 1 mniejszego

Zwróć uwagę, że w edytorze oś OY jest skierowana w dół, więc przy obliczaniu współrzędnych y -owych musimy odejmować odpowiednie wartości. Współrzędne punktu C można obliczyć w następujący sposób. Punkt C jest przesunięty względem punktu F (środka odcinka AE) o odległość $\frac{d \cdot \sqrt{3}}{6}$ (wysokość trójkąta równobocznego o długości boku $\frac{d}{3}$) i obrócony o kąt $\alpha + \frac{\pi}{2}$ względem osi OX . Punkt C znajduje się zatem na okręgu o środku w punkcie F i o promieniu $\frac{d \cdot \sqrt{3}}{6}$ i jest obrócony o kąt $\alpha + \frac{\pi}{2}$ względem osi OX . Ponieważ oś OY jest skierowana w dół, współrzędne punktu C są równe:

$$x_C = x_F + \frac{d \cdot \sqrt{3}}{6} \cdot \cos\left(\alpha + \frac{\pi}{2}\right) = x_F - \frac{d \cdot \sqrt{3}}{6} \cdot \sin \alpha = x_F - \frac{dy \cdot \sqrt{3}}{6}$$

$$y_C = y_F - \frac{d \cdot \sqrt{3}}{6} \cdot \sin\left(\alpha + \frac{\pi}{2}\right) = y_F - \frac{d \cdot \sqrt{3}}{6} \cdot \cos \alpha = y_F - \frac{dx \cdot \sqrt{3}}{6}$$

W funkcji rekurencyjnej rysującej krzywą Kocha wprowadzimy dodatkowy parametr α , określający kąt nachylenia krzywej do osi OX wyrażony w radianach. Zmienna $p3$ przechowuje wartość $\sqrt{3}$. Zapis funkcji Koch w pseudokodzie może być następujący:

```
p3 ← pierwiastek kwadratowy z 3
funkcja Koch(n,x,y,d,alfa)
  dx ← d * cos(alfa)
  dy ← d * sin(alfa)
  jeśli n = 0 to
    Odcinek(x,y,x+dx,y-dy)
  w przeciwnym przypadku
    Koch(n-1,x,y,d/3,alfa)
    Koch(n-1,x+dx/3,y-dy/3,d/3,alfa+pi/3)
    Koch(n-1,x+dx/2-dy*p3/6,y-dy/2-dx*p3/6,
          d/3,alfa-pi/3)
    Koch(n-1,x+2*dx/3,y-2*dy/3,d/3,alfa)
```

Przy wywołaniu funkcji Koch, rysującej krzywą poziomo, w miejsce parametru α podstawimy wartość 0. Jeśli stopień jest większy od zera, to krzywa jest złożona z czterech krzywych stopnia o jeden mniejszego. Pierwsza i ostatnia z tych krzywych są rysowane pod tym samym kątem α , druga krzywa jest obrócona o $\pi/3$ radianów względem pierwszej (czyli jest nachylona pod kątem $\alpha + \pi/3$ do osi OX), a trzecia jest obrócona w drugą stronę o kąt $2\pi/3$ względem drugiej krzywej (czyli znajduje się pod kątem $\alpha + \pi/3 - 2\pi/3 = \alpha - \pi/3$ do osi OX). Funkcję Koch możemy zapisać następująco:

```
1. function Koch(n,x,y,d,alfa) {
2.   var dx=d*cos(alfa);
3.   var dy=d*sin(alfa);
4.   if (n==0) {
5.     line(x,y,x+dx,y-dy);
6.   }
7.   else {
8.     Koch(n-1,x,y,d/3,alfa);
9.     Koch(n-1,x+dx/3,y-dy/3,d/3,alfa+PI/3);
10.    Koch(n-1,x+dx/2-dy*p3/6,y-dy/2-dx*p3/6,d/3,alfa-PI/3);
11.    Koch(n-1,x+2*dx/3,y-2*dy/3,d/3,alfa);
12.  }
13. }
```

Zmienne dx i dy są zmiennymi lokalnymi (dostępnymi tylko dla danej funkcji). W języku JavaScript pierwsze przypisanie wartości zmiennej lokalnej jest poprzedzone słowem kluczowym `var` (linie 2–3).

Dobra rada

$$\sin\left(\alpha + \frac{\pi}{2}\right) = \cos \alpha$$

$$\cos\left(\alpha + \frac{\pi}{2}\right) = -\sin \alpha$$

Dobra rada

Jeśli w algorytmie rekurencyjnym wykorzystujesz wartość, która nie zmienia się przy wywołaniu rekurencyjnym, oblicz tę wartość poza rekurencją. Dzięki temu wyznaczysz ją tylko raz, a nie wielokrotnie.

Warto wiedzieć

W każdym kolejnym etapie konstrukcji krzywej Kocha jej długość powiększa się o $\frac{1}{3}$ długości krzywej z poprzedniego etapu. Jeśli długość początkowego odcinka jest równa 1, to kolejne krzywe mają długości: $1, \frac{4}{3}, \left(\frac{4}{3}\right)^2, \left(\frac{4}{3}\right)^3, \dots$. Długość krzywej Kocha jest zatem nieskończona: $\lim_{n \rightarrow \infty} \left(\frac{4}{3}\right)^n = \infty$

Funkcja rysująca krzywą Kocha stopnia n

Typu zmiennej nie deklarujemy. Jest on określany na podstawie przypisanej wartości i może być zmieniany podczas działania skryptu. Wartość zmiennej `p3` warto przypisać poza funkcją rekurencyjną (aby wielokrotnie nie obliczać wartości pierwiastka kwadratowego z 3) i używać jej jako zmiennej globalnej. Przy pierwszym przypisaniu wartości zmiennej globalnej nie piszemy słowa kluczowego `var`. Wartość tej zmiennej można określić w funkcji `setup`.

Zapis funkcji `setup` i `draw` w skrypcie rysującym krzywą Kocha stopnia 4 może wyglądać następująco:

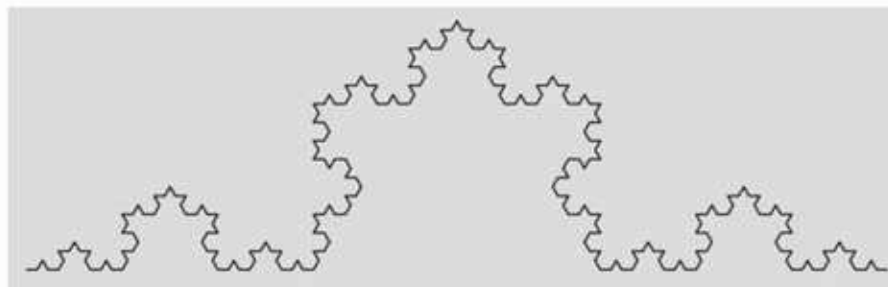
Fragment skryptu 
rysującego krzywą
Kocha stopnia 4 –
przykład definicji funkcji
`setup` i `draw`

```
1. function setup() {
2.   createCanvas(600, 400);
3.   p3=sqrt(3);
4. }
5.
6. function draw() {
7.   background(220);
8.   Koch(4,50,300,500,0);
9.   noLoop();
10. }
```

Dobra rada

Pamiętaj, że długość rysowanego odcinka maleje trzykrotnie wraz z powiększeniem stopnia o 1. Dlatego stopień krzywej i długość odcinka bazowego dobieraj tak, aby długość rysowanego odcinka nie była mniejsza od 1 (1 piksela).

W funkcji `draw` wywołana jest funkcja `Koch`, która rysuje krzywą stopnia 4 o długości odcinka bazowego 500 z punktu początkowego o współrzędnych (50, 300) pod kątem nachylenia 0 radianów do osi *OX*. Efekt działania skryptu ilustruje rysunek 10.9.



Rys. 10.9. Efekt działania skryptu rysującego krzywą Kocha stopnia 4

Ćwiczenie 3

Napisz i uruchom skrypt rysujący krzywą Kocha. Przetestuj działanie skryptu dla różnych stopni krzywej.

Jeśli w trójkącie równobocznym każdy z boków zastąpimy krzywą Kocha, otrzymamy **płatki Kocha** . Fraktal ten przypomina płatki śniegu. Danymi, oprócz stopnia figury oraz długości boku trójkąta, mogą być współrzędne środka ciężkości trójkąta.

Specyfikacja problemu rysowania płatka Kocha może być następująca.

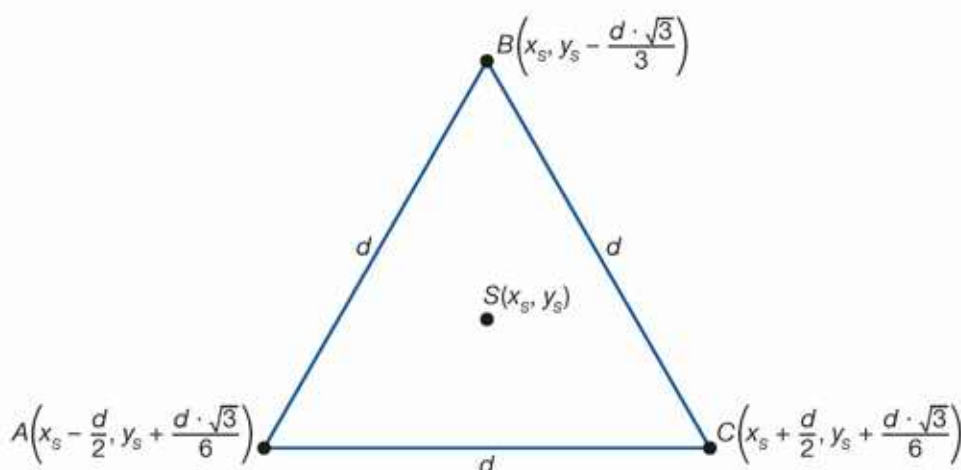
Specyfikacja

Dane: n – liczba całkowita nieujemna, stopień płatka Kocha,
 x_S, y_S – liczby rzeczywiste, współrzędne środka ciężkości trójkąta równobocznego,

d – liczba rzeczywista, długość boku trójkąta równobocznego.

Wynik: rysunek płatka Kocha stopnia n .

Współrzędne wierzchołków trójkąta równobocznego w zależności od współrzędnych środka ciężkości są takie jak na rysunku 10.10.



Rys. 10.10. Współrzędne wierzchołków trójkąta w zależności od współrzędnych jego środka ciężkości

Funkcję rysującą płatek Kocha można zapisać w pseudokodzie następująco:

funkcja PłatekKocha(n, x_S, y_S, d)
 Koch($n, x_S - d/2, y_S + d \cdot \sqrt{3}/6, d, \pi/3$)
 Koch($n, x_S, y_S - d \cdot \sqrt{3}/3, d, -\pi/3$)
 Koch($n, x_S + d/2, y_S + d \cdot \sqrt{3}/6, d, \pi$)

A to ciekawe

Samopodobieństwo w muzyce

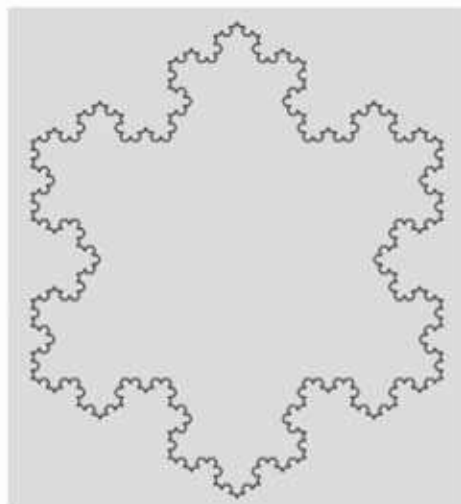
Okazuje się, że w wielu utworach muzycznych można wskazać fragmenty podobne do całości. Według badań cechę tę posiadają np. wybrane dzieła Jana Sebastiana Bacha czy Wolfganga Amadeusza Mozarta. Niektórzy nawet twierdzą, że stanowi to o atrakcyjności tych utworów. Obecnie algorytmy oparte na własnościach fraktali wykorzystuje się przy komponowaniu muzyki za pomocą komputera.



Zwróć uwagę, że jeden wierzchołek nie może być punktem początkowym dwóch krzywych – wówczas jedna z krzywych rysowana byłaby do wnętrza trójkąta, a nie na zewnątrz. Dlatego punktem początkowym trzeciego wywołania funkcji Koch jest punkt *C* (rys. 10.10, s. 177), a krzywa jest rysowana pod kątem równym π . Rysunek 10.11 przedstawia płatek Kocha stopnia 4.

Warto wiedzieć

Płatek Kocha przypominający śnieżynkę jest jednym z najbardziej popularnych fraktali. Opisał go szwedzki matematyk Helge von Koch w 1906 r. Płatek Kocha ma skończone pole, ale jest ograniczony krzywą o nieskończonej długości.



Rys. 10.11. Płatek Kocha stopnia 4

Ćwiczenie 4

Napisz i uruchom skrypt rysujący płatek Kocha. Przetestuj działanie skryptu dla różnych stopni figury.

Zapamiętaj

Aby utworzyć krzywą Kocha, najpierw dzieli się wybrany odcinek na trzy równe części i zastępuje środkową część dwoma bokami trójkąta równobocznego o długości boku takiej jak usuwana część odcinka. Następnie postępuje się w ten sam sposób dla każdego z otrzymanych odcinków. Proces dzielenia odcinków na trzy równe części i zastępowania części środkowych fragmentem trójkąta równobocznego jest nieskończony. Płatek Kocha powstaje przez zastąpienie każdego z boków trójkąta równobocznego krzywą Kocha.

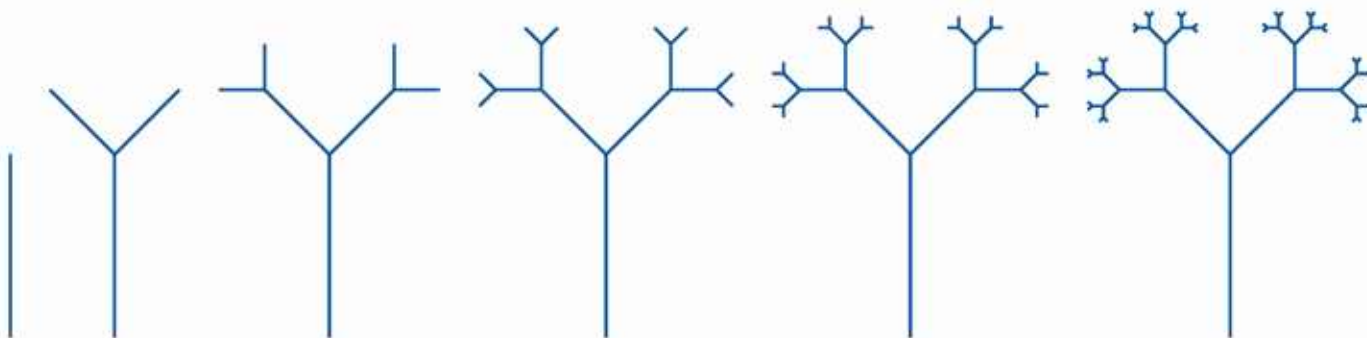
10.3. Drzewo binarne

Drzewo binarne  Przedstawimy teraz fraktal nazywany **drzewem binarnym**. Drzewo binarne stopnia 0 to odcinek o wybranej długości. Drzewo binarne stopnia n ($n > 0$) składa się z odcinka wyjściowego (nazwiemy go pniem drzewa) oraz dwóch pomniejszonych dwa razy drzew binarnych stopnia $n - 1$. Ich początkiem jest koniec odcinka wyjściowego i są one obrócone odpowiednio w lewo oraz prawo pod kątem 45° .

Rysunek 10.12 przedstawia drzewa binarne stopni 0, 1, 2, 3, 4 i 5.

Warto wiedzieć

Kąt między pniem drzewa a dołączanym do niego drzewem binarnym może być inny niż 45° .



Rys. 10.12. Drzewa binarne stopni 0, 1, 2, 3, 4 i 5

Specyfikacja problemu rysowania drzewa binarnego stopnia n ($n \geq 0$) jest następująca:

Specyfikacja

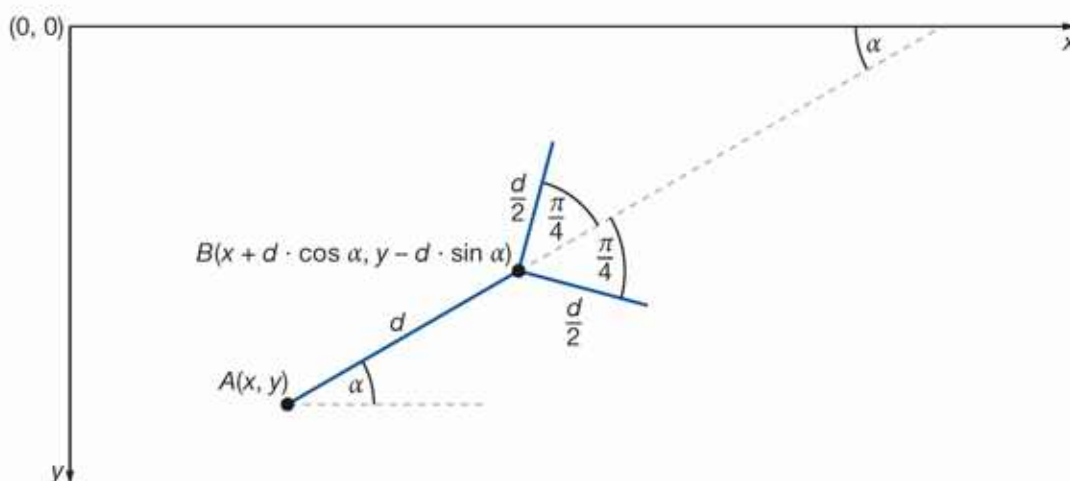
Dane: n – liczba całkowita nieujemna, stopień drzewa binarnego,
 x, y – liczby rzeczywiste, współrzędne punktu początkowego, z którego będzie rysowane drzewo,
 d – liczba rzeczywista, długość odcinka bazowego (pnia).

Wynik: rysunek drzewa binarnego stopnia n .

Warto wiedzieć

Drzewem binarnym nazywamy także strukturę danych – szczególny przypadek grafu skierowanego, w którym z wierzchołków grafu wychodzą maksymalnie dwie krawędzie tworzące rozłączne poddrzewa (nie istnieje żadna krawędź prowadząca od wierzchołka lewego poddrzewa do wierzchołka prawego poddrzewa). Algorytmy przetwarzające drzewa są zazwyczaj rekurencyjne.

W funkcji rekurencyjnej rysującej drzewo binarne wprowadzimy dodatkowy parametr określający kąt nachylenia drzewa do osi OX wyrażony w radianach. Ilustruje to rysunek 10.13.



Rys. 10.13. Współrzędne punktów oraz kąty w drzewie binarnym

Oto funkcja rysująca drzewo binarne zapisana w pseudokodzie:

```
funkcja DrzewoBinarne( $n, x, y, d, \text{alfa}$ )
   $x1 \leftarrow x + d * \cos(\text{alfa})$ 
   $y1 \leftarrow y - d * \sin(\text{alfa})$ 
  Odcinek( $x, y, x1, y1$ )
  jeśli  $n > 0$  to
    DrzewoBinarne( $n-1, x1, y1, d/2, \text{alfa} + \pi/4$ )
    DrzewoBinarne( $n-1, x1, y1, d/2, \text{alfa} - \pi/4$ )
```


Dobra rada

Stopień drzewa i długość odcinka początkowego dobieraj tak, aby żaden rysowany odcinek nie był dłuższy niż 1 (1 piksel).

Obliczamy współrzędne końca odcinka o długości d nachylonego do osi OX pod kątem α (zmienne $x1$ i $y1$). Następnie rysujemy odcinek o końcach w punktach (x, y) i $(x1, y1)$. Jeśli stopień drzewa jest większy od zera, następują dwa wywołania rekurencyjne – dla lewego i prawego poddrzewa stopnia o jeden mniejszego. Poddrzewa o długości odcinka $d/2$ wychodzą z punktu $(x1, y1)$ i są obrócone odpowiednio o $\pi/4$ w lewo lub prawo.

Ćwiczenie 5

Napisz i uruchom skrypt rysujący drzewo binarne. Wykorzystaj w nim omówioną w temacie funkcję `DrzewoBinarne`. Odcinek wyjściowy (pień) powinien być prostopadły do osi poziomej. Przykładowe wywołanie funkcji `DrzewoBinarne` w funkcji `draw` może być następujące: `DrzewoBinarne(5, 300, 400, 200, PI/2)`. Przetestuj działanie skryptu dla różnych stopni figury.

Zapamiętaj

Drzewo binarne stopnia 0 to odcinek o wybranej długości. Drzewo binarne stopnia n ($n > 0$) składa się z odcinka wyjściowego oraz dwóch pomniejszonych dwa razy drzew binarnych stopnia $n - 1$. Ich początkiem jest koniec wyjściowego odcinka i są one obrócone odpowiednio w lewo lub prawo pod kątem 45° . Do rysowania drzewa binarnego wygodnie jest wykorzystać rekurencję.

10.4. Dywan Sierpińskiego**Dywan Sierpińskiego****Warto wiedzieć**

W kolejnych etapach konstrukcji dywanu Sierpińskiego z każdego z kwadratów usuwamy środkowy kwadrat o polu równym $\frac{1}{9}$ całego kwadratu. Sumę pól usuwanych kwadratów można obliczyć następująco:

$$1 \cdot \frac{1}{9} + 8 \cdot \frac{1}{9^2} + 8^2 \cdot \frac{1}{9^3} + \dots = \frac{1}{9} \cdot \sum_{n=1}^{\infty} \left(\frac{8}{9}\right)^{n-1} = 1$$

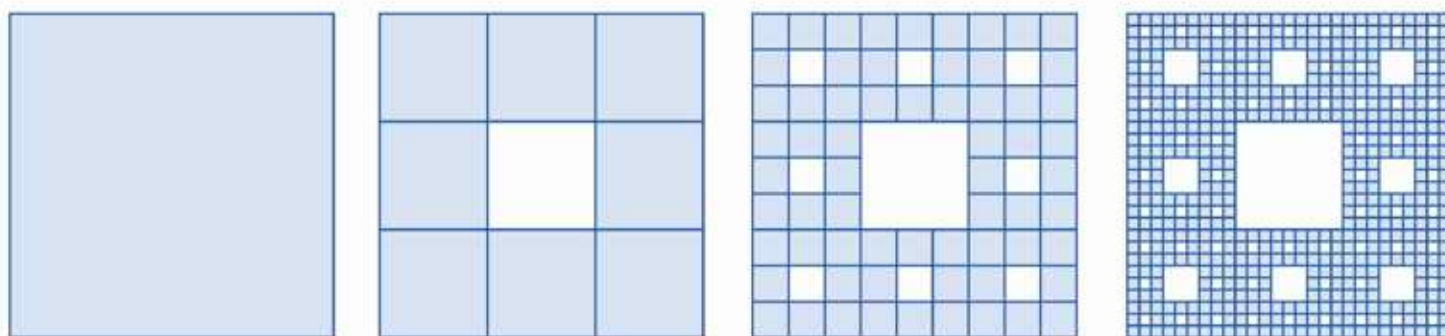
Zatem pole powierzchni dywanu Sierpińskiego jest równe 0.

Dotychczas przedstawione fraktale bazowały na odcinkach. Teraz przedstawimy **dywan Sierpińskiego** – fraktal, którego przybliżenie można utworzyć z kwadratów. Dywan Sierpińskiego to podzbiór pewnych punktów (x, y) jednostkowego kwadratu, gdzie $x, y \in [0; 1]$. Można go opisać w następujący sposób:

- ▶ dywan Sierpińskiego stopnia 0 to kwadrat o boku długości 1,
- ▶ dywan Sierpińskiego stopnia 1 otrzymujemy w następujący sposób: dzielimy kwadrat jednostkowy na dziewięć identycznych kwadratów i usuwamy środkowy kwadrat,
- ▶ dywan Sierpińskiego wyższego stopnia tworzymy z dywanu Sierpińskiego stopnia o jeden mniejszego: dzielimy każdy kwadrat na dziewięć identycznych kwadratów i usuwamy środkowy.

Proces dzielenia kwadratów tworzących dywan Sierpińskiego i usuwania środkowego kwadratu jest nieskończony. Dlatego będziemy generować dywan Sierpińskiego określonego stopnia. Dywan Sierpińskiego stopnia n składa się z ośmiu dywanów stopnia $n - 1$ dla $n > 0$.

Rysunek 10.14 przedstawia poglądowe rysunki dywanów Sierpińskiego stopni od 0 do 3.



Rys. 10.14. Dywany Sierpińskiego stopni 0, 1, 2 i 3

Podczas rysowania dywanu Sierpińskiego stopnia n przyjmiemy, podobnie jak przy **zbiorze Cantora**, że długość boku kwadratu jest większa od 1. Specyfikacja problemu może być następująca:

Zbiór Cantora,
s. 165 [🔗](#)

Specyfikacja

Dane: n – liczba całkowita nieujemna, stopień dywanu Sierpińskiego,
 x, y – liczby rzeczywiste, współrzędne lewego górnego rogu dywanu,
 a – liczba rzeczywista, długość boku dywanu.

Wynik: rysunek dywanu Sierpińskiego stopnia n .

Jeśli stopień jest równy 0, należy narysować kwadrat o określonej długości boku. W przeciwnym przypadku tworzymy osiem dywanów stopnia o 1 mniejszego, a więc należy osiem razy wywołać rekurencyjnie tworzenie dywanu o boku trzy razy krótszym. Dobór współrzędnych punktów początkowych (lewych górnych rogów dywanów stopnia $n - 1$) ilustruje rysunek 10.15 na s. 182.

Warto wiedzieć

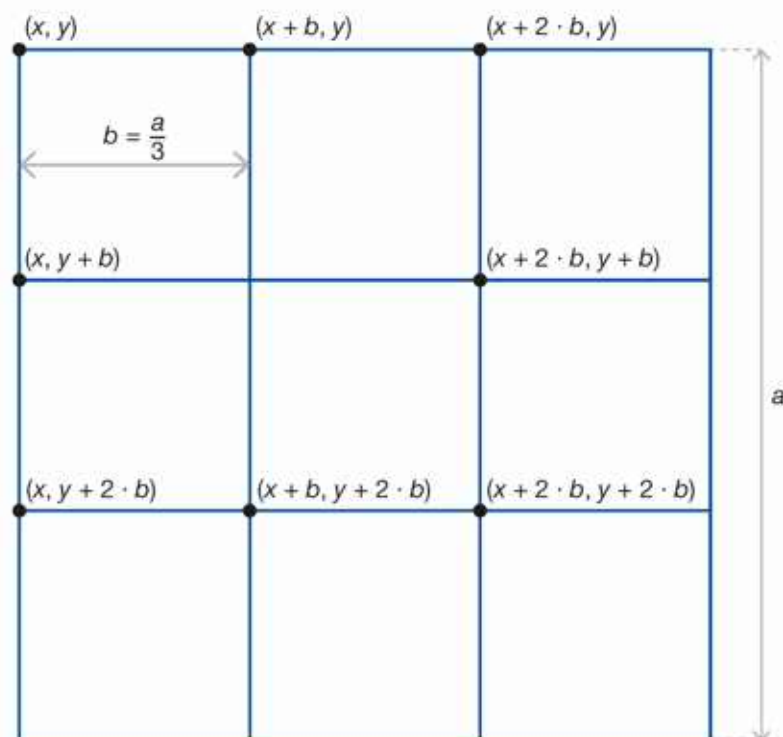
Opis dywanu Sierpińskiego podał polski matematyk Wacław F. Sierpiński (1882–1969). Był on twórcą tzw. warszawskiej szkoły matematycznej i specjalizował się w teorii liczb oraz teorii mnogości.

A to ciekawe

Fraktale w architekturze

Szczególny wzrost zainteresowania fraktalami w różnych dziedzinach życia nastąpił – dzięki Benoît Mandelbrotowi – w drugiej połowie XX w. Figurami samopodobnymi zafascynowali się wtedy także architekci. Okazuje się jednak, że na długo zanim zaczęto używać słowa „fraktal”, powstały budowle o cechach fraktalnych. Elementy takiego budynku wyglądają jak cały obiekt w pomniejszeniu. Cech fraktalnych możemy się doszukać np. w indonezyjskiej świątyni Prambanan pochodzącej z IX w. (na zdjęciu).





Rys. 10.15. Współrzędne lewych górnych rogów dywanów stopnia $n-1$ tworzących dywan stopnia n

👍 Dobra rada

Pamiętaj, że długość boku rysowanego kwadratu maleje trzykrotnie wraz z powiększeniem stopnia o 1. Dlatego stopień dywanu i długość boku kwadratu bazowego dobieraj tak, aby długość boku rysowanego kwadratu nie była mniejsza od 1 (1 piksela).

Funkcję rekurencyjną tworzącą dywan Sierpińskiego stopnia n zgodnie z podaną specyfikacją można zapisać w pseudokodzie tak jak poniżej. Funkcja Kwadrat rysuje kwadrat o określonej długości boku, zaczynając w jego lewym górnym rogu.

```
funkcja Sierpinski( $n, x, y, a$ )
  jeśli  $n = 0$  to Kwadrat( $x, y, a$ )
  w przeciwnym przypadku
     $b \leftarrow a/3$ 
    dla  $i \leftarrow 0, 1, 2$  wykonuj
      Sierpinski( $n-1, x+i*b, y, a/2$ )
    dla  $i \leftarrow 0, 1$  wykonuj
      Sierpinski( $n-1, x+2*i*b, y+b, a/2$ )
    dla  $i \leftarrow 0, 1, 2$  wykonuj
      Sierpinski( $n-1, x+i*b, y+2*b, a/2$ )
```

Definicja funkcji Sierpinski może być następująca:

Funkcja rysująca dywan Sierpińskiego stopnia n

```
1. function Sierpinski( $n, x, y, a$ ) {
2.   if ( $n==0$ ) {
3.     square( $x, y, a$ );
4.   }
5.   else {
6.     var  $b=a/3, i$ ;
7.     for ( $i=0; i<3; i++$ ) {Sierpinski( $n-1, x+i*b, y, b$ );}
8.     for ( $i=0; i<2; i++$ ) {Sierpinski( $n-1, x+2*i*b, y+b, b$ );}
9.     for ( $i=0; i<3; i++$ ) {Sierpinski( $n-1, x+i*b, y+2*b, b$ );}
10.  }
11. }
```

W linii 3 dla stopnia 0 wywołana jest **funkcja square**, rysująca kwadrat. Przy użyciu tej funkcji należy podać minimum trzy parametry: dwa pierwsze parametry określają położenie lewego górnego rogu kwadratu, trzeci – długość boku kwadratu. Kolor wypełnienia figury można ustalić za pomocą **funkcji fill**, której parametrem jest np. liczba z zakresu od 0 do 255 określająca odcień szarości (podobnie jak w funkcji **background**). Można też podać trzy parametry liczbowe z zakresu od 0 do 255 określające kolor wg modelu RGB. Kolor wypełnienia kwadratu można ustalić w funkcji **setup**.

• **Funkcja square**

• **Funkcja fill**

Dobra rada

Kolor rysowanych krawędzi (także odcinków rysowanych przy użyciu funkcji **line**) możesz ustalić poleceniem **stroke**. Jeśli nie chcesz rysować krawędzi figur, można je wyłączyć za pomocą bezparametrowej funkcji **noStroke**.

Ćwiczenie 6

Napisz i uruchom skrypt rysujący dywan Sierpińskiego. Przykładowe wywołanie funkcji **Sierpinski** w funkcji **draw** może być następujące: **Sierpinski(3,0,0,400)**. Przetestuj działanie skryptu dla różnych stopni figury. Ustal kolor wypełnienia.

Zapamiętaj

Aby utworzyć dywan Sierpińskiego, na początku dzieli się kwadrat jednostkowy na dziewięć identycznych kwadratów i usuwa środkowy kwadrat. W kolejnych etapach postępuje się w ten sam sposób dla wszystkich kwadratów z poprzedniego etapu.

10.5. Przekształcenia afiniczne

Znasz już metodę rekurencyjnego generowania fraktali. Nie jest to jednak jedyny sposób – zbiór punktów przybliżających fraktal można także generować z wykorzystaniem przekształceń afinicznych.

Przekształcenie afiniczne odwzorowuje punkt (x, y) na płaszczyźnie na punkt (x', y') , którego współrzędne są równe:

• **Przekształcenie afiniczne**

$$\begin{cases} x' = a_1 \cdot x + b_1 \cdot y + c_1 \\ y' = a_2 \cdot x + b_2 \cdot y + c_2 \end{cases}$$

gdzie $a_1, b_1, c_1, a_2, b_2, c_2$ – liczby rzeczywiste.

Przykładami przekształceń afinicznych są m.in. obrót, przesunięcie (translacja) czy skalowanie, a także złożenie tych przekształceń. W budowie fraktali możemy obserwować złożenie skalowania, obrotu i przesunięcia. Niech s określa współczynnik skalowania, α – kąt obrotu, t_1 i t_2 – współrzędne wektora translacji. Wówczas przekształcenie afiniczne będące złożeniem skalowania, obrotu i translacji wyrażone jest wzorami:

$$\begin{cases} x' = s \cdot (\cos \alpha) \cdot x - s \cdot (\sin \alpha) \cdot y + t_1 \\ y' = s \cdot (\sin \alpha) \cdot x + s \cdot (\cos \alpha) \cdot y + t_2 \end{cases}$$

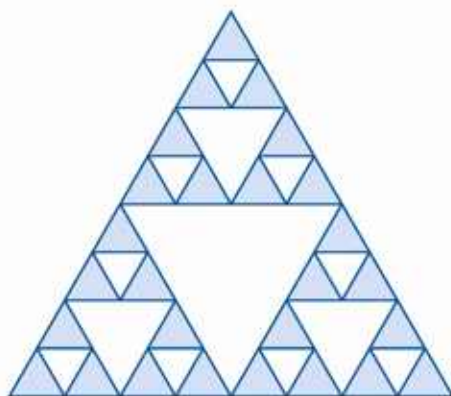
Warto wiedzieć

Można określić więcej parametrów przekształceń, np. inną skalę dla każdej z osi układu.

Jeśli iterujemy wielokrotnie powyższe przekształcenie, tzn. wynik jednego przekształcenia staje się argumentem kolejnego przekształcenia, otrzymujemy ciąg punktów, których większość (z ewentualnym pominięciem początkowego fragmentu) może tworzyć przybliżenie fraktala. Za punkt początkowy można przyjąć dowolny punkt płas-

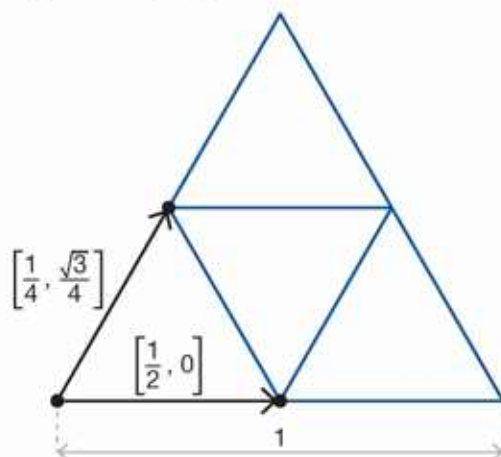
Metoda IFS • czynny. Taki sposób postępowania nazywa się **metodą IFS** (ang. *iterated function system* – iterowany układ funkcyjny).

Wykorzystamy przekształcenia afiniczne do utworzenia przybliżenia **Trójkąt Sierpińskiego** • trójkąta Sierpińskiego. **Trójkąt Sierpińskiego** stopnia 0 to trójkąt równoboczny. Aby uzyskać trójkąt Sierpińskiego stopnia 1, dzielimy trójkąt na cztery identyczne trójkąty równoboczne i usuwamy środkowy trójkąt. Trójkąt Sierpińskiego wyższego stopnia powstaje z trójkąta Sierpińskiego stopnia o jeden mniejszego: każdy z trójkątów tworzących figurę dzieli się na cztery identyczne trójkąty równoboczne i usuwa środkowy trójkąt. Rysunek 10.16 przedstawia trójkąt Sierpińskiego stopnia 3.



Rys. 10.16. Trójkąt Sierpińskiego stopnia 3

Trójkąt Sierpińskiego stopnia n składa się z trzech trójkątów Sierpińskiego stopnia $n - 1$. Te trzy trójkąty są dwa razy mniejsze od trójkąta stopnia n , a więc współczynnik skalowania jest równy $\frac{1}{2}$. Figury nie są obrócone, czyli kąt obrotu jest równy 0 radianów. Jeśli bok trójkąta ma długość 1, poszczególne wektory przesunięcia mają współrzędne $[0, 0]$, $[\frac{1}{2}, 0]$ i $[\frac{1}{4}, \frac{\sqrt{3}}{4}]$. Ilustruje to rysunek 10.17.



Rys. 10.17. Wektory przesunięcia w trójkącie Sierpińskiego

Mamy więc trzy przekształcenia afiniczne różniące się wektorem przesunięcia:

$$\begin{cases} x' = \frac{1}{2} \cdot x + 0 \cdot y + 0 \\ y' = 0 \cdot x + \frac{1}{2} \cdot y + 0 \end{cases}$$

$$\begin{cases} x' = \frac{1}{2} \cdot x + 0 \cdot y + \frac{1}{2} \\ y' = 0 \cdot x + \frac{1}{2} \cdot y + 0 \end{cases}$$

$$\begin{cases} x' = \frac{1}{2} \cdot x + 0 \cdot y + \frac{1}{4} \\ y' = 0 \cdot x + \frac{1}{2} \cdot y + \frac{\sqrt{3}}{4} \end{cases}$$

Przy każdej iteracji wybierzemy losowo z takim samym prawdopodobieństwem jedno z trzech przekształceń i obliczymy współrzędne kolejnego punktu zgodnie z wylosowanym przekształceniem. Proces ten powtórzymy kilka tysięcy razy.

Do symulacji opisanego algorytmu można wykorzystać arkusz kalkulacyjny i zilustrować przybliżenie fraktala na wykresie punktowym. Rysunek 10.18 przedstawia fragment takiej symulacji.

	A	B	C
1			
2	Współczynniki w przekształceniu 1:		
3	0,5	0	0
4	0	0,5	0
5	Współczynniki w przekształceniu 2:		
6	0,5	0	0,5
7	0	0,5	0
8	Współczynniki w przekształceniu 3:		
9	0,5	0	0,25
10	0	0,5	0,433012702
11			
12	Numer wylosowanego przekształcenia	x	y
13		1	1
14	3	0,75	0,933012702
15	3	0,625	0,899519053
16	3	0,5625	0,882772228
17	2	0,78125	0,441386114
18	1	0,390625	0,220693057
19	1	0,1953125	0,110346529
20	2	0,59765625	0,055173264

Rys. 10.18. Fragment arkusza kalkulacyjnego z symulacją przybliżenia trójkąta Sierpińskiego metodą IFS

Komórki w kolumnach A, B i C i wierszach 3–4, 6–7 i 9–10 zawierają współczynniki przekształceń afinicznych. W komórce C10 znajduje się wartość $\frac{\sqrt{3}}{4}$ (=PIERWIASTEK(3)/4). W wierszu 13 określone są wartości początkowe x i y . W wierszu 14 zostały zdefiniowane formuły:

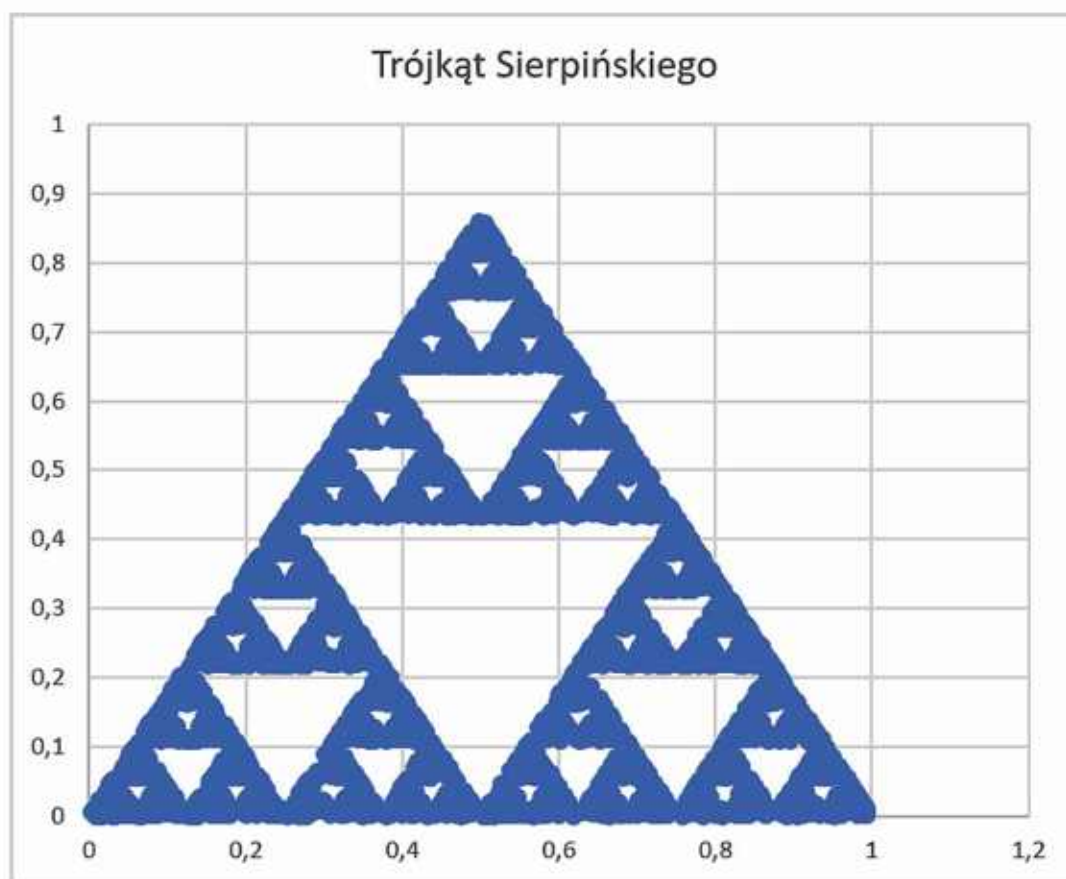
A14=LOS.ZAKR(1;3)

B14=WYBIERZ(A14;\$A\$3*B13+\$B\$3*C13+\$C\$3;
\$A\$6*B13+\$B\$6*C13+\$C\$6;
\$A\$9*B13+\$B\$9*C13+\$C\$9)

C14=WYBIERZ(A14;\$A\$4*B13+\$B\$4*C13+\$C\$4;
\$A\$7*B13+\$B\$7*C13+\$C\$7;
\$A\$10*B13+\$B\$10*C13+\$C\$10)

Kolejne wiersze w kolumnach A, B i C to powielone i odpowiednio zmodyfikowane formuły z wiersza 14.

Rysunek 10.19 przedstawia wykres punktowy wynikający z obliczeń w danej symulacji. Został utworzony na podstawie kolejnych 5000 punktów z pominięciem pierwszych 100.



Rys. 10.19. Wykres punktowy ilustrujący przybliżenie trójkąta Sierpińskiego w arkuszu kalkulacyjnym

Ćwiczenie 7

Utwórz arkusz kalkulacyjny, w którym przybliżysz za pomocą wykresu punktowego trójkąt Sierpińskiego z wykorzystaniem metody IFS.

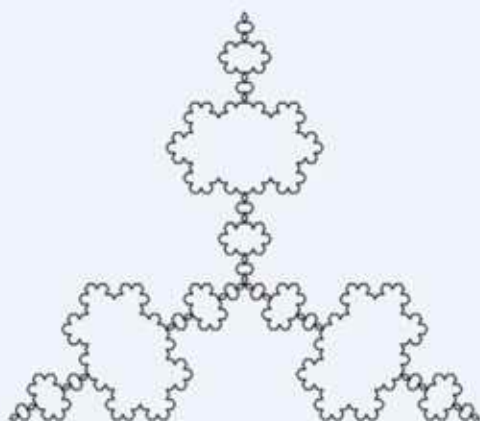
Podsumowanie

- Fraktal to figura, której cechą jest m.in. samopodobieństwo, czyli fragmenty tej figury są podobne do całości.
- Przykładami fraktali są: zbiór Cantora, krzywa Kocha i płatek Kocha, drzewo binarne, dywan Sierpińskiego oraz trójkąt Sierpińskiego.
- Zbiór Cantora stopnia 0 to odcinek o długości 1. Aby otrzymać zbiór stopnia 1, dzieli się odcinek na trzy równe części i usuwa część środkową. Zbiór Cantora wyższego stopnia powstaje ze zbioru Cantora stopnia o jeden mniejszego: każdy odcinek tworzący zbiór dzieli się na trzy równe części i usuwa część środkową.
- Krzywa Kocha stopnia 0 to odcinek o określonej długości. Aby otrzymać krzywą stopnia 1, dzielimy odcinek na trzy równe części i zastępujemy środkową część fragmentem trójkąta równobocznego o długości boku takiej jak usuwana część odcinka – zostawiamy dwa boki trójkąta. Krzywą Kocha wyższego stopnia tworzymy z krzywej Kocha stopnia o jeden mniejszego: dzielimy każdy jej odcinek na trzy równe części i zastępujemy środkową część fragmentem trójkąta równobocznego.
- Płatek Kocha powstaje przez zastąpienie każdego z boków trójkąta równobocznego krzywą Kocha.
- Drzewo binarne stopnia 0 to odcinek określonej długości. Drzewo binarne stopnia n ($n > 0$) składa się z wyjściowego odcinka oraz dwóch pomniejszych dwa razy drzew binarnych stopnia $n - 1$. Ich początkiem jest koniec wyjściowego odcinka i są one obrócone odpowiednio w lewo oraz prawo pod kątem 45° .
- Dywan Sierpińskiego stopnia 0 to kwadrat o jednostkowej długości boku. Aby uzyskać dywan Sierpińskiego stopnia 1, dzielimy kwadrat na dziewięć kwadratów o bokach trzy razy krótszych i usuwamy środkowy kwadrat. Dywan Sierpińskiego wyższego stopnia tworzymy z dywanu Sierpińskiego stopnia o jeden mniejszego, dzieląc każdy kwadrat na dziewięć równych części i usuwając środkową część.
- Trójkąt Sierpińskiego stopnia 0 to trójkąt równoboczny. Aby uzyskać trójkąt Sierpińskiego stopnia 1, dzieli się trójkąt stopnia 0 na cztery identyczne trójkąty równoboczne i usuwa środkowy trójkąt. Trójkąt Sierpińskiego wyższego stopnia powstaje z trójkąta Sierpińskiego stopnia o jeden mniejszego: każdy z trójkątów tworzących figurę dzieli się na cztery identyczne trójkąty równoboczne i usuwa środkowy trójkąt.
- Do tworzenia rysunków przybliżeń wybranych fraktali wygodnie jest zastosować rekurencję.
- Przybliżenia fraktali można także generować metodą IFS.

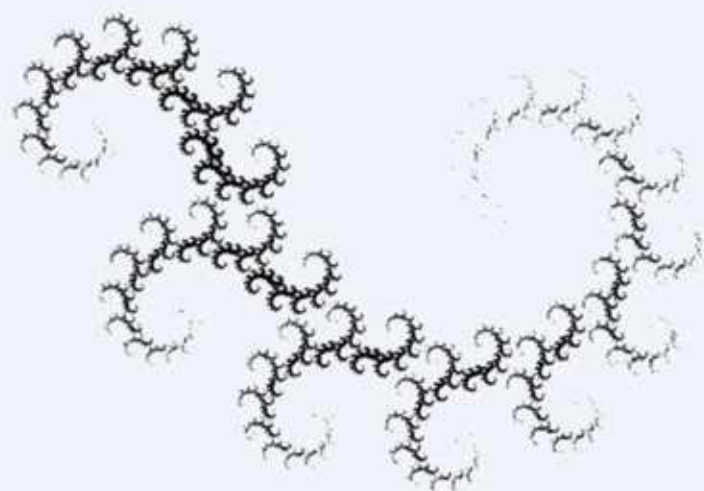
Zadania

- ★ **1** Przygotuj prezentację dotyczącą fraktali oraz ich współczesnych zastosowań.
- ★ **2** Utwórz stronę internetową ze skryptem rysującym dowolny fraktal. Na stronie powinny się znajdować pole wyboru stopnia fraktala oraz przycisk uruchamiający rysowanie fraktala wybranego stopnia.

- ★ **3** Omówiony w temacie skrypt rysujący płatek Kocha stopnia n zmodyfikuj tak, aby powstała figura taka jak na poniższym rysunku. Rysunek przedstawia fraktal stopnia 4.



- ★★ **4** Napisz skrypt w JavaScript z wykorzystaniem biblioteki p5.js rysujący trójkąt Sierpińskiego stopnia n z zastosowaniem rekurencji.
- ★★ **5** Napisz skrypt w JavaScript z wykorzystaniem biblioteki p5.js rysujący przybliżenie trójkąta Sierpińskiego z zastosowaniem metody IFS.
- ★★ **6** Utwórz arkusz kalkulacyjny, w którym za pomocą wykresu punktowego zilustrujesz dywan Sierpińskiego. Zastosuj metodę IFS.
Wskazówka: Musisz zdefiniować osiem przekształceń afinicznych.
- ★★ **7** Napisz skrypt w JavaScript z wykorzystaniem biblioteki p5.js rysujący przybliżenie fraktala nazywanego smokiem Hartera–Heighwaya (rysunek poniżej). Zastosuj metodę IFS.

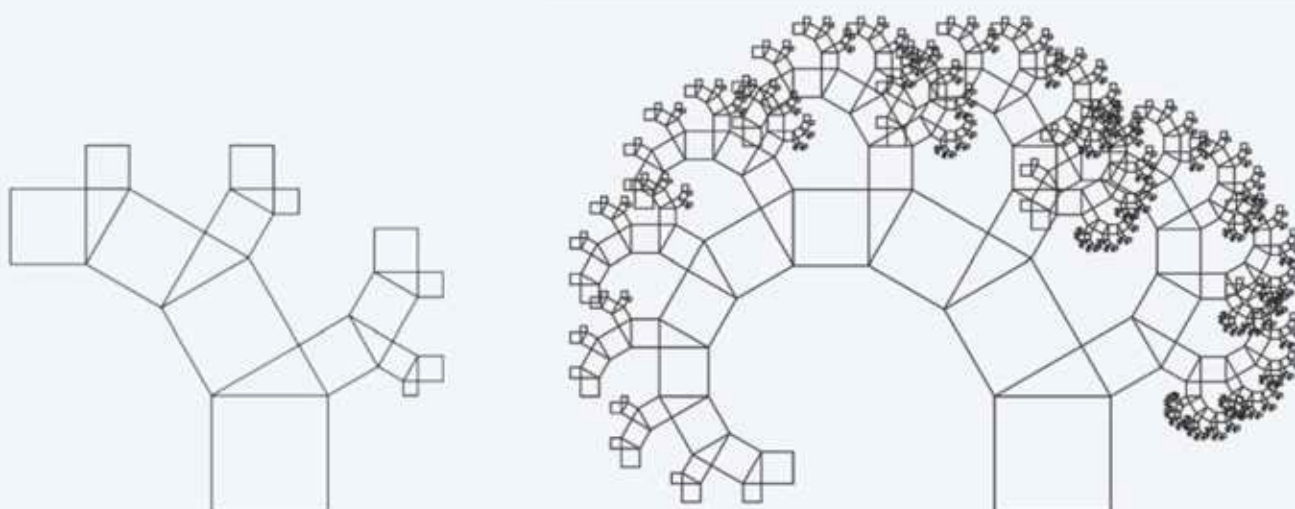


Wskazówka: Możesz wykorzystać poniższe przekształcenia.

$$\begin{cases} x' = -0,4 \cdot x - 1 \\ y' = -0,4 \cdot y + 0,1 \end{cases}$$

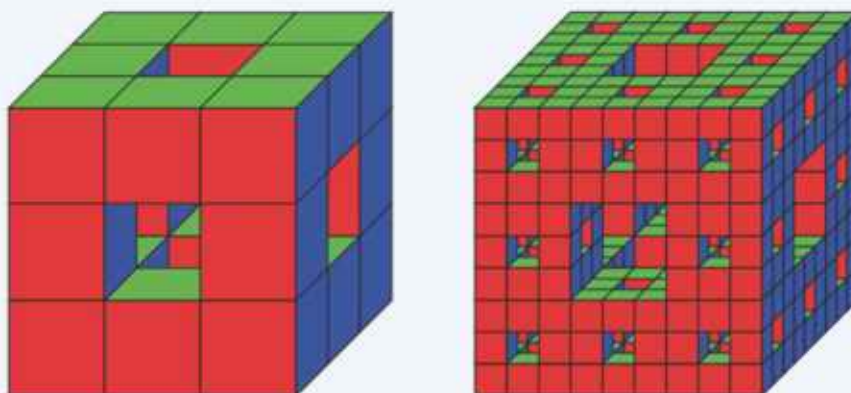
$$\begin{cases} x' = 0,76 \cdot x - 0,4 \cdot y \\ y' = 0,4 \cdot x + 0,76 \cdot y \end{cases}$$

- *** 8 Napisz skrypt w JavaScript z wykorzystaniem biblioteki p5.js rysujący drzewo Pitagorasa stopnia n . Drzewo stopnia 0 to kwadrat o określonej długości boku. Drzewo wyższego stopnia składa się z kwadratu oraz dwóch poddrzew zbudowanych na bokach trójkąta prostokątnego, którego kąty ostre są równe 30° i 60° . Rysunki przedstawiają drzewa stopnia 3 oraz 9.



Wskazówka: Możesz przygotować funkcję rysującą obrócony kwadrat, skorzystać z funkcji **quad** rysującej czworokąt lub zastosować funkcje **rotate** i **translate** z biblioteki p5.js.

- *** 9 Napisz skrypt w JavaScript z wykorzystaniem biblioteki p5.js rysujący kostkę Mengera stopnia n . Kostka Mengera stopnia 0 to sześcian. Kostka Mengera stopnia 1 powstaje przez podział sześcianu na 27 takich samych sześcianów i usunięcie sześcianów środkowych (czyli usuwamy 7 sześcianów: środkowe z 6 ścian oraz jeden w środku sześcianu). Kostkę Mengera stopnia n tworzymy z kostki Mengera stopnia $n - 1$, zmieniając według tych samych reguł każdy sześcian. Rysunek ilustruje kostki Mengera stopni 1 i 2.



Sposób na zadania

Zadanie 1

Reprezentację zmiennoprzecinkową liczby w systemie binarnym w postaci znormalizowanej można wyrazić wzorem:

$$(-1)^{\text{znak}} \cdot \text{mantysa} \cdot 2^{\text{cecha}}$$

gdzie: *znak* – liczba 0 dla liczby dodatniej lub 1 dla liczby ujemnej, *mantysa* – liczba z przedziału $[1; 2]$, *cecha* – liczba całkowita.

Przyjmijmy, że na reprezentację liczby w tej postaci przeznaczono 8 bitów: 1 bit na znak, 3 bity na cechę oraz 4 bity na mantysę.



Znak pamiętany jest w najstarszym bicie (bicie znajdującym się najbardziej z lewej strony). Kolejne trzy bity reprezentują cechę w kodzie z nadmiarem 3. Oznacza to, że aby obliczyć, ile jest równa cecha, od pamiętanej wartości należy odjąć 3, np. 000 reprezentuje wartość -3 , 011 – wartość 0, a 111 – wartość 4. W bitach przeznaczonych na mantysę pamiętana jest tylko jej część ułamkowa. Części całkowitej mantysy nie trzeba pamiętać, ponieważ zawsze jest równa 1.

Zgodnie z powyższymi założeniami zapis 00011001 oznacza liczbę $(-1)^0 \cdot 1,5625 \cdot 2^{-2} = 0,390625$.

W podanym przykładzie bit znaku jest równy 0, więc liczba jest dodatnia. Bity cechy 001 reprezentują wartość 1, od której należy odjąć 3. Cecha jest zatem równa -2 . Bity mantysy 1001 reprezentują liczbę $1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} = 0,5 + 0,0625 = 0,5625$, do której należy dodać część całkowitą (liczbę 1). Otrzymujemy mantysę 1,5625.

Zwróć uwagę, że w omówionym zapisie liczba 0 nie ma reprezentacji. Dlatego przyjmujemy, że jeśli wszystkie cyfry mantysy i cechy będą równe 0 (bit znaku nie ma wówczas znaczenia – może być równy 0 lub 1), to taka reprezentacja oznacza liczbę 0.

Rozwiąż poniższe zadania. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.1 (0–1)

Znajdź binarne reprezentacje zmiennoprzecinkowe najmniejszej liczby dodatniej i największej liczby dodatniej w przedstawionym powyżej systemie zapisu. Podaj także reprezentacje tych liczb w systemie dziesiętnym.

Rozwiązanie:

Tagi: reprezentacja zmiennoprzecinkowa liczb rzeczywistych, postać znormalizowana, system binarny

Krok 1

Szukane liczby są dodatnie, dlatego bit znaku każdej z nich jest równy 0.

Krok 2

Dla najmniejszej liczby dodatniej mantysa i cecha powinny być jak najmniejsze. Ponieważ wszystkie bity cechy i mantysy nie mogą być równe zero, reprezentacja zmiennoprzecinkowa najmniejszej liczby to 00000001. Bity cechy 000 reprezentują liczbę 0. Po odjęciu nadmiaru 3 otrzymujemy cechę -3 . Bity mantysy 0001 odpowiadają liczbie $0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} = 0,0625$, czyli mantysa jest równa $1 + 0,0625 = 1,0625$.

Zatem wartość szukanej liczby w systemie dziesiętnym to: $(-1)^0 \cdot 1,0625 \cdot 2^{-3} = 0,1328125$.

Krok 3

Dla największej liczby dodatniej mantysa i cecha powinny być możliwie największe, zatem wszystkie bity mantysy i cechy muszą być równe 1. Binarna reprezentacja zmiennoprzecinkowa największej liczby to: 01111111. Bity cechy odpowiadają liczbie $1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 7$.

Po odjęciu liczby 3 otrzymujemy cechę 4. Bity mantysy reprezentują liczbę $1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = 0,5 + 0,25 + 0,125 + 0,0625 = 0,9375$. Po dodaniu liczby 1 otrzymujemy mantysę 1,9375. Wartość poszukiwanej liczby w systemie dziesiętnym jest zatem równa: $(-1)^0 \cdot 1,9375 \cdot 2^4 = 31$.

Odpowiedź: Najmniejszą liczbą dodatnią w opisanym systemie jest 00000001, w systemie dziesiętnym odpowiada ona liczbie 0,1328125.

Największą liczbą dodatnią w opisanym systemie jest 01111111, w systemie dziesiętnym odpowiada ona liczbie 31.

Zadanie 1.2 (0–1)

Uzupełnij poniższą tabelę. Znajdź wartości dziesiętne liczb podanych w przedstawionym wcześniej systemie zapisu.

Reprezentacja zmiennoprzecinkowa w systemie binarnym	Wartość dziesiętna
10101001	-0,78125
01110000	
10001111	
01010101	

Rozwiązanie:

Tagi: reprezentacja zmiennoprzecinkowa liczb rzeczywistych, postać znormalizowana, system binarny

Krok 1

W zapisie 01110000 bit znaku jest równy 0, więc liczba jest dodatnia. Bity cechy (111) reprezentują liczbę 7. Po odjęciu od niej liczby 3 otrzymujemy cechę 4. Wszystkie bity mantysy są równe zero, więc mantysa jest równa $0 + 1 = 1$. Wartość dziesiętna liczby jest równa $(-1)^0 \cdot 1 \cdot 2^4 = 16$.

Krok 2

W zapisie 10001111 bit znaku jest równy 1, więc liczba jest ujemna. Bity cechy (000) reprezentują liczbę 0, po odjęciu 3 cecha wynosi -3. Bity mantysy (1111) reprezentują liczbę $0,5 + 0,25 + 0,125 + 0,0625 = 0,9375$, więc mantysa po dodaniu 1 jest równa 1,9375. Wartość dziesiętna liczby jest równa: $(-1)^0 \cdot 1,9375 \cdot 2^{-3} = -0,2421875$.

Krok 3

W zapisie 01010101 bit znaku jest równy 0, więc liczba jest dodatnia. Bity cechy (101) reprezentują liczbę 5, po odjęciu 3 otrzymujemy cechę 2. Bity mantysy (0101) reprezentują liczbę $0,25 + 0,0625 = 0,3125$, więc mantysa po dodaniu 1 jest równa 1,3125. Wartość dziesiętna liczby jest równa $(-1)^0 \cdot 1,3125 \cdot 2^2 = 5,25$.

Odpowiedź: Prawidłowo uzupełniona tabela:

Reprezentacja zmiennoprzecinkowa w systemie binarnym	Wartość dziesiętna
10101001	-0,78125
01110000	16
10001111	-0,2421875
01010101	5,25

Zadanie 1.3 (0–2)

Uzupełnij poniższą tabelę. Zapisz wartości dziesiętne z wykorzystaniem podanego wcześniej wzoru oraz znajdź reprezentację zmiennoprzecinkową tych wartości w opisanym systemie zapisu liczb. Zwróć uwagę, że nie każdą liczbę da się dokładnie przedstawić w tym zapisie. Liczby dziesiętne w tabeli zostały dobrane tak, aby miały dokładną reprezentację binarną.

Wartość dziesiętna	Wartość dziesiętna wyrażona wzorem	Reprezentacja zmiennoprzecinkowa w systemie binarnym
30	$(-1)^0 \cdot 1,875 \cdot 2^4$	01111110
-15,5		
-0,8125		
1,75		

Rozwiązanie:

Tagi: reprezentacja zmiennoprzecinkowa liczb rzeczywistych, postać znormalizowana, system binarny

Krok 1

Liczba -15,5 jest ujemna, więc bit znaku jest równy 1. Należy przekształcić wartość bezwzględną liczby tak, aby mantysa należała do przedziału $[1; 2)$. Ponieważ jest ona większa od 2, wykonujemy dzielenie przez 2 tak długo, aż otrzymamy liczbę z podanego przedziału. Dzielenie przez 2 musimy wykonać trzy razy. Otrzymujemy w ten sposób mantysę 1,9375, a cecha jest równa 3 ($15,5 = 1,9375 \cdot 2^3$). Do cechy dodajemy nadmiar 3 i otrzymujemy wartość 6, w zapisie binarnym 110. Pomijamy część całkowitą mantysy i szukamy reprezentacji binarnej części ułamkowej. $0,9375 = 0,5 + 0,25 + 0,125 + 0,0625 = 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4}$, więc reprezentacja binarna mantysy to 1111. Poszukiwaną reprezentacją liczby -15,5 jest 11101111.

Krok 2

Liczba -0,8125 jest ujemna, więc bit znaku jest równy 1. Przekształcamy wartość bezwzględną liczby tak, aby mantysa należała do przedziału $[1; 2)$. Ponieważ jest ona mniejsza od 1, mnożymy liczbę przez 2 tak długo, aż otrzymamy liczbę z danego przedziału. Liczbę 0,8125 wystarczy pomnożyć raz przez 2. Otrzymujemy w ten sposób mantysę 1,625, a cecha jest równa -1 ($0,8125 = 1,625 \cdot 2^{-1}$). Do cechy dodajemy nadmiar 3 i otrzymujemy 2, w zapisie binarnym 010. Pomijamy część całkowitą mantysy, wartość $0,625 = 0,5 + 0,125 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4}$, więc reprezentacja binarna mantysy to 1010. Poszukiwana reprezentacja binarna liczby -0,8125 to: 10101010.

Krok 3

Liczba 1,75 jest dodatnia, więc bit znaku jest równy 0. Mantysa liczby należy do przedziału $[1; 2)$, więc cecha jest równa 0. Do cechy dodajemy nadmiar 3 i otrzymujemy wartość 3, w zapisie binarnym 011. Pomijamy część całkowitą mantysy, wartość $0,75 = 0,5 + 0,25 = 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4}$, więc reprezentacja binarna mantysy to 1100. Poszukiwana reprezentacja binarna liczby 1,75 to: 00111100.

Odpowiedź: Prawidłowo uzupełniona tabela:

Wartość dziesiętna	Wartość dziesiętna wyrażona wzorem	Reprezentacja zmiennoprzecinkowa w systemie binarnym
30	$(-1)^0 \cdot 1,875 \cdot 2^4$	01111110
-15,5	$(-1)^1 \cdot 1,9375 \cdot 2^3$	11101111
-0,8125	$(-1)^1 \cdot 1,675 \cdot 2^{-1}$	10101010
1,75	$(-1)^0 \cdot 1,75 \cdot 2^0$	00111100

Informacja do zadań 1.4 i 1.5

W pliku, który otrzymasz od nauczyciela (np. *WUZ2_zad1_zmiennop.txt*), w kolejnych stu wierszach znajdują się reprezentacje binarne liczb w przedstawionym sposobie zapisu, w każdym wierszu jedna liczba 8-bitowa. Oto dziesięć pierwszych wierszy pliku:

```
10100101
01101000
10000000
00100111
11010001
10001000
01111100
00000000
10011010
11110001
```

Napisz program lub programy rozwiązujące poniższe zadania. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.4 (0–2)

Podaj, ile jest liczb ujemnych oraz liczb dodatnich w pliku z danymi.

W pierwszych dziesięciu wierszach pliku znajdują się trzy liczby dodatnie i pięć liczb ujemnych.

Liczby dodatnie to: 01101000, 00100111, 01111100.

Liczby ujemne to: 10100101, 11010001, 10001000, 10011010, 11110001.

Rozwiązanie:

Tagi: reprezentacja zmiennoprzecinkowa liczb rzeczywistych, postać znormalizowana, system binarny

Krok 1

Liczby z pliku z danymi odczytamy za pomocą zmiennej plikowej `we` typu `ifstream`. Każdą liczbę potraktujemy jako napis – przechowamy go w zmiennej `s` typu `string`. Liczby dodatnie zliczymy w zmiennej `ile0`, a ujemne – w zmiennej `ile1`. Wartości początkowe tych dwóch zmiennych są równe 0.

Krok 2

Odczytujemy kolejne wiersze z pliku z danymi jako napisy. Sprawdzamy, czy wszystkie znaki napisu od drugiego do ostatniego są zerami. Jeśli tak, to pomijamy tę liczbę, ponieważ jest równa 0, a więc nie jest ani dodatnia, ani ujemna. Przy sprawdzaniu znaków napisu korzystamy z metody `substr`. Zwraca ona fragment napisu od indeksu określonego przez pierwszy parametr. Liczbę zwracanych znaków określa drugi parametr.

Krok 3

Jeśli liczba nie jest zerem, sprawdzamy pierwszy znak napisu. Jeśli jest równy 0, to powiększamy wartość zmiennej `ile0` o 1, w przeciwnym przypadku zwiększamy o 1 wartość zmiennej `ile1`.

Krok 4

Po rozpatrzeniu wszystkich liczb z pliku z danymi zamykamy plik za pomocą metody `close`, a następnie wypisujemy wyniki – wartości zmiennych `ile0` i `ile1`.

Odpowiedź: Kod źródłowy programu rozwiązującego to zadanie może być następujący:

```

1. #include <iostream>
2. #include <fstream>
3. #include <string>
4.
5. using namespace std;
6.
7. int main()
8. {
9.     ifstream we("WUZ2_zad1_zmiennop.txt");
10.    string s;
11.    int ile0=0, ile1=0;
12.    for (int i=0; i<100; i++)
13.    {
14.        we>>s;
15.        if (s.substr(1,7)!="0000000")
16.            if (s[0]=='-')
17.                ile0++;
18.            else ile1++;
19.    }

```

```

20.     we.close();
21.     cout<<"Liczba dodatnich: "<<ile0<<endl;
22.     cout<<"Liczba ujemnych: "<<ile1;
23.     return 0;
24. }

```

Zadanie 1.5 (0–4)

Dla każdej liczby w pliku z danymi oblicz jej wartość dziesiętną. Utwórz plik wynikowy (np. *WUZ2_zad1_5_zmiennop_wynik.txt*), w którym w jednym wierszu znajdzie się jedna liczba rzeczywista zapisana w systemie dziesiętnym. Kolejność liczb w pliku wynikowym ma być taka sama jak w pliku z danymi. Dla pierwszych dziesięciu wierszy pliku z danymi wyniki są następujące:

```

-0.65625
12
0
0.71875
-4.25
-0.1875
28
0
-0.40625
-17

```

Rozwiązanie:

Tagi: reprezentacja zmiennoprzecinkowa liczb rzeczywistych, postać znormalizowana, system binarny

Krok 1

Zdefiniujemy funkcję `BinToDec1` typu `int`, która dla reprezentacji cechy w systemie binarnym znajdzie odpowiadającą jej liczbę w systemie dziesiętnym. Funkcja jest typu `int`, ponieważ szukana wartość jest liczbą całkowitą.

Krok 2

Zdefiniujemy funkcję `BinToDec2` typu `float`, która dla reprezentacji mantysy w systemie binarnym znajdzie odpowiadający jej ułamek dziesiętny. Część ułamkowa mantysy jest przechowywana na czterech bitach, dlatego do jej pamiętania wystarczy użyć liczb pojedynczej precyzji (typu `float`). Wystarczy zsumować te z wartości 2^{-1} , 2^{-2} , 2^{-3} i 2^{-4} , dla których w reprezentacji mantysy występuje 1.

Krok 3

W funkcji `main` odczytujemy kolejne wiersze z pliku z danymi, liczbę z danego wiersza zapisujemy w zmiennej typu `string`. Sprawdzamy, czy wszystkie znaki napisu, zaczynając od drugiego znaku, są zerami. Jeśli tak, to zapisujemy do pliku wynikowego liczbę 0. W przeciwnym przypadku obliczamy wartości cechy i mantysy za pomocą funkcji `BinToDec1` oraz `BinToDec2`. Od cechy odejmujemy nadmiar (czyli 3), a do mantysy dodajemy jej część całkowitą (czyli 1).

Krok 4

Należy pomnożyć mantysę przez odpowiednią potęgę liczby 2 (wykładnikiem potęgi jest cecha). Dla cechy ujemnej oznacza to dzielenie mantysy przez 2 odpowiednią liczbę razy. Jeśli cecha jest dodatnia, mnożymy mantysę przez 2 odpowiednią liczbę razy. Jeśli bit znaku jest równy 1, zmieniamy policzoną wartość na przeciwną. Wyniki zapisujemy do pliku.

Odpowiedź: Kod źródłowy programu rozwiązującego to zadanie może być następujący:

```

1. #include <fstream>
2. #include <string>
3.
4. using namespace std;
5.
6. int BinToDec1(string b)
7. {
8.     int d=0;
9.     for (int i=0;i<b.size();i++)
10.    {
11.        d=d*2;
12.        if (b[i]=='1')
13.            d++;
14.    }
15.    return d;
16. }
17.
18. float BinToDec2(string b)
19. {
20.     float d=0, p=1;
21.     for (int i=0;i<b.size();i++)
22.    {
23.        p=p/2;
24.        if (b[i]=='1')
25.            d=d+p;
26.    }
27.    return d;
28. }
29.
30. int main()
31. {
32.     ifstream we("WUZ2_zad1_zmiennop.txt");
33.     ofstream wy("WUZ2_zad1_5_zmiennop_wynik.txt");
34.     string s;
35.     int c;
36.     float m;

```

```

37.     for (int i=0;i<100;i++)
38.     {
39.         we>>s;
40.         if (s.substr(1,7)=="0000000") wy<<0<<endl;
41.         else
42.         {
43.             c=BinToDec1(s.substr(1,3))-3;
44.             m=1+BinToDec2(s.substr(4,4));
45.             if (c<0)
46.                 for (int j=0;j>c;j--) m=m/2;
47.             else
48.                 for (int j=0;j<c;j++) m=m*2;
49.             if (s[0]=='1') m=-m;
50.             wy<<m<<endl;
51.         }
52.     }
53.     we.close(); wy.close();
54.     return 0;
55. }

```

Trening

Zadanie 1

Odległość d między dwoma punktami o współrzędnych (x_1, y_1) i (x_2, y_2) można wyrazić wzorem:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Rozwiąż poniższe zadania. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.1 (0–2) ROZWIĄŻ BEZ KOMPUTERA

Oto specyfikacja pewnego problemu algorytmicznego:

Specyfikacja

Dane: n – liczba całkowita dodatnia określająca liczbę punktów na płaszczyźnie,

$X[0..n-1]$, $Y[0..n-1]$ – tablice liczb rzeczywistych określające odpowiednio pierwsze i drugie współrzędne n punktów na płaszczyźnie.

Wynik: $x_{\min}$, $y_{\min}$ – liczby rzeczywiste określające współrzędne punktu o najmniejszej odległości od początku układu współrzędnych,

$x_{\max}$, $y_{\max}$ – liczby rzeczywiste określające współrzędne punktu o największej odległości od początku układu współrzędnych.

Zapisz w wybranej przez siebie notacji (lista kroków, pseudokod, język programowania) algorytm wyznaczający współrzędne punktu o najmniejszej odległości od początku układu współrzędnych oraz punktu o największej odległości zgodnie z powyższą specyfikacją. Załóż, że nie występują dwa punkty o tej samej odległości od początku układu współrzędnych.

Informacja do zadań 1.2–1.4

W pliku, który otrzymasz od nauczyciela (np. *WUZ2_zad_1_punkty.txt*), w kolejnych stu wierszach są zapisane po dwie liczby rzeczywiste oddzielone spacją, określające współrzędne punktu na płaszczyźnie. Pierwsza liczba to współrzędna x , druga liczba – współrzędna y . Wszystkie współrzędne mieszczą się w zakresie od -1000 do 1000 .

Napisz program lub programy rozwiązujące poniższe zadania. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.2 (0–2) **ROZWIĄŻ NA KOMPETERZE**

Znajdź współrzędne wierzchołków prostokąta, który ma najmniejsze możliwe pole i boki równoległe do osi układu współrzędnych oraz zawiera wszystkie punkty z pliku z danymi, tzn. punkty znajdują się we wnętrzu prostokąta lub na jego bokach.

Zadanie 1.3 (0–3) **ROZWIĄŻ NA KOMPETERZE**

Utwórz plik wynikowy (np. *WUZ2_zad1_3_punkty_wynik.txt*), w którym znajdą się wszystkie punkty z pliku z danymi uporządkowane względem odległości od początku układu współrzędnych – najpierw punkt o najmniejszej odległości, a na końcu punkt o największej odległości od początku układu współrzędnych. W każdym wierszu pliku wynikowego zapisz współrzędne jednego punktu oddzielone spacją.

Zadanie 1.4 (0–3) **ROZWIĄŻ NA KOMPETERZE**

Punkty w pliku z danymi zostały dobrane tak, że każde cztery kolejne punkty są wierzchołkami czworokąta wypukłego. Znajdź wśród tych figur czworokąt o najmniejszym polu oraz czworokąt o największym polu. Podaj współrzędne wierzchołków tych czworokątów oraz ich pola. Załóż, że w pliku z danymi nie ma dwóch czworokątów o takich samych polach.

Wskazówka: Do obliczenia pola czworokąta możesz wykorzystać poniższy wzór na pole trójkąta o wierzchołkach w punktach $A(x_A, y_A)$, $B(x_B, y_B)$ i $C(x_C, y_C)$.

$$P_{ABC} = \frac{1}{2} \cdot |(x_B - x_A) \cdot (y_C - y_A) - (x_C - x_A) \cdot (y_B - y_A)|$$



3 Zaawansowane algorytmy i techniki programistyczne

- 11. Wyszukiwanie wzorca w tekście
 - 12. Szyfrowanie z kluczem publicznym. Algorytm RSA
 - 13. Programowanie obiektowe
- Wiesz, umiesz, zdasz

Algorytmika i programowanie

```
for i:=2 to j
do
  k:=1;
  while text(k)=text(i-k+1)
  do
    k:=k+1;
    if k>i-k+1
    then
      write ("found at i-th item");
      return
    fi
  od
od;
write ("not found")
```

W 1982 r. na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego pod kierunkiem profesorów Andrzeja Salwickiego i Antoniego Kreczmara powstał **Loglan 82** – nowoczesny język programowania wysokiego poziomu. Pod wieloma względami wyprzedzał współczesne mu języki (bezpieczny system zarządzania pamięcią obiektów, klasy wewnętrzne, łączenie zagnieżdżenia klas i dziedziczenia). Wiele jego rozwiązań zastosowano potem np. w Javie. Współcześnie istnieją kompilatory Loglana na systemy Linux i Windows. Mimo swojego nowatorstwa język ten nie znalazł szerszego zastosowania.

Amerykańska edukatorka Yvonne Marie Andrés uważana jest za najważniejszą prekursorkę **e-learningu**. W 1984 r. założyła fundację Global SchoolNet, której celem było łączenie młodzieży i naukowców z całego świata we wspólnych projektach edukacyjnych. W 2000 r. została uznana za jedną z 25 najbardziej wpływowych osób na świecie w dziedzinie technologii edukacyjnych. Współczesnym przykładem e-learningu są kursy typu MOOC (ang. *massive open online course*), w których może brać udział nieograniczona liczba uczących się z całego świata. Pierwsze kursy MOOC pojawiły się w 2012 r.



1982 r.

1984 r.

1983 r.

1985 r.



Duński informatyk Bjarne Stroustrup w 1983 r. zaproponował rozszerzenie języka C do **języka C++**. Możliwość programowania obiektowego, łatwość korzystania z bibliotek i bardzo efektywny kod wynikowy spowodowały, że C++ stał się niezwykle popularnym językiem ogólnego przeznaczenia i jest powszechnie wykorzystywany, m.in. do tworzenia systemów operacyjnych, gier komputerowych czy oprogramowania systemowego sterującego pracą urządzeń, np. AGD. Korzystają z niego największe światowe korporacje takie jak Adobe, Google, Facebook, Amazon, Intel.

Radia Perlman to amerykańska programistka, która w 1985 r. stworzyła **protokół STP** (ang. *Spanning Tree Protocol*), który stał się podstawą technologii Ethernet. Perlman okrzyknięto nawet matką Internetu, choć ona sama nie lubi tego określenia. W latach 70. razem z Seymourem Papertem uczestniczyła w rozwijaniu języka programowania Logo. Napisała jego prostszą wersję Tortis, której do sterowania robotem mogły używać małe dzieci.





Holenderski informatyk Guido van Rossum stworzył w 1991 r. skryptowy język programowania **Python** o czytelnej i zwięzłej składni. Nazwę zaczerpnął z popularnego serialu komediowego *Latający Cyrk Monthly Pythona*. Składnia języka wymusza poprawne formatowanie kodu źródłowego poprzez ścisłe zasady stosowania wcięć. Współcześnie jest to jeden z najchętniej używanych języków o wielu zastosowaniach. Na przykład oprogramowanie do serwisu YouTube i narzędzi Google napisano właśnie w języku Python. Ze względu na licencję *open source* dodatkowej popularności przysparzają mu liczne biblioteki niezależnych twórców.

1991 r.



W 1995 r. powstały dwa języki o podobnej nazwie, ale różnym przeznaczeniu i odmiennej budowie. **Java** jest obiektowym językiem ogólnego przeznaczenia, kompilowanym do kodu wykonywanego przez maszynę wirtualną. Część jego składni przejęto z języka C++. Autorem Javy jest kanadyjski programista James Gosling (na zdjęciu) z firmy Sun Microsystems. Z kolei **JavaScript** jest skryptowym językiem programowania, wykorzystywanym m.in. do tworzenia stron internetowych. Jego twórcą jest amerykański programista Brendan Eich, pracujący dla firmy Netscape Communications.

1995 r.

2018 r.



Od 2018 r. wyszukiwarka Google stosuje algorytm **BERT** (*Bidirectional Encoder Representations from Transformers*). Pozwala on na lepszą analizę języka naturalnego, uwzględniającą m.in. końcówki fleksyjne oraz tzw. *stopwords* (np. zaimki, przymki, spójniki), czyli słowa o mniej istotnym znaczeniu. Osadza to wyszukiwane słowa w kontekście i daje znaczącą poprawę jakości wyników. Algorytm ten jest przykładem praktycznego zastosowania teorii sieci neuronowych. W końcu 2019 r. został wdrożony również w języku polskim.

2019 r.

Amerykańska badaczka Katherine Bouman w 2019 r. stworzyła algorytm **CHIRP** (ang. *Continuous High-resolution Image Reconstruction using Patch priors*), służący do ciągłej rekonstrukcji obrazu o wysokiej rozdzielczości. Dzięki niemu możliwe stało się opublikowanie pierwszego w historii zdjęcia czarnej dziury, położonej w Galaktyce Messier 87 i oddalonej od Ziemi o 53 mln lat świetlnych. Dzięki algorytmowi CHIRP gigantyczne ilości danych zebrane przez teleskopy rozsięte po całym świecie zostały przekonwertowane na obrazy.



11. Wyszukiwanie wzorca w tekście

Problem wyszukiwania wzorca w tekście jest niezwykle istotny w zastosowaniach informatyki. Praktycznie wszystkie edytory tekstu są wyposażone w narzędzie, które pozwala znaleźć dany ciąg znaków w dokumencie. Szukanie informacji w internecie też często sprowadza się do wpisania w wyszukiwarce fragmentu tekstu. Oczekujemy wtedy, że wyniki nie tylko będą poprawne, lecz także pojawią się szybko. W tym temacie poznasz algorytmy związane z wyszukiwaniem wzorca w tekście.

Cele lekcji

- Wyszukasz wzorzec w tekście z wykorzystaniem algorytmu naiwnego.
- Dowiesz się, na czym polega metoda haszowania.
- Poznasz podstawy arytmetyki modularnej.
- Zastosujesz funkcję haszującą do wyszukiwania wzorca w tekście.
- Poznasz algorytm Karpa–Rabina, służący do znajdowania wzorca w tekście.

Problem wyszukiwania wzorca w tekście

Problem wyszukiwania wzorca w tekście można sformułować w następujący sposób. Dane są dwa napisy: jeden nazwiemy tekstem, a drugi wzorcem. Celem jest określenie, w którym miejscu w tekście znajduje się wzorzec, o ile w ogóle w nim występuje.

Przyjmujemy, że długość wzorca (rozumiana jako liczba znaków) jest nie większa niż długość tekstu, a w praktyce jest ona wielokrotnie mniejsza. Ograniczymy znaki, które mogą występować w tekście i we wzorcu, do małych liter alfabetu łacińskiego.

Oto specyfikacja problemu:

Specyfikacja

Dane: t , w – niepuste napisy złożone z małych liter alfabetu łacińskiego, długość tekstu t jest większa lub równa długości wzorca w .

Wynik: p – liczba całkowita określająca pozycję (indeks) pierwszego wystąpienia wzorca w w tekście t – lub -1 , gdy wzorzec nie występuje w tekście.

Dobra rada

Biblioteki większości języków programowania oferują funkcję do poszukiwania wzorca w tekście. W języku C++ możesz skorzystać z biblioteki `string` i metody `find` dla klasy `string`.

11.1. Algorytm naiwny

Algorytm naiwny wyszukiwania wzorca w tekście

Jednym z najprostszych rozwiązań jest tzw. **algorytm naiwny wyszukiwania wzorca w tekście**, który polega na porównywaniu znak po znaku wzorca z tekstem.

W algorytmie naiwnym ustawiamy wzorec pod tekstem tak, aby pierwszy znak wzorca znajdował się pod pierwszym znakiem tekstu i porównujemy kolejne pary znaków obu napisów.

Jeśli wszystkie znaki wzorca będą odpowiadały znakom fragmentu tekstu, to znaleźliśmy położenie wzorca w tekście. Jeśli jednak znajdziemy pierwszą parę różniących się znaków, to przesuwamy wzorec o jeden znak w prawo względem tekstu i ponownie rozpoczynamy porównywanie kolejnych par znaków.

Algorytm kończy działanie po znalezieniu pozycji wzorca w tekście lub wysunięciu wzorca poza tekst, czyli gdy pozycja początkowa wzorca jest większa od długości tekstu pomniejszonej o długość wzorca.

Opisany algorytm pokażemy na przykładzie. Poszukamy wzorca *ar* w tekście *rabarbar*. Przedstawia to rysunek 11.1.



Rys. 11.1. Naiwne wyszukiwanie wzorca w tekście

Oto algorytm naiwny w postaci funkcji, zapisany w pseudokodzie:

```
funkcja Znajdz(w,t)
  p ← 0
  dopóki p ≤ długość t – długość w wykonuj
    i ← 0
    dopóki i < długość w oraz w[i] = t[p+i] wykonuj
      i ← i + 1
    jeśli i = długość w to zwróć p i zakończ
    w przeciwnym przypadku p ← p + 1
  zwróć -1 i zakończ
```

Dobra rada

Jeśli chcesz znaleźć wszystkie wystąpienia wzorca w tekście, kontynuuj działanie algorytmu, aż pozycja początkowa wzorca będzie większa od długości tekstu pomniejszonej o długość wzorca.

Zmienna pomocnicza i określa indeks porównywanego znaku wzorca. W zależności od przesunięcia wzorca względem tekstu (wartości zmiennej p) porównujemy znaki $w[i]$ z $t[p+i]$ dla i w zakresie od 0 do długości wzorca minus 1, chyba że wcześniej pojawi się para różnych znaków. Wówczas zwiększamy wartość zmiennej p , czyli przesuwamy wzorzec o jeden znak w prawo względem tekstu.

Oto kod źródłowy funkcji `Znajdz`, realizującej algorytm naiwny wyszukiwania wzorca w tekście:

Kod źródłowy funkcji
Znajdz, realizującej
algorytm naiwny
wyszukiwania wzorca
w tekście

```
1. int Znajdz(string w, string t)
2. {
3.     int dw=w.size();
4.     int dt=t.size();
5.     int i, p=0;
6.     while (p<=dt-dw)
7.     {
8.         i=0;
9.         while (i<dw && w[i]==t[p+i]) i++;
10.        if (i==dw) return p;
11.        else p++;
12.    }
13.    return -1;
14. }
```

Zmienne pomocnicze dw i dt (linie 3–4) określają odpowiednio długość wzorca i długość tekstu. Zewnętrzna pętla (linie 6–12) odpowiada za sprawdzenie znaków wzorca ze znakami tekstu. Jeśli kolejne znaki będą sobie równe, a zmienna i przyjmie wartość dw , to funkcja zwróci pozycję wzorca (linia 10).

Rysunek 11.2 przedstawia przykłady uruchomienia programu.

```
Tekst: rabarbar
Szukany wzorzec: ar
Pozycja wzorca w tekście: 3
```

```
Tekst: rabarbar
Szukany wzorzec: rak
Wzorzec nie występuje w tekście
```

Rys. 11.2. Efekt działania programu poszukującego wzorców *ar* i *rak* w tekście *rabarbar*

Ćwiczenie 1

Napisz program, który wczyta tekst oraz wzorzec, zapisane małymi literami alfabetu łacińskiego, a następnie wyszuka wzorzec w tekście, stosując algorytm naiwny. W programie wykorzystaj kod źródłowy funkcji `Znajdz`.

Złożoność obliczeniowa algorytmu naiwnego wynosi $O(n \cdot m)$, gdzie n oznacza długość tekstu, a m – długość poszukiwanego wzorca. Warto się zastanowić, czy nie można znaleźć wzorca w tekście (lub sprawdzić, że on nie występuje), wykonując mniej operacji porównywania znaków.

Jednym ze sposobów jest porównywanie wzorca z fragmentem tekstu tylko czasami – kiedy istnieje duże prawdopodobieństwo zgodności napisów. Wykorzystamy do tego technikę algorytmiczną zwaną haszowaniem.

11.2. Metoda haszowania

Ideę haszowania wytłumaczymy na przykładzie wybierania numeru z listy kontaktów zapisanej w telefonie. Jeśli chcemy do kogoś zadzwonić, podajemy jego nazwisko, które jest powiązane z numerem telefonu. Oprogramowanie telefonu szuka więc właściwego numeru po nazwisku (lub nazwie kontaktu powiązanej z numerem).

Gdyby kontakty w telefonie były nieuporządkowane, oprogramowanie stosowałoby zapewne algorytm **przeszukiwania liniowego**, który ma złożoność czasową $O(n)$, gdzie n to liczba zapisanych kontaktów. Ponieważ jednak są one uporządkowane, oprogramowanie może korzystać z algorytmu **przeszukiwania binarnego**, który działa szybciej, w złożoności $O(\log n)$. Dla niewielkiej wartości n kontakt znajdziemy bardzo szybko.

Powyższy sposób zawodzi jednak w przypadku bardzo dużych zbiorów danych, np. dla bazy operatora telekomunikacyjnego, ponieważ czas odszukania właściwego nazwiska może być nadal zbyt długi.

Rozwiązaniem tego problemu jest funkcja, która w czasie niezależnym od wielkości bazy danych przekształca nazwisko tak, aby umożliwić bezpośredni dostęp do numeru telefonu. Wartość tej funkcji mogłaby być np. indeksem tablicy zawierającej numery telefonów.

Warto wiedzieć

Istnieją algorytmy wyszukiwania wzorca w tekście, które wykorzystują przesunięcie wzorca względem tekstu o więcej niż jedną pozycję.

Przeszukiwanie liniowe i binarne, podręcznik *Informatyka na czasie 2. Zakres rozszerzony*, s. 153–154 



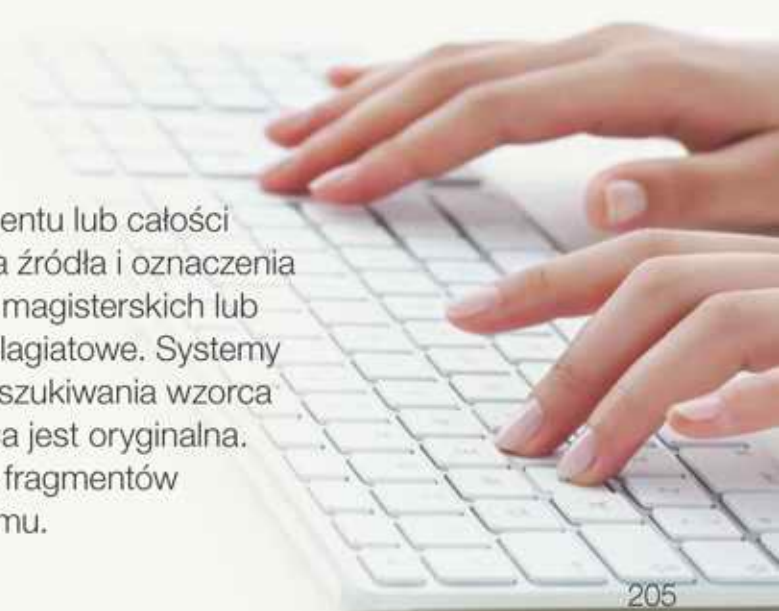
Dobra rada

Jeśli dane są uporządkowane, możesz zastosować algorytm przeszukiwania binarnego.

A to ciekawe

Jak algorytmy pomagają wykrywać plagiaty

Plagiat polega na przypisaniu sobie autorstwa fragmentu lub całości cudzego utworu, czyli wykorzystanie go bez podania źródła i oznaczenia autora. W przypadku utworów tekstowych, np. prac magisterskich lub doktorskich, powszechnie stosuje się systemy antyplagiatowe. Systemy te mają zaimplementowane algorytmy szybkiego wyszukiwania wzorca w tekście, które pozwalają sprawdzić, czy dana praca jest oryginalna. Pozwalają również ustalić, jaka jej część składa się z fragmentów znajdujących się w innych publikacjach z bazy systemu.



Funkcja haszująca (funkcja skrótu) • Taką funkcję nazywamy **funkcją haszującą** (ang. *hash function*) albo **funkcją skrótu**. W ogólnym przypadku przekształca ona napis na pewną wartość, która umożliwia dostęp do danych w czasie niezależnym od ich rozmiaru. Argument funkcji haszującej nazywamy **Klucz, hasz** • **kluczem** (ang. *key*), a jej wartość – **haszem** (ang. *hash*).

Dla danego problemu, np. wyszukiwania numeru na liście kontaktów w telefonie, można zdefiniować wiele różnych funkcji haszujących. Przykładem bardzo prostej funkcji haszującej może być funkcja przekształcająca nazwisko na liczbę na podstawie kodu ASCII pierwszej litery nazwiska.

Wykorzystanie funkcji haszującej

Sposób wykorzystania funkcji haszującej zaprezentujemy na uproszczonym przykładzie wyszukiwania numeru telefonu na liście kontaktów.

1 Wyszukanie nazwiska na liście kontaktów



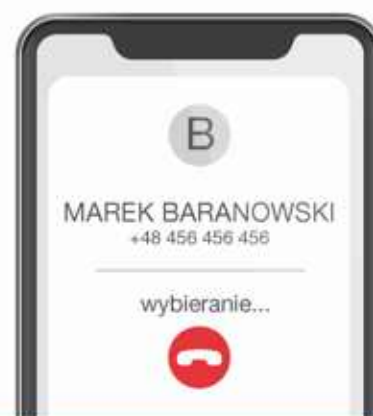
2 Przekształcenie przez funkcję haszującą nazwiska na kod ASCII jego pierwszej litery



3 Odczytanie numeru z bazy – tabela jest indeksowana wartościami funkcji haszującej

Indeks	Numer telefonu
65	123 123 123
66	456 456 456
67	789 789 789
...	...

4 Wybranie numeru



Zauważ, że na liście kontaktów mogą się pojawić przynajmniej dwie osoby o nazwisku rozpoczynającym się na tę samą literę, zatem funkcja haszująca dla różnych kluczy może zwrócić tę samą wartość. Taką sytuację nazywamy **kolizją** (ang. *collision*). Jednak nawet w przypadku kolizji poszukiwanie zostaje zawężone do znacznie mniejszego podzbioru danych.

Ćwiczenie 2

Podaj przykład funkcji haszującej, która zmniejszy liczbę kolizji podczas wyszukiwania kontaktów na liście.

Złożoność algorytmów wykorzystujących haszowanie zależy od funkcji haszującej. Funkcja ta nie powinna zwracać tej samej wartości dla dużego zbioru kluczy, a dla poszczególnych kluczy powinna równomiernie przydzielać wartości. Nie powinna też powodować kolizji dla zbliżonych wartości kluczy.

Warto wiedzieć

Wartością funkcji haszującej nie musi być liczba, może to być również napis. Wartość ta powinna mieć określony rozmiar, np. liczba powinna należeć do określonego przedziału, a napis mieć określoną maksymalną długość.

11.3. Algorytm Karpa–Rabina

Przedstawimy teraz **algorytm Karpa–Rabina**, rozwiązujący problem **Algorytm Karpa–Rabina** znajdowania wzorca w tekście z wykorzystaniem metody haszowania, która znacząco ograniczy liczbę porównań wzorca i fragmentu tekstu.

W algorytmie Karpa–Rabina odpowiednio dobrana **funkcja haszująca** wyznacza wartość haszu dla wzorca oraz dla fragmentu porównywanego tekstu. Jeśli obie wartości haszy są różne, to nie ma zgodności wzorca z fragmentem tekstu. Jeśli wartości haszy są takie same, to musimy dodatkowo potwierdzić, że wzorzec został odnaleziony, ponieważ możemy mieć do czynienia z **kolizją**. Wówczas porównujemy znaki wzorca i fragmentu tekstu analogicznie jak w **algorytmie naiwnym**.

Liczba porównań jest znacząco ograniczona, bo porównujemy tylko te fragmenty tekstu, których hasze zgadza się z haszem wzorca.

Zapiszmy algorytm Karpa–Rabina w pseudokodzie.

```
funkcja Znajdz(w,t)
    hw ← hasz wzorca w
    ht ← hasz początkowego fragmentu tekstu t
    p ← 0
    dopóki p ≤ długość t – długość w wykonuj
        jeśli hw = ht to
            i ← 0
            dopóki i < długość w oraz w[i] = t[p+i] wykonuj
                i ← i + 1
            jeśli i = długość w to zwróć p i zakończ
        p ← p + 1
    jeśli p ≤ długość t – długość w
        ht ← hasz fragmentu tekstu t
            po przesunięciu wzorca
    zwróć -1 i zakończ
```

Zmienna hw pamięta hasz wzorca w – jest to wartość stała. Wartością początkową zmiennej ht jest hasz początkowego fragmentu tekstu o długości równej długości wzorca. Jeśli hasze hw i ht są sobie równe, to prawdopodobnie znaleźliśmy wystąpienie wzorca w tekście. Potwierdzamy to przez porównanie wzorca z fragmentem.

Funkcja haszująca,
s. 206 [↗](#)

Kolizja,
s. 206 [↗](#)

Algorytm naiwny
wyszukiwania wzorca
w tekście,
s. 202 [↗](#)

Jeśli hasze nie były sobie równe lub nastąpiła kolizja, przesuwamy wzorec względem tekstu o jedną pozycję w prawo (zwiększamy wartość zmiennej p o 1). Jeśli wzorec nie wychodzi poza tekst, to obliczamy nową wartość zmiennej ht – wartość funkcji haszującej dla nowego fragmentu tekstu.

Funkcja haszująca,
s. 206 

Istotą algorytmu jest właściwy dobór **funkcji haszującej**, tak aby dawała równomierny rozkład wartości haszu i można było szybko wyznaczyć jego wartość. W takich sytuacjach często stosuje się tzw. arytmetykę modularną.

Zapamiętaj

Funkcja haszująca przekształca napis na pewną wartość umożliwiającą dostęp do danych w czasie niezależnym od ich rozmiaru. Argument funkcji nazywamy kluczem, a jej wartość – haszem.

Arytmetyka modularna

Arytmetyka modularna • **Arytmetyka modularna** (ang. *modular arithmetic*) nazywana jest również arytmetyką reszt lub arytmetyką modulo n . Jest to arytmetyka liczb całkowitych, w której wyniki działań arytmetycznych sprowadza się do przedziału $[0; n)$ w taki sposób, aby wynik działania był przedstawiony jako reszta z dzielenia przez n .

Warto wiedzieć

Arytmetykę modularną stosujemy na co dzień, np. przy odliczaniu czasu. Cztery godziny po godzinie 22 będzie godzina 2, ponieważ $(22 + 4) \bmod 24 = 2$.

Pokażemy przykład dodawania, mnożenia i odejmowania liczb 3 i 13 w arytmetyce modulo 11.

$$(3 + 13) \bmod 11 = 16 \bmod 11 = 5$$

$$(3 \cdot 13) \bmod 11 = 39 \bmod 11 = 6$$

Jeśli wynik odejmowania modulo n jest mniejszy od zera, to do wyniku należy dodać wartość n – w naszym przykładzie liczbę 11:

$$(3 - 13) \bmod 11 = (-10) \bmod 11 = -10 + 11 = 1$$

Ponieważ prawdziwe są poniższe zależności:

$$(a + b) \bmod n = (a \bmod n + b \bmod n) \bmod n$$

$$(a \cdot b) \bmod n = (a \bmod n \cdot b \bmod n) \bmod n$$

można przyjąć, że argumenty działań są także z przedziału $[0; n)$. Na przykład:

$$(3 + 13) \bmod 11 = (3 + 2) \bmod 11 = 5$$

$$(3 \cdot 13) \bmod 11 = (3 \cdot 2) \bmod 11 = 6$$

$$(3 - 13) \bmod 11 = (3 - 2) \bmod 11 = 1$$

Warto wiedzieć

W arytmetyce modularnej nie ma operacji dzielenia. Występuje natomiast odwrotność liczby. Odwrotnością liczby a w arytmetyce modulo n jest taka liczba b , że $(a \cdot b) \bmod n = 1$.

Ćwiczenie 3

Wykonaj dodawanie, odejmowanie i mnożenie liczb:

a. 7 i 15 w arytmetyce modulo 5,

b. 5 i 11 w arytmetyce modulo 13.

Dobór funkcji haszującej w algorytmie Karpa–Rabina

Wyjaśnimy na przykładzie sposób konstruowania funkcji haszującej w algorytmie Karpa–Rabina. Przyjmijmy na moment, że alfabet składa się tylko z trzech liter: a , b i c . Od kodu ASCII danej litery odejmujemy liczbę 97 (kod ASCII litery a), więc litera a będzie reprezentowana przez liczbę 0, litera b – przez liczbę 1, a litera c – przez liczbę 2.

Litery alfabetu możemy potraktować jako cyfry systemu pozycyjnego o podstawie równej liczebności alfabetu – w naszym przykładzie podstawa jest równa 3. Klucz (czyli wzorzec lub fragment tekstu) potraktujemy jako liczbę zapisaną w tym systemie. **Haszem** będzie wartość dziesiętna tej liczby w arytmetyce modulo n , gdzie n jest liczbą pierwszą.

Na przykład dla klucza $bbaabc$ i $n = 29$ wartość funkcji Hash, haszującej w arytmetyce modulo 29, będzie równa:

$$\begin{aligned}\text{Hash}(bbaabc) &= (1 \cdot 3^5 + 1 \cdot 3^4 + 0 \cdot 3^3 + 0 \cdot 3^2 + 1 \cdot 3^1 + 2 \cdot 3^0) \bmod 29 = \\ &= (11 + 23 + 3 + 2) \bmod 29 = 10\end{aligned}$$

Zwróć uwagę, że jest to zapis odpowiadający obliczeniu wartości wielomianu. Zauważ też, że kolejne potęgi liczby 3 w arytmetyce modulo 29 to: 1, 3, 9, 27, 23, 11.

Ćwiczenie 4

Oblicz wartość funkcji Hash dla:

- klucza $bcba$ i $n = 11$,
- klucza $abccbb$ i $n = 29$.

Zapiszemy algorytm działania funkcji haszującej, który będzie właściwy dla dowolnego tekstu utworzonego z małych liter alfabetu łacińskiego. Do obliczenia wartości funkcji wykorzystamy **schemat Hornera** – stopień wielomianu będzie o jeden mniejszy od długości klucza.

Niech n będzie liczbą pierwszą, a m oznacza długość alfabetu. Wartość funkcji jest obliczana w arytmetyce modulo n i przechowywana w zmiennej w . Liczba pierwsza n powinna być dobrana tak, aby z jednej strony uniknąć zbyt dużych wartości funkcji haszującej, a z drugiej strony ograniczyć liczbę **kolizji**.

Oto zapis w pseudokodzie funkcji Hash0:

funkcja Hash0(klucz)

$w \leftarrow 0$

dla $i \leftarrow 0, \dots, \text{długość klucza} - 1$ **wykonuj**

$w \leftarrow ((w * m) \bmod n + \text{kod_ASCII}(\text{klucz}[i]) - 97) \bmod n$

zwróć w i **zakończ**

Jeśli wartość $\text{kod_ASCII}(\text{klucz}[i]) - 97$ jest mniejsza od n , to nie musimy brać reszty z dzielenia przez n .

Hasz,
s. 206 

Dobra rada

Podczas wykonywania obliczeń w arytmetyce modulo n często korzysta się ze standardowych symboli działań arytmetycznych. Pamiętaj jednak, że wynik tego działania zawsze jest przekształcany na resztę z dzielenia przez n , dlatego dla $n = 29$ wynik działania 3^5 to 11.

Schemat Hornera,
s. 125 

Kolizja,
s. 206 

Zauważ, że kolejne fragmenty porównywanego tekstu różnią się od siebie tylko nieznacznie – wystarczy pominąć pierwszy znak aktualnego fragmentu tekstu i uwzględnić kolejny znak. Dzięki temu można szybko obliczyć wartość funkcji haszującej dla kolejnego fragmentu na podstawie dotychczasowej wartości funkcji. Zapiszmy w pseudokodzie funkcję Hash1, która to realizuje.

```
funkcja Hash1(h0,jd,ch)
    w ← (h0 – jd) mod n
    jeśli w<0 to w ← w + n
    zwróć ((w*m) mod n + kod_ASCII(ch) – 97) mod n i zakończ
```

Parametry funkcji Hash1 to: h0 – dotychczasowa wartość funkcji haszującej dla fragmentu tekstu, jd – wartość jednomianu odpowiadająca usuwanemu znakowi w arytmetyce modulo n, ch – kolejny znak tekstu. Wartość m określa liczbę znaków alfabetu.

W naszym przykładzie wartość funkcji dla klucza *bbaabc* jest równa 10. Dla tekstu *bbaabccac* kolejnym kluczem będzie *baabcc*. Od dotychczasowej wartości funkcji należy odjąć wartość jednomianu $1 \cdot 3^5$ (odpowiadającą pierwszej literze *b*), czyli wartość 11 w arytmetyce modulo 29. Następnie należy przesunąć pozostałe litery o jedną pozycję w lewo (czyli pomnożyć otrzymaną wartość przez 3) i dodać wartość odpowiadającą nowemu znakowi fragmentu tekstu (litera *c*, $2 \cdot 3^0$):

$\text{Hash}(bbaabc) = 10$

$\text{Hash}(baabcc) = ((10 - 11) \cdot 3 + 2) \bmod 29 = (28 \cdot 3 + 2) \bmod 29 =$
 $= (26 + 2) \bmod 29 = 28$

Do wyznaczenia wartości odejmowanego jednomianu potrzebna będzie jeszcze funkcja wyznaczająca wartość potęgi w arytmetyce modulo *n*. Będzie to funkcja Potega.

Przedstawimy teraz definicje stałych globalnych określających wartość liczby pierwszej oraz długość alfabetu, a także kody źródłowe funkcji: Potega, Hash0, Hash1 oraz Znajdz.

Definicje stałych globalnych i kod źródłowy funkcji

Potega w programie realizującym algorytm Karpa–Rabina

```
1. const int N=997; // liczba pierwsza
2. const int M=26; // długość alfabetu
3.
4. int Potega(int podst, int wykl)
5. {
6.     int w=1;
7.     while (wykl>0)
8.     {
9.         if (wykl%2==1) w=(w*podst)%N;
10.        wykl=wykl/2;
11.        if (wykl>0) podst=(podst*podst)%N;
12.    }
13.    return w;
14. }
```

W liniach 1–2 zdefiniowane są stałe N i M , określające odpowiednio liczbę pierwszą oraz liczebność alfabetu zgodną ze specyfikacją. W liniach 4–14 zdefiniowana jest funkcja *Potega*, która służy do **szybkiego obliczania wartości potęgi** w arytmetyce modulo N .

Oto kod źródłowy funkcji *Hash0* oraz *Hash1*:

Algorytm szybkiego podnoszenia do potęgi, podręcznik *Informatyka na czasie 2. Zakres rozszerzony*, s. 57–58 [↗](#)

```

1. int Hash0(string s)
2. {
3.     int w=0;
4.     for (int i=0;i<s.size();i++)
5.         w=((w*M)%N+s[i]-'a')%N;
6.     return w;
7. }
8.
9. int Hash1(int h0, int jd, char ch)
10. {
11.     int w=(h0-jd)%N;
12.     if (w<0) w=w+N;
13.     return ((w*M)%N+ch-'a')%N;
14. }
```

🔗 Kod źródłowy funkcji *Hash0* oraz *Hash1* w programie realizującym algorytm Karpa–Rabina

W liniach 1–7 zdefiniowana jest funkcja *Hash0*, którą wykorzystamy do obliczenia haszy wzorca i pierwszego fragmentu tekstu. Zdefiniowana w liniach 9–14 funkcja *Hash1* posłuży do obliczania wartości haszy dla kolejnych fragmentów tekstu.

Powyższe funkcje wykorzystuje funkcja *Znajdz*. Oto jej kod źródłowy:

```

1. int Znajdz(string w, string t)
2. {
3.     int i, p=0, dw=w.size(), dt=t.size();
4.     int hw=Hash0(w);
5.     int ht=Hash0(t.substr(0,dw));
6.     int pot=Potega(M,dw-1);
7.     while (p<=dt-dw)
8.     {
9.         if (hw==ht)
10.        {
11.            i=0;
12.            while (i<dw && w[i]==t[p+i]) i++;
13.            if (i==dw) return p;
14.        }
15.        if (p<dt-dw)
16.            ht=Hash1(ht,(pot*(t[p]-'a'))%N,t[p+dw]);
17.        p++;
18.    }
19.    return -1;
20. }
```

🔗 Kod źródłowy funkcji *Znajdz* w programie realizującym algorytm Karpa–Rabina

Kod źródłowy funkcji
Znajdz, realizującej
algorytm naiwny
wyszukiwania wzorca
w tekście,
s. 204 

Dobra rada

Zauważ, że powiększenie wartości zmiennej p warto wykonać po ewentualnym wywołaniu funkcji Hash1, aby przy określaniu wartości parametrów funkcji nie odejmować liczby 1.

Znaczenie zmiennych i , p , dw i dt w funkcji Znajdz jest takie samo jak w programie realizującym algorytm naiwny. W linii 5 zmienna ht przyjmuje wartość haszu dla pierwszego fragmentu tekstu. W linii 6 w zmiennej pot zapamiętana jest wartość M^{dw-1} . Pętla główna w liniach 7–18 działa podobnie do pętli głównej programu realizującego algorytm naiwny, ale porównywanie wzorca z fragmentem tekstu następuje tylko w przypadku takich samych wartości funkcji haszującej (linia 9).

W liniach 15–16 obliczamy wartość funkcji haszującej dla kolejnego fragmentu tekstu i przypisujemy ją zmiennej ht . Dzieje się tak pod warunkiem, że wzorec można jeszcze przesunąć (linia 15).

Funkcję Hash1 wywołujemy dla parametrów:

- ▶ ht – dotychczasowa wartość funkcji haszującej,
- ▶ $(pot * (t[p] - 'a')) \% N$ – wartość jednomianu reprezentującego pierwszy usuwany znak dotychczasowego fragmentu tekstu,
- ▶ $t[p+dw]$ – kolejny znak tekstu, który należy dodać do fragmentu tekstu.

Ćwiczenie 5

Napisz program, który wczyta tekst oraz wzorec, składające się z małych liter alfabetu łacińskiego, a następnie wyszuka wzorec w tekście z wykorzystaniem algorytmu Karpa–Rabina.

Złożoność obliczeniowa algorytmu Karpa–Rabina

Funkcja Potega jest wywoływana w funkcji Znajdz tylko raz, a funkcja Hash0 dwa razy. Czas wykonania tych funkcji nie ma więc wpływu na czas wyszukiwania wzorca w długim tekście.

Funkcja Hash1 jest wielokrotnie wywoływana w pętli (linia 16 w kodzie źródłowym funkcji Znajdz). Działa ona szybko i wykonuje jedynie kilka podstawowych działań arytmetycznych. Jej złożoność czasowa nie zależy ani od długości wzorca, ani od długości tekstu, wynosi więc $O(1)$. Algorytm Karpa–Rabina działa w złożoności czasowej $O(n + m)$, gdzie n oznacza długość tekstu, a m – długość wzorca.

Warto wiedzieć

Istnieją także inne algorytmy szybkiego wyszukiwania wzorca w tekście mające złożoność czasową $O(n + m)$, gdzie n oznacza długość tekstu, a m – długość wzorca.

A to ciekawe

Kryptograficzne funkcje haszujące

Funkcje haszujące są wykorzystywane również w kryptografii. Wiele serwisów internetowych przechowuje hasła użytkowników w postaci haszy. W czasie logowania podane przez użytkownika hasło jest haszowane i porównywane z haszem zapisanym w bazie danych. Funkcje haszujące stosuje się również m.in. w podpisach cyfrowych oraz w systemach kontroli wersji.



USERNAME :

PASSWORD :



Podsumowanie

- Algorytm naiwny wyszukiwania wzorca w tekście polega na porównywaniu wzorca z tekstem znak po znaku, a następnie przesuwaniu położenia wzorca względem tekstu o jedną pozycję w prawo.
- Złożoność obliczeniowa algorytmu naiwnego wynosi $O(n \cdot m)$, gdzie n oznacza długość tekstu, a m – długość wzorca.
- Metoda haszowania polega na przyporządkowaniu napisowi odpowiadającej mu wartości o określonym rozmiarze.
- Złożoność czasowa algorytmów wykorzystujących metodę haszowania zależy od doboru funkcji haszującej. Powinna ona działać w złożoności czasowej niezależnej od rozmiaru danych.
- Algorytm Karpa–Rabina, służący do wyszukiwania wzorca w tekście, wykorzystuje metodę haszowania – zamiast porównywać wzorzec z fragmentem tekstu, porównuje odpowiadające im hasze. Funkcja haszująca wyznacza wartość dla kolejnego fragmentu tekstu na podstawie dotychczasowej wartości w czasie $O(1)$ za pomocą arytmetyki modularnej.
- W arytmetyce modulo n wyniki działań arytmetycznych sprowadza się do przedziału $[0; n)$. Wynikiem jest reszta z dzielenia przez n .
- Złożoność obliczeniowa algorytmu Karpa–Rabina wynosi $O(n + m)$, gdzie n oznacza długość tekstu, a m – długość wzorca.



Zadania

- * **1** Przygotuj prezentację wyjaśniającą ideę metody haszowania.
- * **2** Dane są dwa napisy określające autora i tytuł książki. Zaproponuj funkcję haszującą, która umożliwi szybkie wyszukiwanie książek.
- * **3** Przygotuj prezentację wyjaśniającą zasady arytmetyki modularnej liczb całkowitych.
- ** **4** Program realizujący algorytm naiwny wyszukiwania wzorca w tekście zmodyfikuj tak, aby znajdował wszystkie wystąpienia wzorca w tekście.
- ** **5** Program realizujący algorytm Karpa–Rabina zmodyfikuj tak, aby znajdował wszystkie wystąpienia wzorca w tekście.
- ** **6** Poszukaj w internecie informacji na temat innych algorytmów wyszukiwania wzorca w tekście niż omówione w tym temacie i przygotuj prezentację na ten temat.
- *** **7** Poszukaj w internecie informacji na temat uproszczonego algorytmu Boyera–Moore’a, służącego do znajdowania wzorca w tekście, i napisz program realizujący ten algorytm.
- *** **8** Poszukaj w internecie informacji na temat algorytmu Morrisa–Pratta, służącego do znajdowania wzorca w tekście, i napisz program realizujący ten algorytm.

12. Szyfrowanie z kluczem publicznym. Algorytm RSA

Niektóre zagadnienia dotyczące cyberbezpieczeństwa znasz już z wcześniejszych lekcji informatyki. Być może pamiętasz, że z szyfrowania z użyciem klucza publicznego korzystasz praktycznie codziennie – podczas przeglądania szyfrowanej strony internetowej czy logowania do bankowości elektronicznej. W tym temacie przyjrzymy się algorytmom związanym z kryptografią asymetryczną oraz zaprogramujemy jeden z najważniejszych algorytmów kryptograficznych – algorytm RSA.

Cele lekcji

- Utrwalisz wiadomości dotyczące kryptografii symetrycznej i asymetrycznej.
- Poznasz algorytm RSA i jego przykładową implementację.
- Wygenerujesz klucze: publiczny i prywatny.
- Zaszzyfrujesz informacje kluczem publicznym i odszyfrujesz je kluczem prywatnym.

Podstawy kryptografii,
podręcznik *Informatyka*
na czasie 1. Zakres
rozszerzony,
s. 214–221 

Podstawami kryptografii, czyli zagadnieniami zawiązanymi z szyfrowaniem informacji, zajmowaliśmy się już w części pierwszej podręcznika. Przypomnijmy, że podczas szyfrowania informacja jawna jest

Szyfrogram (kryptogram)  przekształcana w **szyfrogram (kryptogram)**. Wykorzystuje się przy tym **klucz** , czyli dane umożliwiające zaszyfrowanie lub odszyfrowanie informacji.

Jeśli do deszyfrowania służy ten sam klucz, którego użyto do szyfrowania, lub można go łatwo wyznaczyć na podstawie tego klucza, to

Kryptografia symetryczna  mamy do czynienia z **kryptografią symetryczną**. Taki klucz musi być utrzymywany w tajemnicy.

Bezpieczniejszym sposobem szyfrowania jest zastosowanie **kryptografii asymetrycznej**, nazywanej też **kryptografią klucza publicznego**, w której występują dwa klucze: klucz publiczny oraz klucz prywatny. Na podstawie jednego klucza nie da się w rozsądnym czasie wyznaczyć drugiego, dlatego **klucz publiczny** może być przekazywany wszystkim zainteresowanym, którzy chcą zaszyfrować informacje. Odszyfrować może ją jednak tylko właściciel **klucza prywatnego** – nie można tego zrobić za pomocą klucza użytego do szyfrowania.

Możliwa jest także czynność odwrotna: szyfrowanie kluczem prywatnym i odszyfrowywanie kluczem publicznym. Wówczas informację może odczytać każdy, kto ma dostęp do klucza publicznego. Wykorzystuje się to w **podpisie elektronicznym**.

Kryptografia asymetryczna
(kryptografia klucza publicznego)

Klucz publiczny 

Klucz prywatny 

Podpis elektroniczny,
podręcznik *Informatyka*
na czasie 1. Zakres
rozszerzony,
s. 219–221 

12.1. Algorytm RSA

Jedną z najbardziej popularnych metod szyfrowania z kluczem publicznym jest **algorytm RSA**. Wykorzystuje on fakt, że nie potrafimy w rozsądnym czasie wykonać **faktoryzacji** bardzo dużej (kilkusetcyfrowej) liczby, czyli rozłożyć jej na czynniki pierwsze.

W metodzie RSA można wyróżnić trzy główne etapy: generowanie kluczy, szyfrowanie z kluczem publicznym oraz deszyfrowanie z kluczem prywatnym.

Algorytm RSA – generowanie kluczy

W pierwszym etapie generuje się klucze: publiczny i prywatny. Można to zrobić w trzech krokach.

► Krok 1. Obliczenia wstępne

Wybieramy dwie duże liczby pierwsze p i q oraz obliczamy iloczyn:

$$n = p \cdot q$$

$$m = (p - 1) \cdot (q - 1)$$

Liczby pierwsze p i q nie będą już potrzebne. Liczba n będzie elementem zarówno klucza publicznego, jak i prywatnego.

► Krok 2. Generowanie klucza publicznego

Poszukujemy liczby e ($1 < e < m$) względnie pierwszej z liczbą m utworzoną w poprzednim kroku, czyli takiej, która spełnia równość:

$$\text{NWD}(e, m) = 1$$

Można wykorzystać do tego **algorytm Euklidesa**. Sprawdzimy za jego pomocą, dla jakiej liczby e wartość $\text{NWD}(e, m)$ będzie równa 1. Liczba e jest liczbą nieparzystą, ponieważ m jest liczbą parzystą (jako iloczyn liczb parzystych), ale nie musi ona być liczbą pierwszą.

Para liczb (e, n) to klucz publiczny.

► Krok 3. Generowanie klucza prywatnego

Poszukujemy liczby d odwrotnej do liczby e w **arytmetyce modularnej** modulo m , tzn. szukamy liczby d spełniającej równość:

$$(d \cdot e) \bmod m = 1$$

W poprzednim kroku znaleźliśmy liczbę e , która spełnia równanie $\text{NWD}(e, m) = 1$. Spełnia ona również równanie:

$$\text{NWD}(e, m) \bmod m = 1$$

Z **rozszerzonego algorytmu Euklidesa** wiemy, że istnieją liczby całkowite x i y , które spełniają równość $\text{NWD}(e, m) = x \cdot e + y \cdot m$. Możemy więc zapisać:

$$(x \cdot e + y \cdot m) \bmod m = 1 \Leftrightarrow (x \cdot e) \bmod m + (y \cdot m) \bmod m = 1$$

$$\Leftrightarrow (x \cdot e) \bmod m = 1$$

Ostatnia równość wynika z tego, że $(y \cdot m) \bmod m = 0$, ponieważ liczba $y \cdot m$ dzieli się bez reszty przez m . Zauważ, że poszukiwana liczba d to współczynnik x przy liczbie e w ostatnim zapisie. Para liczb (d, n) to klucz prywatny.

► Algorytm RSA

► Faktoryzacja liczby

Warto wiedzieć

Nazwa algorytmu RSA pochodzi od pierwszych liter nazwisk jego twórców, profesorów Massachusetts Institute of Technology – Rona Rivesta, Adiego Shamira i Leonarda Adlemana. Opisali oni ten algorytm w 1977 r.

Warto wiedzieć

Liczba m jest równa wartości funkcji Eulera $\varphi(n)$ określającej, ile jest liczb względnie pierwszych z n mniejszych od n .

Algorytm Euklidesa, podręcznik *Informatyka na czasie 2. Zakres rozszerzony*, s. 102–108 [↗](#)

Arytmetyka modularna, s. 208 [↗](#)

Rozszerzony algorytm Euklidesa, podręcznik *Informatyka na czasie 2. Zakres rozszerzony*, s. 250–252 [↗](#)

Warto wiedzieć

Liczby 0 i 1 nie są szyfrowane, ponieważ po podniesieniu ich do potęgi otrzymamy niezmienną wartość.

Klucz publiczny,
s. 214

**Algorytm szybkiego
podnoszenia do potęgi,**
podręcznik *Informatyka
na czasie 2. Zakres
rozszerzony,*
s. 57–58

Klucz prywatny,
s. 214

Algorytm RSA – szyfrowanie informacji

W algorytmie RSA szyfrowane są liczby całkowite z przedziału $(1, n)$. Jeśli szyfrowany jest tekst, to jego kolejne znaki są zamieniane na liczby z tego przedziału. Kody znaków można przekształcić w jedną dużą liczbę. Można wykorzystać kody ASCII znaków i łączyć je np. po dwa znaki. Wtedy kod pierwszego znaku wystarczy pomnożyć przez 256 (liczbę wszystkich znaków ASCII) i dodać kod drugiego znaku.

Liczbę $x \in (1, n)$ szyfrujemy z wykorzystaniem **klucza publicznego** (e, n) w następujący sposób:

$$y = x^e \bmod n$$

Liczba y jest szyfrogramem liczby x . Potęgowanie wykonujemy w arytmetyce modulo n , więc wynik będzie liczbą mniejszą od n . Do szyfrowania można wykorzystać **algorytm szybkiego podnoszenia do potęgi** w arytmetyce modulo n .

Algorytm RSA – deszyfrowanie informacji

Żeby odszyfrować kryptogram y , należy wykonać potęgowanie z wykorzystaniem danych z **klucza prywatnego** (d, n) :

$$x = y^d \bmod n$$

Wartość wykładnika d zna tylko właściciel klucza prywatnego – nie da się jej wyznaczyć w rozsądnym czasie na podstawie znajomości klucza publicznego (e, n) .

Zapamiętaj

Algorytm RSA jest metodą szyfrowania z wykorzystaniem kluczy publicznego i prywatnego. Do wyznaczenia klucza publicznego można zastosować algorytm Euklidesa, a do wyznaczenia klucza prywatnego – rozszerzony algorytm Euklidesa. Szyfrowanie i odszyfrowywanie informacji zapisanej w postaci liczby sprowadza się do wykonania potęgowania w arytmetyce modulo n , np. za pomocą algorytmu szybkiego podnoszenia do potęgi.

A to ciekawe**Rewolucja w kryptografii**

Przez tysiące lat systemy kryptograficzne działały na zasadzie klucza symetrycznego. Jego postać i sposób użycia stawały się coraz bardziej skomplikowane, a zarazem nierozwiązany pozostawał problem, jak ten klucz bezpiecznie przekazywać i przechowywać. W 1976 r. Whitfield Diffie oraz Martin Hellman zaprezentowali koncepcję systemu z kluczem publicznym, która rozpoczęła nową erę w kryptografii. W konsekwencji powstały znacznie bezpieczniejsze systemy takie jak RSA, DSA czy ElGamal.

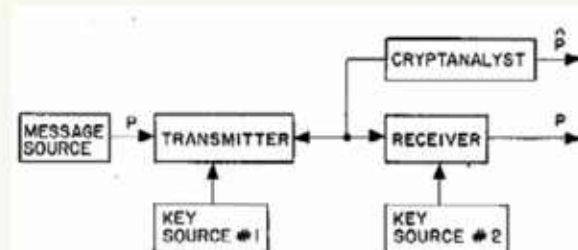


Fig. 2. Flow of information in public key system.

12.2. Program generujący klucz publiczny i klucz prywatny

Napiszemy teraz program generujący klucze zgodnie z algorytmem RSA. Nie będziemy się posługiwać wielkimi liczbami (kilkusetcyfrowymi), lecz wykorzystamy dwie małe liczby pierwsze $p = 13$ i $q = 23$, które pozwolą używać typu `int`. Prześledźmy kolejne kroki obliczeń.

► Krok 1. Obliczenia wstępne

$$n = 13 \cdot 23 = 299$$

$$m = 12 \cdot 22 = 264$$

► Krok 2. Generowanie klucza publicznego

Poszukujemy liczby względnie pierwszej z liczbą 264. Ponieważ musi to być liczba nieparzysta, rozpoczynamy poszukiwanie od liczby 3. Korzystając z algorytmu Euklidesa, możemy obliczyć, że $\text{NWD}(3, 264) = 3$, zatem będziemy przeglądać kolejne liczby nieparzyste (znajdziemy najmniejszą taką liczbę). Następną liczbą jest 5, dla której $\text{NWD}(5, 264) = 1$.

Znaleźliśmy więc parę (5, 299), która jest kluczem publicznym.

► Krok 3. Generowanie klucza prywatnego

Za pomocą **rozszerzonego algorytmu Euklidesa** znajdujemy rozkład:

$$\text{NWD}(5, 264) = 53 \cdot 5 + (-1) \cdot 264$$

Para (53, 299) jest kluczem prywatnym dla klucza publicznego (5, 299).

Warto wiedzieć

Liczb e większych od 1 i mniejszych od m spełniających warunek $\text{NWD}(e, n) = 1$ może być wiele, a więc na podstawie liczb pierwszych p i q można wyznaczyć więcej niż jeden klucz publiczny.

Rozszerzony algorytm Euklidesa,
podręcznik *Informatyka na czasie 2. Zakres rozszerzony*,
s. 250–252 [↗](#)

Ćwiczenie 1

- Poszukaj innej pary kluczy dla liczb pierwszych 13 i 23.
- Znajdź klucze publiczny i prywatny dla liczb pierwszych 17 i 19.

Specyfikację problemu generowania kluczy w algorytmie RSA możemy sformułować następująco:

Specyfikacja

Dane: p, q – liczby pierwsze, $p > 2, q > 2$.

Wynik: d, e, n – liczby całkowite dodatnie tworzące odpowiednio klucz publiczny (e, n) i klucz prywatny (d, n) szyfru RSA.

Zapiszmy w pseudokodzie algorytm generowania kluczy, zgodnie z powyższą specyfikacją.

```
n ← p*q
m ← (p-1)*(q-1)
e ← 3
dopóki NWD(e,m) ≠ 1 wykonuj e ← e + 2
(x,y) ← NWDroz(e,m)
d ← x
jeśli d < 0 to d ← d + m
```


W pseudokodzie wykorzystaliśmy funkcje NWD oraz NWDroz, realizujące odpowiednio algorytm Euklidesa oraz rozszerzony algorytm Euklidesa. Oto kod źródłowy programu generującego klucze w algorytmie RSA:

Fragment kodu
źródłowego programu
generującego
klucz publiczny
i klucz prywatny
w algorytmie RSA

```

1.  int NWD(int a, int b)
2.  {
3.      int pom;
4.      while (b!=0)
5.      {
6.          pom=b;
7.          b=a%b;
8.          a=pom;
9.      }
10.     return a;
11. }
12.
13. pair<int,int> NWDroz(int a, int b)
14. {
15.     if (b==0) return make_pair(1,0);
16.     pair<int,int> pom=NWDroz(b,a%b);
17.     return make_pair(pom.second,pom.first-(a/b)*pom.second);
18. }
19.
20. int main()
21. {
22.     int d, e, n, m, p, q;
23.     cout<<"Podaj dwie liczby pierwsze p i q:"<<endl;
24.     cout<<"p = "; cin>>p;
25.     cout<<"q = "; cin>>q;
26.     n=p*q;
27.     m=(p-1)*(q-1);
28.     e=3;
29.     while (NWD(e,m)!=1) e=e+2;
30.     d=NWDroz(e,m).first;
31.     if (d<0) d=d+m;
32.     cout<<"Klucz publiczny: "<<e<<" "<<n<<endl;
33.     cout<<"Klucz prywatny: "<<d<<" "<<n;
34.     return 0;
35. }

```

Dobra rada

Omówienie kodu źródłowego funkcji NWD oraz NWDroz znajdziesz w podręczniku *Informatyka na czasie 2. Zakres rozszerzony* na s. 106–108 oraz 250–252.

W liniach 1–11 zdefiniowana jest funkcja NWD, realizująca algorytm Euklidesa, a w liniach 13–18 – funkcja NWDroz, realizująca rozszerzony algorytm Euklidesa.

W funkcji main w liniach 24–25 wczytywane są dwie liczby pierwsze. W linii 26 obliczana jest wartość zmiennej n, w liniach 28–29 – zmiennej e, a w liniach 30–31 – zmiennej d. Jeśli funkcja NWDroz zwróci ujemną wartość d, to dodawana jest do niej wartość m (linia 31). W liniach 32–33 wypisywane są wartości obliczonych kluczy.

Ćwiczenie 2

Napisz program, który wczyta dwie liczby pierwsze p i q oraz wygeneruje klucz publiczny i klucz prywatny według algorytmu RSA. Wykorzystaj funkcje NWD oraz NWDroz.

12.3. Program szyfrujący informacje kluczem publicznym

Przypomnijmy, że w algorytmie RSA szyfrowane są liczby mniejsze od n . W naszym przykładzie n jest równe 299, więc możemy szyfrować kody ASCII pojedynczych znaków (liczby mniejsze od 256).

Na przykład szyfrowanie wielkiej litery A (kod ASCII 65) kluczem publicznym (5, 299) można zapisać następująco:

$$\begin{aligned} y &= 65^5 \bmod 299 = (65 \cdot (65^4 \bmod 299)) \bmod 299 = \\ &= (65 \cdot ((65^2 \bmod 299) \cdot (65^2 \bmod 299)) \bmod 299) \bmod 299 = \\ &= (65 \cdot (39 \cdot 39) \bmod 299) \bmod 299 = (65 \cdot 26) \bmod 299 = 195 \end{aligned}$$

Ćwiczenie 3

Wyznacz szyfrogram litery Z (kod ASCII 90) algorytmem RSA z kluczem (5, 299).

Napiszemy teraz program, który będzie szyfrował plik tekstowy szyfrem RSA. Będziemy szyfrować wszystkie znaki tekstu łącznie ze spacjami, znakami przestankowymi i sterującymi, np. znacznikiem końca wiersza.

Ponieważ będziemy używać klucza bazującego na małych liczbach, będziemy szyfrować pojedyncze znaki (ich kody ASCII). Wynikiem działania programu będzie plik tekstowy złożony z liczb całkowitych oddzielonych spacjami, czyli szyfrogramów poszczególnych znaków.

Oto specyfikacja problemu:

Specyfikacja

Dane: e, n – liczby całkowite dodatnie tworzące klucz publiczny RSA, plik wejściowy – plik tekstowy do zaszyfrowania.

Wynik: plik wynikowy – plik tekstowy składający się z liczb oddzielonych spacjami, który jest szyfrogramem tekstu z pliku plik wejściowy utworzonym zgodnie z algorytmem RSA z kluczem publicznym (e, n).

Zapiszemy w pseudokodzie algorytm szyfrowania informacji. Szyfrowanie pojedynczego znaku realizuje funkcja RSA. Zapiszemy również w języku C++ kod źródłowy funkcji RSA oraz funkcji main.

Warto wiedzieć

Innym sposobem na szyfrowanie znaków tekstu jawnego mogłoby być łączenie kodów kilku znaków w jedną liczbę. Na przykład dla dwóch znaków byłby to wzór $a \cdot 256 + b$, gdzie a jest kodem ASCII pierwszego znaku, a b – kodem ASCII drugiego znaku. Liczba n powinna być wówczas większa od 65 535.


```

dopóki nie koniec pliku wejściowego wykonuj
    ch ← znak z pliku wejściowego
    x ← kod_ASCII(ch)
    y ← RSA(x,e,n)
    zapisz y do pliku wynikowego

```

Fragment kodu
źródłowego programu
szyfrującego tekst
z wykorzystaniem
klucza publicznego
w algorytmie RSA

```

1. int RSA(int podst, int wykl, int n)
2. {
3.     int w=1;
4.     while (wykl>0)
5.     {
6.         if (wykl%2==1) w=(w*podst)%n;
7.         wykl=wykl/2;
8.         if (wykl>0) podst=(podst*podst)%n;
9.     }
10.    return w;
11. }
12.
13. int main()
14. {
15.     ifstream we("t_jawny_RSA.txt");
16.     ofstream wy("szyfrogram_RSA.txt");
17.     unsigned char ch;
18.     int e,n;
19.     cout<<"Podaj klucz publiczny:"<<endl;
20.     cout<<"e = "; cin>>e;
21.     cout<<"n = "; cin>>n;
22.     we>>noskipws;
23.     while (we>>ch) wy<<RSA(ch,e,n)<<" ";
24.     we.close(); wy.close();
25.     cout<<"Tekst zostal zaszyfrowany";
26.     return 0;
27. }

```

Dobra rada

Pamiętaj, że obsługa plików w programie wymaga dołączenia do kodu źródłowego informacji o korzystaniu z biblioteki `fstream`.

Algorytm Karpa–Rabina,
s. 207 

Funkcja RSA (linie 1–11) realizuje algorytm szybkiego podnoszenia do potęgi w arytmetyce modulo n , analogicznie jak funkcja Potega w algorytmie Karpa–Rabina. W linii 15 otwierany jest do odczytu plik wejściowy o nazwie `t_jawny_RSA.txt`, a w linii 16 otwierany jest do zapisu plik wynikowy `szyfrogram_RSA.txt`.

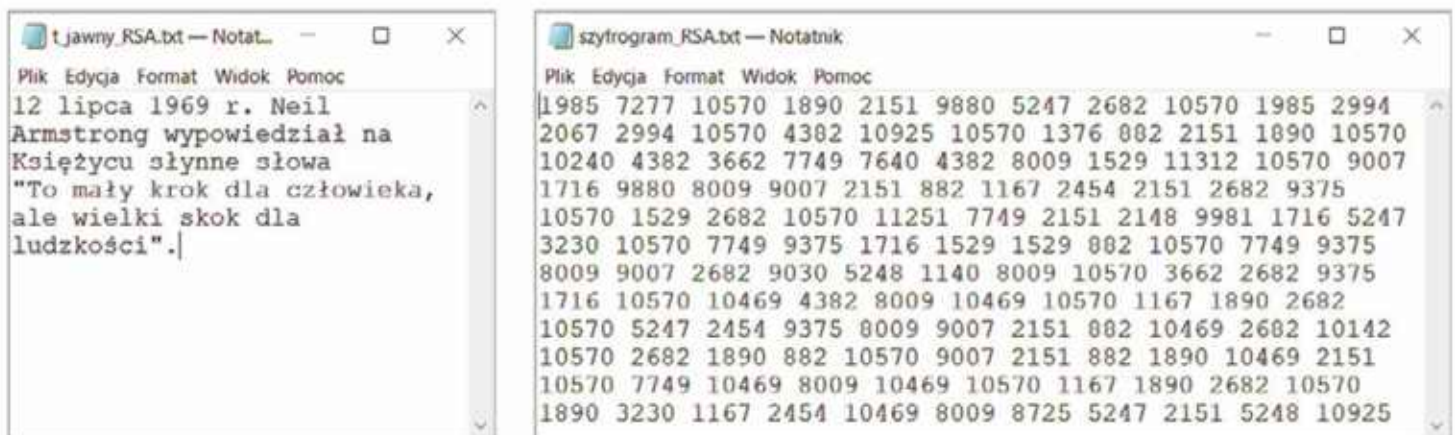
Typ unsigned char W linii 17 deklarujemy zmienną `ch` typu `unsigned char`, który reprezentuje wartości od 0 do 255. Ponieważ chcemy szyfrować wszystkie znaki zawarte w tablicy ASCII, a typ `char` reprezentuje liczby z zakresu od -128 do 127 . Znaki, których kody ASCII przekraczają 127, byłyby reprezentowane przez liczby ujemne.

W liniach 19–21 wczytywany jest klucz publiczny służący do szyfrowania. Wyjaśnijmy linie 22 i 23. Domyślnie tzw. **białe znaki** (ang. *whitespaces*), np. spacje lub znaczniki końca wiersza, są pomijane podczas odczytywania danych wejściowych za pomocą instrukcji `we>>ch`.

Ponieważ chcemy odczytywać również białe znaki ze strumienia wejścia, należy to wymusić z wykorzystaniem **manipulatora** `noskipws` (linia 22). W warunku pętli w linii 23 znajduje się instrukcja odczytu danych `we>>ch`. Jeśli zakończy się ona niepowodzeniem, to warunek przyjmie wartość liczbową 0, oznaczającą fałsz, w przeciwnym przypadku – wartość różną od zera, interpretowaną jako prawda. Do pliku zapisywane są: wynik funkcji RSA (czyli liczba będąca zakodowanym znakiem) oraz spacja oddzielająca kolejne liczby.

Pierwszym parametrem funkcji RSA jest znak reprezentowany przez kod ASCII w zakresie do 255. Parametr ten jest wykorzystywany jako zmienna lokalna `podst` typu `int`, więc jej wartość może przekroczyć 255. Dwa następne parametry to liczby określające klucz publiczny.

Rysunek 12.1 przedstawia tekst jawny i utworzony szyfrogram.



Rys. 12.1. Tekst jawny i jego szyfrogram dla klucza publicznego (11, 11561)

Ćwiczenie 4

Napisz program wczytujący klucz publiczny i szyfrujący dane z pliku wejściowego o nazwie `t_jawny_RSA.txt` do pliku `szyfrogram_RSA.txt`. Wykorzystaj funkcję `RSA`.

A to ciekawe

Czy szyfr RSA można złamać?

Obecnie powszechnie stosuje się klucze RSA o długości 2048 bitów. Najdłuższy klucz, który dotychczas złamano, składał się z 768 bitów. Udało się to w 2009 r. z wykorzystaniem klastra, czyli wielu połączonych jednostek komputerowych. Zastosowano wówczas metodę brute-force, a łączny czas prac nad projektem wyniósł 2 lata. Szacuje się, że komputery kwantowe dzięki swojej szybkości będą mogły złamać klucz o długości 2048 bitów w ok. 8 godzin. Należy się więc spodziewać, że w niedalekiej przyszłości do użytku wejdą klucze składające się z 4096 bitów.



12.4. Program deszyfrujący kryptogram kluczem prywatnym

Żeby odszyfrować informację, należy wykonać takie same czynności jak podczas szyfrowania, ale z wykorzystaniem liczb z szyfrogramu i klucza prywatnego.

Odszyfrujemy szyfrogram 195 kluczem (53, 299).

$$x = 195^{53} \bmod 299 = 65$$

Otrzymujemy liczbę 65, która interpretowana jako kod ASCII reprezentuje literę A.

Kod źródłowy funkcji RSA,
s. 220 [🔗](#)

Prześledźmy obliczenia, które wykonuje **funkcja** RSA dla powyższych argumentów. Pośrednie kroki wykonywane w celu wyznaczenia wartości funkcji $\text{RSA}(195, 53, 299)$ ilustruje tabela 12.1.

Iteracja pętli	Wartość w	Wartość wyk1	Wartość podst
–	1	53	195
1	$(1 \cdot 195) \bmod 299 = 195$	26	$(195 \cdot 195) \bmod 299 = 52$
2	195	13	$(52 \cdot 52) \bmod 299 = 13$
3	$(195 \cdot 13) \bmod 299 = 143$	6	$(13 \cdot 13) \bmod 299 = 169$
4	143	3	$(169 \cdot 169) \bmod 299 = 156$
5	$(143 \cdot 156) \bmod 299 = 182$	1	$(156 \cdot 156) \bmod 299 = 117$
6	$(182 \cdot 117) \bmod 299 = 65$	0	–

Tabela 12.1. Kolejne kroki obliczania wartości $195^{53} \bmod 299$

Ćwiczenie 5

Odszyfruj szyfrogram 129 kluczem prywatnym (53, 299).

Napišemy teraz program, który będzie deszyfrował kryptogram utworzony szyfrem RSA. Danymi będą plik tekstowy złożony z liczb oddzielonych spacjami oraz klucz prywatny. Wynikiem będzie plik tekstowy z odszyfrowanym tekstem.

Oto specyfikacja problemu:

Specyfikacja

Dane: d, n – liczby całkowite dodatnie tworzące klucz prywatny RSA, plik wejściowy – plik tekstowy złożony z liczb całkowitych dodatnich oddzielonych spacjami – zaszyfrowany tekst.

Wynik: plik wynikowy – plik tekstowy zawierający kryptogram z pliku plik wejściowy odszyfrowany kluczem prywatnym (d, n).

Zapiszmy w pseudokodzie algorytm deszyfrowania informacji.

```
dopóki nie koniec pliku wejściowego wykonuj
    y ← liczba z pliku wejściowego
    x ← RSA(y,d,n)
    zapisz znak_ASCII(x) do pliku wynikowego
```

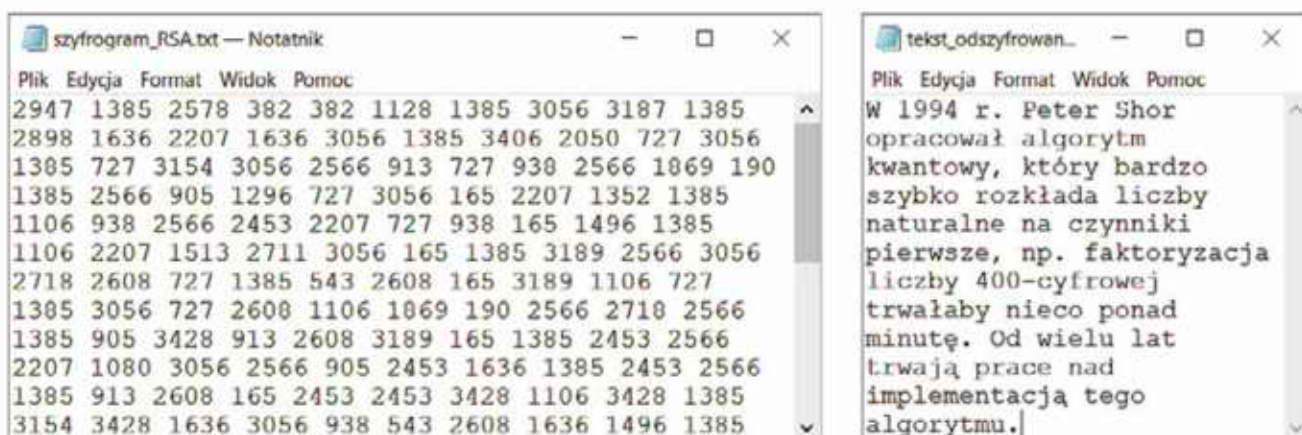
Oto kod źródłowy funkcji main w programie deszyfrującym:

```
1. int main()
2. {
3.     ifstream we("szyfrogram_RSA.txt");
4.     ofstream wy("tekst_odszyfrowany.txt");
5.     int d,n,x;
6.     cout<<"Podaj klucz prywatny:"<<endl;
7.     cout<<"d = "; cin>>d;
8.     cout<<"n = "; cin>>n;
9.     while (we>>x) wy<<char(RSA(x,d,n));
10.    we.close(); wy.close();
11.    cout<<"Tekst został odszyfrowany";
12.    return 0;
13. }
```

❶ Kod źródłowy funkcji main programu deszyfrującego szyfrogram z wykorzystaniem klucza prywatnego w algorytmie RSA

W linii 9 znajduje się pętla odczytująca liczby, analogiczna do tej, która umożliwia odczytywanie znaków tekstu jawnego. Jeśli nie uda się odczytać liczby (np. gdy program dojdzie do końca pliku), instrukcja `we>>x` zwróci wartość liczbową 0, czyli fałsz. Do pliku zapisujemy znak, więc dokonujemy konwersji kodu ASCII na typ `char`. Kod źródłowy funkcji RSA jest taki sam jak w programie szyfrującym. Rysunek 12.2 przedstawia szyfrogram i tekst jawny.

Kod źródłowy funkcji RSA, s. 220 [↗](#)



Rys. 12.2. Fragment szyfrogramu i tekst odszyfrowany z kluczem prywatnym (2107, 3487)

Ćwiczenie 6

Napisz program wczytujący klucz prywatny i deszyfrujący szyfrogram z pliku `szyfrogram_RSA.txt`. Odszyfrowany tekst zapisz do pliku `tekst_odszyfrowany.txt`. Wykorzystaj funkcję RSA.

Podsumowanie

- W kryptografii asymetrycznej wykorzystuje się dwa klucze: klucz publiczny (ogólnie dostępny) i prywatny (dostępny tylko dla właściciela). Informację zaszyfrowaną kluczem publicznym można odszyfrować tylko kluczem prywatnym.
- W podpisie elektronicznym stosuje się szyfrowanie kluczem prywatnym i odszyfrowywanie kluczem publicznym. Wówczas informację może odszyfrować każdy, kto ma dostęp do klucza publicznego.
- Jednym z algorytmów szyfrowania z kluczem publicznym jest algorytm RSA. Jego działanie opiera się na fakcie, że nie znamy szybkiego algorytmu faktoryzacji (rozkładu na czynniki pierwsze) bardzo dużych liczb.
- W algorytmie RSA klucz publiczny to para liczb (e, n) , gdzie n jest iloczynem dwóch dużych liczb pierwszych p i q , natomiast e jest liczbą względnie pierwszą z liczbą $m = (p - 1) \cdot (q - 1)$, $1 < e < m$. Klucz prywatny to para (d, n) , gdzie d jest odwrotnością liczby e w arytmetyce modulo m .
- W algorytmie RSA szyfrowane są liczby całkowite większe od 1 i mniejsze od n . Kryptogramem liczby x jest liczba $y = x^e \bmod n$. Żeby odszyfrować informację, należy wykonać działanie $y^d \bmod n$.

Zadania

- ★ **1** Wygeneruj klucze publiczny i prywatny z użyciem liczb pierwszych $p = 251$ i $q = 263$.
- ★ **2** Przygotuj prezentację na temat szyfrowania z kluczem publicznym.
- ★★ **3** Poszukaj w dostępnych źródłach informacji na temat algorytmów kryptografii asymetrycznej innych niż RSA i przygotuj prezentację na ten temat.
- ★★ **4** Program generujący klucze publiczny i prywatny na podstawie liczb pierwszych p i q zmodyfikuj tak, aby generował wszystkie pary kluczy dla liczb p i q .
- ★★★ **5** Napisz program szyfrujący plik tekstowy (np. *tajny_tekst.txt*), który otrzymasz od nauczyciela, algorytmem RSA w taki sposób, aby na podstawie dwóch kolejnych znaków z pliku wyznaczał liczbę do szyfrowania. Zastosuj klucz publiczny (3, 66013). Wynik w postaci liczb całkowitych oddzielanych spacjami zapisz w pliku tekstowym *zaszyfrowany_RSA.txt*.
Wskazówka: Parametr podst. funkcji szyfrującej powinien być typu **long long**.
- ★★★ **6** Napisz program deszyfrujący kryptogram zaszyfrowany algorytmem RSA zawarty w pliku tekstowym (np. *tajny_szyfrogram.txt*), który otrzymasz od nauczyciela. Kryptogram składa się z liczb całkowitych oddzielonych spacjami. Jedna liczba stanowi szyfrogram dwóch znaków tekstu jawnego. Do odszyfrowania użyj klucza prywatnego (43667, 66013). Wynik zapisz w pliku tekstowym *tajny_odszyfrowany.txt*.
Wskazówka: Parametr podst. funkcji deszyfrującej powinien być typu **long long**.

13. Programowanie obiektowe

Wiele systemów informatycznych i programów ma budowę okienkową. Dzięki niej aplikacje stały się łatwiejsze w obsłudze i dostępne dla szerszej grupy użytkowników. Szybki rozwój tego typu oprogramowania umożliwiły m.in. zasady programowania obiektowego, którego podstawy poznasz w tym temacie.

Cele lekcji

- Poznasz pojęcia klasy i obiektu oraz atrybutu i metody.
- Dowiesz się, na czym polega hermetyzacja danych.
- Zbudujesz hierarchię klas.
- Poznasz pojęcie polimorfizmu i zastosujesz metody wirtualne.

Do tej pory tworzyliśmy programy zgodnie z zasadami **programowania strukturalnego**. Instrukcje wykonywały się sekwencyjnie, do sterowania wykorzystywaliśmy instrukcje warunkowe i pętle, a rozwiązania mniejszych problemów zapisywaliśmy w postaci funkcji. Dane i operacje na nich traktowane były jednak niezależnie. Komunikacja między funkcjami odbywała się za pośrednictwem parametrów oraz zwracanych wartości.

● Programowanie strukturalne

Postawiony problem staraliśmy się podzielić na mniejsze problemy, określić sposób komunikacji między funkcjami i zapisać algorytm rozwiązania problemu głównego, bez zajmowania się na początku szczegółami. Następnie zapisywaliśmy rozwiązania mniejszych problemów. Taki sposób postępowania podczas projektowania programów nazywa się **metodą zstępującą** (ang. *top-down approach*). Postępowanie odwrotne, czyli od szczegółu do ogółu, nazywa się **metodą wstępującą** (ang. *bottom-up approach*). Jeśli ją zastosujemy, rozwiązania problemów szczegółowych może być trudno połączyć w całość bez odpowiednio uzgodnionej komunikacji między funkcjami.

● Metoda zstępująca

● Metoda wstępująca

W **programowaniu obiektowym** (ang. *object-oriented programming*) dane są ściśle połączone z operacjami na nich wykonywanymi. Program jest traktowany jako zbiór obiektów współpracujących ze sobą w celu rozwiązania problemu. Programowanie strukturalne i obiektowe często są łączone. Wiele języków programowania, w tym C++, umożliwia korzystanie z obu tych podejść.

● Programowanie obiektowe

My także w tym podręczniku stosowaliśmy programowanie obiektowe, wykorzystując chociażby typ **string**. Nauczymy się teraz definiować własne struktury obiektowe.

13.1. Podstawowe pojęcia

Dwa podstawowe pojęcia związane z programowaniem obiektowym to

- Klasa** • klasa i obiekt. **Klasa** (ang. *class*) jest abstrakcyjnym modelem pewnych
- Atrybut (pole)** • danych, które nazywamy **atrybutami** (ang. *attributes*) lub **polami**, oraz
- Metoda** • operacji wykonywanych na tych danych, które nazywamy **metodami**
- Obiekt** • (ang. *methods*). Metody są zapisywane w postaci funkcji. **Obiekt** (ang. *object*) to konkretny przedstawiciel klasy (nazywany też instancją klasy), a więc dane, na których metody tej klasy mogą wykonywać operacje.

Na przykład deklaracja `string s;` oznacza utworzenie obiektu o nazwie należącego do s klasy `string`. Obiekt jest więc zmienną, a klasa – typem zmiennej. Wykorzystując notację z kropką, możemy wykonywać różne operacje na obiekcie s za pomocą metod klasy, np. sprawdzić długość napisu (liczbę znaków): `s.size()`.

Warto wiedzieć

W podręczniku *Informatyka na czasie 2. Zakres rozszerzony* (s. 109) omówiliśmy reprezentację liczb wymiernych z wykorzystaniem struktury. Opisywała ona jedynie dane: licznik i mianownik ułamka. Operacje wykonywane na ułamku nie były bezpośrednio powiązane z jego reprezentacją.

Definiowanie własnej klasy

Zdefiniujemy teraz własną klasę o nazwie Wymierna. Klasa ta będzie opisywała liczbę wymierną oraz podstawowe operacje: wczytanie ułamka z klawiatury, skrócenie go i wypisanie skróconego ułamka na ekranie. Definicja klasy jest podobna do definicji struktury, ale w obrębie klasy zadeklarujemy także metody.

Definicja klasy Wymierna

```
1. class Wymierna
2. {
3.     public:
4.         void Czytaj();
5.         void Skroc();
6.         void Pisz();
7.     private:
8.         int licz, mian;
9.         int NWD(int a, int b);
10. };
```

Definicja klasy • **Definicję klasy** rozpoczynamy od słowa kluczowego **class**, następnie podajemy nazwę klasy, a w nawiasach klamrowych – opis klasy w postaci deklaracji atrybutów i metod. Po nawiasie zamykającym stawiamy średnik.

W klasie Wymierna zadeklarowaliśmy dwa atrybuty: `licz` i `mian`, oba typu `int`. Reprezentują one licznik i mianownik liczby wymiernej (linia 8), analogicznie jak wyglądałoby to w strukturze opisującej liczbę wymierną. W liniach 4–6 występują nagłówki metod `Czytaj`, `Skroc` i `Pisz`, umożliwiających odpowiednio: wczytanie ułamka z klawiatury, jego skrócenie i wypisanie na ekranie. Dodatkowo w linii 9 zadeklarowany jest nagłówek pomocniczej metody `NWD`, wykorzystywanej do skrócenia ułamka.

W definicji klasy użyte zostały słowa kluczowe **public** i **private**, które są nazywane **specyfikatorami dostępu** i określają prawa dostępu do atrybutów oraz metod (składowych klasy). Specyfikatry dostępu umożliwiają tzw. **hermetyzację danych**, czyli ukrywanie atrybutów i metod tak, że są one dostępne tylko dla metod danej klasy.

Specyfikator **public** oznacza udostępnienie występujących po nim atrybutów i metod poza klasę, natomiast specyfikator **private** ukrywa wybrane atrybuty i metody.

W klasie `Wymierna` metody `Czytaj`, `Skroc` i `Pisz` są publiczne, natomiast atrybuty `licz` i `mian` oraz metoda `NWD` są prywatne. Dlatego kiedy korzystamy z obiektów klasy `Wymierna`, mamy do dyspozycji tylko metody `Czytaj`, `Skroc` i `Pisz`. Próba bezpośredniego odwołania się do atrybutów reprezentujących ułamek lub metody `NWD` spowoduje błąd kompilacji.

Dzięki takiemu podejściu dostęp do danych z klasy realizuje się wyłącznie za pomocą metod. Zaleca się ukrywanie danych i umożliwianie do nich dostępu właśnie za pomocą metod, ponieważ zapewnia to kontrolę nad danymi i eliminuje potencjalne błędy.

Oto definicje wybranych metod występujących w klasie `Wymierna`:

```

1. void Wymierna::Czytaj()
2. {
3.     cout<<"Licznik = "; cin>>licz;
4.     do
5.     {
6.         cout<<"Mianownik = "; cin>>mian;
7.     }
8.     while (mian<1);
9. }
10.
11. void Wymierna::Skroc()
12. {
13.     int n=NWD(abs(licz),mian);
14.     licz=licz/n;
15.     mian=mian/n;
16. }
17.
18. void Wymierna::Pisz()
19. {
20.     if (licz<0) cout<<"(";
21.     cout<<licz;
22.     if (mian>1) cout<<"/"<<mian;
23.     if (licz<0) cout<<")";
24. }
```

Specyfikatry dostępu

Hermetyzacja danych

Dobra rada

Specyfikatorów dostępu możesz używać wielokrotnie. Dany specyfikator obowiązuje do momentu zastosowania innego. Domyślnym specyfikatorem (jeśli nie został użyty żaden) jest **private**.

Kod źródłowy wybranych metod klasy `Wymierna`

Dobra rada

Aby zdefiniować metodę `NWD` w klasie `Wymierna`, możesz skorzystać z algorytmu Euklidesa w wersji z dzieleniem. Kod źródłowy tej metody może mieć postać:

```

int Wymierna::
    NWD(int a, int b)
{
    int pom;
    while(b!=0)
    {
        pom=b;
        b=a%b;
        a=pom;
    }
    return a;
}
```


Zwróć uwagę na nagłówki definiowanych metod. Za pomocą

Operator zasięgu `::` **operatora zasięgu** `::` wskazujemy, z jakiej klasy pochodzi dana metoda.

**Dodawanie i mnożenie
liczb wymiernych**,
podręcznik *Informatyka
na czasie 2. Zakres
rozszerzony*,
s. 110–114 [🔗](#)

W mianowniku ułamka nie może wystąpić liczba 0. Zakładamy ponadto, że znak ułamka jest pamiętany w liczniku, a więc mianownik jest liczbą większą lub równą 1. Założenia te uwzględniliśmy w definicji metody Czytaj.

Pozostałe metody są analogiczne do funkcji używanych podczas **dobawania i mnożenia liczb wymiernych**. Oto kod źródłowy funkcji main wykorzystujący klasę Wymierna:

Kod źródłowy `main`
funkcji `main`
wykorzystujący
klasę `Wymierna`

```
1. int main ()
2. {
3.     Wymierna a;
4.     a.Czytaj();
5.     a.Skroc();
6.     a.Pisz();
7.     return 0;
8. }
```

W linii 3 zadeklarowana jest zmienna `a` typu `Wymierna`, czyli obiekt klasy `Wymierna`. W następnych liniach są wywoływane dla tego obiektu metody `Czytaj`, `Skroc` i `Pisz`, które zapisujemy z wykorzystaniem notacji z kropką.

Rysunek 13.1 przedstawia przykładowe wywołanie programu. Metoda `Czytaj` pobiera od użytkownika wartości licznika i mianownika. Jeśli wartość mianownika jest mniejsza lub równa 0, to program umożliwia ponowne wpisanie poprawnej wartości. Metoda `Skroc` skraca ułamek, a metoda `Pisz` wypisuje jego wartość po skróceniu.

```
Licznik = 8
Mianownik = 0
Mianownik = 36
2/9
```

Rys. 13.1. Przykładowe wywołanie programu

A to ciekawe

Pierwszy język programowania obiektowego

Programowanie obiektowe stało się popularne wraz z rozwojem systemów okienkowych na przełomie lat 70. i 80. XX w. Współcześnie niemal wszystkie języki umożliwiają programowanie obiektowe, jednak jego idea powstała znacznie wcześniej. Pierwszym językiem, który wprowadził pojęcia klasy i obiektu, był język Simula 67. Opracowali go w latach 60. XX w. norwescy naukowcy Ole-Johan Dahl, Bjørn Myhrhaug i Kristen Nygaard.



Ćwiczenie 1

Napisz z wykorzystaniem programowania obiektowego program wczytujący z klawiatury licznik i mianownik ułamka, skracający go i wypisujący na ekranie. Zdefiniuj w tym celu klasę `Wymierna`.

Zapamiętaj

W programowaniu obiektowym klasa opisuje abstrakcyjny model danych i operacji na nich. Dane są reprezentowane przez atrybuty, a operacje są realizowane przez metody. Obiekt to przedstawiciel danej klasy. Hermetyzacja danych oznacza ukrywanie danych w klasie po to, aby dostęp do atrybutów obiektu był ograniczony.

13.2. Konstruktor i przeciążanie metod

W klasie można zdefiniować specjalną metodę, którą nazywa się **konstruktorem**. Konstruktor nie wywołujemy samodzielnie, ale jest on automatycznie wywoływany podczas tworzenia obiektu, czyli np. w momencie wywołania funkcji, w której obiekt jest zadeklarowany.

Konstruktor ma taką samą nazwę jak klasa i nie określa się typu zwracanej przez niego wartości. Parametrom konstruktora można również przypisać wartość domyślną.

Uzupełnijmy klasę `Wymierna` o deklarację konstruktora przypisującego wartości początkowe licznika i mianownika ułamka.

Warto wiedzieć

Jeśli jest taka potrzeba, można również wywołać konstruktor samodzielnie.

```
1. class Wymierna
2. {
3.     public:
4.         Wymierna(int alicz=0, int amian=1);
5.         void Czytaj();
6.         void Skroc();
7.         void Pisz();
8.     private:
9.         int licz, mian;
10.        int NWD(int a, int b);
11. };
```

➊ Definicja klasy `Wymierna` uzupełniona o deklarację konstruktora

Deklaracja konstruktora klasy `Wymierna` znajduje się w linii 4. Zwróć uwagę na nazwy parametrów formalnych konstruktora. Żeby odróżnić je od nazw atrybutów (linia 9), będziemy je poprzedzać literą `a`. Jeśli podczas deklaracji obiektu klasy `Wymierna` nie określimy jego wartości początkowych, to domyślnie licznik otrzyma wartość 0, a mianownik – wartość 1.

Definicja konstruktora może mieć następującą postać:

Kod źródłowy 
konstruktora
odpowiadającego za
przypisanie wartości
początkowych obiektom
klasy Wymierna

Warto wiedzieć

W definicji metody (także konstruktora) nie trzeba powtarzać listy parametrów formalnych określonych w nagłówku metody w definicji klasy.

```
1. Wymierna::Wymierna(int alicz, int amian)
2. {
3.     if (amian > 0)
4.     {
5.         licz = alicz; mian = amian;
6.         Skroc();
7.     }
8.     else
9.     {
10.        licz = 0; mian = 1;
11.    }
12. }
```

Konstruktor sprawdza, czy wartość mianownika jest większa od 0. Jeśli tak, to przypisuje wartości atrybutom obiektu i skraca ułamek (wywołuje metodę Skroc). Jeśli mianownik jest mniejszy lub równy 0, konstruktor przypisuje atrybutom obiektu domyślne wartości.

Oto przykładowy kod źródłowy funkcji main przedstawiającej działanie konstruktora klasy Wymierna:

Kod źródłowy 
funkcji main
przedstawiającej
działanie konstruktora
klasy Wymierna

```
1. int main ()
2. {
3.     Wymierna a(-6,9), b(5,-10), c(3), d;
4.     cout << "a = "; a.Pisz();
5.     cout << "b = "; b.Pisz();
6.     cout << "c = "; c.Pisz();
7.     cout << "d = "; d.Pisz();
8.     return 0;
9. }
```

W linii 3 zadeklarowane są obiekty a, b, c, d typu Wymierna. W momencie tworzenia każdego z nich wywoływany jest konstruktor. Rysunek 13.2 przedstawia efekt działania powyższej funkcji main.

Dla obiektu a wartości początkowe to -6 i 9. Po skróceniu ułamek przyjmuje wartość $-\frac{2}{3}$. Dane dla obiektu b są niezgodne z założeniami, więc przyjmuje on wartość domyślną 0. Dla obiektu c został podany tylko jeden parametr. Oznacza to określenie wartości pierwszego parametru na liście parametrów, czyli licznika. Mianownik przyjmie wartość domyślną 1.

```
a = (-2/3)
b = 0
c = 3
d = 0
```

Rys. 13.2. Wartości zadeklarowanych obiektów klasy Wymierna

Dobra rada

Pamiętaj, że domyślna wartość obiektu może być dowolnie ustalona przez programistę. W przypadku ułamka ustaliliśmy ją na 0 (licznik 0, mianownik 1).

Dla obiektu `d` nie są określone wartości początkowe, więc będą one domyślne.

Każdy z zadeklarowanych obiektów ma własne kopie atrybutów, ale metody są wspólne dla wszystkich obiektów – występują tylko raz.

Ćwiczenie 2

- a. Uzupełnij klasę `Wymierna` o definicję konstruktora i napisz program sprawdzający jego działanie.
- b. Zmodyfikuj metodę `Czytaj` tak, aby skracala ułamek po wczytaniu.

Przeciwieństwem konstruktora, a więc specjalną metodą wywoływaną automatycznie, gdy obiekt jest niszczone (np. gdy kończy się wykonanie funkcji, w której obiekt jest zadeklarowany), jest **destruktor**. Najczęściej definiuje się go w sytuacjach, kiedy programista przydziela samodzielnie dodatkową pamięć na pomocnicze struktury obiektu w konstruktorze. Wówczas należy ją zwolnić w destruktorze.

Destruktor

Warto wiedzieć

Jeśli programista nie zdefiniuje konstruktora lub destruktora, to kompilator utworzy ich domyślne wersje.

Przeciążanie metod

W obrębie klasy może być zdefiniowanych kilka konstruktorów. Muszą się one jednak różnić listą parametrów tak, aby kompilator mógł rozróżnić, która wersja konstruktora jest wywoływana. Zdefiniowanie w obrębie tej samej klasy wielu metod o tej samej nazwie, różniących się parametrami, nazywamy **przeciążaniem metod**.

Przeciążanie metod

Oto definicja konstruktora, który odczyta ułamek z pliku:

```
1. Wymierna::Wymierna(string nazwa)
2. {
3.     ifstream we(nazwa.c_str());
4.     if (we.is_open())
5.     {
6.         we >> licz >> mian;
7.         we.close();
8.         if (mian > 0) Skroc();
9.         else
10.        {
11.            licz=0; mian=1;
12.        }
13.    }
14.    else
15.    {
16.        licz=0; mian=1;
17.    }
18. }
```

Kod źródłowy konstruktora w klasie `Wymierna` odpowiadającego za wczytanie ułamka z pliku

Warto wiedzieć

Metody można zdefiniować bezpośrednio w klasie – nie trzeba wówczas używać operatora zasięgu (`::`). Jest to jednak rozwiązanie mało czytelne.

Dobra rada

Od standardu języka C++ w wersji 11 jako parametru obiektu typu `ifstream` możesz użyć napisu w stylu języka C++ (obektu typu `string`).

Metoda `c_str` dla klasy `string`

Parametrem konstruktora jest nazwa pliku. Wyjaśnienia wymaga otwarcie pliku w linii 3. Typ `ifstream` jest typem obiekowym (klasą), a zmienna `we` – obiektem. Podanie w nawiasach nazwy pliku, który chcemy otworzyć, oznacza wywołanie konstruktora, który to robi. Parametrem jest napis w stylu języka C, czyli ciąg znaków zakończony znakiem sterującym NUL. Konwersji z typu `string` na napis w stylu języka C dokonuje metoda `c_str`.

Jeśli plik został poprawnie otwarty (linia 4), to konstruktor odczytuje dane z pliku (linia 6) i skraca ułamek (linia 8). Jeśli dane w pliku są niepoprawne lub plik nie został otwarty, to atrybuty otrzymają wartości domyślne (linie 11 i 16). Należy też pamiętać, aby zamknąć plik, z którego odczytujemy dane (linia 7).

Ćwiczenie 3

Uzupełnij klasę `Wymierna` o definicję konstruktora odczytującego ułamek z pliku tekstowego. Napisz program testujący działanie tak zdefiniowanej klasy `Wymierna`. Uruchom program dla pliku, który otrzymasz od nauczyciela (np. `ulamek.txt`).

Zapamiętaj

W programowaniu obiekowym wyróżnia się dwie specjalne metody: konstruktor (uruchamiany automatycznie w momencie tworzenia obiektu) i destruktor (uruchamiany automatycznie w momencie usuwania obiektu). Zdefiniowanie wielu metod (w tym konstruktorów) o tej samej nazwie, ale różniących się listą parametrów formalnych, nazywamy przeciążaniem metod.

13.3. Hierarchia klas

Klasy zwykle są ze sobą powiązane – mogą mieć część atrybutów i metod wspólnych. Nie ma wtedy sensu definiować każdej klasy jako niezależnej i powielać części wspólnych. Na przykład podczas tworzenia aplikacji okienkowej możemy mieć klasę opisującą ogólnie okno i zbudować na tej podstawie klasę bardziej szczegółową, np. opisującą okno do odczytu lub zapisu plików. Definiowane klasy tworzą w ten

Hierarchia klas • sposób **hierarchię klas**.

Zbudujemy teraz klasę `Figura`, która będzie opisywać położenie figury na płaszczyźnie oraz sposób zmiany tego położenia. Położenie określimy jako współrzędne punktu charakterystycznego dla danej figury. Klasa będzie zawierała również trzy metody: konstruktor, służący do określenia położenia figury, metodę wypisującą to położenie oraz metodę do zmiany położenia (przesunięcia) figury.

Zapiszmy definicję klasy `Figura`.

```

1. class Figura
2. {
3.     public:
4.         Figura(float ax=0, float ay=0);
5.         void Pisz();
6.         void Przesun(float dx, float dy);
7.     private:
8.         float x, y;
9. };

```

Definicja klasy Figura

Zdefiniujemy teraz klasę opisującą okrąg. Położenie okręgu na płaszczyźnie możemy jednoznacznie określić przez podanie współrzędnych jego środka oraz długości promienia. Ponieważ klasa *Figura* określa już położenie charakterystycznego punktu (czyli środka okręgu), wystarczy dodać do nowo tworzonej klasy promień i dodatkowe funkcjonalności. Pozostałe elementy będą pochodziły z klasy *Figura*.

Dobra rada

Pamiętaj, żeby definicję struktury i klasy kończyć średnikiem po zamykającym nawiasie klamrowym.

Klasę, na podstawie której tworzymy nową klasę, nazywamy **klasą bazową**. Natomiast klasa definiowana na podstawie klasy bazowej to **klasa pochodna**.

Klasa bazowa

Klasa pochodna

Oto definicja klasy *Okrag*, utworzonej na podstawie klasy *Figura*:

```

1. class Okrag:public Figura
2. {
3.     public:
4.         Okrag(float ar=1, float ax=0, float ay=0);
5.         void Pisz();
6.         float Obwod();
7.     private:
8.         float r;
9. };

```

Definicja klasy Okrag – klasy pochodnej od klasy Figura

Warto wiedzieć

Oprócz specyfikatorów dostępu **public** i **private** istnieje jeszcze trzeci – **protected**, który udostępnia atrybuty i metody klasom pochodnym.

W nagłówku definicji klasy pochodnej (linia 1) po pojedynczym dwukropku podajemy nazwę klasy bazowej poprzedzoną specyfikatorem dostępu. Określa on, w jaki sposób będą traktowane **dziedziczone atrybuty i metody**, czyli te pochodzące z klasy bazowej. Specyfikator **public** (najczęściej stosowany) oznacza, że dziedziczone składowe mają taki sam specyfikator dostępu jak w klasie bazowej.

Dziedziczenie atrybutów i metod

W klasie *Okrag* zadeklarowany jest atrybut *r*, natomiast atrybuty *x* i *y* są dziedziczone z klasy *Figura*. Konstruktor nie jest dziedziczony w klasie pochodnej, dlatego w definicji klasy znajduje się również nagłówek konstruktora. Oprócz niego mamy nagłówki metod *Pisz* i *Obwod*. Metoda *Pisz* będzie przedefiniowana – mówimy wówczas, że **przesłania** ona metodę *Pisz* z klasy bazowej. Metoda *Obwod* jest nową metodą w klasie *Okrag*, natomiast metoda *Przesun* jest dziedziczona.

Przesłanianie metod

Definicje konstruktora oraz metody Pisz dla klasy Okrag można zapisać następująco:

Kod źródłowy definicji konstruktora oraz metody Pisz w klasie Okrag

```
1. Okrag::Okrag(float ar, float ax, float ay):Figura(ax,ay)
2. {
3.     if (ar>0) r=ar;
4.     else r=1;
5. }
6.
7. void Okrag::Pisz()
8. {
9.     Figura::Pisz();
10.    cout<<"r = "<<r<<endl;
11. }
```

Choć klasa Okrag nie dziedziczy konstruktora klasy bazowej, powinien on być wykonany, aby zainicjować wszystkie niezbędne elementy klasy bazowej. Często nawet nie wiemy, jakie to są elementy, np. gdy klasa bazowa jest zdefiniowana w dodatkowej bibliotece i nie znamy jej kodu źródłowego.

W nagłówku definicji konstruktora klasy pochodnej (linia 1) po dwukropku wywołujemy konstruktor klasy bazowej (Figura) z listą parametrów aktualnych (ax oraz ay). Zadaniem konstruktora w klasie Okrag jest określenie dodatniej wartości atrybutu r (linie 3–4).

W metodzie Pisz korzystamy z przesłoniętej metody Pisz z klasy bazowej. Metoda Pisz w klasie Okrag wywoła metodę Pisz z klasy Figura (która wypisuje współrzędne punktu) i dodatkowo wypisze wartość atrybutu r.

Oto kod źródłowy przykładowej funkcji main:

Dobra rada

Żeby wywołać w metodzie klasy pochodnej przesłoniętą metodę klasy bazowej, poprzedź jej nazwę nazwą klasy bazowej i operatorem dostępu.

Kod źródłowy funkcji main wykorzystującej obiekt i metody klasy Okrag

```
1. int main ()
2. {
3.     Okrag o(10);
4.     cout<<"Polozenie i promien okregu:"<<endl;
5.     o.Pisz();
6.     cout<<"Obwod: "<<o.Obwod()<<endl;
7.     o.Przesun(2,3);
8.     cout<<"Dane okregu po przesunieciu:"<<endl;
9.     o.Pisz();
10.    return 0;
11. }
```

W linii 3 został zadeklarowany obiekt o klasy Okrag o promieniu równym 10. Położenie okręgu będzie domyślne (środek w punkcie o współrzędnych (0, 0)). W linii 7 wywoływana jest dziedziczona metoda Przesun, przesuwająca okrąg o wektor o współrzędnych [2, 3].

Rysunek 13.3 przedstawia przykład wywołania programu korzystającego z obiektu klasy Okrag.

```
Polozenie i promien okregu:
x = 0
y = 0
r = 10
Obwod: 62.8319
Dane okregu po przesunieciu:
x = 2
y = 3
r = 10
```

Rys. 13.3. Przykładowe wywołanie programu korzystającego z obiektu klasy Okrag

Warto wiedzieć

W języku C++ jest możliwe dziedziczenie wielokrotne, tzn. można wskazać więcej niż jedną klasę bazową. W tym podręczniku nie będziemy wykorzystywać tego typu dziedziczenia.

Ćwiczenie 4

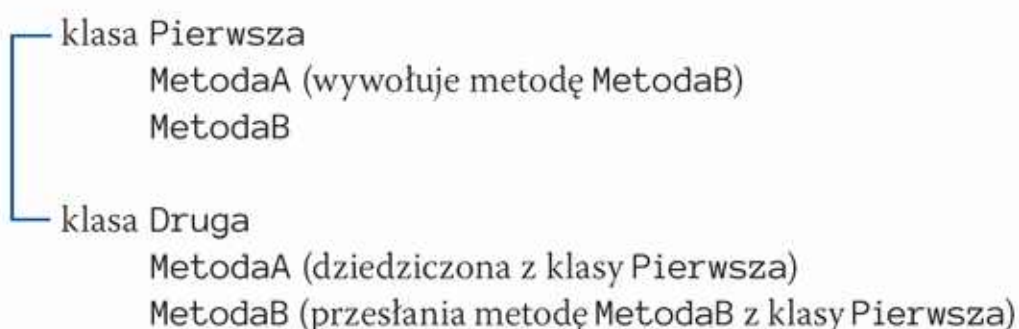
- Zdefiniuj klasy Figura i Okrag oraz ich metody. Napisz program wykorzystujący obiekt klasy Okrag.
- Uzupełnij hierarchię klas o definicję klasy Kwadrat, a program o wykorzystanie obiektu klasy Kwadrat.

Zapamiętaj

Klasy mogą tworzyć hierarchię. Klasę, na podstawie której tworzymy nową klasę, nazywamy klasą bazową. Klasa definiowana na podstawie klasy bazowej to klasa pochodna. Klasa pochodna dziedziczy z klasy bazowej atrybuty i metody. Metody klasy bazowej mogą być przededefiniowane (przesłonięte) w klasie pochodnej.

13.4. Polimorfizm i metody wirtualne

Założmy, że mamy zdefiniowaną hierarchię klas przedstawioną na rysunku 13.4:



Rys. 13.4. Przykładowa hierarchia klas

W klasie Pierwsza zdefiniowana jest metoda MetodaA, która wywołuje metodę MetodaB. Klasa ta jest **klasą bazową** dla klasy Druga.

Klasa bazowa,
s. 233 

Warto wiedzieć

Kompilator w momencie kompilowania kodu źródłowego klasy Pierwsza „widzi” tylko jej metody.

W klasie Druga metoda MetodaB przesłania metodę MetodaB z klasy Pierwsza oraz dziedziczy metodę MetodaA. Zauważ, że jeśli dla obiektu klasy Druga zostanie wywołana MetodaA, to zostanie w niej wywołana MetodaB z klasy Pierwsza.

Czy można w takim razie spowodować, żeby ustalenie wywoływanej wersji metody MetodaB zostało odłożone na czas wykonania programu (np. do czasu utworzenia obiektu), a nie było realizowane już podczas

Metoda wirtualna • kompilacji? Okazuje się, że można to zrobić z wykorzystaniem **metod wirtualnych** – ich wywołanie będzie zależało od typu obiektu. Zjawisko wyboru jednej z wielu postaci tej samej metody o tych samych

Polimorfizm • parametrach podczas działania programu nazywamy **polimorfizmem** (wielopostaciowość) (**wielopostaciowością**).

Polimorfizm jest realizowany za pośrednictwem metod wirtualnych – umożliwia zdefiniowanie abstrakcyjnych operacji z wykorzystaniem metod, które będą dopiero w przyszłości zdefiniowane w klasach pochodnych.

Definiowanie metod wirtualnych

Opracujmy hierarchię klas, która będzie opisywała bryły i metody obliczania ich objętości. Dla wielu brył można sformułować ogólnie sposób obliczenia objętości jako iloczyn pola podstawy i wysokości bryły, który jest mnożony jeszcze przez pewną stałą. Dla graniastopu lub walca stała ta jest równa 1, a dla ostrosłupa lub stożka ma ona wartość $\frac{1}{3}$. Sposób obliczenia pola podstawy będzie zależał od figury, która jest podstawą bryły.

Zdefiniujmy klasę Bryla, opisującą ogólną postać bryły, dla której możemy określić podstawę, wysokość oraz metodę obliczającą jej objętość.

Definicja klasy Bryla •

```
1. class Bryla
2. {
3.     public:
4.         Bryla(float aH=1, float ac_V=1);
5.         float Objetosc();
6.         virtual float PolePodstawy()=0;
7.     private:
8.         float c_V, H;
9. };
```

Klasa Bryla ma dwa atrybuty: c_V – do pamiętania stałej wykorzystywanej do obliczania objętości – oraz H – określający wysokość bryły.

Metoda PolePodstawy jest wirtualna – w deklaracji nagłówek takiej metody poprzedzamy słowem kluczowym **virtual**. Metoda ta w klasie Bryla jest pusta – będzie zdefiniowana w klasach pochodnych.

Język C++ wprowadza specjalną definicję pustej metody wirtualnej. Wystarczy dopisać w nagłówku, że jest ona równa 0. Taką metodę nazywamy **metodą czysto wirtualną**. Klasa, która zawiera metody czysto wirtualne, nazywana jest **klasą abstrakcyjną**. Służy ona tylko jako klasa bazowa dla klas pochodnych (nie można tworzyć obiektów takiej klasy). Zapiszmy definicje konstruktora oraz metody Objetosc:

```
1. Bryla::Bryla(float aH, float ac_V)
2. {
3.     if (aH>0) H=aH; else H=1;
4.     if (ac_V>0) c_V=ac_V; else c_V=1;
5. }
6.
7. float Bryla::Objetosc()
8. {
9.     return c_V*PolePodstawy()*H;
10. }
```

• Kod źródłowy konstruktora oraz metody Objetosc dla klasy Bryla

Zdefiniujmy teraz dwie klasy pochodne od klasy Bryla: Graniastoslup i Ostroslop, które opiszą graniastoslupy i ostrosłupy (rys. 13.5). Klasy te będą miały określony tylko konstruktor, który przypisze odpowiednią wartość stałej c_V.



Rys. 13.5. Hierarchia klas utworzona dla brył

Zapiszmy definicje klas Ostroslop i Graniastoslup:

```
1. class Ostroslop:public Bryla
2. {
3.     public:
4.         Ostroslop(float aH=1):Bryla(aH,1.0/3){}
5. };
6.
7. class Graniastoslup:public Bryla
8. {
9.     public:
10.        Graniastoslup(float aH=1):Bryla(aH){}
11.};
```

• Definicje klas Graniastoslup oraz Ostroslop

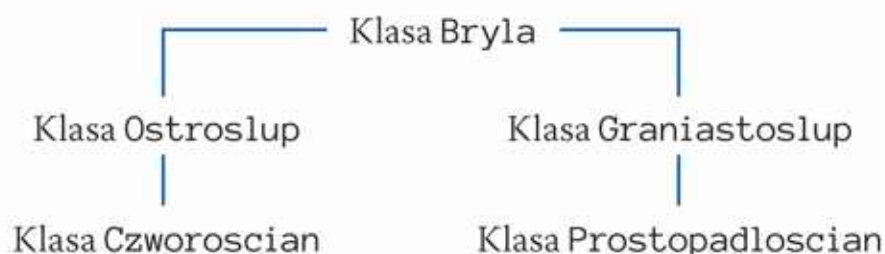
Kod konstruktorów jest pusty – wartość przypisywana stałej c_V została określona jako parametr aktualny wywołania konstruktora klasy bazowej.

Warto wiedzieć

Konstruktor nie może być metodą wirtualną, więc nie można użyć zapisu z zerem, żeby określić pusty zestaw instrukcji.

Dla ostrosłupa jest to wartość $\frac{1}{3}$, a dla graniastosłupa – wartość domyślna 1. Zwróć uwagę, że metodę można też zdefiniować bezpośrednio w klasie.

Zdefiniujemy teraz kolejne dwie klasy: Prostopadloscian (pochodną od klasy Graniastoslup) oraz Czworoscian (pochodną od klasy Ostroslup), których obiekty będą mogły reprezentować konkretne bryły. Rysunek 13.6 przedstawia rozbudowaną hierarchię klas.



Rys. 13.6. Rozbudowana hierarchia klas dla brył

Oto definicja klasy Prostopadloscian, która jest klasą pochodną od klasy Graniastoslup:

Definicja klasy

Prostopadloscian –
klasy pochodnej od
klasy Graniastoslup

```

1. class Prostopadloscian:public Graniastoslup
2. {
3.     public:
4.         Prostopadloscian(float aa=1, float ab=1, float ah=1);
5.         virtual float PolePodstawy();
6.     private:
7.         float a, b;
8. };
  
```

Zapiszmy również definicję klasy Czworoscian (pochodnej od klasy Ostroslup):

Definicja klasy

Czworoscian –
klasy pochodnej od
klasy Ostroslup

```

1. class Czworoscian:public Ostroslup
2. {
3.     public:
4.         Czworoscian(float aa=1, float ab=1, float ac=1,
5.                     float ah=1);
6.         virtual float PolePodstawy();
7.     private:
8.         float a, b, c;
9. };
  
```

W obydwu klasach zadeklarowane są konstruktor oraz wirtualna metoda PolePodstawy.

Oto definicje metod klasy Czworoscian:

```

1. Czworoscian::Czworoscian(float aa, float ab, float ac,
2.                           float aH):Ostroslop(aH)
3. {
4.     if (aa>0) a=aa; else a=1;
5.     if (ab>0) b=ab; else b=1;
6.     if (ac>0) c=ac; else c=1;
7.     if (a>=b+c || b>=a+c || c>=a+b)
8.     {
9.         a=1; b=1; c=1;
10.    }
11. }
12.
13. float Czworoscian::PolePodstawy()
14. {
15.     float p=(a+b+c)/2;
16.     return sqrt(p*(p-a)*(p-b)*(p-c));
17. }
```

W konstruktorze, w liniach 4–6, ustalamy długości boków podstawy zgodnie z podanymi wartościami parametrów aa, ab i ac. Dodatkowo sprawdzamy, czy parametry określające długości boków spełniają warunek trójkąta (linie 7–10).

Podstawa czworoscianu jest trójkątem, dlatego do obliczenia jej pola wykorzystamy wzór Herona (linie 15–16).

Oto kod źródłowy funkcji main tworzącej przykładowe bryły i wyświetlającej ich objętości:

```

1. int main ()
2. {
3.     Prostopadloscian p(2,3,4);
4.     Czworoscian c(3,3,3,sqrt(6));
5.     cout<<"Obj. prostopadloscianu: "<<p.Objetosc();
6.     cout<<endl;
7.     cout<<"Obj. czworoscianu: "<<c.Objetosc();
8.     return 0;
9. }
```

Rysunek 13.7 przedstawia przykład działania powyższej funkcji main.

```

Obj. prostopadloscianu: 24
Obj. czworoscianu: 3.18198
```

Rys. 13.7. Przykład działania programu obliczającego objętość brył

Kod źródłowy konstruktora oraz metody wirtualnej PolePodstawy dla klasy Czworoscian

Warto wiedzieć

Czworościan foremny to czworościan zbudowany z czterech trójkątów równobocznych. Wysokość w czworościanie foremnym jest równa $\frac{a\sqrt{6}}{3}$, gdzie a jest długością krawędzi.

Dobra rada

Pamiętaj, że pole trójkąta, dla którego są podane długości wszystkich boków a, b, c , wygodnie obliczysz ze wzoru Herona:

$$P = \sqrt{p(p-a)(p-b)(p-c)},$$

gdzie p jest połową obwodu trójkąta:

$$p = \frac{1}{2}(a+b+c)$$

Kod źródłowy funkcji main tworzącej prostopadłościan o wymiarach 2, 3 i 4 i czworościan foremny o długości krawędzi 3 oraz wypisującej objętości brył

Podsumowanie

- Klasa to abstrakcyjny model pewnych danych i operacji wykonywanych na tych danych. Obiekt to przedstawiciel danej klasy. Dane są reprezentowane przez atrybuty (pola), a operacje na tych danych są wykonywane przez metody klasy.
- Hermetyzacja oznacza ukrywanie atrybutów i metod tak, że na zewnątrz klasy są dostępne tylko wybrane metody, a dostęp do atrybutów jest realizowany wtedy za pośrednictwem metod.
- Konstruktor to specjalna metoda, która jest wywoływana automatycznie podczas tworzenia obiektu danej klasy.
- Klasy mogą tworzyć hierarchię. Klasa definiowana na podstawie innej klasy to klasa pochodna. Klasa, na podstawie której jest definiowana nowa klasa, to klasa bazowa. W klasie pochodnej atrybuty i metody mogą być dziedziczone z klasy bazowej. Metody klasy bazowej mogą być przesłonięte (zdefiniowane) w klasie pochodnej.
- Polimorfizm (wielopostaciowość) to możliwość wyboru jednej z wielu postaci danej metody podczas wykonywania programu w zależności od kontekstu. Polimorfizm jest realizowany z wykorzystaniem metod wirtualnych, czyli takich, dla których odwołanie do metody jest ustalane podczas wykonywania programu, a nie podczas kompilacji.
- Metoda czysto wirtualna to pusta metoda wirtualna, która będzie przesłonięta w klasach pochodnych. Klasa abstrakcyjna to klasa zawierająca metody czysto wirtualne.

Zadania

- * **1** Zdefiniuj klasę `Wymierna` w taki sposób, aby zawierała metodę do zmiany wartości ułamka podczas działania programu oraz metodę do odczytu wartości ułamka. Napisz program testujący tak zdefiniowaną klasę.
- * **2** Zdefiniuj hierarchię klas przedstawioną na rysunku 13.6 na s. 238 oraz metody dla tych klas. Napisz program testujący obiekty klas `Prostopadloscian` i `Czworoscian`.
- ** **3** Zdefiniuj klasę `Trojkat`, reprezentującą trójkąt jako trzy liczby określające długości boków. Zadbaj, aby liczby zawsze spełniały warunek trójkąta. Uwzględnij zdefiniowanie konstruktora, metody wypisującej długości boków oraz metod obliczających obwód i pole trójkąta. Napisz program sprawdzający tak zdefiniowaną klasę.
- ** **4** Uzupełnij klasę `Wymierna` o metody do zmiany wartości ułamka oraz odczytu jego wartości. Zdefiniuj funkcje wykonujące dodawanie i mnożenie liczb wymiernych jako obiektów klasy `Wymierna`. Napisz program, który to realizuje.
- *** **5** Uzupełnij hierarchię klas opisujących bryły o klasę `CzworoscianForemny` (pochodną od klasy `Czworoscian`) oraz o metodę obliczającą pole powierzchni całkowitej bryły. Metodę tę zdefiniuj w klasie `Bryla` z wykorzystaniem metody czysto wirtualnej obliczającej pole powierzchni bocznej bryły. Napisz program, który to realizuje.

Sposób na zadania

Zadanie 1

Anagram to słowo powstałe z innego słowa po przestawieniu liter. Przez słowo rozumiemy w tym zadaniu dowolny ciąg małych liter alfabetu łacińskiego (niekoniecznie znaczący). Rozważ następujący algorytm sprawdzający, czy dwa słowa są anagramami.

Dla danego słowa konstruujemy funkcję, której wartością jest liczba z zakresu od 0 do 255. Aby otrzymać wartość funkcji, sumujemy kody ASCII znaków słowa w arytmetyce modulo 256 (wynik każdego działania arytmetycznego jest przekształcany na resztę z dzielenia przez 256). Kody ASCII małych liter alfabetu łacińskiego przedstawia poniższa tabela.

a	b	c	d	e	f	g	h	i	j	k	l	m
97	98	99	100	101	102	103	104	105	106	107	108	109
n	o	p	q	r	s	t	u	v	w	x	y	z
110	111	112	113	114	115	116	117	118	119	120	121	122

Na przykład dla słowa *kod* otrzymamy wartość funkcji $(107 + 111 + 100) \bmod 256 = 62$.

Funkcję przyporządkowującą napisowi wartość z określonego przedziału nazywamy funkcją haszującą, a jej wartość – haszem. Jeśli dwa słowa są anagramami, to otrzymamy takie same wartości haszy. Możemy jednak otrzymać takie same wartości haszy dla słów, które nie są anagramami. Taką sytuację nazywamy kolizją. Rozwiąż poniższe zadania. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.1 (0–1)

Uzupełnij poniższą tabelę. Obydwa słowa w wierszu tabeli są takiej samej długości. Policz wartości haszy dla danych słów oraz określ, czy są anagramami. W ostatniej kolumnie tabeli wpisz:

TAK – gdy wartości haszy są takie same i słowa są anagramami,

NIE – gdy wartości haszy są różne,

KOLIZJA – gdy wartości haszy są takie same, ale słowa nie są anagramami.

Słowo s	Słowo t	Hash(s)	Hash(t)	Anagramy (TAK, NIE, KOLIZJA)
epoka	nalep	16	16	KOLIZJA
tabor	torba			
abakan	abakus			
ekran	rodak			

Rozwiązanie

Tagi: anagramy, arytmetyka modulo, funkcja haszująca, kolizja

Krok 1

Obliczamy sumę kodów ASCII słów *tabor* i *torba* w arytmetyce modulo 256. Możemy wykorzystać zależność:

$$(a + b) \bmod n = (a \bmod n + b \bmod n) \bmod n$$

Oznacza to, że gdy suma cząstkowa jest większa lub równa 256, można od niej odjąć 256.

Obliczone wartości dla słów *tabor* i *torba* są równe 24 i słowa składają się z tych samych liter, więc są anagramami.

Krok 2

Obliczamy sumę kodów ASCII dla słów *abakan* i *abakus* w arytmetyce modulo 256. Otrzymane wartości to 94 i 119. Wartości są różne, więc słowa nie są anagramami.

Krok 3

Obliczamy sumę kodów ASCII słów *ekran* i *rodak* w arytmetyce modulo 256. Obydwie wartości są równe 17. Słowa składają się jednak z różnych liter, więc mamy do czynienia z kolizją.

Odpowiedź: Prawidłowo uzupełniona tabela:

Słowo s	Słowo t	Hash(s)	Hash(t)	Anagramy (TAK, NIE, KOLIZJA)
epoka	nalep	16	16	KOLIZJA
tabor	torba	24	24	TAK
abakan	abakus	94	119	NIE
ekran	rodak	17	17	KOLIZJA

Zadanie 1.2 (0–1)

Uzupełnij poniższą tabelę. Oblicz minimalną liczbę potrzebnych zmian, które spowodują, że dwa słowa będą anagramami. Słowa w wierszu mogą być różnej długości. Przez pojedynczą zmianę należy rozumieć wymianę litery w słowie, dodanie litery do słowa lub usunięcie litery ze słowa. Zwróć uwagę, że dla słów będących anagramami liczba zmian jest równa 0.

Słowo s	Słowo t	Liczba zmian
abak	abakus	2
tabor	torba	
epoka	nalepka	
komar	rodak	

Rozwiązanie

Tagi: anagramy, zliczanie

Krok 1

Słowa *tabor* i *torba* są anagramami, więc liczba zmian jest równa 0.

Krok 2

Zliczamy liczbę wspólnych liter w słowach *epoka* i *nalepka*. Są cztery takie litery (zaznaczyliśmy je na zielono):

epoka

nalepka

Od długości dłuższego słowa odejmujemy liczbę wspólnych liter. Minimalna liczba zmian to: $7 - 4 = 3$. Wystarczy jedną literę wymienić i dwie dodać lub usunąć.

Krok 3

Zliczamy liczbę wspólnych liter w słowach *komar* i *rodak*. Są cztery takie litery. Obydwa słowa mają taką samą długość (5), więc wystarczy zmienić jedną literę.

Odpowiedź: Prawidłowo uzupełniona tabela:

Słowo s	Słowo t	Liczba zmian
abak	abakus	2
tabor	torba	0
epoka	nalepka	3
komar	rodak	1

Informacja do zadań 1.3 i 1.4

W pliku z danymi, który otrzymasz od nauczyciela (np. *WUZ3_zad1_slowa.txt*), w kolejnych 100 wierszach są zapisane po dwa słowa o długości nie większej niż 20 liter każde. W pierwszych 50 wierszach obydwa słowa w danym wierszu mają taką samą długość, w następnych 50 mogą mieć różną długość.

Oto 5 pierwszych wierszy pliku:

```
kamienica naciekami
powymiatawszy termowizyjnym
starcza czasami
realistka salaterki
lingwistka kserografy
```

Wiersze od 51 do 55 są następujące:

```
ratyfikacyjna narracyjna
harmonizacja szympanse
legendarny nylon
potulny osobliwsze
obserwator egiptolog
```

Napisz program lub programy rozwiązujące poniższe zadania. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.3 (0–4)

Dla pierwszych 50 wierszy z pliku policz liczbę wierszy, w których:

- ▶ słowa są anagramami,
- ▶ występuje kolizja,
- ▶ wartości haszy dla słów są różne.

Na przykład dla pierwszych 5 wierszy pliku z danymi wyniki są następujące: 2 wiersze z anagramami, 2 wiersze z kolizją, 1 wiersz z różnymi wartościami haszy.

Rozwiązanie

Tagi: anagramy, arytmetyka modulo, funkcja haszująca, kolizja

Krok 1

Konstruujemy funkcję `Hash`, obliczającą wartość haszu dla danego słowa zgodnie z treścią zadania. Wartość zmiennej `kod` ustalamy na 0. Dla każdej litery słowa dodajemy jej kod ASCII do zmiennej `kod` w arytmetyce modulo 256: `kod = (kod + kod_ascii_litery) mod 256`.

Krok 2

Jeśli dla dwóch słów otrzymamy taką samą wartość haszy, należy jeszcze sprawdzić, czy rzeczywiście słowa są anagramami. Konstruujemy więc funkcję `Anagramy`, która będzie to sprawdzać. Dwa algorytmy badające, czy słowa są anagramami, zostały omówione w podręczniku *Informatyka na czasie 2. Zakres rozszerzony* na s. 185–188. Zastosujemy algorytm, który zlicza i porównuje liczbę wystąpień liter w obu słowach z wykorzystaniem jednej tablicy. Dla pierwszego słowa będziemy w tablicy powiększać liczniki odpowiadające literom występującym w tym słowie. Liczniki odpowiadające literom drugiego słowa będziemy pomniejszać. Jeśli słowa są anagramami, to wszystkie liczniki liter muszą mieć po zliczeniu wartości równe zero. Wykorzystamy pomocniczą funkcję `Zlicz`, której trzecim parametrem będzie informacja, czy liczniki powiększamy czy pomniejszamy.

Krok 3

Zerujemy dwie zmienne `ile_a` i `ile_k`, zliczające wiersze z anagramami i wiersze z kolizjami. Przeglądamy kolejne wiersze pliku i odczytujemy dwa słowa z danego wiersza. Dla słów obliczamy wartości haszy. Jeśli są takie same, to sprawdzamy, czy słowa są anagramami. Jeśli są, to powiększamy wartość zmiennej `ile_a`, w przeciwnym przypadku powiększamy wartość zmiennej `ile_k`.

Krok 4

Wypisujemy wyniki: wartości zmiennych `ile_a` i `ile_k` oraz liczbę wierszy z różnymi wartościami haszy, którą obliczamy (różnica liczby wszystkich wierszy i sumy wartości zmiennych `ile_a` i `ile_k`).

Odpowiedź: Kod źródłowy programu rozwiązującego to zadanie może być następujący:

```

1. #include <iostream>
2. #include <fstream>
3. #include <string>
4.
5. using namespace std;
6.
7. int Hash(string s)
8. {
9.     int kod=0;
10.    for (int i=0;i<s.size();i++) kod=(kod+s[i])%256;
11.    return kod;
12. }
13.

```

```

14. void Zlicz(string s, int Ile[], int znak)
15. {
16.     for (int i=0;i<s.size();i++)
17.         if (znak==1) Ile[s[i]-'a']++;
18.         else Ile[s[i]-'a']--;
19. }
20.
21. bool Anagramy(string s1, string s2)
22. {
23.     int Ile[26];
24.     for (int i=0;i<26;i++) Ile[i]=0;
25.     Zlicz(s1,Ile,1); Zlicz(s2,Ile,-1);
26.     for (int i=0;i<26;i++)
27.         if (Ile[i]!=0) return false;
28.     return true;
29. }
30.
31. int main()
32. {
33.     ifstream we("WUZ3_zad1_slowa.txt");
34.     string s1,s2;
35.     int ile_a=0, ile_k=0;
36.     for (int i=0;i<50;i++)
37.     {
38.         we>>s1>>s2;
39.         if (Hash(s1)==Hash(s2))
40.             if (Anagramy(s1,s2)) ile_a++;
41.             else ile_k++;
42.     }
43.     we.close();
44.     cout<<"Anagramy: "<<ile_a<<endl;
45.     cout<<"Kolizje: "<<ile_k<<endl;
46.     cout<<"Rozne hasze: "<<50-(ile_a+ile_k);
47.     return 0;
48. }

```


Zadanie 1.4 (0–4)

Dla każdego wiersza pliku z danymi oblicz minimalną liczbę potrzebnych zmian, które spowodują, że dwa słowa będą anagramami. Przez pojedynczą zmianę należy rozumieć wymianę litery w słowie, dodanie litery do słowa lub usunięcie litery ze słowa. Zwróć uwagę, że dla słów będących anagramami liczba zmian jest równa 0.

Utwórz plik wynikowy (np. *WUZ3_zad1_4_slowa_wynik.txt*), w którym w każdym wierszu znajdzie się jedna liczba całkowita nieujemna, określająca minimalną liczbę zmian potrzebnych do uzyskania anagramów w wierszach pliku z danymi.

Na przykład dla pierwszych 5 wierszy pliku z danymi wyniki są następujące:

0
5
2
0
6

Dla wierszy od 51 do 55 wyniki są następujące:

5
7
6
8
6

Rozwiązanie

Tagi: anagramy, zliczanie, tablice

Krok 1

Napiszemy funkcję zliczającą liczbę wystąpień liter w danym słowie. Wyniki zapamiętamy w tablicy.

Krok 2

Dla danych dwóch słów policzymy liczbę wspólnych liter. W tym celu obliczymy liczbę wystąpień liter w każdym słowie. Na przykład jeśli w jednym słowie litera *a* występuje trzy razy, a w drugim słowie dwa razy, to słowa mają dwie wspólne litery *a*. Wystarczy więc zsumować wartości minimalne z wystąpień poszczególnych liter w słowach.

Krok 3

Jeśli słowa są tej samej długości, to minimalna liczba zmian jest równa liczbie różnych liter, czyli różnicy długości słowa i liczby wspólnych liter. Dla słów o różnej długości należy jeszcze dodać liczbę dodawanych lub usuwanych liter (obojętnie, czy skracamy dłuższe słowo czy wydłużamy krótsze). Minimalna liczba zmian jest więc równa różnicy długości dłuższego słowa i liczby wspólnych liter.

Krok 4

Przeglądamy kolejne wiersze pliku i odczytujemy dwa słowa z danego wiersza. Dla słów obliczamy minimalną liczbę zmian i zapisujemy ją do pliku wynikowego.

Odpowiedź: Kod źródłowy programu rozwiązującego to zadanie może być następujący:

```

1. #include <iostream>
2. #include <fstream>
3. #include <string>
4.
5. using namespace std;
6.
7. void Zlicz(string s, int Ile[])
8. {
9.     for (int i=0;i<26;i++) Ile[i]=0;
10.    for (int i=0;i<s.size();i++) Ile[s[i]-'a']++;
11. }
12.
13. int Wspolne(string s1, string s2)
14. {
15.     int Ile1[26], Ile2[26], suma=0;
16.     Zlicz(s1,Ile1); Zlicz(s2,Ile2);
17.     for (int i=0;i<26;i++) suma=suma+min(Ile1[i],Ile2[i]);
18.     return suma;
19. }
20.
21. int main()
22. {
23.     ifstream we("WUZ3_zad1_slowa.txt");
24.     ofstream wy("WUZ3_zad1_4_slowa_wynik.txt");
25.     string s1,s2;
26.     for (int i=0;i<100;i++)
27.     {
28.         we>>s1>>s2;
29.         wy<<max(s1.size(),s2.size())-Wspolne(s1,s2)<<endl;
30.     }
31.     we.close();
32.     wy.close();
33.     return 0;
34. }

```


Trening

Zadanie 1

Palindrom to słowo, które czytane od przodu i od tyłu jest takie samo. Prefiks to początkowy fragment słowa, natomiast sufix to jego końcowy fragment. Całe słowo jest zarówno swoim prefiksem, jak i sufixem. Przez słowo rozumiemy w tym zadaniu dowolny ciąg małych liter alfabetu łacińskiego (niekoniecznie znaczący). Rozwiąż poniższe zadania. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.1 (0–1) ROZWIĄŻ BEZ KOMPUTERA

Uzupełnij poniższą tabelę. Dla każdego słowa znajdź najdłuższy palindrom będący prefiksem lub sufixem słowa. Jeśli palindromy będące prefiksem i sufixem słowa są tej samej długości, to wpisz palindrom, który jest prefiksem.

Słowo	Najdłuższy prefiks lub sufix, który jest palindromem
papa	pap
kajak	
moment	
papardelle	

Zadanie 1.2 (0–1) ROZWIĄŻ BEZ KOMPUTERA

Uzupełnij poniższą tabelę. Dla podanej pary słów utwórz najkrótsze możliwe słowo zawierające obydwa słowa. Jeśli najkrótsze słowo można utworzyć na dwa sposoby, najpierw powinno w nim wystąpić pierwsze słowo z pary.

Słowo s	Słowo t	Najkrótsze słowo zawierające słowa s i t
abrakadabra	abraham	abrakadabraham
jakkolwiek	kajak	
prefiks	fiksacja	
palindrom	sufiks	

Informacja do zadań 1.3 i 1.4

W pliku z danymi, który otrzymasz od nauczyciela (np. *WUZ3_zad1_trening.txt*), w kolejnych 100 wierszach są zapisane po dwa słowa o długości nie większej niż 15 liter każde. Oto 5 pierwszych wierszy pliku:

```
owocowo kwaknij
szparag agrestowemu
jeszcze deklarujesz
mikroskop optymalizacja
tetryk myszkowianin
```

Napisz program lub programy rozwiązujące poniższe zadania. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.3 (0–4) **ROZWIĄŻ NA KOMPETERZE**

Utwórz plik wynikowy (np. *WUZ3_zad1_3_trening_wynik.txt*), w którym również znajdzie się 100 wierszy, a w każdym wierszu dwa słowa oddzielone spacją. Dla każdego słowa z pliku z danymi znajdź najdłuższy palindrom będący prefiksem lub sufiksem słowa. Jeśli palindromy będące prefiksem i sufiksem słowa są tej samej długości, to w pliku wynikowym umieść palindrom, który jest prefiksem. Kolejność słów w pliku wynikowym powinna być taka sama jak w pliku z danymi.

Na przykład dla pierwszych 5 wierszy pliku z danymi plik wynikowy będzie następujący:

```
owocowo k
s a
j d
m o
tet nin
```

Zadanie 1.4 (0–4) **ROZWIĄŻ NA KOMPETERZE**

Utwórz plik wynikowy (np. *WUZ3_zad1_4_trening_wynik.txt*), w którym znajdzie się 100 słów, każde w oddzielnym wierszu. Każde słowo powinno być najkrótszym możliwym słowem zawierającym obydwa słowa z odpowiadającego wiersza pliku z danymi. Jeśli najkrótsze słowo można utworzyć na dwa sposoby, najpierw powinno wystąpić pierwsze słowo z danego wiersza pliku z danymi. W pliku wynikowym kolejność słów powinna odpowiadać kolejności słów w pliku z danymi.

Na przykład dla pierwszych 5 wierszy pliku z danymi plik wynikowy będzie następujący:

```
owocowokwaknij
szparagrestowemu
deklarujeszczce
mikroskoptymalizacja
tetrykmyszkowianin
```


Algorytmika i programowanie

Wyobraź sobie, że...

...pracujesz na stanowisku analityka w przedsiębiorstwie, które wytwarza żywność ekologiczną. Celem firmy jest dostarczenie świeżych produktów w rozsądnej cenie. Jeden z głównych problemów to optymalizacja kosztu i czasu dostaw. Spotykasz się z przedstawicielami firm produkujących wyspecjalizowaną nawigację. Dzięki znajomości **dynamicznych struktur danych** i **algorytmów grafowych** rozumiesz, jak działa oprogramowanie prezentowane przez potencjalnych kontrahentów.



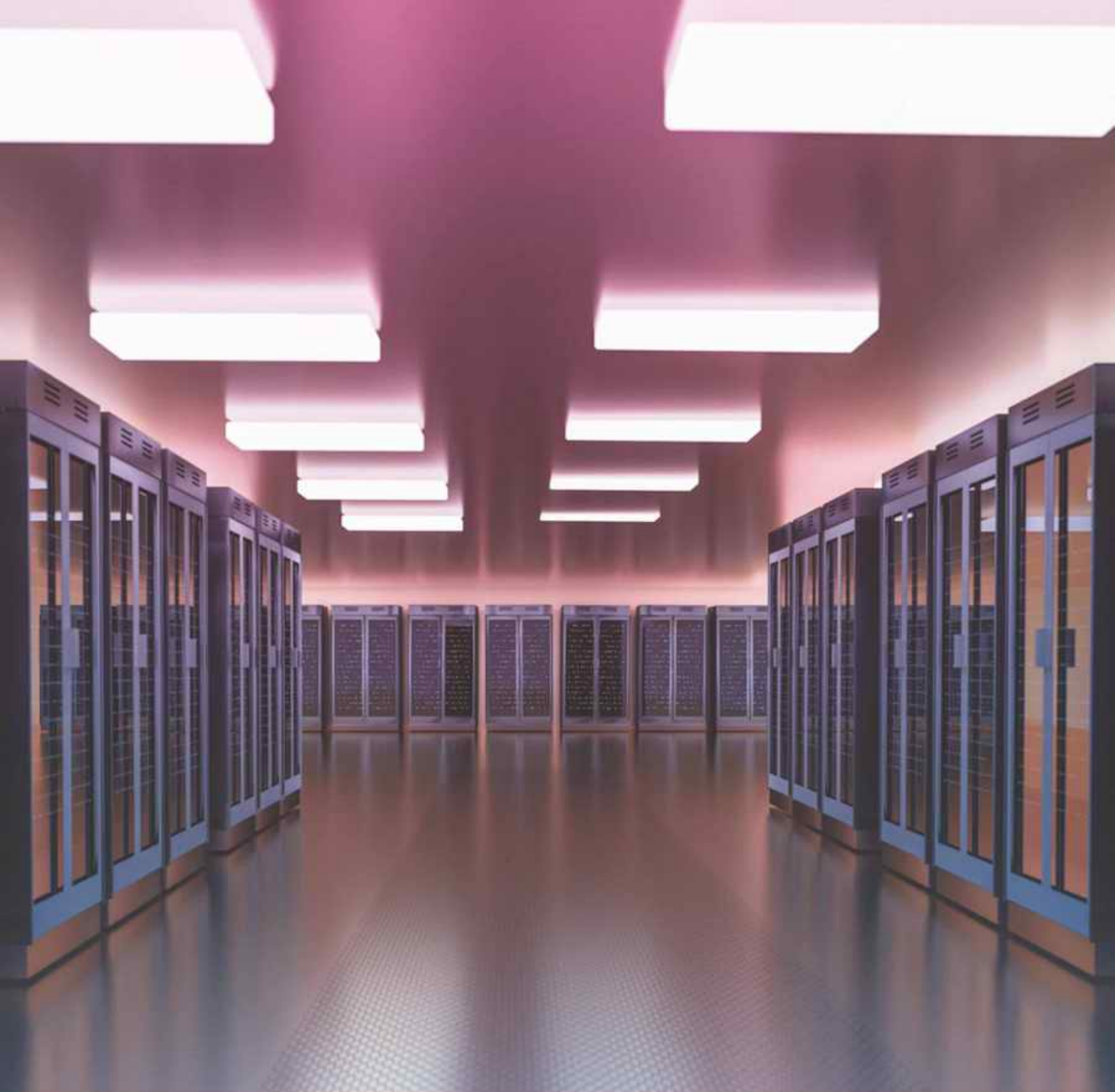
...w czasie studiów ekonomicznych odbywasz praktyki w centrali banku. Opiekun praktyk uwzględnił twoją wiedzę informatyczną wyniesioną ze szkoły i skierował cię do działu bezpieczeństwa. Pracujesz w zespole odpowiedzialnym za bezpieczeństwo transakcji w aplikacji internetowej banku. Radzisz sobie świetnie dzięki podstawowej znajomości **kryptografii klucza publicznego** oraz **umiejętnościom programistycznym**.

...po zdaniu matury kontynuujesz naukę na studiach informatycznych. Kończysz je z wyróżnieniem i rozpoczynasz pracę w znanej firmie software'owej, która przygotowuje biblioteki oprogramowania wykorzystywane w różnych językach programowania. W pracy na co dzień używasz **programowania obiektowego**, którego uczyłeś się w szkole średniej i na studiach.



...jesteś astronomem i na co dzień obserwujesz odległe obiekty we Wszechświecie. Używasz do tego niezwykle precyzyjnych teleskopów oraz komputerów o dużej mocy obliczeniowej. Bardzo przydaje ci się to, czego nauczyłeś się w szkole o **metodach numerycznych**, a także o **reprezentacji liczb rzeczywistych w komputerze** i **błędach przybliżeń**.





4 Relacyjne bazy danych

- 14. Wprowadzenie do relacyjnych baz danych
 - 15. Wykorzystanie danych pochodzących z kwerend
 - 16. Podstawy języka SQL
 - 17. Zapytania w języku SQL
 - 18. Tworzenie aplikacji korzystającej z sieciowej bazy danych
- Wiesz, umiesz, zdasz

Relacyjne bazy danych

Od czasów starożytnych ludzie gromadzą i przetwarzają różnego rodzaju informacje. Dziś pomagają w tym systemy zarządzania bazami danych. Dzięki szybkiemu rozwojowi internetu współczesne bazy danych często są częścią aplikacji webowych, czyli stron wzbogaconych o przydatne funkcjonalności.



W starożytnych bibliotekach gromadzono cenne pisma. Katalogi biblioteczne, z których korzystano, można uznać za pierwsze **bazy danych**. Za pierwszy spisany na papierze katalog uważa się spis zbiorów Biblioteki Aleksandryjskiej – w jej zbiorach znajdowało się ok. 500 tysięcy zwojów. Katalog ten, liczący 120 ksiąg, sporządził Kallimach z Cyreny.

III w. p.n.e.

Pierwszy **system zarządzania bazami danych** o nazwie IDS (ang. *Integrated Data Store*) został opracowany na komputer GE 235 przez Charlesa Bachmana w latach 60. XX w. System składał się z plików z danymi i narzędzi do ich obsługi, które wysyłały zapytania do IDS. Bachman zaproponował również graficzne przedstawienie struktur danych, zwane diagramami Bachmana.



lata 60. XX w.

1959 r.

W 1959 r. powstało konsorcjum **CODASYL**, składające się z przedstawicieli organizacji przemysłowych i rządowych. CODASYL zajmował się m.in. promowaniem efektywnej analizy danych i standaryzacją języków programowania, a jego głównym celem było stworzenie języka programowania, którego można używać na komputerach różnych producentów. Tak powstał język COBOL, wspierający sieciowy model baz danych.



1970 r.



W 1970 r. Edgar F. Codd przedstawił pracę *A Relational Model of Data for Large Shared Data Banks*, w której opisał **relacyjny model danych**. Jedną z udanych implementacji tego modelu na mikrokomputery był program dBase (na zdjęciu). Na podstawie prac Codd'a powstał język SQL, który do dziś jest standardem w komunikacji z serwerami relacyjnych baz danych.


```

44 GO
45 SELECT p.Name AS ProductName,
46 NonDiscountSales = (OrderQty * UnitPr
47 Discounts = ((OrderQty * UnitPr
48 FROM Production.Product AS p
49 INNER JOIN Sales.SalesOrderData
50 ON p.ProductID = sod.ProductID
51 ORDER BY ProductName DESC;
52 GO

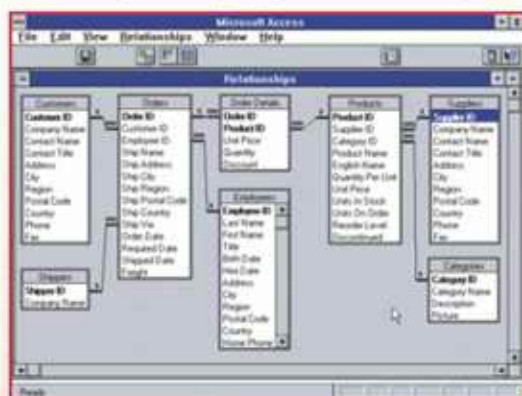
```

W 1974 r. Donald D. Chamberlin i Raymond F. Boyce z IBM Research Laboratory opracowali język **SEQUEL** (ang. *Structured English Query Language*). Miał on służyć użytkownikom baz danych, którzy nie byli profesjonalnymi programistami. SEQUEL dał początek strukturalnemu językowi zapytań SQL i jego różnym dialektom. Język SQL jest obecnie powszechnie używany do obsługi baz danych wykorzystywanych przez aplikacje internetowe.

1974 r.

1992 r.

W 1992 r. debiutowała pierwsza wersja programu **Microsoft Access**. Jest on docenianym przez wielu użytkowników wizualnym środowiskiem do tworzenia i edycji baz danych, budowania relacji między tabelami oraz wydobywania wiedzy z danych dzięki możliwości budowania kwerend. MS Access pozwala nie tylko tworzyć własne bazy danych, można go także wykorzystać do obsługi zewnętrznych źródeł danych.



1995 r.



W 1995 r. Rasmus Lerdorf udostępnił publicznie **PHP** – własny skryptowy język programowania. Był to zestaw skryptów służących do śledzenia odwiedzin jego witryny internetowej. W tym języku można tworzyć skrypty uruchamiane na stronach internetowych, a następnie wykonywane przez serwer. Ten mechanizm jest wykorzystywany przez aplikacje internetowe, czyli programy, które można obsługiwać za pośrednictwem przeglądarki z dowolnego urządzenia podłączonego do sieci.

2019 r.



W 2019 r. Ośrodek Przetwarzania Informacji jako jedna z pierwszych polskich instytucji publicznych wdrożył technologię **autonomicznej bazy danych**. Jest to baza danych w chmurze obliczeniowej, wykorzystująca sztuczną inteligencję do zautomatyzowania prac, które wykonywali dotąd administratorzy baz danych (np. tworzenie kopii zapasowych czy aktualizacji).

14. Wprowadzenie do relacyjnych baz danych

Serwisy internetowe wykorzystują olbrzymie bazy danych, aby przygotowywać informacje dopasowane do użytkownika. Jeśli stu twoich znajomych na portalu społecznościowym udostępni w tym samym czasie jakieś informacje, trzeba je gdzieś przechować i skatalogować, a także zdecydować, które z nich ci wyświetlić. Bez bazy danych i zaawansowanych algorytmów byłoby to niemożliwe. W profesjonalnych zastosowaniach baz danych dominuje model relacyjny, który poznasz w tym temacie.

Cele lekcji

- Zrozumiesz, czym jest baza danych, i poznasz związane z nią pojęcia.
- Poznasz praktyczne zastosowanie relacyjnych baz danych.
- Zaprojektujesz prostą bazę danych.
- Nauczysz się zarządzać bazą danych w programie Microsoft Access.
- Dowiesz się, jak przechowywać informacje w bazie danych oraz jak je modyfikować.
- Nauczysz się tworzyć tabele i relacje między nimi, kwerendy, formularze oraz raporty w programie Microsoft Access.

14.1. Projektowanie bazy danych

Baza danych • **Bazą danych** będziemy nazywać zbiór uporządkowanych i powiązanych ze sobą informacji dotyczących pewnej dziedziny życia, zorganizowanych w taki sposób, aby łatwo można było je przeszukiwać, wyciągać z nich wnioski i na tej podstawie podejmować decyzje.

Podczas projektowania bazy danych przede wszystkim należy się zastanowić, do czego będzie wykorzystana, czyli w jakim celu powstaje.

Założmy, że wraz z przyjaciółmi prowadzisz w formie bloga serwis internetowy o nowoczesnych technologiach. Aby utrzymać stałych czytelników oraz pozyskać nowych, powinniście regularnie publikować ciekawe i dobrze opracowane artykuły.

Utworzenie bazy danych w tym wypadku pozwoliłoby:

- ▶ gromadzić dane dotyczące opublikowanych i planowanych artykułów,
- ▶ kontrolować terminowe przygotowanie artykułów,
- ▶ przechowywać dane kontaktowe autorów,
- ▶ sprawdzać, czy długości artykułów zgadzają się z ustaleniami.

Aby to wszystko było możliwe, trzeba się zastanowić, które informacje należałoby zebrać w bazie danych. Mogą to być m.in.: tytuł, liczba stron i termin dostarczenia artykułu, nazwa działu, w którym ukaże się artykuł, imię i nazwisko autora oraz kontakt do niego, a także informacja, czy tekst został dostarczony.

Warto wiedzieć

Tematyka serwisu internetowego i związanej z nim bazy danych nie ma większego znaczenia. Projekt bazy po niewielkich modyfikacjach może również posłużyć do prowadzenia np. gazetki szkolnej, bloga modowego lub profilu szkoły w mediach społecznościowych.

Tabela, w której gromadzilibyśmy wymienione dane, mogłaby wyglądać tak jak na rysunku 14.1. Zostały w niej umieszczone informacje, które wydają się niezbędne, aby skutecznie zarządzać pracami nad tekstami na blog.

Lp.	Tytuł	Strony	Termin	Dział	Dostarczono	Imię	Nazwisko	Telefon	Email
1	Sztuczna inteligencja	6	11.10.2020	Nowe technologie	Tak	Jan	Kowalski	991123123	jkow@mail.com
2	Instagram jako narzędzie osobistej marki	4	13.10.2020	Samouczki	Nie	Eliza	Makulska	991231231	emaku@mail.com
3	Autonomiczne pojazdy – analiza SWOT	3	12.10.2020	Rynek	Nie	Jan	Kowalski	991123122	jkow@mail.com

Rys. 14.1. Tabela z danymi dotyczącymi bloga

Uzupełniając tak zaprojektowaną tabelę, możemy jednak zauważyć, że w rzeczywistości przechowujemy w niej informacje o dwóch niezależnych obiektach: artykułach oraz autorach. Łączenie informacji o nich w taki sposób, jak w tabeli na rysunku 14.1, stwarza kilka potencjalnych problemów.

Problem 1: Dodawanie kolejnych artykułów tej samej osoby wymaga wprowadzania za każdym razem tych samych informacji. Zajmuje to czas oraz miejsce, a poza tym łatwo popełnić błąd (np. w numerze telefonu Jana Kowalskiego, jak na rysunku 14.1).

Problem 2: Jeśli wprowadzimy błędne dane dotyczące tego samego autora, trudno potem rozstrzygnąć, które są właściwe.

Problem 3: Jeśli autor zmieni swoje dane (np. adres e-mail), trzeba będzie tę zmianę wprowadzić przy wszystkich jego artykułach.

Projektując bazę danych, powinniśmy więc unikać powielania informacji o tym samym obiekcie, czyli tzw. **redundancji**. Dlatego w naszej sytuacji należy oddzielić dane dotyczące autorów od tych dotyczących artykułów. Można w tym celu utworzyć dwie tabele, które są ze sobą powiązane, np. takie jak na rysunkach 14.2 i 14.3 (na s. 256).

Warto wiedzieć

Ten sam tekst napisany na komputerze może mieć różną liczbę stron w zależności od szerokości marginesów, wybranej czcionki, jej wielkości itp. Dlatego do rozliczeń wykorzystuje się tzw. znormalizowaną stronę maszynopisu – zwyczajowo zawiera ona 1800 znaków (łącznie ze spacjami i znakami interpunkcyjnymi).

Redundancja

ID	Tytuł	Strony	Termin	Dział	Dostarczono	Autor
1	Sztuczna inteligencja	6	11.10.2020	Nowe technologie	Tak	1
2	Instagram jako narzędzie osobistej marki	4	13.10.2020	Samouczki	Nie	2
3	Autonomiczne pojazdy – analiza SWOT	3	12.10.2020	Rynek	Nie	1

Rys. 14.2. Tabela Artykuły

ID	Imię	Nazwisko	Telefon	Email
1	Jan	Kowalski	991123123	jkow@mail.com
2	Eliza	Makulska	991231231	emaku@mail.com

Rys. 14.3. Tabela Autorzy

Dobra rada

W tabelach bazy danych zadbaj o to, aby każdy wiersz dotyczył tylko jednego obiektu oraz aby ten sam obiekt nie występował w dwóch różnych wierszach.

Tabela z rysunku 14.1 została więc podzielona na dwie tabele o nazwach **Artykuły** i **Autorzy**. Tabela **Artykuły** zawiera teraz dane ściśle powiązane z artykułami, w tym informację o autorze. Dane autorów zostały jednak w niej zapisane nie wprost, lecz za pomocą liczb, które są identyfikatorami w tabeli **Autorzy**. Dlatego też nazwa pierwszej kolumny tabeli **Autorzy** to **ID** (skrót słowa „identyfikator”).

W tabeli **Autorzy** znajdują się teraz wszystkie dane dotyczące poszczególnych autorów. Zauważ, że nie trzeba, a wręcz nie powinno się umieszczać w niej dwa razy jednej osoby. Zakładamy przy tym, że jeden artykuł ma jednego autora, a przynajmniej jednego autora głównego, który reprezentuje zespół autorów.

W opisany sposób zaprojektowaliśmy dwie tabele bazy danych.

Tabela bazy danych • W **tabeli bazy danych** (ang. *database table*) są gromadzone informacje na temat obiektów jednego, określonego typu. Obiekty te są opisywane za pomocą cech podanych jako wartości w kolumnach tabeli.

Atrybut • W bazach danych kolumny tabeli nazywa się **atrybutami** (ang. *attributes*). Nazwy tych atrybutów są nagłówkami kolumn – w naszym przykładzie to np. **Tytuł**, **Termin** czy **Dział**. Atrybuty zapisane

Schemat tabeli • w nawiasie i poprzedzone nazwą tabeli tworzą **schemat tabeli**. Dla tabeli **Artykuły** schemat będzie wyglądał następująco: **Artykuły (ID, Tytuł, Strony, Termin, Dział, Dostarczono, Autor)**.

Każdy wiersz tabeli zawiera informacje o jednym obiekcie i jest

Rekord • nazywany **rekordem** (ang. *data record*). Każdy rekord składa się

Pole rekordu • z wielu **pól rekordu** (ang. *values*), które znajdują się na przecięciu rekordu z wybranym atrybutem.

Rysunek 14.4 przedstawia te pojęcia na przykładzie tabeli **Artykuły**.

ID	Tytuł	Strony	Termin	Dział	Dostarczono	Autor
1	Sztuczna inteligencja	6	11.10.2020	Nowe technologie	Tak	1
2	Instagram jako narzędzie osobistej marki	4	13.10.2020	Samouczki	Nie	2
3	Autonomiczne pojazdy – analiza SWOI	3	12.10.2020	Rynek	Nie	1

Rys. 14.4. Elementy składowe tabeli bazy danych na przykładzie tabeli **Artykuły**

Atrybuty o nazwie ID, znajdujące się w obu tabelach, pełnią funkcję **klucza głównego** (ang. *primary key*), nazywanego również **kluczem podstawowym**, czyli niepowtarzalnego identyfikatora, który jednoznacznie wskazuje konkretny rekord danej tabeli. Klucz główny może się składać z jednej lub wielu kolumn i nie musi przechowywać wartości liczbowych.

Kolumna Autor tabeli Artykuły pełni funkcję tzw. **klucza obcego** (ang. *foreign key*), ponieważ wskazuje na klucz główny innej tabeli, w naszym przypadku kolumnę ID tabeli Autorzy (rys. 14.5).



Warto wiedzieć

Nazwy kolumn zawierających wartości klucza obcego często tworzy się z wykorzystaniem skrótu ID oraz nazwy tabeli, do której klucza głównego się odnoszą. Na przykład kolumna o nazwie ID_Autorzy zawiera klucze obce odnoszące się do rekordów z tabeli Autorzy.

Warto wiedzieć

Atrybut pełniący funkcję klucza głównego tabeli zwyczajowo nazywany jest ID. Jest to skrót słowa „identyfikator”.

Rys. 14.5. Klucze główne i klucz obcy tabel Artykuły i Autorzy

Takie powiązanie między rekordami w dwóch tabelach nazywamy relacją. **Relacja** (ang. *relationship*) to połączenie logiczne między dwiema tabelami, określające, które atrybuty są wspólne dla tych tabel.

W tym przypadku mamy do czynienia z **relacją typu jeden do wielu** (zapisywaną jako 1:∞), ponieważ:

- ▶ jeden autor może napisać wiele artykułów, tzn. rekord z tabeli Autorzy może być powiązany z wieloma rekordami w tabeli Artykuły,
- ▶ jeden artykuł ma tylko jednego autora, tzn. jednemu rekordowi z tabeli Artykuły odpowiada tylko jeden rekord w tabeli Autorzy.

Relacja jeden do wielu jest często występującym typem relacji. Może być zastosowana w bardzo wielu sytuacjach, w których pojedyncza informacja z jednej tabeli jest powiązana z wieloma informacjami z innej tabeli, np. jedna klasa jest powiązana z wieloma uczniami, którzy są do niej przypisani.

Relacja typu jeden do jednego

Oprócz relacji jeden do wielu istnieją także relacje: jeden do jednego i wiele do wielu. W **relacji typu jeden do jednego** (1:1) jeden rekord tabeli odpowiada dokładnie jednemu rekordowi innej tabeli. Relację tego typu wprowadza się czasem z powodu ograniczeń technicznych, np. kiedy program, w którym gromadzimy dane, nie pozwala umieścić wymaganej liczby atrybutów w jednej tabeli.

Relacja typu wiele do wielu

W **relacji typu wiele do wielu** ($\infty:\infty$) wiele rekordów jednej tabeli może odpowiadać wielu rekordom innej tabeli. Taka relacja sprawdziłaby się również w naszej bazie, jeśli przyjęlibyśmy założenie, że każdy autor może napisać wiele artykułów, a jeden artykuł może być współtworzony przez wielu autorów. Na potrzeby relacji $\infty:\infty$ tworzy się dodatkową tabelę, w której są przechowywane pary kluczy głównych tabel powiązanych ze sobą.

Schemat relacyjnej bazy danych

Zestaw wszystkich schematów tabel to **schemat relacyjnej bazy danych**. W naszym przykładzie ten schemat wygląda następująco: Artykuły (ID, Tytuł, Strony, Termin, Dział, Dostarczono, Autor) Autorzy (ID, Imię, Nazwisko, Telefon, Email)

Warto wiedzieć

W relacyjnych bazach danych często tabelę nazywa się relacją, ponieważ wiąże ona cechy opisujące określony obiekt.

Ćwiczenie 1

Zaprojektuj bazę danych dowolnego muzeum, zawierającą rejestr zbiorów. Baza powinna składać się z dwóch tabel: Ekspozycje i Sale. Ustal nazwy atrybutów w tabelach, klucze główne oraz relacje między tabelami. Projekt możesz przygotować na kartce lub w wybranym oprogramowaniu.

14.2. Od projektu do bazy danych

W praktyce do korzystania z relacyjnych baz danych używa się tzw. **systemu zarządzania bazą danych – SZBD** (ang. *Database Management System – DBMS*). Jest to program lub zestaw programów służących do tworzenia bazy danych i zarządzania nią. Jednym z takich programów jest Microsoft Access.

System zarządzania bazą danych (SZBD)**Warto wiedzieć**

Aby podkreślić relacyjny charakter systemu zarządzania bazą danych, stosuje się niekiedy skrót SZRBD – system zarządzania relacyjną bazą danych.

Dane w programie Microsoft Access są przechowywane w tabelach. Między tabelami można tworzyć relacje opisanych wcześniej typów. Za pomocą formularzy można wygodnie wprowadzać i przeglądać pojedyncze rekordy.

Znajdowanie odpowiednich informacji w bazie danych umożliwiają kwerendy, natomiast dzięki raportom można zbierać wygenerowane przez kwerendy dane i przygotowywać je do wydruku.

Główne okno programu (s. 260–261) zawiera w lewej części listę wszystkich utworzonych w bazie danych tabel, formularzy, kwerend oraz raportów. W obszarze roboczym znajdują się natomiast aktualnie otwarte obiekty, między którymi można się przełączać, klikając w zakładki widoczne w górnej części.

Tabele w programie Access

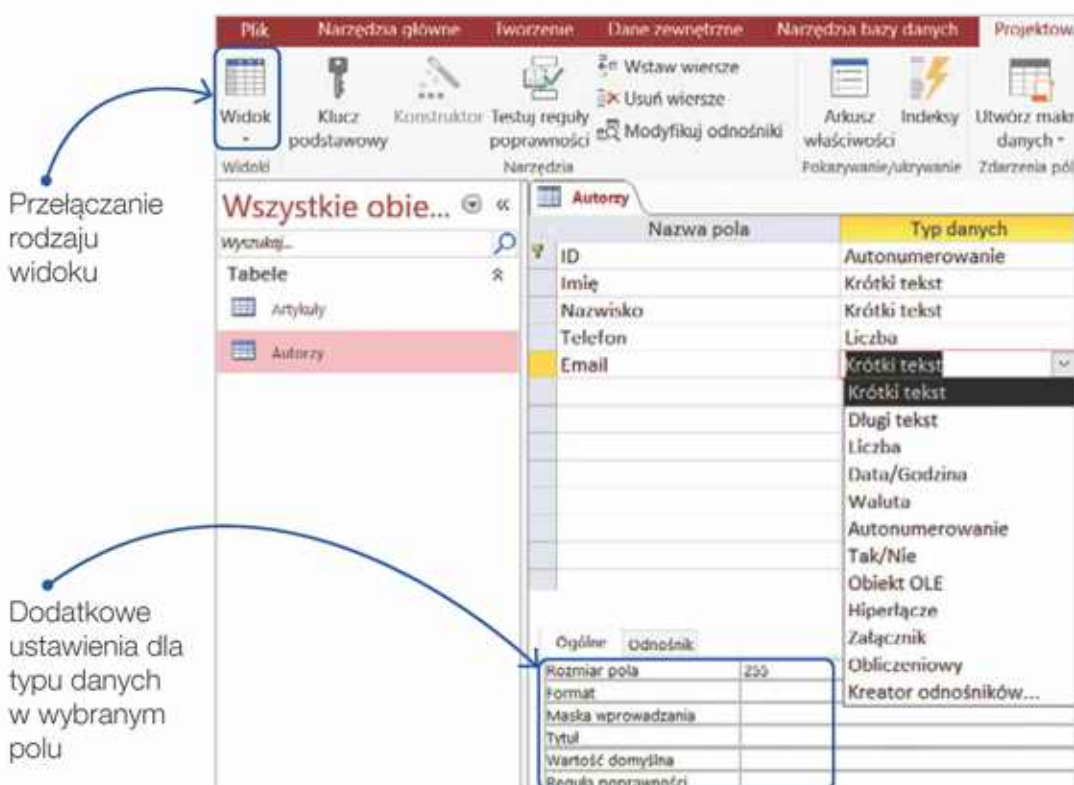
Podstawowym elementem bazy danych w programie Access jest **tabela** – w niej przechowywane są dane dotyczące opisywanych obiektów. Aby dodać do bazy nową tabelę, należy wybrać **Tworzenie** → **Tabele** → **Tabela**. Otworzy się nowa zakładka z tabelą zawierającą tylko jeden atrybut o domyślnej nazwie **Identyfikator**.

Założmy, że chcemy dodać tabelę opisującą autorów, taką jak zaprojektowana na rysunku 14.5 ze s. 257. W naszym projekcie nie ma jednak atrybutu **Identyfikator**, tylko **ID**. Aby zmienić nazwę atrybutu, należy ją kliknąć prawym przyciskiem myszy i z menu kontekstowego wybrać opcję **Zmień nazwę pola**.

Aby dodać kolejny atrybut, wybieramy opcję **Kliknij, aby dodać**, znajdującą się po prawej stronie ostatniego atrybutu tabeli (w naszym przypadku atrybutu **ID**). Następnie wybieramy typ danych, które będą w nim przechowywane, i wprowadzamy nazwę atrybutu, np. **Imię**.

Projektując tabelę bazy danych i jej atrybuty, należy określić **typ** poszczególnych danych. W przypadku imienia, nazwiska lub adresu e-mail odpowiednie będzie pole tekstowe (**Krótki tekst**) o długości do 255 znaków. W większości przypadków będzie to nawet więcej, niż potrzeba. Dla numeru telefonu można wybrać pole typu **Liczba**. W tabeli **Autorzy** wprowadzamy zatem następujące typy atrybutów: **Nazwisko** (**Krótki tekst**), **Telefon** (**Liczba**), **E-mail** (**Krótki tekst**).

Na rysunku 14.6 pokazano typy danych przypisane do pól tabeli **Autorzy**. Widać na nim również pełną listę typów danych programu Access.



Rys. 14.6. Zarządzanie tabelą w Widoku projektu

Warto wiedzieć

Atrybut opisuje pewną własność obiektu, np. cenę, kolor, kod pocztowy, nazwisko, markę. Dla danego rekordu wybrany atrybut nazywa się polem. W programie Microsoft Access używa się tego terminu również do określenia nazwy nagłówka tabeli.

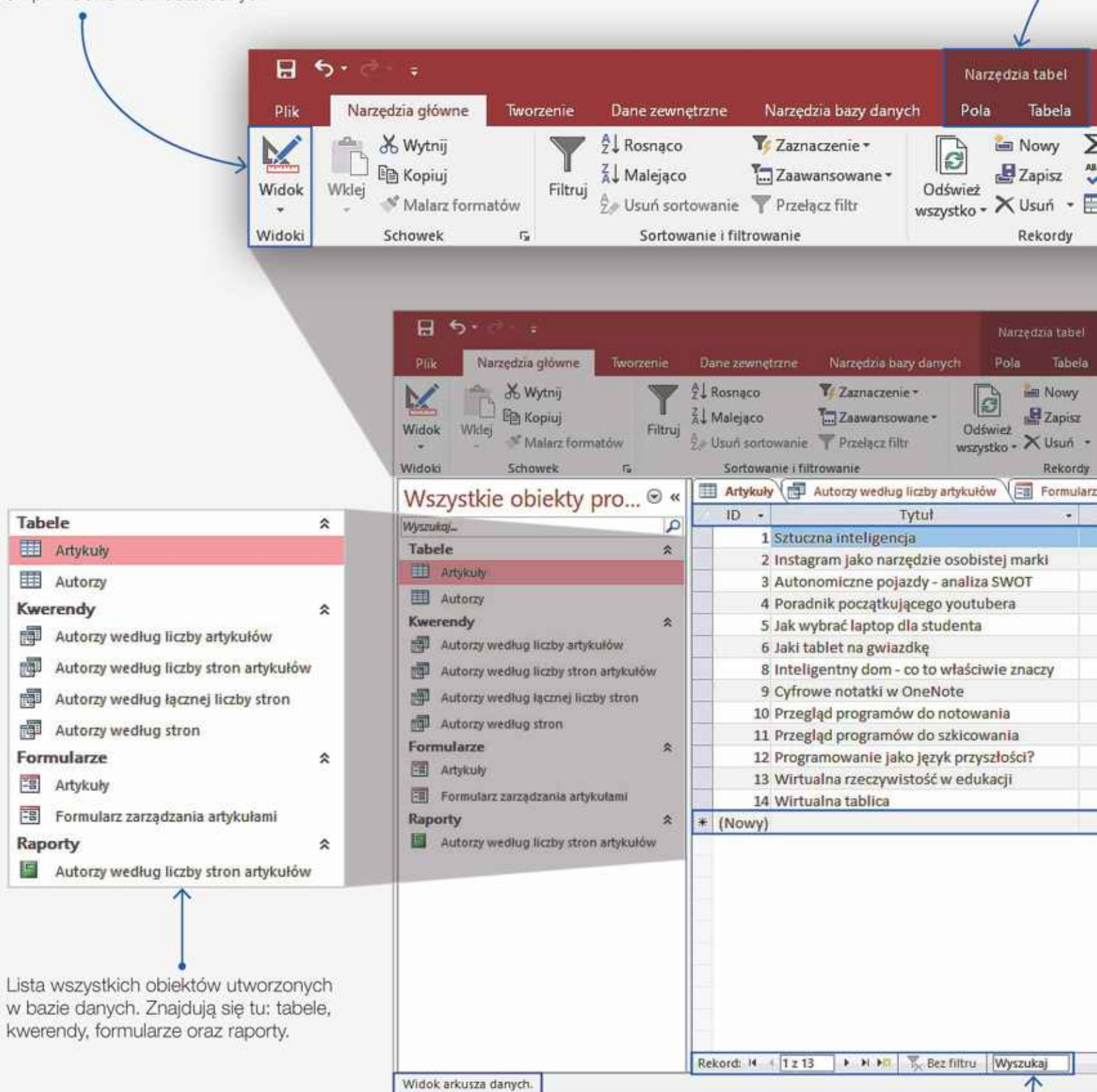
Typy danych i maski w programie Microsoft Access,
s. 432

Warto wiedzieć

W przypadku numeru telefonu ustalenie typu pola nie jest oczywiste. Zazwyczaj wystarczy podać tylko 9 cyfr, a w polu ustawić typ **Liczba**. Dla numerów z zagranicy to jednak za mało, gdyż trzeba uwzględnić choćby numer kierunkowy kraju (np. +33). Dla tak zapisanych danych można skorzystać z typu **Krótki tekst**.

Okno programu Microsoft Access 2016

W programie Access na kartach głównych są pogrupowane narzędzia o podobnym przeznaczeniu. Jednym z podstawowych narzędzi jest **Widok**, dzięki któremu można się przełączać między widokiem projektu obiektu bazy (np. tabeli lub kwerendy) a np. widokiem arkusza danych.



Lista wszystkich obiektów utworzonych w bazie danych. Znajdują się tu: tabele, kwerendy, formularze oraz raporty.

Informacja o aktualnym widoku aktywnego obiektu bazy danych

Pasek nawigacji po rekordach tabeli

Po uaktywnieniu wybranego obiektu (w tym wypadku tabeli **Artykuły**) pojawiają się dodatkowe karty, zawierające narzędzia dotyczące tylko tego obiektu (tu: **Narzędzia tabel**, a w nich karty **Pola** oraz **Tabela**).

W kolejnych zakładkach można przeglądać i modyfikować zawartość elementów bazy danych w różnych trybach widoku. Zakładkę danego obiektu możemy otworzyć po dwukrotnym kliknięciu jego nazwy na liście wszystkich obiektów.

blog_artykuły

Powiedz mi, co chcesz zrobić...

Sumy

Pisownia

Więcej

Znajdź

Zamień

Przejdź do

Zaznacz

Znajdowanie

Calibri (Szczegół)

11

Formatowanie tekstu

blog_artykuły

Powiedz mi, co chcesz zrobić...

Sumy

Pisownia

Więcej

Znajdź

Zamień

Przejdź do

Zaznacz

Znajdowanie

Formatowanie tekstu

zarządzania artykułami

Autorzy według liczby stron artykułów

Strony	Termin	Dział	Dostarczone	Autor	Kliknij, aby dodać
6	11.10.2020	Nowe technologie	☒	Kowalski	
4	13.10.2020	Samouczki	☒	Makulska	
3	12.10.2020	Rynek	☐	Kowalski	
4	20.10.2020	Samouczki	☐	Waleczny	
3	21.10.2020	Rynek	☐	Walczuk	
3	23.11.2020	Rynek	☒	Wysocka	
6	16.11.2020	Nowe technologie	☐	Bobrzak	
6	21.11.2020	Samouczki	☐	Moczulski	
4	23.11.2020	Aplikacje	☒	Babicz	
4	23.11.2020	Aplikacje	☐	Krynicky	
5	25.11.2020	Nowe technologie	☐	Poplawska	
4	26.11.2020	Nowe technologie	☒	Kowalski	
4	04.12.2020	Nowe technologie	☐	Moczulski	
0			☐		

Przycisk filtrowania i sortowania

Pole pozwalające dodać nowy atrybut w tabeli

Nazwy atrybutów tabeli

Wypełnienie ostatniego wiersza tabeli powoduje dodanie do niej nowego rekordu z danymi.

Po dodaniu wszystkich kolumn (atrybutów) należy pamiętać o zapisaniu tabeli. W tym celu możemy kliknąć prawym klawiszem jej zakładkę, a następnie z menu kontekstowego wybrać opcję **Zapisz**. Ponieważ zapisujemy tabelę po raz pierwszy, pojawi się okno **Zapisz jako**, w którym możemy wprowadzić nazwę tabeli – w naszym przypadku **Autorzy**.

Dodawanie nowych atrybutów oraz określenie typów danych możemy również w **Widoku projektu**. Aby przełączyć się na ten widok, należy zaznaczyć zakładkę danego obiektu (np. tabeli **Autorzy**), a następnie wybrać **Narzędzia główne** → **Widoki** → **Widok** → **Widok projektu** (rys. 14.6, s. 259).

Pliki bazy danych programu Access zapisywane są w pliku z domyślnym rozszerzeniem *accdb*.

Warto wiedzieć

Po zapisaniu tabeli jej nazwa pojawi się na liście wszystkich obiektów bazy.

Ćwiczenie 2

W bazie danych udostępnionej przez nauczyciela (np. w pliku *baza_blog_2.accdb*), korzystając z narzędzia **Widok projektu**, utwórz tabelę **Autorzy** z atrybutami odpowiednich typów. Plik zapisz pod nazwą *baza_blog_3.accdb*.

Zapamiętaj

Relacyjna baza danych to zorganizowany zbiór tabel połączonych relacjami. Każda tabela powinna opisywać jeden typ obiektu i składać się z kolumn zwanych atrybutami, które są zbiorem cech opisujących ten typ. Tabela może zawierać wiele rekordów, czyli pojedynczych wierszy opisujących cechy (atrybuty) różnych obiektów danego (pojedynczego) typu.

Importowanie danych do tabeli

Dane do tabeli w bazie danych można zaimportować z różnych źródeł, m.in. z arkusza kalkulacyjnego, z innej bazy danych lub z pliku tekstowego. Pokażemy, jak zaimportować dane z pliku tekstowego. Taki plik powinien być odpowiednio przygotowany. W jego pierwszym wierszu znajdują się nazwy **atrybutów**, oddzielone od siebie separatorami (w naszym przypadku tabulatorami, ale mogą to być też np. spacje lub przecinki). Kolejne wiersze to rekordy, których poszczególne **poła** są rozdzielone takimi samymi separatorami jak w pierwszym wierszu (rys. 14.7).

Atrybut,
s. 256 

Pole rekordu,
s. 256 

ID	Imię	Nazwisko	Email	Telefon
1	Jan	Kowalski	jkow@mail.com	991123123
2	Eliza	Makulska	emaku@mail.com	991231231
3	Piotr	Waleczny	pwal@mail.com	990212121
4	Paweł	Walczyk	pawal@mail.com	991023212
5	Gabriela	Wysocka	gwys@mail.com	929023212
6	Marta	Popławska	mpop@mail.com	992345432
7	Elżbieta	Bobrzak	ebob@mail.com	991238765
8	Kondrad	Moczulski	kmoc@mail.com	989218743
9	Eliza	Babicz	ebab@mail.com	978987654

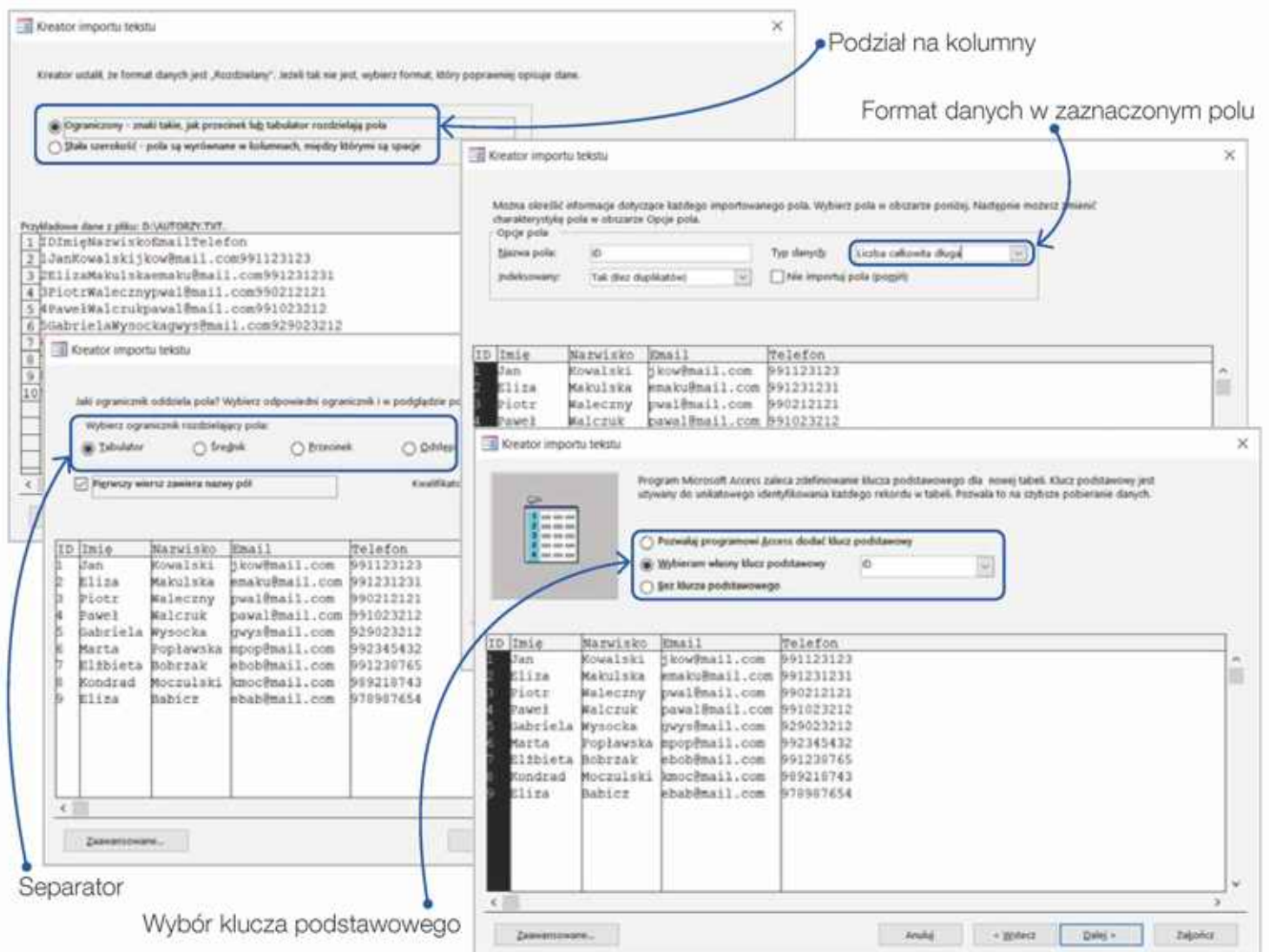
Rys. 14.7. Plik tekstowy zawierający dane importowane do tabeli

Aby zaimportować te dane, wybieramy **Dane zewnętrzne** → **Importowanie i łączenie** → **Plik tekstowy** i wskazujemy odpowiedni plik na dysku. W kolejnych krokach kreatora określamy następujące parametry:

- ▶ format danych **Ograniczony**, ponieważ dane w pliku są rozdzielone separatorami,
- ▶ rodzaj separatora – **Tabulator**, zaznaczamy również **Pierwszy wiersz zawiera nazwy pól**,
- ▶ typy danych w poszczególnych kolumnach (atrybutach), a jeśli w pliku jest już kolumna z **kluczem głównym**, ustawiamy ją jako **indeksowaną i bez duplikatów**,
- ▶ w ostatnim oknie wskazujemy, w której kolumnie znajduje się klucz główny lub pozwalamy programowi samodzielnie dodać taki klucz.

Klucz główny,
s. 257 ↗

Kreator importowania danych przedstawiono na rysunku 14.8.



Rys. 14.8. Importowanie danych do tabeli bazy danych

Przeglądanie i modyfikowanie danych w tabelach

Aby ułatwić sobie przeglądanie danych w pojedynczej tabeli, warto korzystać z filtrowania i sortowania. Działają one tak samo jak narzędzia dostępne w arkuszu kalkulacyjnym.

Jeśli chcemy poprawić dane, w otwartej zakładce wybranej tabeli wskazujemy odpowiednie pole w widoku arkusza danych i wprowadzamy poprawkę lub dodajemy informację, której brakowało.

Zapisywanie zmian w tabeli

Aby **zapisać zmiany**, klikamy zakładkę z nazwą tabeli prawym przyciskiem myszy i wybieramy opcję **Zapisz**.

Warto wiedzieć

Program Access pozwala stosować polskie znaki diakrytyczne, ale nie wszystkie SZBD oferują taką możliwość. Dlatego w bazach danych zazwyczaj unika się polskich znaków. Wielkość liter w nazwach tabel i nazwach atrybutów nie ma znaczenia.

Warto wiedzieć

O tym, że na dane w tabeli został założony filtr, świadczy ikona lejka wyświetlona przy nazwie danej kolumny (obok ikony narzędzia filtrowania).

Warto wiedzieć

Często zamiast usuwać rekordy z bazy, oznacza się je dodatkowym atrybutem jako nieaktywne lub nieaktualne. W niektórych zastosowaniach takie postępowanie wręcz jest konieczne, a usuwanie danych zabronione – np. w Biuletynach Informacji Publicznej (BIP).

Warto wiedzieć

Systemy zarządzania relacyjnymi bazami danych zazwyczaj dbają o zachowanie zasady niepowtarzalności klucza głównego i automatycznie uzupełniają pole identyfikatora stanowiącego ten klucz.

Ćwiczenie 3

Otwórz plik *baza_blog_3.accdb*, utworzony w ćwiczeniu 2, a następnie:

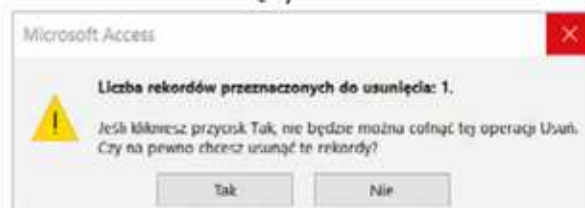
- posortuj rekordy tabeli *Artykuły* od najwcześniejszego terminu dostarczenia utworu,
- wyświetl tylko tytuły artykułów, których nie dostarczono,
- zmień tytuł artykułu *Sztuczna inteligencja* na *Systemy sztucznej inteligencji*.

Plik bazy zapisz pod nazwą *baza_blog_4.accdb*.

Usuwanie i dodawanie rekordu

Jeśli wiemy, że któryś z artykułów do naszego serwisu nie zostanie napisany, wówczas możemy usunąć z bazy danych cały rekord, który go dotyczy. W tym celu zaznaczamy odpowiadający mu wiersz, a następnie z menu kontekstowego wybieramy opcję **Usuń rekord**. Po potwierdzeniu operacji (rys. 14.9) wiersz zostanie usunięty z tabeli.

Aby dodać nowy rekord, wystarczy uzupełnić pola w pustym wierszu na końcu tabeli. Pole identyfikatora program uzupełni automatycznie.



Rys. 14.9. Potwierdzenie usunięcia rekordu

Ćwiczenie 4

W pliku *baza_blog_4.accdb* z ćwiczenia 3 wykonaj następujące czynności:

- usuń z tabeli *Artykuły* artykuł o ID 4 i tytule *Roboty wokół nas*,
- uzupełnij w tabeli *Autorzy* listę autorów zgodnie z projektem bazy danych z rysunku 14.5 ze s. 257,
- dodaj nowy, wymyślony przez siebie artykuł Elizy Makulskiej.

W tabeli nie mogą istnieć dwa rekordy z takim samym kluczem głównym. Jeśli dany rekord zostanie usunięty, przyporządkowany do niego klucz główny nie powinien zostać przydzielony innemu rekordowi (także nowemu).

Gdy w tabeli bazy danych programu Access usuniemy rekord o danym identyfikatorze, to żaden z nowych rekordów nie otrzyma już tego, teoretycznie zwolnionego, numeru. Jest to jedno z wielu zabezpieczeń oferowanych przez systemy zarządzania bazami danych.

Maski wprowadzania i format wyświetlania danych

W wielu sytuacjach określenie typu danych nie wystarczy. W przypadku numeru telefonu zapisanego jako liczba system nie powinien dopuszczać wartości składających się np. wyłącznie z jednej cyfry, ponieważ takie numery telefonów nie istnieją.

Aby zapewnić poprawność informacji wpisywanych do bazy, możemy użyć tzw. **maski wprowadzania**, która ma postać ciągu znaków określającego wzorzec dozwolonych znaków. W programie Access maski wprowadzania określa się w widoku projektu wybranej tabeli.

W przypadku numeru telefonu możemy założyć, że składa się on przynajmniej z 9 cyfr, ale nie więcej niż z 15 cyfr. W programie Access maska taka będzie mieć postać: 000000000999999, gdzie 0 oznacza cyfrę obowiązkową, a 9 – opcjonalną.

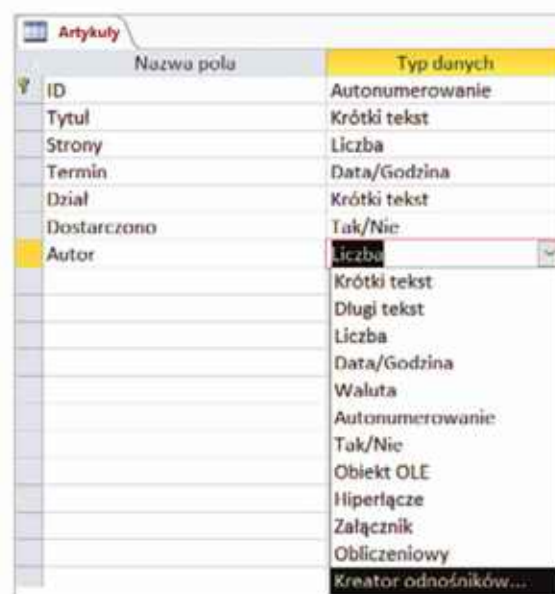
W widoku projektu możemy także zmienić format danych widocznych w polu – m.in. zdecydować, jak będzie wyświetlana data (np. „16.11.2020” czy „poniedziałek, 16 listopada 2020”). **Format** możemy wybrać z listy rozwijanej lub użyć do jego określenia odpowiednich symboli. Aby zobaczyć wprowadzoną zmianę formatu, należy zapisać zmiany w tabeli jeszcze przed przełączeniem się do **Widoku arkusza danych**.

Definiowanie relacji

W naszej bazie zaplanowaliśmy relację między tabelami **typu jeden do wielu**. Do tej pory jednak nie stworzyliśmy żadnych mechanizmów, które by z niej korzystały. Możliwość tworzenia relacji jest jedną z podstawowych zalet bazy danych. Dzięki temu będziemy mogli wyświetlać wybrane dane o autorach, mimo że formalnie znajdują się one w odrębnej tabeli.

Do stworzenia relacji w programie Microsoft Access wykorzystamy **Kreator odnośników**, który pozwoli nie tylko powiązać ze sobą pola tabel, lecz także wyświetlać w obu tabelach powiązane ze sobą informacje (rys. 14.14 i 14.15, s. 267–268).

Aby uruchomić kreator, przechodzimy do **Widoku projektu** dla tabeli **Artykuły**. Następnie dla pola **Autor** w kolumnie **Typ danych** rozwijamy menu kontekstowe i wybieramy narzędzie **Kreator odnośników** (rys. 14.10). Najpierw wskazujemy tabelę **Autorzy**, a następnie pola, których chcemy użyć w tabeli **Artykuły** (rys. 14.11, s. 266).



Rys. 14.10. Wybór Kreatora odnośników

Maska wprowadzania

Warto wiedzieć

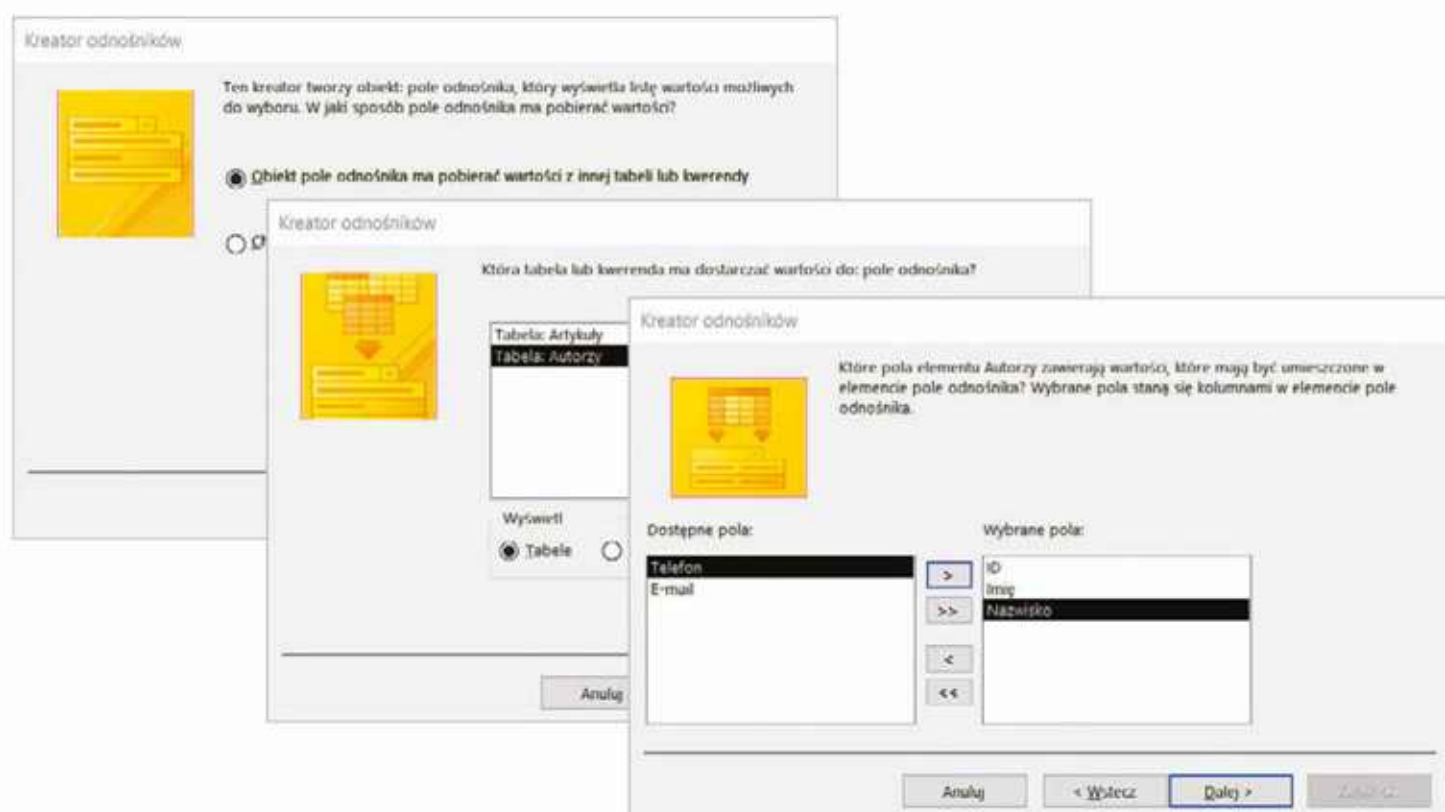
Międzynarodowe numery telefoniczne mogą mieć do 15 cyfr. Wynika to z zalecenia Międzynarodowego Związku Telekomunikacyjnego – najstarszej tego typu organizacji na świecie.

Relacja typu jeden do wielu, s. 257

Kreator odnośników

Dobra rada

Jeśli w twojej bazie danych znajdują się dwie osoby o takim samym imieniu i nazwisku, to podczas tworzenia odnośnika zawierającego te dane dodaj dodatkowy atrybut, który będzie je odróżniał, np. adres e-mail.

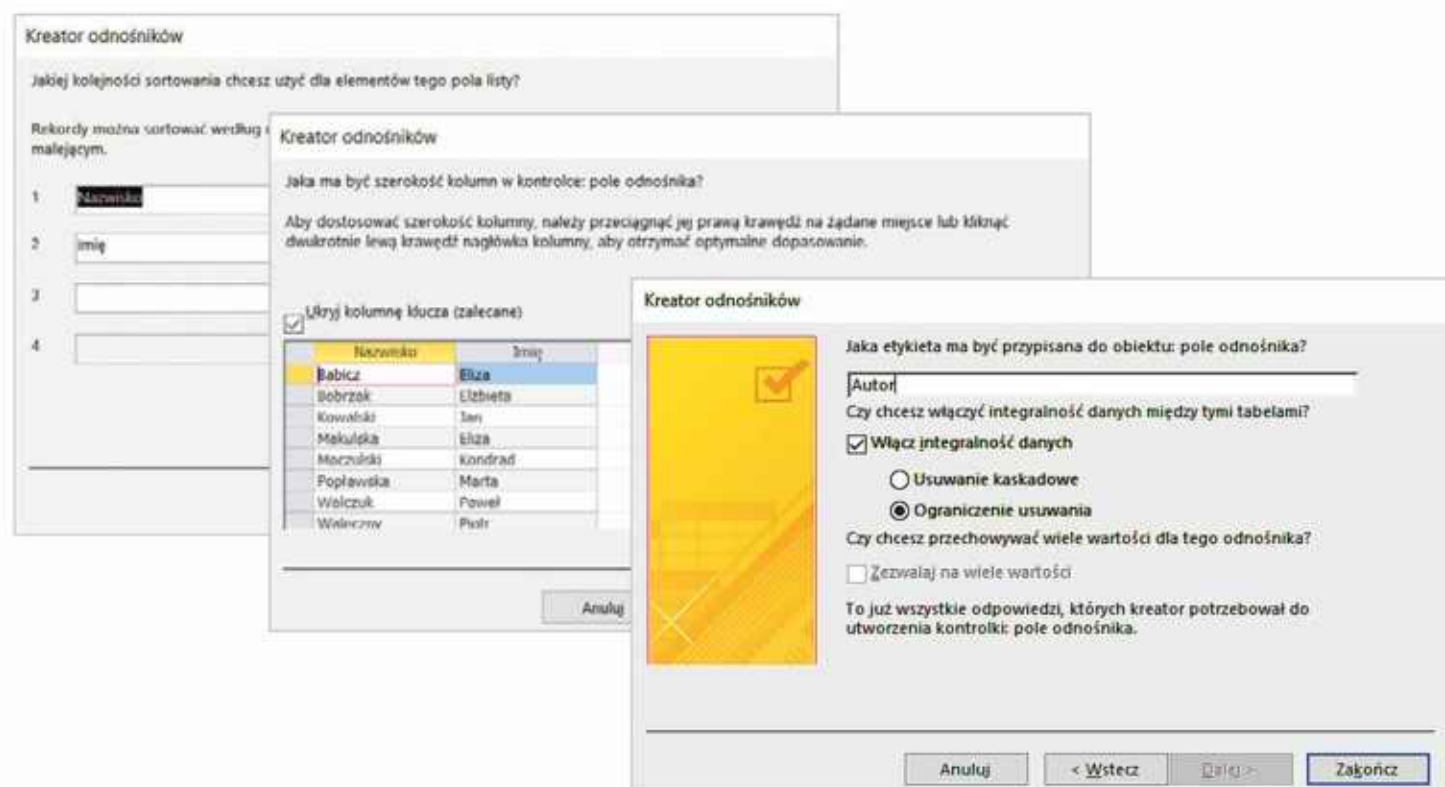


Rys. 14.11. Wybór pól tabeli w Kreatorze odnośników

Dobra rada

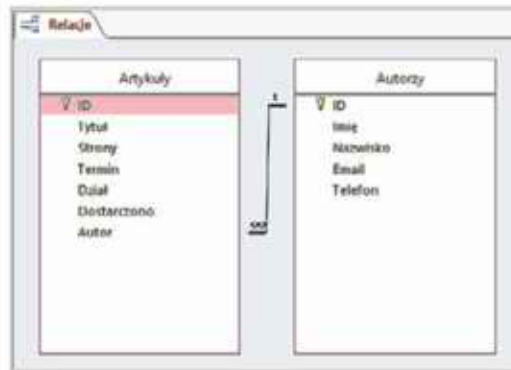
Aby zamienić miejscami wybrane kolumny, zaznacz jedną z nich, a następnie chwyć kursorem nazwę kolumny i przeciągnij ją na właściwe miejsce.

Po wybraniu pól i ich zatwierdzeniu możemy zdecydować, w jaki sposób zostanie posortowana lista z ich wartościami. Wybieramy atrybuty Nazwisko oraz Imię i w obu przypadkach zostawiamy opcję sortowania rosnącego, które dla tekstu oznacza kolejność alfabetyczną. Ustawiamy również szerokość kolumn i ich kolejność, a także nazwę etykiety. Włączamy również integralność danych (rys. 14.12). Domyślnie pole ID (które jest kluczem obcym) będzie ukryte.



Rys. 14.12. Ustawienia wyświetlania pól w Kreatorze odnośników

Po zapisaniu zmian w tabeli możemy sprawdzić utworzoną relację między tabelami Autorzy oraz Artykuły. W tym celu zaznaczamy jedną z tabel i wybieramy: **Narzędzia tabel** → **Tabela** → **Relacje** → **Relacje** (rys. 14.13). Powinna być widoczna linia łącząca atrybut Autor tabeli Artykuły z atrybutem ID tabeli Autorzy.



Rys. 14.13. Relacja między tabelami Artykuły i Autorzy

Dobra rada

Jeśli po utworzeniu odnośnika kolejność kolumn jest niewłaściwa, kliknij prawym przyciskiem myszy nazwę kolumny, wybierz z menu kontekstowego opcję **Modyfikuj odnośniki** i przejdź do kroku, w którym możesz zmienić kolejność kolumn.

Klucz obcy,
s. 257

Poza utworzeniem relacji między tabelami w tabeli zawierającej **klucz obcy** pojawią się dane z powiązanej z nią tabeli. Aby to zobaczyć, należy przełączyć się na **Widok arkusza danych**. W kolumnie Autor powinny być widoczne nazwiska autorów zamiast ich identyfikatorów. Po wybraniu pola z nazwiskiem autora powinna się pojawić lista wszystkich autorów. Pokazano to na rysunku 14.14.

ID	Tytuł	Strony	Termin	Dział	Dostarczone	Autor	Kliknij, aby dodać
1	Sztuczna inteligencja	6	11.10.2020	Nowe technologie	☒	Kowalski	
2	Instagram jako narzędzie osobistej marki	4	13.10.2020	Samouczki	☒	Babicz	Eliza
3	Autonomiczne pojazdy - analiza SWOT	3	12.10.2020	Rynek	☒	Bobrzak	Elżbieta
4	Poradnik początkującego youtubera	4	20.10.2020	Samouczki	☒	Kowalski	Ian
5	Jak wybrać laptop dla studenta	3	21.10.2020	Rynek	☒	Makulska	Eliza
6	Jaki tablet na gwiazdkę	3	23.11.2020	Rynek	☒	Moczulski	Konrad
8	Inteligentny dom - co to właściwie znaczy	6	16.11.2020	Nowe technologie	☒	Popławska	Marta
9	Cyfrowe notatki w OneNote	6	21.11.2020	Samouczki	☒	Walczuk	Paweł
10	Przegląd programów do notowania	4	23.11.2020	Aplikacje	☒	Walczyński	Piotr
11	Przegląd programów do szkicowania	4	23.11.2020	Aplikacje	☒	Wysocka	Gabriela
12	Programowanie jako język przyszłości?	5	25.11.2020	Nowe technologie	☒	Kowalski	
13	Wirtualna rzeczywistość w edukacji	4	26.11.2020	Nowe technologie	☒	Makulska	
14	Wirtualna tablica	4	04.12.2020	Nowe technologie	☒	Kowalski	

Rys. 14.14. Widok tabeli Artykuły po powiązaniu z tabelą Autorzy

W **Widoku arkusza danych** tabeli Autorzy przy każdym rekordzie pojawia się znak plusa. Jego wybranie spowoduje wyświetlenie wszystkich rekordów z tabeli Artykuły powiązanych z danym rekordem tabeli Autorzy (rys. 14.15, s. 268).

A to ciekawe

Bieżące raportowanie sprzedaży

Jedna z największych firm technologicznych wdrożyła we wszystkich swoich sklepach system, który w czasie rzeczywistym zbiera dane na temat sprzedanych produktów. Zaprojektowano w tym celu specjalną bazę danych. Dzięki temu rozwiązaniu można przewidywać zapotrzebowanie na produkty i wytwarzać je tak, żeby zaspokoić popyt bez ryzyka nadprodukcji.



Autorzy						
ID	Imię	Nazwisko	Email	Telefon	Kliknij, aby dodać	
1	Jan	Kowalski	jkow@nowaera.pl	991123123		
ID	Tytuł	Strony	Termin	Dział	Dostarczone	
1	Sztuczna inteligencja	6	11.10.2020	Nowe technologie	☒	
12	Programowanie jako język przyszłości?	5	25.11.2020	Nowe technologie	☐	
14	Wirtualna tablica	4	04.12.2020	Nowe technologie	☐	
(Nowy)		0			☐	
2	Eliza	Makulska	emak@nowaera.pl	991231231		
3	Piotr	Waleczny	piowa@nowaera.pl	990212121		
4	Paweł	Walczuk	pawa@nowaera.pl	991023212		
5	Gabriela	Wysocka	gawy@nowaera.pl	929023212		
6	Marta	Popławska	mapo@nowaera.pl	992345432		
7	Elżbieta	Bobrzak	elbo@nowaera.pl	991238765		
8	Konrad	Moczulski	komo@nowaera.pl	989218743		
9	Eliza	Babicz	elba@nowaera.pl	978987654		

Dane z tabeli Artykuły związane z wybranym autorem

Rys. 14.15. Widok tabeli Autorzy po utworzeniu relacji

Dobra rada

Jeśli dwie tabele twojej bazy danych są związane relacją, to w **Widoku arkusza danych** jednej z tych tabel możesz wprowadzać zmiany dotyczące rekordów drugiej tabeli. Możesz również dodawać nowe rekordy.

Ćwiczenie 5

W pliku bazy danych, który otrzymasz od nauczyciela (np. *baza_blog_5.accdb*), stwórz odnośnik powodujący, że w kolumnie Autor tabeli Artykuły wyświetli się nazwisko autora z odpowiadającego mu rekordu w tabeli Autorzy. Plik zapisz pod nazwą *baza_blog_6.accdb*.

Zapamiętaj

Za pomocą Kreatora odnośników w programie Microsoft Access można jednocześnie stworzyć relację między tabelami bazy danych oraz spowodować, że w polu klucza obcego będą wyświetlane odpowiadające mu dane z tabeli, na którą ten klucz wskazuje.

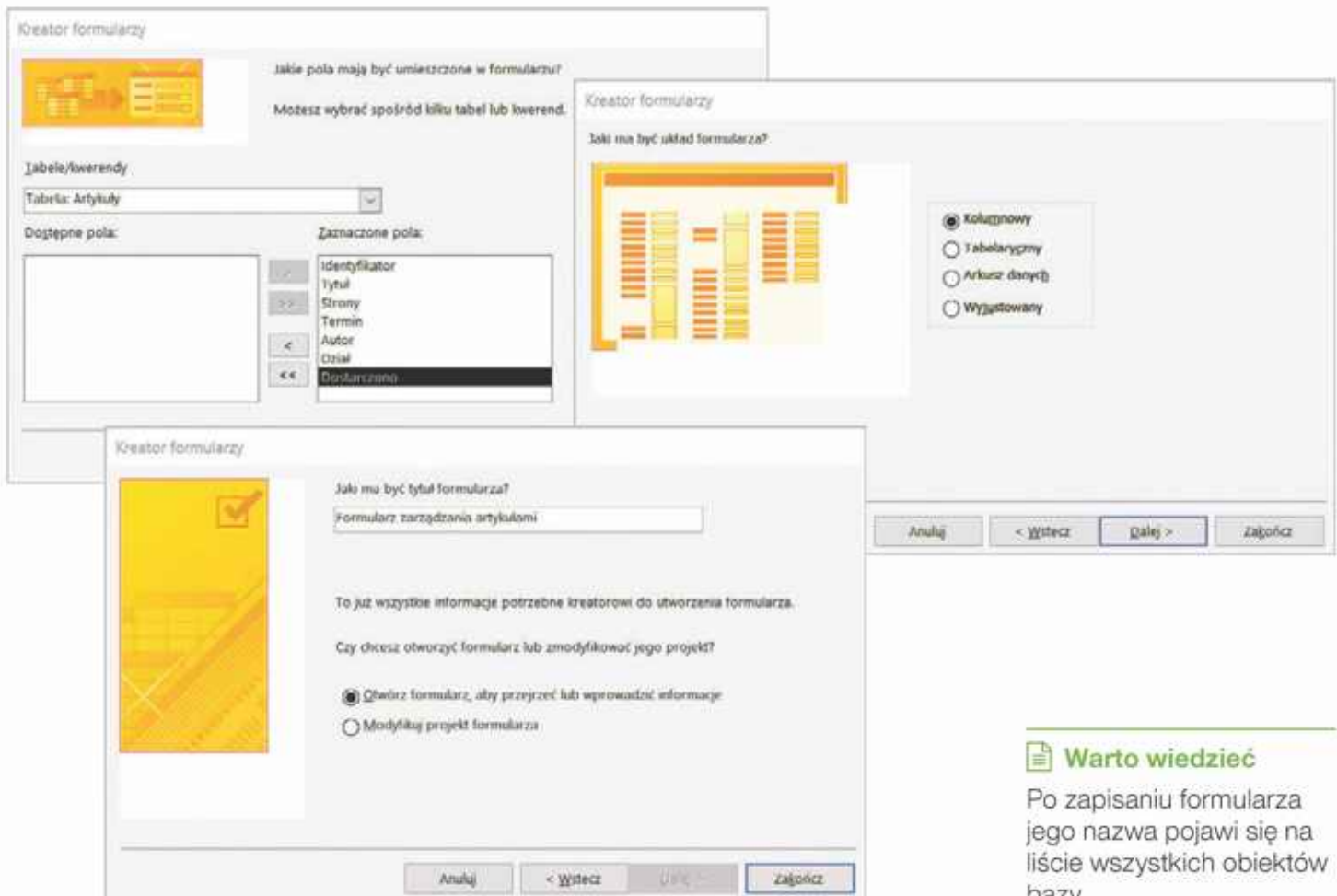
14.3. Formularze – zarządzanie danymi i ich przeglądanie

Formularze • **Formularze** programu Microsoft Access umożliwiają przeglądanie i modyfikowanie danych zgromadzonych w bazie oraz dodawanie nowych. Dzięki formularzom możemy skupić się na pojedynczym rekordzie zamiast na całej tabeli.

Tworzenie formularza

Aby utworzyć nowy formularz, wybieramy **Tworzenie** → **Formularze** → **Kreator formularzy**. Wskazujemy wszystkie pola tabeli Artykuły.

W naszym przykładzie możemy zostawić domyślnie wybrany układ kolumnowy i nadać formularzowi nazwę: Formularz zarządzania artykułami (rys. 14.16).



Rys. 14.16. Tworzenie formularza za pomocą Kreatora formularzy

W formularzu wyświetlone są wartości poszczególnych pól wybranego rekordu (rys. 14.17). Możemy je modyfikować, a zmiany automatycznie zostaną dodane do tabeli po przejściu do innego rekordu.

Rys. 14.17. Widok formularza utworzonego dla danych z tabeli Artykuły

Warto wiedzieć

Po zapisaniu formularza jego nazwa pojawi się na liście wszystkich obiektów bazy.

Dobra rada

Korzystając z przycisków umieszczonych na pasku pod formularzem, możesz przechodzić do pierwszego, poprzedniego, kolejnego, ostatniego lub nowego rekordu tabeli.

Rekord: 1 z 13

Ćwiczenie 6

W pliku *baza_blog_6.accdb* z ćwiczenia 5 utwórz formularz dla tabeli Artykuły o nazwie Formularz zarządzania artykułami. Plik zapisz pod nazwą *baza_blog_7.accdb*.

Modyfikowanie formularza

Formularz możemy dostosować poprzez zmianę wielkości pól, czcionki czy dopasowanie tła i kolorystyki poszczególnych elementów. Możemy także dodać przyciski, np. ułatwiające mniej zaawansowanym użytkownikom korzystanie z bazy.

Aby zmodyfikować formularz, wybieramy **Narzędzia główne** → **Widoki** → **Widok** → **Widok projektu**. Wygląd poszczególnych elementów (pól i etykiet) zmieniamy w podobny sposób jak w innych programach pakietu Microsoft Office – zaznaczamy wybrany element, a następnie korzystamy z odpowiednich narzędzi formatowania, które znajdziemy m.in. na kartach **Formatowanie**, **Rozmieszczanie** oraz **Projektowanie**.

Ćwiczenie 7

Otwórz plik *baza_blog_7.accdb*, utworzony w ćwiczeniu 6. Zmodyfikuj wygląd istniejącego w nim formularza, zmieniając wielkość i kolor czcionek etykiet. Dodaj miejsce pod polami formularza, przesuując jego stopkę nieco w dół. Zapisz plik pod nazwą *baza_blog_8.accdb*.

Dobra rada

W widoku projektu możesz powiększyć domyślny obszar formularza, chwytając i przeciągając jego brzegi kursorzem myszy. Możesz też przesunąć do dołu **Stopkę formularza**, by zyskać więcej miejsca np. na przyciski.

14.4. Kwerendy – wyszukiwanie i łączenie danych

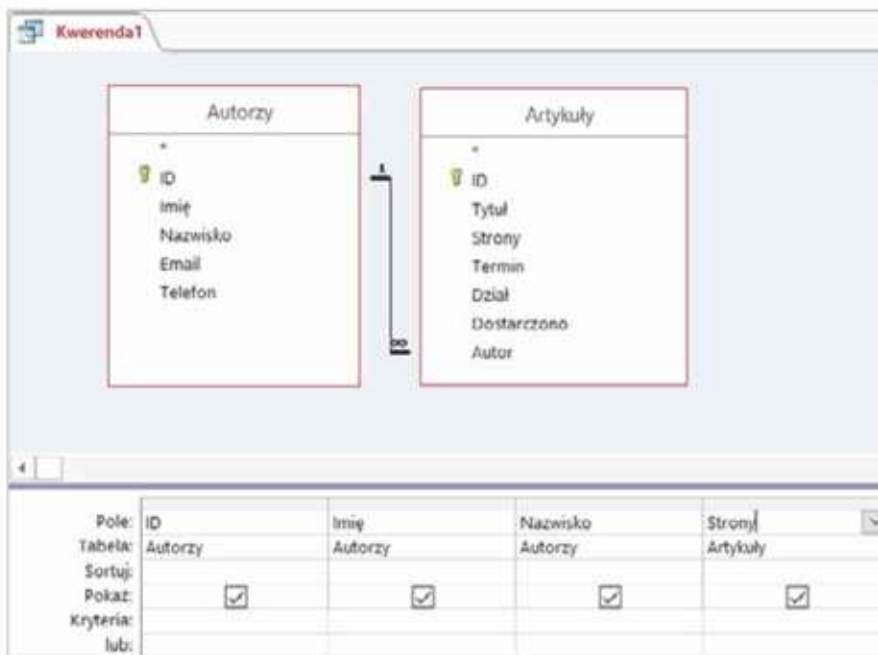
Kwerenda  **Kwerenda** pozwala znajdować i pobierać z tabel lub innych kwerend dane spełniające określone kryteria. Tworzenie kwerend jest jedną z podstawowych operacji wykonywanych przez użytkowników baz danych. Dzięki kwerendom można odpytywać bazę z informacji dotyczących przechowywanych w niej danych. W naszej bazie utworzymy kwerendę, dzięki której dowiemy się, ile w sumie stron artykułów zobowiązał się dostarczyć każdy z autorów. Będzie to kwerenda wybierająca. Program Access pozwala również na tworzenie kwerend dodających, usuwających i aktualizujących dane, a także kwerend krzyżowych, które zostaną omówione w kolejnym temacie.

Aby utworzyć nową kwerendę, wybieramy **Tworzenie** → **Kwerendy** → **Projekt kwerendy**. W ten sposób przejdziemy do widoku projektu pustej kwerendy. Na ekranie pojawi się okno **Pokazywanie tabeli**, w którym możemy dodać wybrane tabele do projektu. Dodajemy tabele **Artykuły** oraz **Autorzy**, ponieważ w obu tych tabelach znajdują się dane, które chcemy wyświetlić. Dodane tabele pojawią się w widoku projektu, pokaże się także łącząca je relacja.

Z obu tabel wybieramy nazwy pól, przeciągamy je do dolnej części ekranu projektu i umieszczamy na liście pól tworzących daną kwerendę (rys. 14.18). Ponieważ chcemy mieć dane autora, na początek przeciągniemy z tabeli **Autorzy** pola **ID**, **Imię** oraz **Nazwisko**. Następnie z tabeli **Artykuły** wybierzemy pole **Strony**, bo to dzięki niemu będzie można obliczyć, ile stron zobowiązał się napisać każdy z autorów.

Warto wiedzieć

Do kwerendy można dodawać nie tylko tabele, lecz także inne, wcześniej utworzone kwerendy. Jest to możliwe, ponieważ wynikiem działania kwerendy jest w rzeczywistości zbiór wybranych danych ułożonych w formie tabeli.



Rys. 14.18. Dodawanie tabel i wybieranie z nich pól tworzących kwerendę

Na tym etapie możemy już sprawdzić działanie kwerendy – wystarczy na karcie **Projektowanie** wybrać przycisk **Uruchom**. Powinniśmy zobaczyć listę autorów, a przy każdym nazwisku – liczbę stron.

Autorzy się jednak powtarzają, a my chcemy znać łączną liczbę stron dla każdego autora. Należy więc zmodyfikować kwerendę. W tym celu przechodzimy do **Widoku projektu** kwerendy i wybieramy **Projektowanie** → **Pokazywanie/ukrywanie** → **Sumy**.

Zauważ, że na liście pól wybranych do kwerendy pojawiła się opcja **Grupuj według**. Aby zsumować liczbę stron, rozwijamy tę opcję pod polem **Strony** i wybieramy **Suma**. Jeżeli chcemy, żeby lista była posortowana w kolejności od autora z największą łączną liczbą stron, powinniśmy jeszcze wybrać opcję sortowania **Malejąco**. Możemy także odznaczyć wyświetlanie identyfikatora autora. Na rysunku 14.19 na s. 272 pokazano ustawienia kwerendy oraz jej wywołanie.

Warto wiedzieć

SQL to język służący do tworzenia zapytań w relacyjnych bazach danych. W programie Access po utworzeniu kwerendy możemy zobaczyć jej zapis w języku SQL. W tym celu wystarczy przełączyć się z **Widoku projektu** lub **Widoku arkusza danych** na **Widok SQL**.

Dobra rada

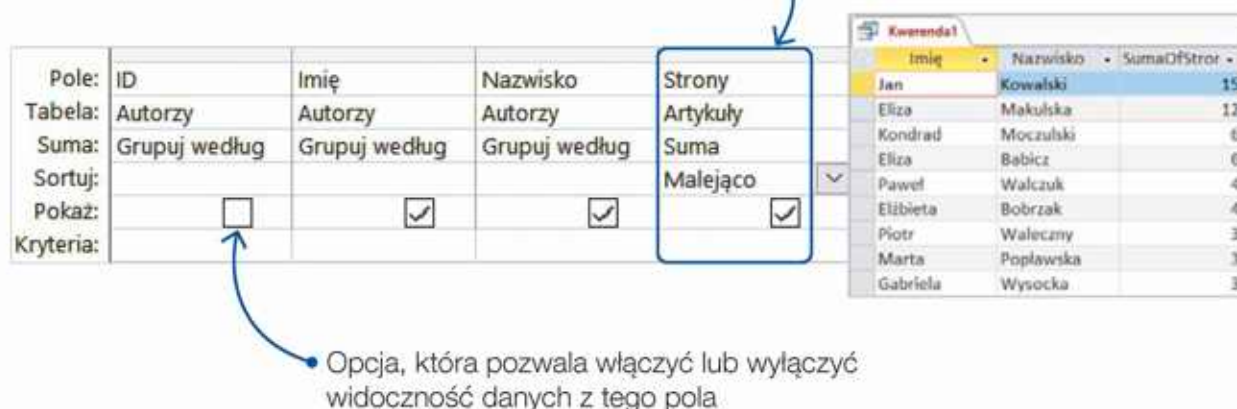
Do widoku projektu kwerendy przejdziesz analogicznie jak w przypadku innych obiektów (np. formularzy), wybierając: **Narzędzia główne** → **Widoki** → **Widok** → **Widok projektu**.

A to ciekawe

Bazy danych a *big data*

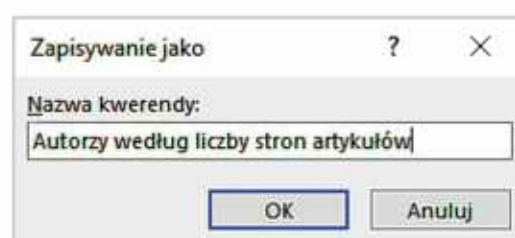
W biznesie dużą popularność zyskują rozwiązania typu *big data*, pozwalające na analizę ogromnych zbiorów danych. Rozwiązania te umożliwiają powiązanie różnorodnych danych, znajdowanie relacji między nimi i wyciąganie wniosków. W ten sposób można np. na podstawie analizy zapytań w wyszukiwarkach internetowych ustalić miejsce wybuchu epidemii grypy lub przewidywać trendy na rynkach finansowych. Nazwa *big data* ma swoje uzasadnienie, bowiem odnosi się do gigantycznych ilości danych zmieniających się w czasie. Dane te liczone są w tera- lub petabajtach.

Pole Strony, w którym dane sumujemy i sortujemy malejąco



Rys. 14.19. Ustawienia kwerendy wyświetlającej autorów według liczby stron

Jeżeli kwerenda działa zgodnie z naszymi oczekiwaniami, możemy ją zapisać. W tym celu klikamy nazwę otwartej zakładki kwerendy prawym przyciskiem myszy i z menu kontekstowego wybieramy opcję **Zapisz**. W oknie **Zapisywanie jako** należy wprowadzić nazwę, która będzie odpowiadała działaniu kwerendy (rys. 14.20).



Rys. 14.20. Zapisywanie kwerendy

Jeżeli wynik kwerendy jest inny od oczekiwanego, należy przełączyć się do widoku projektu i wprowadzić poprawki.

Warto wiedzieć

Po zapisaniu kwerendy jej nazwa pojawi się na liście wszystkich obiektów bazy.

Ćwiczenie 8

W pliku *baza_blog_8.accdb* z ćwiczenia 7 utwórz kwerendę wyświetlającą imiona i nazwiska autorów oraz łączną liczbę stron artykułów, do których napisania się zobowiązali. Posortuj listę według nazwisk. Plik zapisz pod nazwą *baza_blog_9.accdb*.

14.5. Raporty – wizualne przedstawienie danych

Raport Dzięki **raportom** można w czytelny sposób przedstawić dane pochodzące z bazy, aby je potem wyświetlić lub wydrukować.

Tworzenie raportu z użyciem kreatora

Założmy, że chcemy przygotować wydruk zawierający listę autorów wraz z łączną liczbą stron wszystkich ich artykułów. Lista ma być posortowana malejąco według liczby stron. Jeśli kilku autorów ma przygotować tę samą liczbę stron, powinni być w ramach grupy posortowani alfabetycznie według nazwisk, a następnie według imion.

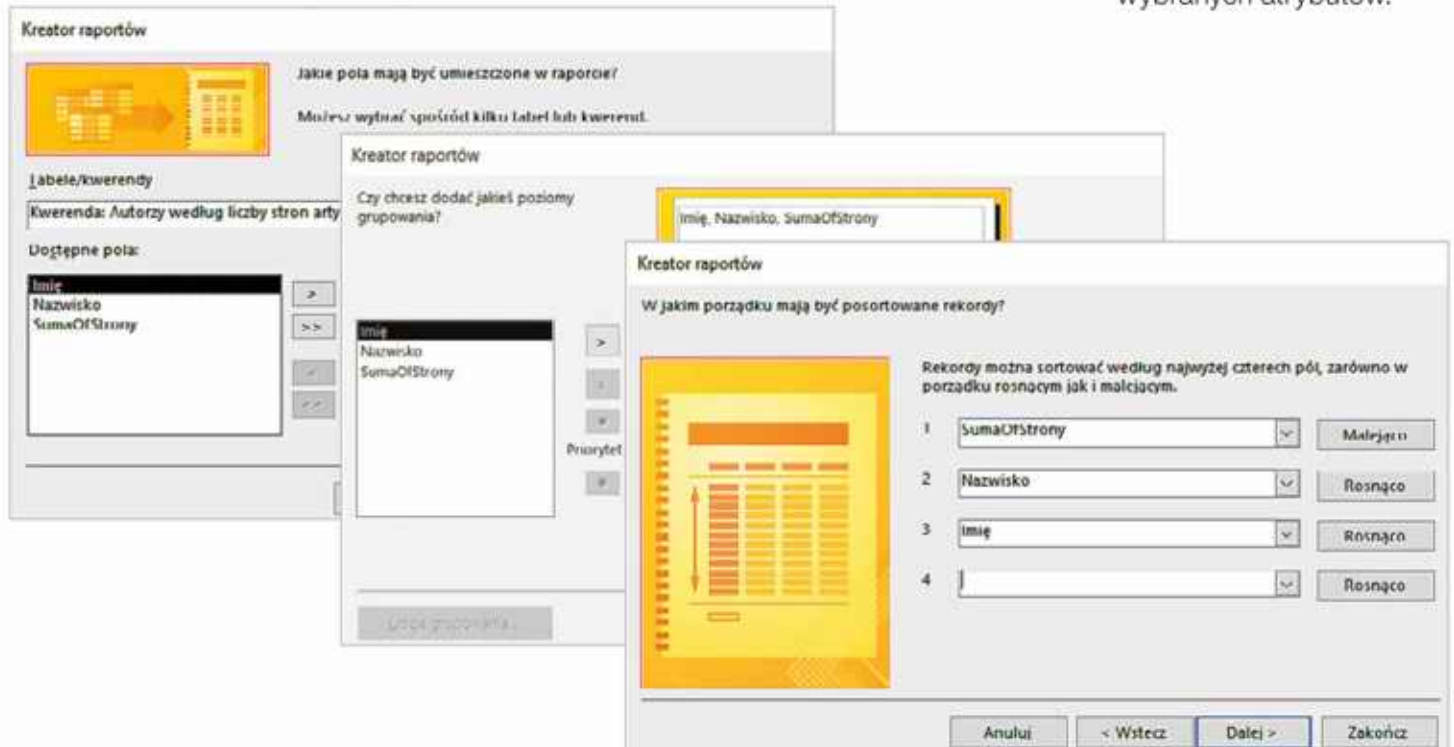
Zauważ, że w bazie znajduje się już kwerenda, która zwraca potrzebne nam dane. Do utworzenia prostego raportu możemy wykorzystać **Kreator raportów** (opcję tę można znaleźć na karcie **Tworzenie**).

W kreatorze (rys. 14.21) wybieramy kolejno:

- ▶ obiekt, na którego podstawie zostanie przygotowany raport – np. utworzoną wcześniej kwerendę,
- ▶ pole, według którego chcielibyśmy ustalić poziomy grupowania,
- ▶ pola, według których dane zostaną posortowane: SumaOfStrony (malejąco), Nazwisko (rosnąco) i Imię (rosnąco).

Warto wiedzieć

Dzięki określeniu poziomów grupowania można grupować informacje w raporcie według wartości wybranych atrybutów.



Rys. 14.21. Początkowe etapy tworzenia raportu

W kolejnych krokach możemy wybrać typ raportu. W naszym przypadku dobrze się sprawdzi układ tabelaryczny (rys. 14.22), ponieważ zależy nam na odpowiednio posortowanej liście osób. Lista będzie zawierała tylko trzy kolumny, dlatego zostawimy układ pionowy.

Autorzy według liczby stron artykułów

SumaOfStrony	Nazwisko	Imię
15	Kowalski	Jan
12	Makulski	Elixa
6	Baliński	Elixa
6	Mocniński	Ryszard
4	Bobrowski	Elixieta
4	Walczyk	Paweł
3	Popławski	Maria
3	Walczyński	Piotr
3	Wysocki	Gabriela

Rys. 14.22. Przykładowy podgląd wydruku raportu

Warto wiedzieć

Kolumny w raporcie zostaną domyślnie umieszczone w kolejności wybranej dla sortowania. Kolejność można później zmodyfikować w widoku projektu formularza.

Po zakończeniu pracy kreatora warto zobaczyć, jak wygląda raport, i zastanowić się nad wprowadzeniem ewentualnych zmian.

Dobra rada

Jeśli tabela wykorzystywana w raporcie ma wiele kolumn, rozważ poziomy układ raportu.

Ćwiczenie 9

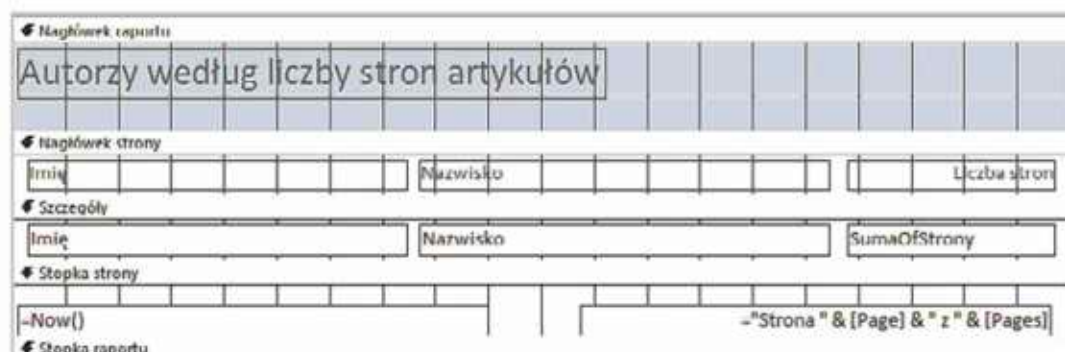
Otwórz plik *baza_blog_9.accdb*, utworzony w ćwiczeniu 8, i przygotuj raport na podstawie kwerendy *Autorzy według liczby stron artykułów*. Raport powinien zawierać imię, nazwisko oraz łączną liczbę stron artykułów napisanych lub zadeklarowanych przez danego autora. Plik zapisz pod nazwą *baza_blog_10.accdb*.

Dostosowanie raportu

W celu zwiększenia czytelności raportu możemy w widoku projektu zmienić automatycznie utworzony tekst etykiety SumaOfStrony na Liczba stron. Możemy też zmienić kolejność wyświetlanych kolumn – na pierwszym miejscu ustawić imię, potem nazwisko, a na końcu liczbę stron (rys. 14.23).

Dobra rada

W widoku projektu raportu możesz zaznaczyć wiele obiektów i przenieść je jednocześnie.



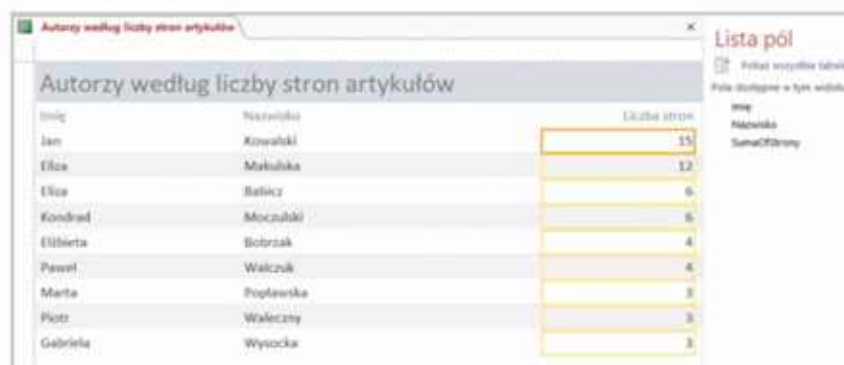
Rys. 14.23. Dostosowywanie raportu w widoku projektu

Jeśli chcemy zamienić miejscami dwie kolumny raportu, należy w widoku projektu przenieść zarówno etykiety kolumn w sekcji **Nagłówek strony**, jak i same kolumny w sekcji **Szczegóły**.

Wygląd raportu możemy również zmienić w widoku układu, edytując poszczególne jego elementy na podglądzie już gotowego raportu (rys. 14.24).

Warto wiedzieć

Usuniętą przez przypadek kolumnę można łatwo przywrócić, korzystając z widocznej obok raportu listy pól. Wystarczy przeciągnąć wybrane pole do raportu.



Rys. 14.24. Widok układu raportu

Drukowanie raportu i eksport do pliku

Po dopracowaniu raportu możemy przełączyć się do podglądu wydruku, wybierając: **Narzędzia główne** → **Widoki** → **Widok** → **Podgląd wydruku**. Jeśli chcemy wydrukować raport, wybieramy **Podgląd wydruku** → **Drukowanie** → **Drukuj**.

Raport możemy również wyeksportować do formatów XLS, PDF lub TXT, korzystając z odpowiednich przycisków na karcie **Podgląd wydruku** w sekcji **Dane**. Aby wrócić do pracy z raportem, należy wybrać przycisk **Zamknij podgląd wydruku**.

Ćwiczenie 10

Otwórz plik *baza_blog_10.accdb*, utworzony w ćwiczeniu 9. Wyeksportuj przygotowany przez siebie raport do pliku PDF.

Podsumowanie

- Bazą danych nazywamy zbiór powiązanych ze sobą informacji dotyczących pewnej dziedziny życia i zorganizowanych w taki sposób, aby łatwo było je przeszukiwać i na ich podstawie wyciągać wnioski oraz podejmować decyzje.
- Relacyjna baza danych w programie Microsoft Access składa się z tabel połączonych relacjami, kwerend, formularzy oraz raportów.
- W tabeli bazy danych przechowywane są informacje o jednym typie obiektów.
- Kolumny tabeli, zwane atrybutami, stanowią zbiór cech opisujących dany typ obiektu.
- Każdy wiersz tabeli, zwany rekordem, opisuje jeden obiekt danego typu poprzez określenie wartości poszczególnych jego atrybutów.
- Relacje opisują powiązania między danymi gromadzonymi w poszczególnych tabelach. Do tworzenia tych powiązań służą klucze główne oraz obce.
- Kwerendy pozwalają wyszukiwać w bazie danych informacje spełniające określone kryteria.
- Raporty umożliwiają wydrukowanie znalezionych danych w czytelnej, łatwej do zrozumienia formie.
- Formularze ułatwiają wprowadzanie oraz modyfikowanie danych w bazie.



Zadania

- ★ **1** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_blog_z1.accdb*) przefiltruj i posortuj tabelę *Artykuły* tak, aby wyświetlić tylko nieoddane artykuły przeznaczone do działu *Nowe technologie* w kolejności, w jakiej powinny zostać dostarczone.
- ★ **2** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_blog_z2.accdb*) przy użyciu kreatora stwórz na podstawie kwerendy *Nieoddane artykuły* raport zawierający listę niedostarczonych artykułów. Raport powinien zawierać: tytuł artykułu, nazwę działu, termin oddania, nazwisko autora. Lista powinna zostać posortowana według planowanej daty dostarczenia, a następnie – tytułu artykułu. Gotowy raport wyeksportuj do pliku PDF.
- ★ **3** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_blog_z3.accdb*) utwórz formularz pozwalający łatwo zarządzać danymi autorów artykułów.
- ★ **4** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_wol_z4.accdb*) w tabeli *Uczniowie* dla pól *Imię* i *Nazwisko* ustal maskę wprowadzania, która będzie wymuszała podanie imienia i nazwiska dużą literą. Określ również poprawną maskę wprowadzania dla pola *Kod pocztowy* w tabeli *Organizacje*.
- ★ **5** Korzystając z bazy danych otrzymanej od nauczyciela (np. w pliku *baza_wol_z5.accdb*), przygotuj raport zawierający listę wolontariuszy i nazwy organizacji, z którymi współpracują. Lista powinna być uporządkowana alfabetycznie według nazwisk.

- ★ **6** Korzystając z bazy danych otrzymanej od nauczyciela (np. w pliku *baza_wol_z6.accdb*), utwórz kwerendę, która zwróci listę uczniów z klas IIB i IIC. W pierwszej kolumnie ma być zapisane imię, w drugiej – nazwisko, a w trzeciej – nazwa klasy.
- ★★ **7** Korzystając z bazy danych otrzymanej od nauczyciela (np. w pliku *baza_wol_z7.accdb*), przygotuj formularz kolumnowy, który pozwoli wprowadzić do tabeli Uczniowie dane nowych uczniów wraz z ich przynależnością do określonej organizacji.
- ★★ **8** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_blog_z8.accdb*) dodaj w tabeli Autorzy atrybuty pozwalające przechowywać adres korespondencyjny autora, czyli: ulicę, numer domu, numer lokalu, kod pocztowy. Przyporządkuj właściwy typ danych do każdego z tworzonych atrybutów. Dla kodu pocztowego określ maskę wprowadzania.
- ★★ **9** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_blog_z9.accdb*) przygotuj kwerendę wyświetlającą listę autorów, którzy zobowiązali się dostarczyć artykuły między 20.10 a 20.11.2020 r. W kwerendzie powinny się również znaleźć tytuły tych artykułów.
- ★★ **10** Zaproponuj, jak zmodyfikować bazę danych autorów bloga opisaną w podręczniku tak, aby można w niej było gromadzić dodatkowo informacje na temat nie tylko autorów, lecz także innych osób współpracujących przy tworzeniu bloga – np. grafików, animatorów, redaktorów itd.
- ★★★ **11** Zaprojektuj bazę danych, która pozwoliłaby samorządowi szkolnemu gromadzić informacje na temat uczniów gotowych pomagać kolegom w nauce przed maturą. Baza powinna zawierać dane kontaktowe wolontariuszy oraz umożliwiać odnajdywanie ich ze względu na przedmioty, w których mogą pomóc. Uwzględnij to, że jeden wolontariusz może pomagać w wielu przedmiotach oraz że dla jednego przedmiotu pomoc może oferować wielu wolontariuszy.
- ★★★ **12** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_blog_z12.accdb*) dodaj nową tabelę służącą do przechowywania informacji na temat działów, w których są publikowane artykuły. Wprowadź w niej nazwy działów oraz ich opisy na podstawie danych gromadzonych dotychczas w kolumnie Dział tabeli Artykuły. Następnie dodaj nowe pole IdDziału w tabeli Artykuły. Za pomocą kreatora odnośników stwórz powiązanie między tabelami Artykuły oraz Działy. Przyporządkuj artykuły do działów, których nazwy zostały umieszczone w tabeli Działy. Następnie usuń kolumnę Dział z tabeli Artykuły.
- ★★★ **13** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_blog_z13.accdb*) stwórz kwerendę wyświetlającą liczbę artykułów opublikowanych na blogu w danym roku. Wyświetl dane z lat 2017–2020.
- ★★★ **14** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_blog_z14.accdb*) zaprojektuj kwerendę wyświetlającą tytuły artykułów, które powinny zostać dostarczone między 01.11 a 31.12.2020 r., ale nie zostały jeszcze przekazane przez autorów. Niech kwerenda wyświetla również imiona i nazwiska autorów artykułów.

15. Wykorzystanie danych pochodzących z kwerend

Odnalezienie interesujących nas informacji w ogromnych bazach danych jest nie lada wyzwaniem. Dlatego kluczowym elementem każdego systemu zarządzania bazą danych jest możliwość wysyłania do niej precyzyjnych zapytań, nazywanych kwerendami. Dzięki nim możemy szybko wybrać dokładnie takie informacje, jakich potrzebujemy.

Cele lekcji

- Zrozumiesz, jak można zastosować relację typu wiele do wielu.
- Dowiesz się, jak ustalić precyzyjne kryteria kwerend wybierających.
- Poznasz możliwości kwerend krzyżowych.
- Wykorzystasz arkusz kalkulacyjny do przetwarzania danych z kwerend.

15.1. Łączenie danych za pomocą tabeli pomostowej

Posłużymy się przykładem z poprzedniego tematu, w którym została omówiona baza danych składająca się z dwóch tabel: *Artykuły* i *Autorzy*. W obu tabelach występuje atrybut ID, który pełni funkcję klucza podstawowego. Schematy tych tabel można zapisać następująco:

Artykuły(ID, Tytuł, Strony, Termin, Dział, Dostarczone, Autor)
Autorzy(ID, Imię, Nazwisko, Telefon, Email)

Podkreślenie atrybutu ID oznacza, że został on wybrany jako klucz podstawowy. Jednocześnie atrybut *Autor* w tabeli *Artykuły* stanowi jej klucz obcy, ponieważ wskazuje na klucz główny tabeli *Autorzy*. Tabele są więc połączone relacją typu jeden do wielu, co w tym przypadku oznacza, że do każdego artykułu możemy przypisać jednego autora, a każdy autor może napisać wiele artykułów.

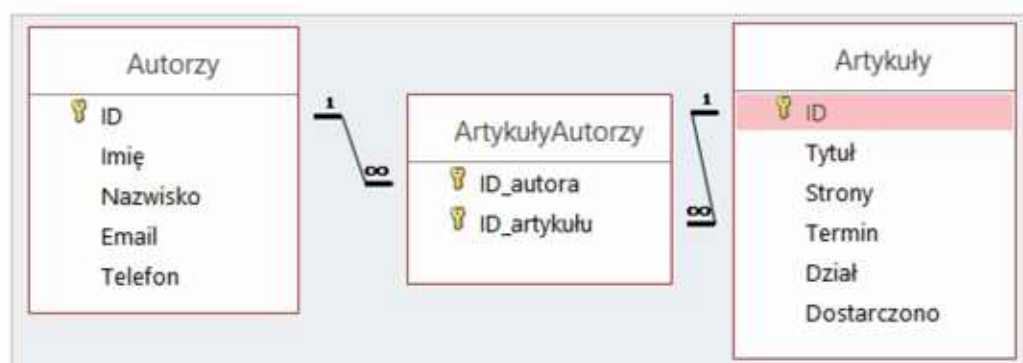
Jeśli przyjmiemy jednak, że jeden artykuł może być dziełem wielu autorów, potrzebna jest zmiana relacji na **relację wiele do wielu** ($\infty:\infty$). Aby to zrobić, potrzebujemy trzeciej tabeli, której głównym zadaniem będzie połączenie ze sobą rekordów z tabel *Artykuły* i *Autorzy*, choć można będzie w niej umieszczać również dodatkowe informacje. Jest to tzw. **tabela pomostowa**, określana też jako **tabela łącząca** lub **tabela łączenia**. Naszą tabelę pomostową nazwiemy *ArtykułyAutorzy*. Jej atrybuty oraz relacje z pozostałymi tabelami pokazano na rysunku 15.1 na s. 278.

Warto wiedzieć

W literaturze można spotkać poszerzony opis schematu tabel, uwzględniający typy poszczególnych atrybutów. Typ atrybutu oddziela się od jego nazwy dwukropkiem, np. *Tytuł*: VARCHAR(30) oznacza, że atrybut *Tytuł* może przyjmować wartości typu tekstowego, które mają długość do 30 znaków.

Relacja typu wiele do wielu,
s. 258 

● **Tabela pomostowa**
(tabela łącząca, tabela łączenia)



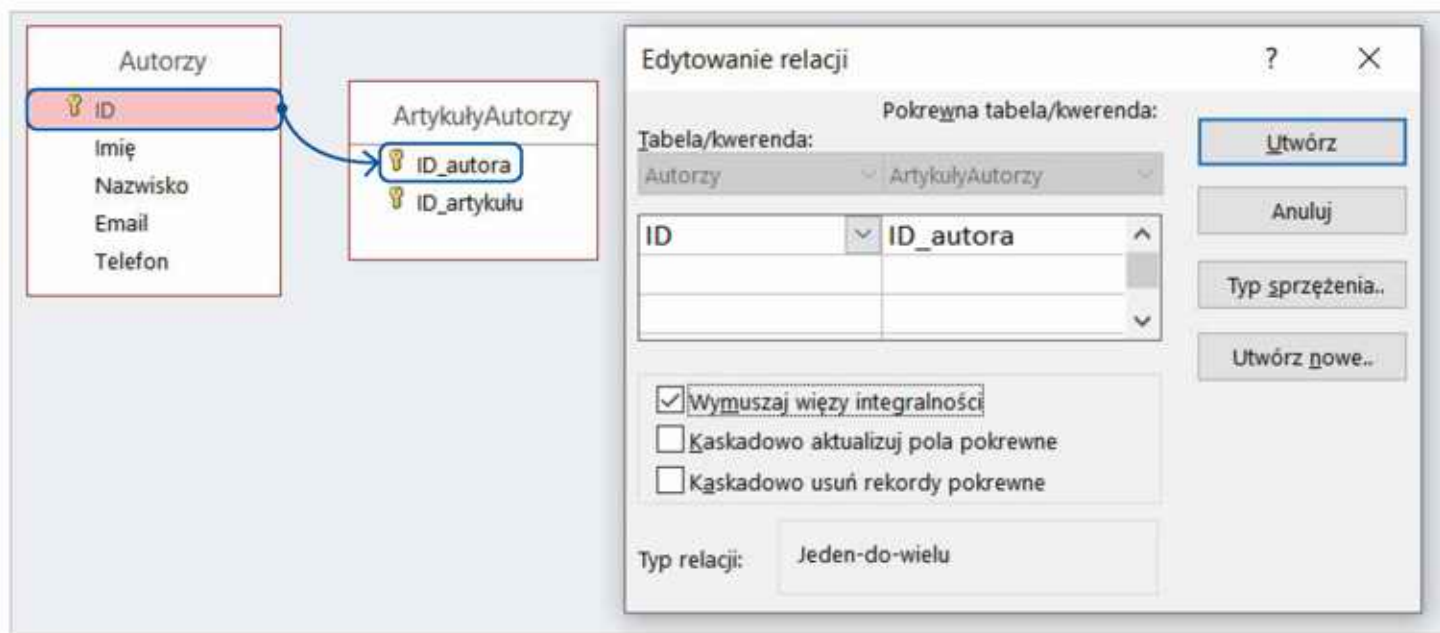
Rys. 15.1. Tabele w bazie danych i relacje między nimi

Klucz podstawowy tabeli ArtykułyAutorzy stanowią dwa atrybuty, ponieważ każda występująca w tabeli para identyfikatorów musi być unikatowa. Klucz podstawowy składający się z kilku atrybutów nazywamy **kluczem złożonym**. Aby ustawić taki klucz w programie Access, należy w widoku projektu zaznaczyć oba identyfikatory (z przytrzymanym klawiszem **Ctrl**), a następnie wybrać **Narzędzia tabel** → **Projektowanie** → **Narzędzia** → **Klucz podstawowy**.

Na rysunku 15.1 widoczne są dwie relacje jeden do wielu (1:∞), razem stanowiące jedną relację wiele do wielu (∞:∞). Aby te relacje utworzyć, najpierw wybieramy **Narzędzia baz danych** → **Relacje** → **Relacje** i dodajemy wszystkie tabele. Następnie z listy atrybutów tabeli Autorzy przeciągamy atrybut ID i upuszczamy go na atrybucie ID_autora z tabeli ArtykułyAutorzy. Tak samo postępujemy z atrybutem ID z tabeli Artykuły – przeciągamy go na atrybut ID_artykułu z tabeli ArtykułyAutorzy. Wartości w kolumnach stanowiących klucz główny tabeli ArtykułyAutorzy muszą być zgodne z wartościami w kolumnach ID pozostałych tabel. Dlatego w ustawieniach obu relacji zaznaczamy opcję **Wymuszaj więzy integralności** (rys. 15.2).

Dobra rada

Jeśli zaznaczysz opcję **Kaskadowo usuń rekordy pokrewne**, usunięcie dowolnego rekordu z tabeli podstawowej spowoduje usunięcie wszystkich powiązanych z nim rekordów w połączonej tabeli.



Rys. 15.2. Tworzenie relacji i wymuszanie więzów integralności między tabelami

Ćwiczenie 1

W bazie danych, którą otrzymasz od nauczyciela (np. w pliku *baza_15_1.accdb*), utwórz tabelę *ArtykułyAutorzy*. Powinna ona zawierać atrybuty oraz relacje z innymi tabelami przedstawione na rysunku 15.1. Uzupełnij tabelę danymi tak, aby każdy artykuł miał co najmniej dwóch autorów i aby każdy autor był przypisany przynajmniej do jednego artykułu. Plik zapisz pod nazwą *baza_15_2.accdb*.

Zapamiętaj

Relacja typu wiele do wielu pozwala na połączenie wielu rekordów z jednej tabeli z wieloma rekordami z drugiej tabeli. Wykorzystuje się do tego dodatkową tabelę, tzw. tabelę pomostową, która w kolejnych wierszach łączy ze sobą rekordy z obu tabel.

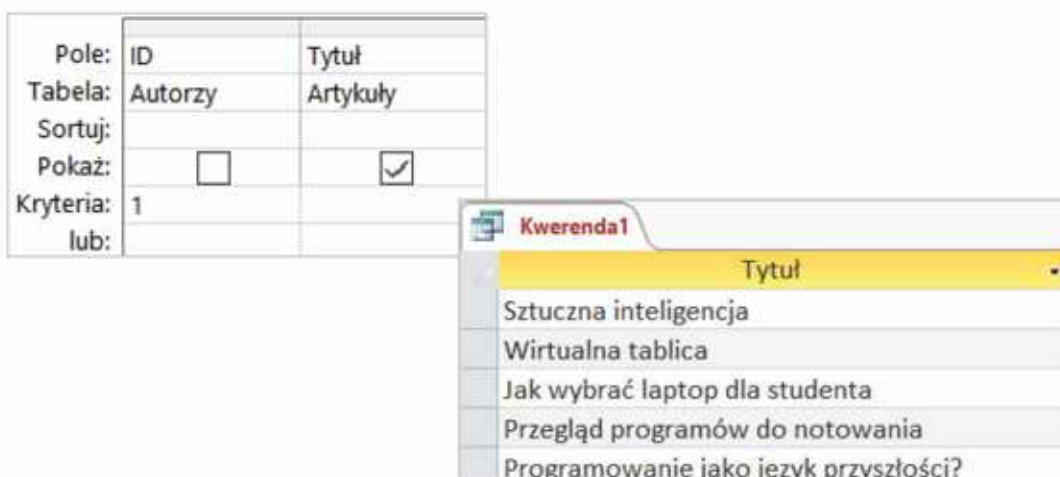
15.2. Parametry i kryteria kwerend wybierających

Kwerenda wybierająca pozwala wybrać dokładnie takie informacje, jakie są nam potrzebne. Taką kwerendę tworzyliśmy w poprzednim temacie. Teraz pokażemy, jak jeszcze precyzyjniej ustawić jej kryteria.

W widoku projektu kwerendy możemy wpisać kryteria ograniczające wyświetlane wyniki. Jeśli chcemy np. wyświetlić tytuły artykułów przypisane konkretnemu autorowi, dodajemy do kwerendy atrybut ID z tabeli *Autorzy* oraz Tytuł z tabeli *Artykuły*, a jako kryterium pierwszego pola wpisujemy numer ID interesującego nas autora (rys. 15.3).

Dobra rada

Możesz łatwo zmienić typ kwerendy. W tym celu w widoku projektu przejdź do **Narzędzi kwerend** i na karcie **Projektowanie** wybierz interesujący cię typ kwerendy.



Rys. 15.3. Ustawienia i wyniki kwerendy wyświetlającej artykuły autora o ID równym 1

Gwiazdką * możemy zastąpić dowolny ciąg znaków. Jeśli w kryteriach powyższej kwerendy dla pola *Tytuł* wpisujemy *p**, wyświetlą się te artykuły autora o numerze ID równym 1, których tytuły zaczynają się na literę *p* (rys. 15.4, s. 280).

Warto wiedzieć

Po wpisaniu gwiazdki program Access automatycznie dodaje słowo *Like*, które pochodzi od angielskiego *like*, oznaczającego „podobny do”, „taki jak”.

Pole:	ID	Tytuł
Tabela:	Autorzy	Artykuły
Sortuj:		
Pokaż:	☐	☒
Kryteria:	1	Like "p*"
lub:		

Kwerenda1	Tytuł
	Przegląd programów do notowania
	Programowanie jako język przyszłości?

Rys. 15.4. Ustawienia i wyniki kwerendy wyświetlającej artykuły konkretnego autora o tytułach zaczynających się na literę p

Co zrobić w przypadku, kiedy chcemy np. wyświetlić tylko tych autorów, którzy mają do napisania artykuły o liczbie stron równej 5 lub więcej? W kwerendzie umieszczamy pola Imię i Nazwisko z tabeli Autorzy i pole Strony z tabeli Artykuły. Następnie dodajemy pokazywanie sumy, wybierając **Narzędzia kwerend** → **Projektowanie** → **Pokazywanie/ukrywanie** → **Sumy**. Dla atrybutu Strony ustawiamy **Suma** na **Gdzie**, a w kryteriach wykorzystujemy operator porównania $\geq$ (rys. 15.5).

Pole:	Imię	Nazwisko	Strony
Tabela:	Autorzy	Autorzy	Artykuły
Suma:	Grupuj według	Grupuj według	Gdzie
Sortuj:			
Pokaż:	☒	☒	☐
Kryteria:			≥ 5
lub:			

Kwerenda1	Imię	Nazwisko
	Eliza	Makulska
	Gabriela	Wysocka
	Jan	Kowalski
	Janusz	Wierzbicki
	Paweł	Krynicky
	Paweł	Walczuk

Rys. 15.5. Ustawienia i wyniki kwerendy wyświetlającej autorów, którzy mają napisać najdłuższe artykuły

Dobra rada

Wiersza **Suma** w widoku projektu kwerendy możesz również użyć do obliczania średniej wartości pól lub wskazywania minimalnej i maksymalnej wartości.

Jeśli dodamy do siatki widoku projektu pole Tytuł z tabeli Artykuły i ustawimy dla tego pola **Suma** na **Policz**, możemy zobaczyć, ile takich artykułów ma do napisania każdy z autorów. Wyniki posortujemy malejąco według liczby artykułów (rys. 15.6).

Pole:	Imię	Nazwisko	Strony	Tytuł
Tabela:	Autorzy	Autorzy	Artykuły	Artykuły
Suma:	Grupuj według	Grupuj według	Gdzie	Policz
Sortuj:				Malejąco
Pokaż:	☒	☒	☐	☒
Kryteria:			≥ 5	
lub:				

Kwerenda1	Imię	Nazwisko	PoliczOfTytuł
	Eliza	Makulska	2
	Piotr	Walczyński	1
	Paweł	Walczuk	1
	Paweł	Krynicky	1
	Janusz	Wierzbicki	1

Rys. 15.6. Ustawienia i wyniki kwerendy pokazującej, kto ma do napisania najwięcej długich artykułów

Możemy też w kryteriach wykorzystywać elementy logiki matematycznej i za pomocą operatorów And i Or jeszcze bardziej precyzować wyniki. Na rysunku 15.7 widoczne są ustawienia kwerendy wyświetlającej autorów, których imiona kończą się na *a* lub *z* i którzy mają do napisania więcej niż jeden, ale mniej niż pięć artykułów.

Pole:	Imię	Nazwisko	Tytuł
Tabela:	Autorzy	Autorzy	Artykuły
Suma:	Grupuj według	Grupuj według	Policz
Sortuj:			Malejąco
Pokaż:	☒	☒	☒
Kryteria:	Like "**a" Or Like "**z"		> 1 And < 5
lub:			

Rys. 15.7. Parametry kwerendy wykorzystującej operatory logiczne

Możemy również wykorzystać kolejne wiersze siatki widoku projektu. Nie oznacza to jednak, że zapis z rysunku 15.8 jest równoznaczny z tym z rysunku 15.7.

Pole:	Imię	Nazwisko	Tytuł
Tabela:	Autorzy	Autorzy	Artykuły
Suma:	Grupuj według	Grupuj według	Policz
Sortuj:			Malejąco
Pokaż:	☒	☒	☒
Kryteria:	Like "**a"		> 1 And < 5
lub:	Like "**z"		

Rys. 15.8. Zapisanie kryteriów w oddzielnych wierszach siatki widoku projektu

W przypadku pokazanym na rysunku 15.8 kwerenda wyświetli wszystkich autorów, których imiona kończą się na *a* i mają oni do napisania więcej niż 1 i mniej niż 5 artykułów, oraz wszystkich tych, których imiona kończą się na *z*, ale do napisania mają dowolną liczbę artykułów.

Warto wiedzieć

Kolejne wiersze kryteriów związane są alternatywą logiczną (lub), a pola w danym wierszu związane są koniunkcją logiczną (i).

Ćwiczenie 2

W bazie danych utworzonej w ćwiczeniu 1 przygotuj kwerendę wyświetlającą autorów, którzy powinni dostarczyć 10 lub więcej stron artykułów. Posortuj kwerendę według liczby stron artykułów. Plik zapisz pod nazwą *baza_15_3.accdb*.

Zapamiętaj

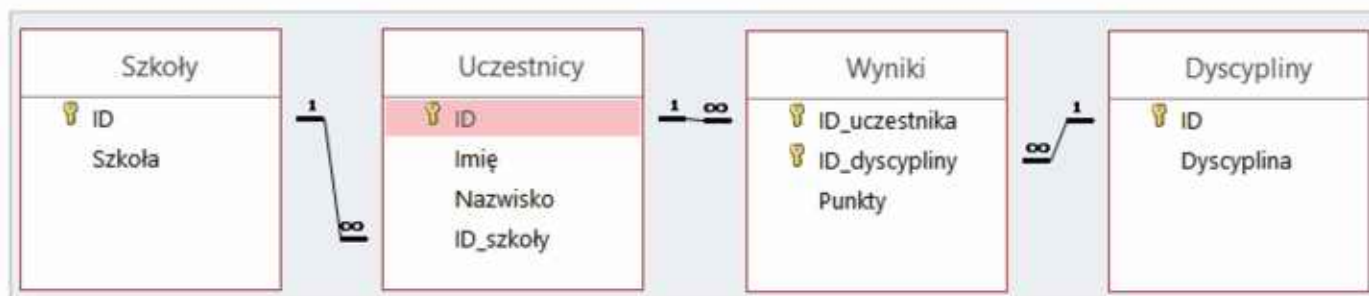
W widoku projektu kwerendy można definiować szczegółowe dane do wyświetlenia w kwerendzie. Kryteria te mogą być tekstowe, również uwzględniające tylko fragment tekstu, oraz liczbowe, w tym wykorzystujące logikę matematyczną.

15.3. Agregowanie danych w kwerendach krzyżowych

W programie Excel mogliśmy tworzyć zestawienia danych, łącząc je w postaci tabel przestawnych. Program Access pozwala przygotowywać podobne zestawienia, zebrane z kilku tabel. Wykorzystuje się do tego

Kwerenda krzyżowa • **kwerendy krzyżowe**, w których można zdecydować, jakie informacje będą w nagłówkach kolumn, jakie w nagłówkach wierszy, a także jak zostaną przedstawione wartości w komórkach tabeli.

Wykorzystamy kwerendę krzyżową do przedstawienia wyników zawodów sportowych, w których wzięli udział uczniowie kilku szkół. W oddzielnych tabelach zapiszemy dane uczestników oraz ich szkół, a także nazwy dyscyplin i wyniki osiągnięte przez uczniów w poszczególnych dyscyplinach. Poszczególne tabeli oraz układ relacji między nimi przedstawiono na rysunku 15.9.



Rys. 15.9. Tabele bazy danych przygotowane na potrzeby zawodów sportowych

Tabela pomostowa,
s. 277 [🔗](#)

Widzimy, że tabela Wyniki jest **tabelą pomostową**. Mamy więc do czynienia z relacją typu wiele do wielu. Oznacza to, że każdy uczeń mógł rywalizować w więcej niż jednej dyscyplinie.

Tak przygotowane tabeli możemy wypełnić danymi. Po zakończeniu zawodów wpisujemy wyniki uczestników do tabeli Wyniki. Możemy wykorzystać **kwerendy wybierające** do tworzenia różnego rodzaju podsumowań po zawodach. Jeśli wykorzystamy kwerendę krzyżową, będziemy mogli podsumować w jednym miejscu wyniki poszczególnych zawodników w kolejnych dyscyplinach.

Kwerendę krzyżową włączamy za pomocą opcji **Narzędzia kwerend** → **Projektowanie** → **Typ kwerendy** → **Krzyżowa**. Następnie dodajemy do niej kolejne pola, jednocześnie decydując, gdzie powinny się one znaleźć. My zaczniemy od nagłówka wiersza, który nazwiemy Uczestnik i w którym połączymy nazwiska i imiona uczestników za pomocą następującej definicji:

Uczestnik: [Imię] & " " & [Nazwisko]

Według tego pola będziemy sortowali kwerendę. Nagłówkiem kolumny będzie natomiast nazwa dyscypliny i tutaj również ustawimy sortowanie alfabetyczne. Na przecięciu informacji o uczestnikach oraz dyscyplinach powinna się znaleźć liczba punktów. Dodamy więc pole Punkty z tabeli Wyniki jako **Wartość** kwerendy krzyżowej.

Kwerenda wybierająca,
s. 279 [🔗](#)

👍 Dobra rada

Do kwerendy krzyżowej możesz dodać wiele nagłówków wierszy, ale tylko jeden nagłówek kolumny.

Dla tego pola musimy też ustawić sposób **agregowania danych**, czyli **Aggregowanie danych** podsumowywania wartości w kwerendzie krzyżowej. Robimy to w wierszu **Suma**.

Wszystkie ustawienia kwerendy widoczne są na rysunku 15.10.

Pole:	Uczestnik: [Imię] & " " & [Nazwisko]	Dyscyplina	Punkty
Tabela:	Uczestnicy	Dyscypliny	Wyniki
Suma:	Grupuj według	Grupuj według	Suma
Krzyżowe:	Nagłówek wiersza	Nagłówek kolumny	Wartość
Sortuj:	Rosnąco		
Kryteria:			
lub:			

Rys. 15.10. Ustawienia kwerendy krzyżowej

Po uruchomieniu kwerendy otrzymamy tabelaryczne zestawienie informujące, ile punktów w każdej dyscyplinie uzyskał każdy z uczestników (patrz s. 284–285).

Zwróć uwagę, że w tej kwerendzie zebrane są dane z aż trzech tabel. Imiona i nazwiska uczestników zostały pobrane z tabeli Uczestnicy, nazwy dyscyplin, w których startowali uczniowie – z tabeli Dyscypliny, a liczba uzyskanych przez nich punktów – z tabeli Wyniki.

Ćwiczenie 3

W bazie danych przekazanej przez nauczyciela (np. w pliku *zawody_baza.accdb*) utwórz kwerendę krzyżową, w której podsumowane będą punkty uzyskane przez uczestników zawodów w poszczególnych konkurencjach. W nagłówku wiersza umieść imiona i nazwiska zawodników, a w nagłówku kolumn – nazwy dyscyplin. Plik zapisz pod nazwą *zawody_baza_3.accdb*.

A to ciekawe

Społecznościowe bazy danych

Jednymi z największych baz danych są te, do których treści dodajemy my sami, czyli portale społecznościowe. Według informacji z 2020 r. korzysta z nich blisko połowa ludzi na świecie. Zdecydowanym liderem jest Facebook, który zrzesza ok. 2,5 miliarda osób. Z posiadaniem tak ogromnej bazy danych wiąże się duża odpowiedzialność, ale też bardzo duże zyski uzyskiwane z reklam umieszczanych w portalu.



Kwerenda krzyżowa

W oddzielnych tabelach bazy danych zebraliśmy informacje o uczestnikach zawodów sportowych, dyscyplinach, w jakich ze sobą konkurowali, oraz wynikach rywalizacji. Tabele połączyliśmy relacjami, dzięki którym uczestnikom przypisane są dyscypliny i zdobyta liczba punktów. Informacje te możemy podsumować w kwerendzie krzyżowej i wyświetlić w formie czytelnego, tabelarycznego zestawienia.

Projekt

Pole:	Uczestnik: [Imię] & " " & [Nazwisko]	Dyscyplina	Punkty
Tabela:	Uczestnicy	Dyscypliny	Wyniki
Suma:	Grupuj według	Grupuj według	Suma
Krzyżowe:	Nagłówek wiersza	Nagłówek kolumny	Wartość
Sortuj:	Rosnąco	Rosnąco	
Kryteria:			

Nagłówek wiersza

Wiersze są kolejnym po kolumnach poziomem grupowania wartości. Nagłówek wiersza może pochodzić z innej tabeli niż nagłówek kolumny. Kwerenda krzyżowa może zawierać jeden nagłówek wiersza lub więcej.

ID	Imię	Nazwisko	ID_szkoły
1	Jan	Nowak	6
2	Maria	Kowalska	2
3	Przemysław	Zalewski	3
4	Robert	Rycki	1
5	Zofia	Stefańska	4
6	Michał	Kowalewski	2

Pola Imię i Nazwisko z tabeli Uczestnicy

→ Nagłówek kolumny

Wartości grupowane są w kolumnach, których nagłówki zostały utworzone na podstawie pól wybranego atrybutu jednej z tabel bazy danych. W kwerendzie krzyżowej może być tylko jeden nagłówek kolumny.

ID	Dyscyplina
1	Pływanie
2	Skoki
3	Tenis
4	Bieg

Pole Dyscyplina z tabeli Dyscypliny

Wynik

Uczestnik	Pływanie	Skoki	Tenis	Bieg
Jan Nowak	18	31	81	36
Maria Kowalska	66	93	32	72
Michał Kowalewski	38	18	63	36
Przemysław Zalewski	27	36	51	60
Robert Rycki	58	75	18	18
Zofia Stefańska	61	24	62	15

→ Wartości

Dane z wybranego pola są agregowane w tym miejscu i grupowane w kolumnach oraz wierszach o ustalonych nagłówkach.

ID_uczestnika	ID_dyscypliny	Punkty
1	1	18
1	2	31
1	3	81
1	4	36
2	1	66
2	2	93

Pole Punkty z tabeli Wyniki

Eksportowanie wyników kwerendy do arkusza kalkulacyjnego

Wyeksportujemy dane uzyskane w kwerendzie do arkusza kalkulacyjnego, aby wyłonić zwycięzców w każdej dyscyplinie oraz uczestnika, który zdobył najwięcej punktów (rys. 15.11).

	A	B	C	D	E
1	Uczestnik	Bieg	Pływanie	Skoki	Tenis
2	Andrzej Cebulski	26	13	42	18
3	Angelika Jureszyn	76	90	21	62
4	Franciszek Kochański	22	18	22	31
5	Jan Nowak	36	18	31	81
6	Janusz Sosnowski	33	18	3	43
7	Marcin Bieredzewski	91	79	83	28
8	Maria Kowalska	72	66	93	32
9	Michał Kowalewski	36	38	18	63
10	Monika Leśniewska	64	75	35	89
11	Przemysław Zalewski	60	27	36	51

Rys. 15.11. Dane z kwerendy po skopiowaniu do arkusza kalkulacyjnego

Dobra rada

Wyniki kwerendy możesz też skopiować do arkusza kalkulacyjnego, wykorzystując systemowy schowek, jednak w przypadku dużej ilości danych ograniczeniem może być pojemność pamięci operacyjnej komputera.

Aby to zrobić, zaznaczamy utworzoną kwerendę na liście obiektów bazy danych i wybieramy **Dane zewnętrzne** → **Eksportuj** → **Excel**. W otwartym oknie wybieramy miejsce zapisu pliku i jego nazwę. Tak zapisany plik możemy otworzyć w arkuszu kalkulacyjnym.

Teraz wystarczy już tylko posortować dane według wybranej kolumny, aby sprawdzić, który uczestnik zawodów zdobył najwięcej punktów w danej dyscyplinie.

Aby wyłonić osobę z największą liczbą punktów, dodajemy w arkuszu kolumnę, w której sumujemy punkty uzyskane przez każdego uczestnika. Następnie sortujemy tabelę według tej kolumny (rys. 15.12).

	A	B	C	D	E	F
1	Uczestnik	Bieg	Pływanie	Skoki	Tenis	Suma
2	Marcin Bieredzewski	91				
3	Monika Leśniewska	64				
4	Maria Kowalska	72				
5	Angelika Jureszyn	76				
6	Przemysław Zalewski	60				
7	Robert Rycki	18				
8	Jan Nowak	36				
9	Zofia Stefańska	15				
10	Michał Kowalewski	36				

Rys. 15.12. Wyłanianie uczestnika zawodów z największą liczbą punktów

Ćwiczenie 4

Dane z kwerendy utworzonej w ćwiczeniu 3 skopiuj do arkusza kalkulacyjnego. Zsumuj wyniki poszczególnych zawodników i wyznacz uczestnika, który zdobył najwięcej punktów. Plik zapisz pod nazwą *zawody_wyniki.xlsx*.

Podsumowanie

- Do budowania relacji typu wiele do wielu stosuje się tabelę pomostową, której kolejne wiersze łączą ze sobą rekordy z innych tabel.
- W kwerendach wybierających można stosować rozmaite kryteria filtrujące.
- Kryteria filtrujące w kwerendach można ze sobą łączyć, wykorzystując koniunkcję logiczną (i) oraz alternatywę logiczną (lub).
- Pojedyncze pole kwerendy może zawierać wyrażenia pozwalające np. sumować zawartość kilku pól tabel bazy danych.
- Kwerenda krzyżowa pozwala na agregowanie danych w formie tabelarycznej, przypominającej tabele przestawne z arkusza kalkulacyjnego.

Zadania

- ★ **1** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_kw_z1.accdb*) znajdują się dwie tabele: Oceny i Uczniowie. Utwórz trzecią tabelę o nazwie Wyniki, w której połączysz relacją wiele do wielu tabele Oceny i Uczniowie tak, żeby można było każdemu uczniowi przypisać dowolną liczbę ocen. Uczeń może również otrzymać kilka takich samych ocen.
- ★ **2** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_kw_z2.accdb*) utwórz kwerendę zawierającą zestawienie wszystkich ocen uczniów. W kwerendzie powinny się znaleźć imiona i nazwiska uczniów oraz oceny.
- ★ **3** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_kw_z3.accdb*) utwórz zestawienie liczby poszczególnych ocen. Posortuj dane malejąco według tej liczby.
- ★★ **4** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_kw_z4.accdb*) utwórz kwerendę wyświetlającą imiona i nazwiska uczniów z ocenami niedostatecznymi. Każdy uczeń powinien być na liście tylko raz, nawet jeśli ma kilka ocen niedostatecznych. Posortuj listę alfabetycznie według nazwisk uczniów.
- ★★ **5** Na stronie dane.gov.pl znajdź informacje o najpopularniejszych imionach żeńskich nadanych dzieciom w Polsce w 2020 r. Pobierz je w formacie CSV i zaimportuj do bazy danych. Utwórz zestawienie najpopularniejszych imion w województwie dolnośląskim i posortuj je według liczby wystąpień. Ukryj kolumnę z nazwą województwa.
- ★★ **6** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_kw_z6.accdb*) utwórz kwerendę krzyżową wyświetlającą, ile poszczególnych ocen otrzymał każdy uczeń. Posortuj dane alfabetycznie według nazwisk.
- ★★ **7** Zaprojektuj bazę danych, w której będziesz przechowywać informacje o swoich ocenach. Powinna ona zawierać w oddzielnych tabelach listę przedmiotów szkolnych i kategorie ocen (np. sprawdzian, odpowiedź ustna, praca na lekcji). Wykorzystaj relację typu wiele do wielu.

- *** **8** Na stronie dane.gov.pl znajdź aktualną listę nazwisk męskich zapisanych w rejestrze PESEL. Utwórz kwerendę wyświetlającą nazwiska zaczynające się na literę *N*, a kończące się na *-ski*, których łączna liczba wystąpień jest większa niż tysiąc.
- *** **9** W bazie danych otrzymanej od nauczyciela (np. w pliku *baza_kw_z9.accdb*) utwórz kwerendę zawierającą informacje o pensjach i premiach pracowników poszczególnych działów firmy. Oblicz kwoty do wypłaty dla każdego pracownika.
- *** **10** Korzystając z bazy danych otrzymanej od nauczyciela (np. w pliku *baza_kw_z10.accdb*), oblicz, jakie kwoty firma wydała w poszczególnych miesiącach na premie dla pracowników.
- *** **11** W bazie danych, którą otrzymasz od nauczyciela (np. w pliku *baza_kw_z11.accdb*), znajdują się informacje o cenach produktów w pewnym sklepie. Stały klient sklepu otrzymuje rabat w wysokości 10%. Przygotuj zestawienie produktów uwzględniające ceny wyjściowe oraz ceny z naliczonym rabatem.

16. Podstawy języka SQL

Na co dzień korzystacie z różnych aplikacji internetowych na swoich komputerach i smartfonach. Wiele z nich do prawidłowego działania potrzebuje dostępu do dużych ilości informacji zebranych w relacyjnych bazach danych. Aby zapewnić ten dostęp, programy wykorzystują język zapytań SQL. Warto go poznać, aby zrozumieć działanie tego typu aplikacji oraz móc samodzielnie je tworzyć.

Cele lekcji

- Poszerzysz swoją wiedzę na temat relacyjnego modelu baz danych.
- Utworzysz bazę danych i tabele.
- Dodasz rekordy do tabel.
- Zaimportujesz dane z pliku tekstowego do bazy.
- Nauczysz się edytować rekordy w bazie danych.

Język SQL (ang. *Structured Query Language*) to strukturalny język [Język SQL](#) zapytań do bazy danych, pozwalający na pełną komunikację między użytkownikiem lub aplikacją a bazą danych (rys. 16.1).



Rys. 16.1. Komunikacja między aplikacją a bazą danych za pomocą języka SQL

Za pomocą języka SQL możemy tworzyć i modyfikować bazy danych, tabele oraz rekordy w tabelach, a przede wszystkim wybierać z bazy potrzebne nam informacje.

Język SQL ewoluował przez lata, a jednocześnie producenci SZBD dostosowywali go do swoich systemów. Dlatego między różnymi systemami mogą występować nieznaczne różnice w składni języka. Jego podstawowe mechanizmy działają jednak tak samo.

W poprzednich tematach omówiliśmy możliwości programu Microsoft Access jako **systemu zarządzania bazą danych**. Zdefiniowaliśmy podstawowe pojęcia zgodnie z nazewnictwem stosowanym w tym programie. Na początku pracy z językiem SQL wprowadzimy jednak pojęcia ogólne, odnoszące się do wszystkich systemów zarządzania bazami danych. Kiedy mówi się o bazach danych, lepiej jest stosować nazewnictwo, które wprowadzimy w tym temacie. Jednak podczas pracy w programie Microsoft Access można korzystać z pojęć odnoszących się tylko do niego.

System zarządzania bazą danych,
s. 258 [🔗](#)

16.1. Pojęcia używane w różnych systemach zarządzania bazą danych

Relacja w modelu relacyjnym • **Relacja w modelu relacyjnym**, jako pojęcie dotyczące baz danych, to pojedyncza tabela, która zawiera zestaw informacji określających cechy zapisanych w niej obiektów (rys. 16.2). Baza danych zawierająca zbiór

Relacyjna baza danych • takich informacji nazywana jest **relacyjną bazą danych**.

ID	Imie	Nazwisko	ID_szkoły
1	Jan	Nowak	6
2	Maria	Kowalska	2
3	Przemysław	Zalewski	3
4	Robert	Rycki	1
5	Zofia	Stefańska	4

Rys. 16.2. Relacja (tabela) w modelu relacyjnym

Relacja, s. 257  W odniesieniu do programu Access mówiliśmy, że **relacja** to połączenie ze sobą rekordów z dwóch tabel. Takie połączenie w bazach **Związek** • danych nazywamy **związkiem** (rys. 16.3).

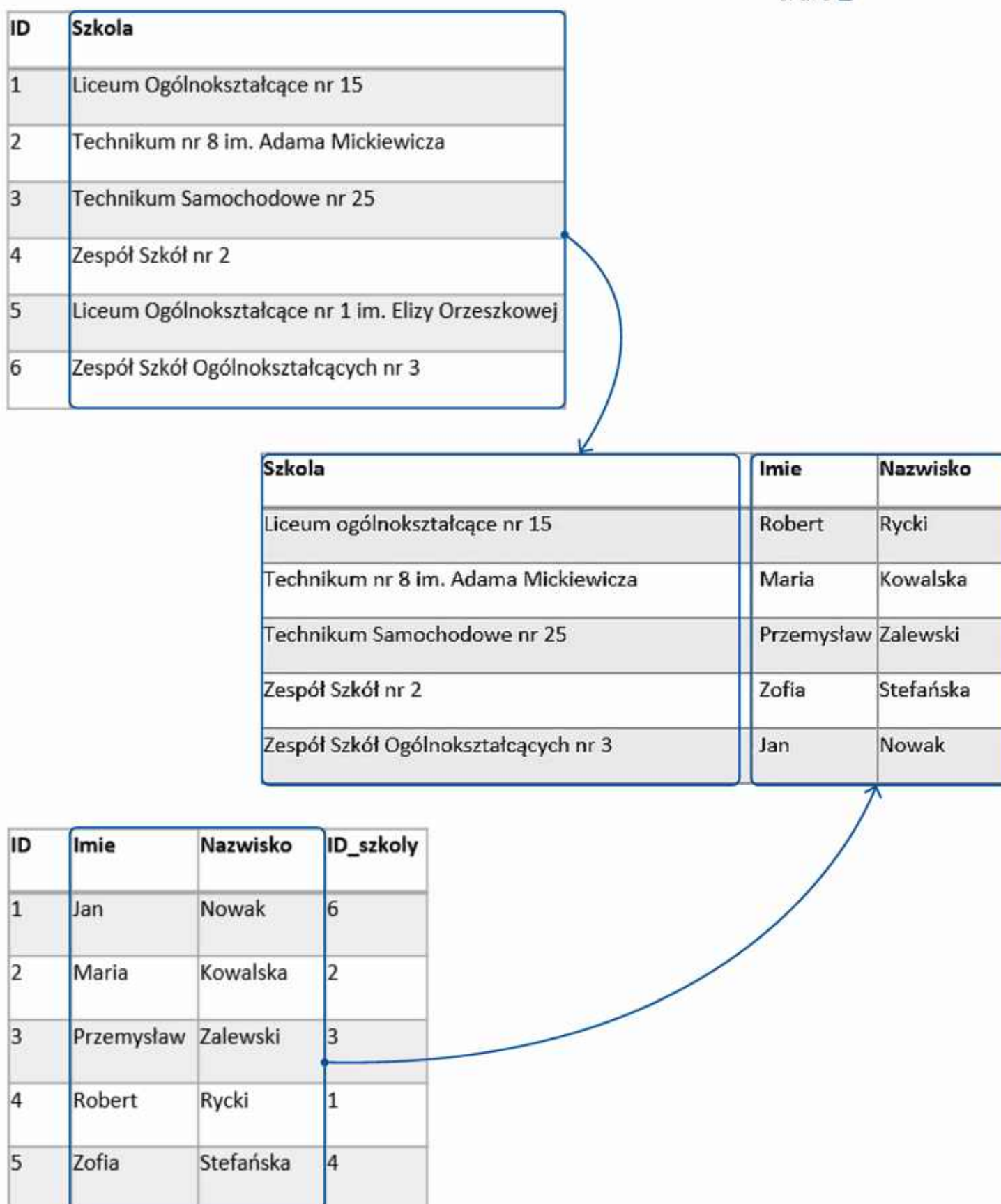
ID	Imie	Nazwisko	ID_szkoły	ID	Szkola
1	Jan	Nowak	6	1	Liceum Ogólnokształcące nr 15
2	Maria	Kowalska	2	2	Technikum nr 8 im. Adama Mickiewicza
3	Przemysław	Zalewski	3	3	Technikum Samochodowe nr 25
4	Robert	Rycki	1	4	Zespół Szkół nr 2
5	Zofia	Stefańska	4	5	Liceum Ogólnokształcące nr 1 im. Elizy Orzeszkowej
				6	Zespół Szkół Ogólnokształcących nr 3

Rys. 16.3. Związek między rekordami dwóch tabel

Zapytanie do bazy danych • W dalszej części tego tematu skupimy się na omówieniu **zapytań do bazy danych**. Pozwalają one tworzyć bazy danych, dodawać do nich tabele, a do tabel rekordy oraz wykonywać wiele innych operacji, w szczególności wybierać z całej bazy potrzebne informacje.

Wynikiem takiego wyboru jest tabela, która może zawierać wzajemnie się uzupełniające informacje, połączone za pomocą związków (rys. 16.4). W programie Access taką tabelę nazywaliśmy **kwerendą**.

Kwerenda,
s. 270 [🔗](#)



Rys. 16.4. Wynik zapytania do bazy danych w modelu relacyjnym

Zapytania budujemy z **klauzul**, które tworzymy ze słów kluczowych • **Klauzula** języka SQL.

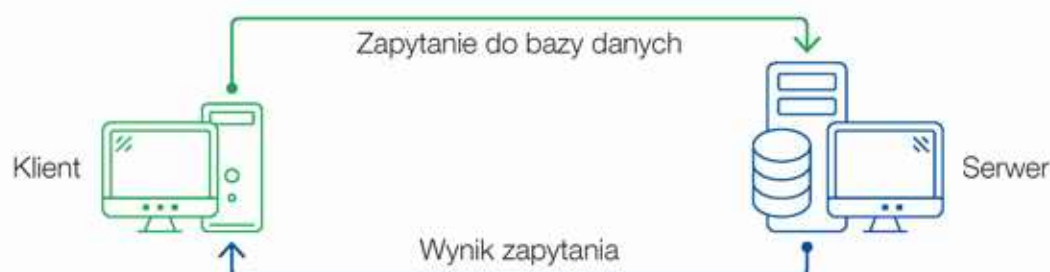
Architektura klient-serwer

Jeśli za pośrednictwem języka SQL chcemy utworzyć bazę danych, powinniśmy pamiętać, że konstruowanie poleceń języka SQL może się trochę różnić w zależności od wykorzystywanego systemu zarządzania bazą danych. Można wyróżnić dwa komponenty takiego systemu:

- ▶ standard języka SQL, który jest obsługiwany przez system,
- ▶ inne własności, w tym konfigurację serwera.

Architektura klient-serwer

Ze względu na szeroki zakres zastosowań sieciowych dużą popularnością cieszą się bazy danych pracujące w **architekturze klient-serwer**, czyli umieszczone najczęściej na odległych serwerach i komunikujące się z klientem (aplikacją) za pośrednictwem internetu. Systemy pracujące w ten sposób to m.in.: MySQL, Microsoft SQL Server, PostgreSQL, Firebird, Informix i DB2. Zasady działania bazy danych w architekturze klient-serwer wyjaśnia rysunek 16.5.



Rys. 16.5. Baza danych w architekturze klient-serwer

16.2. Tworzenie i modyfikowanie relacyjnej bazy danych

Aby poznać możliwości języka SQL oraz jego składnię, potrzebujemy dostępu do serwera baz danych. Wykorzystamy jeden z popularnych systemów – MySQL, dostępny w pakiecie XAMPP, który można pobrać ze strony apachefriends.org.

MySQL dostępny w pakiecie XAMPP to w rzeczywistości MariaDB – alternatywna wersja stworzona przez twórców MySQL. Rozwijana jest niezależnie, od kiedy oryginalny MySQL został zakupiony przez firmę Oracle.

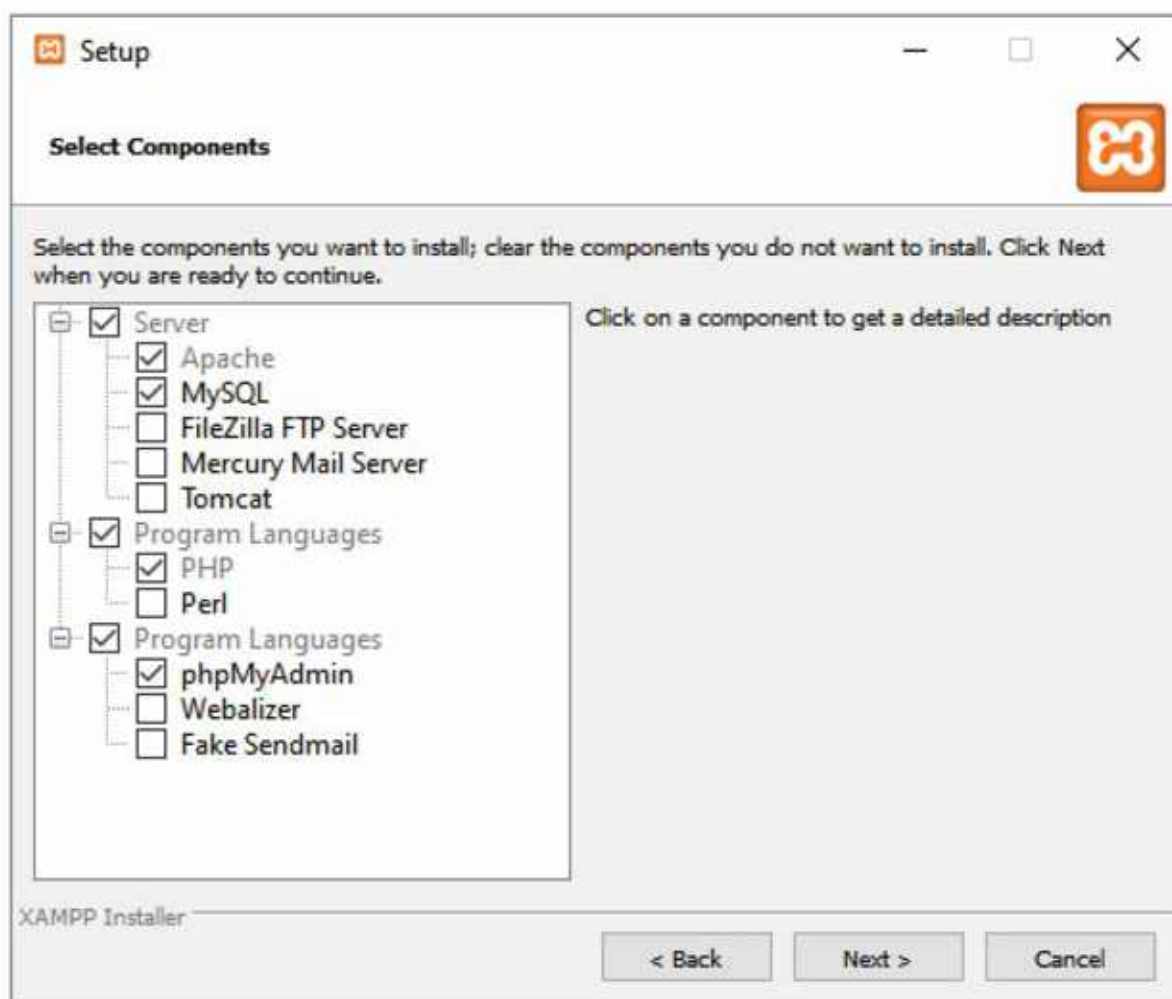
Podczas instalowania pakietu pamiętajmy, aby na etapie wyboru komponentów zaznaczyć MySQL, a także narzędzie phpMyAdmin, które posłuży do obsługi bazy danych (rys. 16.6).

Serwer stron internetowych Apache jest podstawowym składnikiem pakietu XAMPP i nie można z niego zrezygnować podczas wyboru komponentów. Dzięki niemu możemy uruchomić narzędzie phpMyAdmin, które w gruncie rzeczy jest właśnie stroną internetową.

W pakiecie XAMPP instalowany jest również interpreter języka skryptowego PHP. Język ten jest powszechnie używanym narzędziem do tworzenia aplikacji internetowych obsługujących bazy danych, a jego podstawy zostaną omówione w temacie 18.

Warto wiedzieć

W omówionym przykładzie serwer MySQL instalowany jest lokalnie (czyli na naszym komputerze). Jednak w komercyjnych zastosowaniach bazy danych zawsze znajdują się w zewnętrznych i specjalnie do tego przygotowanych magazynach danych.



Rys. 16.6. Wybór komponentów pakietu XAMPP

Po zainstalowaniu pakietu otwieramy panel kontrolny (XAMPP Control Panel) i korzystając z przycisków **Start**, włączamy serwery Apache i MySQL. Zielone podświetlenie wskazuje, że elementy te zostały uruchomione prawidłowo (rys. 16.7).



Rys. 16.7. Uruchomione serwery Apache i MySQL

Możemy teraz przejść do konsoli phpMyAdmin, czyli narzędzia do tworzenia baz danych oraz zarządzania bazami dostępnymi na serwerze i modyfikowania ich zawartości. Aby uruchomić tę konsolę, w pasku adresu przeglądarki internetowej wpisujemy „localhost”, a następnie wybieramy **phpMyAdmin**.

Warto wiedzieć

Fraza „localhost” to nazwa sieciowa komputera, na którym aktualnie pracujemy. Ma on też swój adres IP, który zawsze jest taki sam: 127.0.0.1. Można więc zamiast nazwy wpisać w przeglądarce adres IP i w ten sposób przejść do zawartości zainstalowanego serwera.

Dobra rada

Możesz nadać bazie danych nazwę zawierającą spację i polskie znaki, jednak nie jest to zalecane. Jeśli zechcesz użyć takiej bazy w aplikacji internetowej, możesz mieć problem z dostępem do niej.

Tworzenie bazy danych

Rozpocznijmy budowanie bazy danych dla zarządu zieleni miejskiej. Zgromadzimy w niej informacje o rodzajach drzew rosnących w mieście oraz o ulicach miasta.

Aby utworzyć nową bazę danych, w oknie konsoli wybieramy **Utwórz bazę danych**. Nadajemy jej nazwę *zarzad_zieleni*. Wybieramy sposób kodowania znaków **utf8_polish_ci** i zatwierdzamy przyciskiem **Utwórz** (rys. 16.8).

Baza danych	Metoda porównywania napisów	Działanie
☐ information_schema	utf8_general_ci	Sprawdź uprawnienia
☐ mysql	utf8mb4_general_ci	Sprawdź uprawnienia
☐ performance_schema	utf8_general_ci	Sprawdź uprawnienia
☐ phpmyadmin	utf8_bin	Sprawdź uprawnienia

Ogółem: 4

Rys. 16.8. Tworzenie nowej bazy danych w konsoli phpMyAdmin

Bazę danych możemy też utworzyć w inny sposób – przejść do zakładki **SQL** i wpisać zapytanie:

Zapytanie tworzące bazę danych

```
1. CREATE DATABASE zarzad_zieleni
2. CHARACTER SET utf8
3. COLLATE utf8_polish_ci;
```

Klauzula CREATE DATABASE ➤ W linii 1 zapytania, po **klauzuli** **CREATE DATABASE**, wpisujemy nazwę

Klauzula CHARACTER SET ➤ bazy danych. W linii 2 używamy **klauzuli** **CHARACTER SET**, aby ustawić

Klauzula COLLATE ➤ kodowanie znaków na UTF-8. W linii 3 korzystamy z **klauzuli** **COLLATE**, która dla ustawionego kodowania uaktywnia porównywanie znaków dla języka polskiego. Dzięki temu w bazie danych będą prawidłowo wyświetlane polskie znaki diakrytyczne.

Dobra rada

Zapytania wpisane w oknie zapytań możesz też wykonać, używając skrótu klawiszowego **Ctrl + Enter**.

Każde zapytanie można wpisać w jednej linii lub w wielu (jak w powyższym przykładzie), a średnik wskazuje jego zakończenie. Po średniku możemy umieścić kolejne zapytania i wykonać wiele zapytań jednocześnie – naciskamy w tym celu przycisk **Wykonaj**.

Po utworzeniu bazy danych klikamy na jej nazwę widoczną w panelu po lewej stronie, aby uaktywnić bazę. Możemy też użyć do tego zapytania:

Zapytanie uaktywniające bazę danych

```
USE zarzad_zieleni;
```

W tym zapytaniu została użyta **klauzula** `USE`, służąca do wskazania aktywnej bazy danych, czyli tej, dla której będą realizowane kolejne zapytania.

Ćwiczenie 1

Korzystając z pakietu XAMPP, zainstaluj serwery Apache i MySQL oraz konsolę phpMyAdmin. Uruchom zainstalowane serwery, a następnie utwórz bazę danych o nazwie *zarzad_zieleni*. Ustaw w niej kodowanie polskich znaków `utf8_polish_ci`. Uaktywnij utworzoną bazę danych.

Bazy danych w konsoli

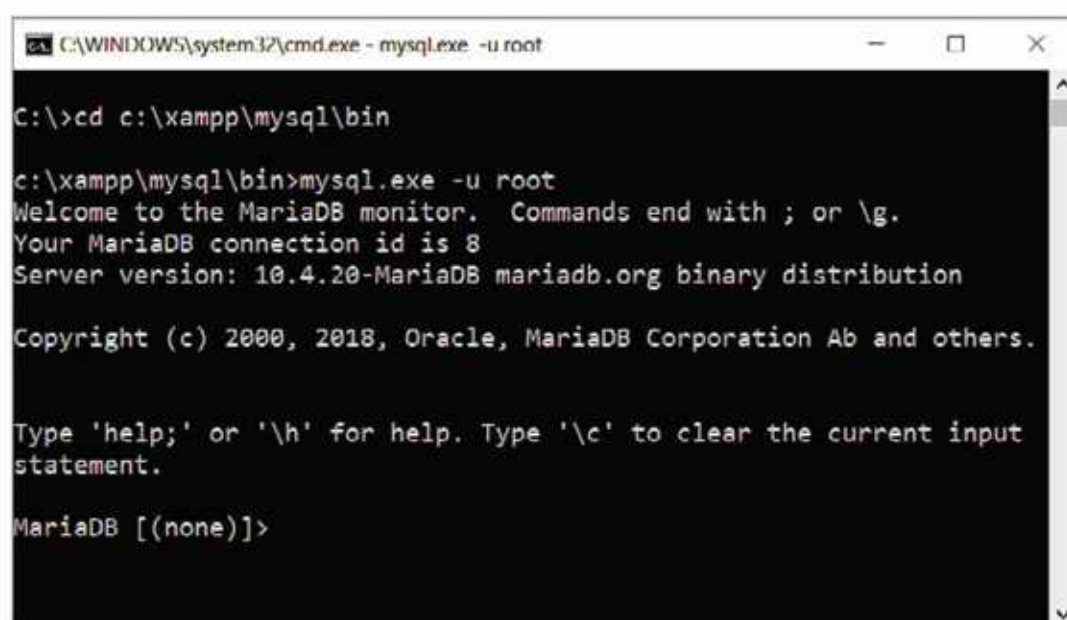
Do wpisywania zapytań w języku SQL nie musimy wykorzystywać konsoli phpMyAdmin. Możemy użyć np. konsoli systemowej. W przypadku systemu Windows będzie to Wiersz poleceń. Po jego uruchomieniu trzeba przejść do folderu zawierającego aplikację `mysql.exe`. W tym celu wpisujemy poniższe polecenie i naciskamy klawisz **Enter**.

```
cd C:\xampp\mysql\bin
```

Następnie wpisujemy nazwę programu, a po nim parametr `-u` i nazwę użytkownika:

```
mysql.exe -u root
```

Domyślnym użytkownikiem jest właśnie *root*, bez ustawionego hasła. Po uruchomieniu programu możemy już tworzyć zapytania (rys. 16.9).



```

C:\WINDOWS\system32\cmd.exe - mysql.exe -u root

C:\>cd c:\xampp\mysql\bin

c:\xampp\mysql\bin>mysql.exe -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 10.4.20-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>

```

Rys. 16.9. Dostęp do serwera baz danych przez konsolę systemową

Podobnie jak w phpMyAdmin zapytania w konsoli systemowej można wpisywać w wielu wierszach. Dopiero wprowadzenie średnika i naciśnięcie klawisza **Enter** spowoduje wykonanie zapytania.

Dobra rada

Jeśli korzystasz z systemu Linux lub macOS, do wpisywania poleceń języka SQL możesz użyć aplikacji Terminal.

Dobra rada

Aby podczas pracy w konsoli wyświetlić listę dostępnych poleceń, wpisz znak zapytania.

Dodawanie tabel do bazy danych

Na początku utworzymy dwie tabele. Tabela drzewa będzie zawierała gatunki i rodzaje drzew, a tabela ulice – nazwy ulic (rys. 16.10).

Tabela drzewa			Tabela ulice	
IDD	Gatunek	Rodzaj	IDU	Ulica
1	świerk	iglaste	1	Piaskowa
2	klon	liściaste	2	Kwiatowa
3	topola	liściaste	3	Transportowa
4	sosna	iglaste	4	Szeroka
5	brzoza	liściaste	5	Kręta
6	buk	liściaste	6	Długa
7	jesion	liściaste		
8	olsza	liściaste		
9	lipa	liściaste		
10	modrzew	iglaste		

Rys. 16.10. Dane w tabelach drzewa i ulice

Aby utworzyć te dwie tabele, wprowadzamy zapytania:

Zapytanie tworzące tabele

1. `CREATE TABLE drzewa (IDD integer PRIMARY KEY AUTO_INCREMENT,`
2. `Gatunek varchar(30), Rodzaj varchar(30));`
3. `CREATE TABLE ulice (IDU integer PRIMARY KEY AUTO_INCREMENT,`
4. `Ulica varchar(30));`

Klauzula CREATE TABLE

Instrukcja tworzenia nowej tabeli zaczyna się od **klauzuli** `CREATE TABLE` (linia 1). Składnia tej instrukcji wygląda następująco:

`CREATE TABLE nazwa tabeli (atrybuty i typy danych);`

Użyta w linii 1 **klauzula** `PRIMARY KEY` wskazuje, że atrybut `IDD` będzie w tej tabeli **kluczem głównym**. Do kolumny `IDD` będziemy mogli wpisać tylko liczby typu całkowitego (`integer`). Jeśli będziemy tworzyć nowy rekord i nie wpisujemy ręcznie żadnej wartości w tym polu, to MySQL uzupełni je automatycznie kolejną dostępną liczbą całkowitą dzięki zastosowaniu **klauzuli** `AUTO_INCREMENT` – jest to tzw. autoinkrementacja.

Zapisy w linii 2 wskazują, że do kolumn `Gatunek` i `Rodzaj` będziemy mogli wprowadzić napisy (`varchar`). W nawiasach podaliśmy maksymalną długość tekstów, jakie można wpisać jako wartości tych atrybutów.

Średnik na końcu linii 2 kończy pierwszą instrukcję. W liniach 3–4 znajduje się instrukcja tworząca drugą tabelę.

Ćwiczenie 2

W bazie danych z ćwiczenia 1 utwórz dwie tabele: `drzewa` i `ulice`. Powinny one mieć atrybuty takie jak na rysunku 16.10.

• **Klauzula** `PRIMARY KEY`

Klucz główny,
s. 257 

• **Klauzula** `AUTO_INCREMENT`

Warto wiedzieć

W systemie MySQL każdą instrukcję kończymy średnikiem, jednak nie każdy system tego wymaga. Średnik nie jest konieczny np. w systemie Microsoft SQL Server.

Dodawanie rekordów do tabel

Kiedy tabele są już gotowe, trzeba uzupełnić je rekordami. Możemy dodać je ręcznie lub zaimportować z zewnętrznego pliku. Importowanie pokażemy w dalszej części tematu, a na razie skupimy się na ręcznym dodawaniu rekordów do tabeli. W tabeli `drzewa` umieścimy gatunki i rodzaje drzew.

Poniższe zapytanie wprowadzi do tabeli trzy pierwsze gatunki drzew:

```
1. INSERT INTO drzewa (Gatunek, Rodzaj) VALUES
2. ("świerk", "iglaste"),
3. ("klon", "liściaste"),
4. ("topola", "liściaste");
```

• **Zapytanie dodające rekordy do tabeli**

W linii 1 po **klauzuli** `INSERT INTO` wpisujemy nazwę tabeli, a następnie w nawiasie nazwy atrybutów, do których zostaną przyporządkowane wartości. Całkowicie pomijamy tutaj atrybut `IDD`, ponieważ podczas tworzenia tabeli ustawiliśmy mu właściwość autoinkrementacji za pomocą **klauzuli** `AUTO_INCREMENT`.

Na końcu linii 1 znajduje się **klauzula** `VALUES`, po której – również w nawiasie – wpisujemy wartości. Wartości muszą być wpisane w takiej kolejności, w jakiej wcześniej wpisaliśmy nazwy atrybutów. Możemy uzupełnić jednocześnie dowolną liczbę rekordów tabeli. Kolejne rekordy oddzielamy od siebie przecinkiem (linie 2–4).

• **Klauzula** `INSERT INTO`

• **Klauzula** `VALUES`

Ćwiczenie 3

Do tabel `drzewa` i `ulice` w bazie danych z ćwiczenia 2 dodaj gatunki i rodzaje drzew oraz nazwy ulic zgodnie z rysunkiem 16.10.

Dobra rada

Koniec i początek łańcucha znaków w zapytaniach możesz oznaczać zarówno cudzysłowem, jak i apostrofem.

 Zapamiętaj

Zapytania w języku SQL budujemy, wykorzystując klauzule złożone ze słów kluczowych tego języka. Do tworzenia tabeli służy klauzula `CREATE TABLE`, a do dodawania rekordów do tabeli – klauzula `INSERT INTO`. Każde zapytanie języka SQL kończymy średnikiem.

Wyświetlanie zawartości tabeli

O wyświetlaniu informacji zawartych w bazie danych więcej powiemy w kolejnym temacie. Warto jednak już na tym etapie poznać podstawowe zapytanie wyświetlające wszystkie rekordy pojedynczej tabeli:

Zapytanie wyświetlające wszystkie dane z tabeli drzewa

```
1. SELECT *
2. FROM drzewa;
```

- Klauzula SELECT** • W pierwszej linii korzystamy z **klauzuli SELECT** – pozwala ona wybrać atrybuty, które chcemy wyświetlić. Gwiazdka w tym przypadku
- Klauzula FROM** • oznacza wszystkie atrybuty wybranej tabeli. Za pomocą **klauzuli FROM** w linii 2 wskazujemy tabelę, której rekordy chcemy wyświetlić.

 Dobra rada

Dane do tabel bazy danych możesz też zaimportować z pliku pobranego z serwisu, który udostępnia zbiory danych. Przykładem takiego serwisu jest strona Głównego Urzędu Statystycznego: stat.gov.pl.

Importowanie danych do tabeli

Dane do bazy danych możemy zaimportować np. z zewnętrznego pliku tekstowego. Taki plik może być wynikiem eksportu z innego systemu zarządzania bazą danych. Dzięki importowaniu unikamy czasochłonnego ręcznego wpisywania danych do tabeli.

Aby zaimportować dane z pliku tekstowego, najpierw kopiujemy ten plik do folderu zawierającego bazę danych. W naszym przypadku jest to `C:\xampp\mysql\data\zarzad_zieleni`. Dla pliku o nazwie `ulice.txt` zapytanie będzie wyglądało następująco:

Zapytanie importujące dane z pliku tekstowego do bazy danych

```
1. LOAD DATA INFILE 'ulice.txt' INTO TABLE ulice
2. CHARACTER SET cp1250
3. LINES TERMINATED BY '\r\n';
```

- Klauzula LOAD DATA INFILE** • W linii 1 tego zapytania korzystamy z **klauzuli LOAD DATA INFILE**, po której w apostrofach wpisujemy nazwę pliku. W tej samej linii używamy **klauzuli INTO TABLE**, aby wskazać, do której tabeli importujemy dane. Za pomocą **klauzuli CHARACTER SET** (linia 2) określamy, w jaki sposób znaki są kodowane w dokumencie tekstowym. W linii 3 z użyciem **klauzuli LINES TERMINATED BY** wpisujemy kod ASCII znaku końca wiersza. Jeśli pierwszy wiersz dokumentu zawiera nagłówki kolumn, możemy je pominąć podczas importowania – służy do tego **klauzula IGNORE n LINES** (n – liczba wierszy do pominięcia).

Ćwiczenie 4

Do tabeli `ulice` w bazie danych z ćwiczenia 3 zaimportuj listę ulic otrzymaną od nauczyciela (np. w pliku `ulice.txt`). Sprawdź, czy dane zostały dodane prawidłowo.

W danych wprowadzonych ręcznie lub zaimportowanych z pliku mogą się pojawić błędy. Dlatego warto wiedzieć, jak poprawiać lub usuwać błędnie wprowadzone rekordy oraz jak usuwać tabele, a nawet całe bazy danych.

Edycja rekordów

W naszej bazie danych umieściliśmy ulicę Baśniową zamiast Bajkowej. Poprawimy ten błąd, wykorzystując **klauzulę** `UPDATE`, aby wprowadzić zmiany w wybranym rekordzie.

```
1. UPDATE ulice
2. SET Ulica="Bajkowa"
3. WHERE Ulica="Baśniowa";
```

Warto wiedzieć

Konsola phpMyAdmin oferuje również wizualne narzędzie do importowania danych z pliku. Można z niego skorzystać po przejściu do zakładki **Import**.

• Zapytanie modyfikujące rekord w tabeli

W linii 1 po klauzuli `UPDATE` wpisujemy nazwę tabeli, w której chcemy wprowadzić zmiany. W linii 2 za pomocą **klauzuli** `SET` wybieramy atrybut i przypisujemy mu wartość. W linii 3 używamy **klauzuli** `WHERE`, aby wskazać, że zmiany mają objąć tylko te rekordy, których atrybut `Ulica` ma wartość `Baśniowa`. Wynik takiego zapytania widoczny jest na rysunku 16.11.

IDU	Ulica
25	Baśniowa

→

IDU	Ulica
25	Bajkowa

Rys. 16.11. Wynik zapytania zmieniającego pojedynczy rekord tabeli

Usuwanie rekordów

W naszej bazie danych przez przypadek umieściliśmy ulicę Nową, której nie ma w mieście. Usuniemy ją, wykorzystując **klauzulę** `DELETE FROM`.

```
1. DELETE FROM ulice
2. WHERE Ulica="Nowa";
```

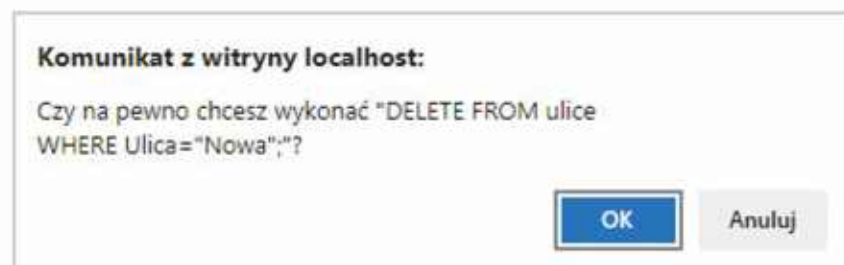
• Zapytanie usuwające rekord z tabeli

Po klauzuli `DELETE FROM` wprowadzamy nazwę tabeli, z której chcemy usunąć rekord (linia 1). W linii 2 za pomocą **klauzuli** `WHERE` określamy filtr, który wskaże rekordy do usunięcia. W naszym przypadku zostanie usunięty rekord, w którym atrybut `Ulica` ma wartość `Nowa`. Zwróć uwagę, że rekordów spełniających warunek może być więcej – wtedy wszystkie zostaną usunięte.

Dobra rada

Zapytania usuwające rekordy z tabeli wpisuj bardzo uważnie, ponieważ wykonanych zapytań nie można już cofnąć.

Przed usunięciem rekordu serwer zapyta, czy na pewno chcemy wykonać to zapytanie (rys. 16.12).



Rys. 16.12. Potwierdzenie usunięcia rekordu z tabeli

Usuwanie tabel i bazy danych

Zdarza się, że chcemy skasować tabelę lub całą bazę danych. Do usuwania tabeli służy **klauzula DROP TABLE**. Jej postać ogólna jest następująca:

```
DROP TABLE nazwa_tabeli;
```

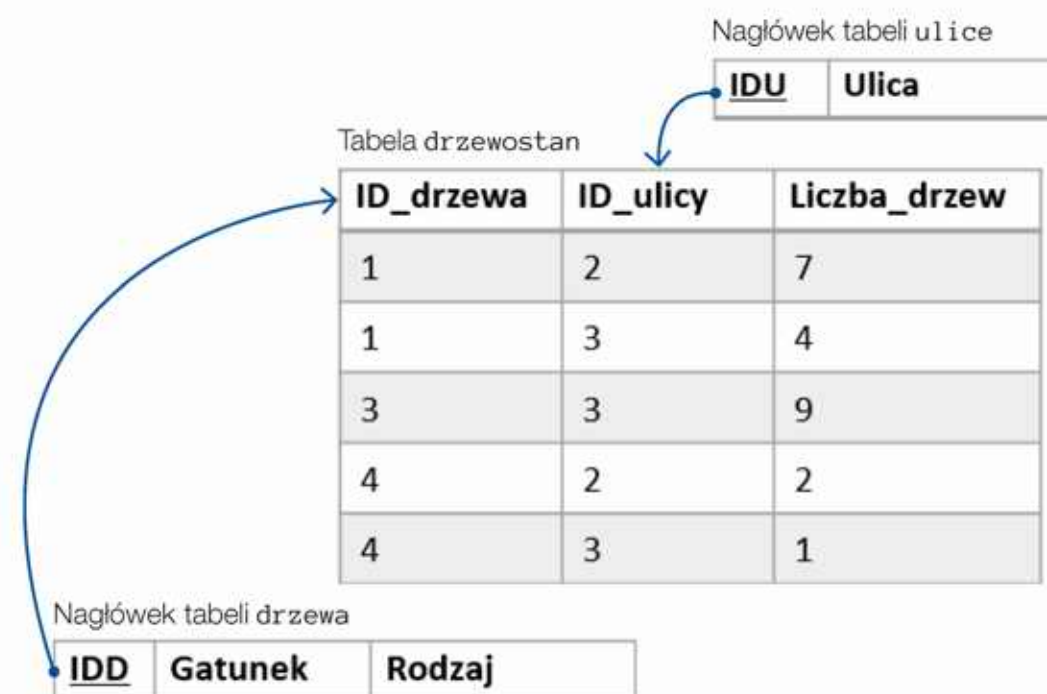
Klauzula DROP DATABASE Do usuwania całej bazy danych używamy **klauzuli DROP DATABASE** w następujący sposób:

```
DROP DATABASE nazwa_bazy_danych;
```

Tworzenie związków między tabelami

Utworzone przez nas tabele nie są ze sobą powiązane żadnym kluczem. Chcielibyśmy jednak dodać informację o tym, przy jakich ulicach rosną określone drzewa i ile ich jest. W tym celu wprowadzamy dodatkową tabelę drzewostan, która będzie **tabelą pomostową** pozwalającą na utworzenie związku typu wiele do wielu między tabelami drzewa i ulice (rys. 16.13).

Tabela pomostowa,
s. 277 [↗](#)



Rys. 16.13. Przykładowe dane tabeli drzewostan i związki między tabelami w bazie danych

Do utworzenia tabeli drzewostan użyjemy poniższej instrukcji. Aby była bardziej czytelna, rozbijemy ją na 5 linii:

```
1. CREATE TABLE drzewostan(
2. ID_drzewa integer, ID_ulicy integer, Liczba_drzew integer,
3. FOREIGN KEY (ID_drzewa) REFERENCES drzewa (IDD),
4. FOREIGN KEY (ID_ulicy) REFERENCES ulice (IDU)
5. );
```

🔗 Zapytanie tworzące tabelę pomostową

W linii 1 widzimy znane już słowo kluczowe `CREATE TABLE`, a po nim nazwę tworzonej tabeli. W tej samej linii utworzyliśmy nawias, w którym umieściliśmy atrybuty tabeli (linia 2). Ostatni z atrybutów (`Liczba_drzew`) będzie przechowywał informacje o liczbie drzew rosnącej przy danej ulicy.

W linii 3 znajduje się **klauzula** `FOREIGN KEY`, określająca, że atrybut `ID_drzewa` jest kluczem obcym tej tabeli. Użycie **klauzuli** `REFERENCES` łączy ten klucz obcy z tabelą `drzewa`, wskazując jednocześnie na jej klucz główny, czyli `IDD`.

W linii 4 ustawiliśmy `ID_drzewa` jako drugi klucz obcy i połączyliśmy go z tabelą `ulice`, wskazując na jej klucz główny, czyli `IDU`.

Jeśli przejdziemy w phpMyAdmin do zakładki **Widok projektu**, możemy podejrzeć związki między tabelami (rys. 16.14).



Rys. 16.14. Związki między tabelami widoczne w zakładce **Widok projektu** narzędzia phpMyAdmin

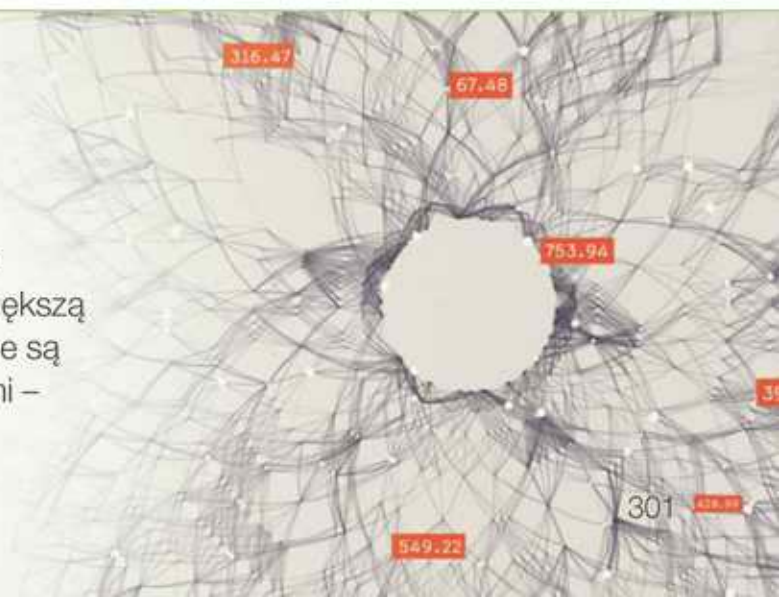
Ćwiczenie 5

Do bazy danych z ćwiczenia 4 dodaj tabelę `drzewostan`, uwzględniając związki z innymi tabelami zgodnie z rysunkiem 16.13. Zaimportuj do tabeli dane, które otrzymasz od nauczyciela (np. w pliku `drzewostan.txt`).

A to ciekawe

Bazy danych bez relacji

W ostatnich latach systematycznie wzrasta znaczenie nierelacyjnych baz danych (NoSQL). Wśród nich największą popularność zdobywają bazy grafowe, w których dane są reprezentowane przez węzły, a połączenia między nimi – przez krawędzie. Takie bazy są bardzo elastyczne.



Integralność danych

Integralność referencyjna danych

Dzięki temu, że klucze obce tabeli `drzewostan` wskazują na klucze główne tabel `drzewa` i `ulice`, uzyskaliśmy **integralność referencyjną danych**. Oznacza to, że jeśli spróbujemy utworzyć w tabeli `drzewostan` rekord, którego atrybut `ID_drzewa` ma wartość niewystępującą jako atrybut `IDD` w tabeli `drzewa`, to taka operacja się nie powiedzie. Podobnie nie uda się utworzyć w tabeli `drzewostan` rekordu, którego atrybut `ID_ulicy` ma wartość niewystępującą jako atrybut `IDU` w tabeli `ulice`. Integralność sprawia też, że nie można usunąć rekordu, który jest połączony z rekordem z innej tabeli. Wpiszmy np. zapytanie usuwające świerk z tabeli `drzewa`:

Zapytanie usuwające świerk z tabeli drzewa

```
1. DELETE FROM drzewa
2. WHERE Gatunek="Świerk";
```

Zapytanie się nie wykona, a my zobaczymy komunikat taki jak na rysunku 16.15.

```
#1451 - Cannot delete or update a parent row: a foreign
key constraint fails (`zarzad_zieleni`.`drzewostan`,
CONSTRAINT `drzewostan_ibfk_1` FOREIGN KEY (`ID_drzewa`)
REFERENCES `drzewa` (`IDD`))
```

Rys. 16.15. Odmowa usunięcia rekordu połączanego z rekordem z innej tabeli bazy danych

Usunięcie tabeli złączonej z inną również nie będzie możliwe. Trzeba jednak pamiętać, że integralność nie chroni przed usunięciem całej bazy danych.

16.3. Zabezpieczanie bazy danych

Powiedzieliśmy już, że domyślnym użytkownikiem serwera baz danych jest *root* i nie ma on ustawionego hasła. Nie stanowi to problemu, jeżeli baza danych jest używana tylko na naszym lokalnym komputerze. Jeśli jednak zamierzamy wykorzystać ją w aplikacji sieciowej, która będzie dostępna w internecie, powinniśmy taką bazę zabezpieczyć, tworząc dla każdego użytkownika konto chronione hasłem i przydzielając mu uprawnienia do wybranej bazy danych.

Tworzenie konta i przydzielanie uprawnień

Utworzymy konto dla użytkownika *admin* i ustalimy mu **hasło** (warto skorzystać z zaleceń zawartych w części pierwszej podręcznika). Hasło będzie brzmiało: `@dm1n1Str@t0r`. Użytkownikowi przydzielimy również uprawnienia do bazy danych *zarzad_zieleni*. Aby to zrobić, otwieramy konsolę `phpMyAdmin` lub logujemy się przez konsolę systemową na konto *root* i wpisujemy następujące zapytania.

Zasady tworzenia i używania haseł, podręcznik Informatyka na czasie 1. Zakres rozszerzony, s. 22 

```

1. CREATE USER 'admin' @'localhost'
2. IDENTIFIED BY '@dm1n1Str@t0r';
3. GRANT ALL PRIVILEGES
4. ON zarzad_zieleni.*
5. TO 'admin' @'localhost';

```

🔹 Zapytania tworzące użytkownika i przydzielające mu uprawnienia

W linii 1 wykorzystujemy **klauzulę** `CREATE USER` do utworzenia konta. Po niej wpisujemy nazwę użytkownika w apostrofach i po znaku `@` nazwę serwera. W linii 2 za pomocą **klauzuli** `IDENTIFIED BY` ustawiamy hasło dla nowego użytkownika. **Klauzula** `GRANT ALL PRIVILEGES` w linii 3 rozpoczyna drugie zapytanie, które udziela pełnego dostępu do wszystkich tabel bazy danych *zarzad_zieleni* (linia 4) użytkownikowi *admin* (linia 5).

🔹 **Klauzula** `CREATE USER`
 🔹 **Klauzula** `IDENTIFIED BY`
 🔹 **Klauzula** `GRANT ALL PRIVILEGES`

Jeśli chcemy sprawdzić, czy konto nowego użytkownika działa, logujemy się przez konsolę systemową (rys. 16.16).

```

C:\WINDOWS\system32\cmd.exe - mysql.exe -u admin -p

c:\xampp\mysql\bin>mysql.exe -u admin -p
Enter password: *****
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 93
Server version: 10.4.20-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>

```

📄 Warto wiedzieć

Uruchomienie aplikacji `mysql.exe` z parametrem `-p` spowoduje, że wyświetli się monit o podanie hasła. Bez tego parametru nie można się zalogować na konto zabezpieczone hasłem.

Rys. 16.16. Logowanie do MySQL na nowe konto użytkownika

Aby zmienić hasło aktualnie zalogowanego użytkownika, korzystamy z **klauzuli** `SET PASSWORD`, której postać ogólna wygląda następująco:

🔹 **Klauzula** `SET PASSWORD`

```
SET PASSWORD = PASSWORD('nowe_haslo');
```

Zamiast `nowe_haslo` wpisujemy hasło, które chcemy ustawić.

Ćwiczenie 6

Dodaj do serwera MySQL nowego użytkownika o wybranych przez siebie nazwie i hasle. Przydziel temu użytkownikowi pełne uprawnienia do bazy danych z ćwiczenia 5. Korzystając z konsoli systemowej, zaloguj się na utworzone konto i uaktywnij bazę danych. Napisz zapytanie wyświetlające wszystkie rekordy z tabeli *drzewa*.

Podsumowanie

- W relacyjnym modelu bazy danych podstawowym pojęciem jest relacja, czyli zestawienie zbioru danych w formie tabeli.
- Dzięki swoim zastosowaniom sieciowym bardzo popularne są bazy danych w architekturze klient–serwer.
- Podstawą konstruowania zapytań w języku SQL są klauzule.
- Pomimo różnic między systemami zarządzania bazą danych podstawowa składnia języka SQL jest taka sama we wszystkich systemach.
- Serwer baz danych udostępniony w internecie powinien być zabezpieczony przed dostępem osób niepowołanych.

Zadania

- ★ **1** Jaki będzie efekt wykonania poniższych instrukcji zapisanych w języku SQL?

```
1. CREATE TABLE sklepy (IDS integer PRIMARY KEY, Sklep varchar(50));
2. INSERT INTO sklepy (IDS, Sklep) VALUES
3. (1, "sklep osiedlowy"),
4. (2, "sklep miejski"),
5. (3, "sklep wielobranżowy");
```

- ★ **2** Utwórz bazę danych dla zarządu komunikacji miejskiej. Zastanów się, jakich tabel użyjesz. Utwórz te tabele i uzupełnij je przykładowymi danymi.
- ★★ **3** Utwórz bazę danych o nazwie *biblioteka*, składającą się z dwóch tabel:
ksiązki(IDK, Tytuł, Autor)
wypożyczający(IDW, Imie, Nazwisko, ID_książki)
Atrybut ID_książki z tabeli wypożyczający powinien być kluczem obcym i wskazywać na atrybut IDK tabeli książki.
- ★★ **4** Utwórz bazę danych o nazwie *zawody*, składającą się z dwóch tabel:
zawodnicy(IDZ, Imie, Nazwisko, ID_szkoly)
szkoly(IDS, Szkola)
Atrybut ID_szkoly z tabeli zawodnicy powinien być kluczem obcym i wskazywać na atrybut IDS tabeli szkoły. Do tabel zaimportuj dane otrzymane od nauczyciela (np. z plików *zawodnicy.txt* i *szkoly.txt*).
- ★★★ **5** Utwórz bazę danych o nazwie *szkola*, składającą się z dwóch tabel:
klasy(IDK, Klasa)
nauczyciele(IDN, Imie, Nazwisko)
W jaki sposób możesz połączyć te dane, wykorzystując trzecią tabelę? Utwórz tę tabelę i związki między wszystkimi tabelami.

17. Zapytania w języku SQL

Dobrze zbudowane zapytania pomagają wydobyć z bazy danych wiele ciekawych informacji. Jeśli znasz strukturę bazy, możesz wybrać z niej dokładnie takie dane, jakich potrzebujesz, niezależnie od liczby tabel czy rekordów w poszczególnych tabelach. W tym temacie wykorzystamy zapytania języka SQL do przeszukiwania bazy danych.

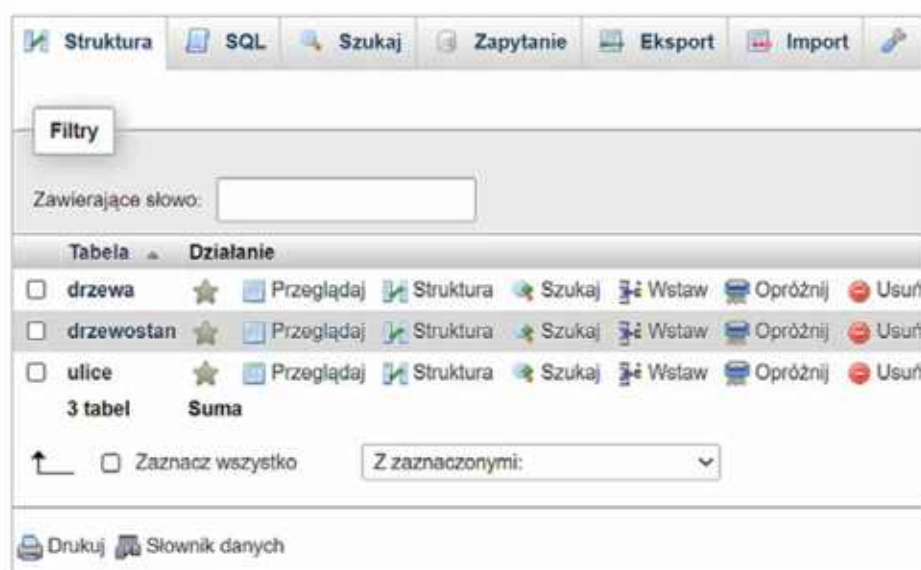
Cele lekcji

- Nauczysz się konstruować zapytania w języku SQL zwracające określone dane.
- Dowiesz się, jak zapisywać operacje selekcji, projekcji i złączenia.
- Poznasz notację opisującą działania na wielu tabelach i wykorzystującą złączenia między nimi.
- Zobaczysz, w jaki sposób język SQL jest wykorzystywany w kwerendach programu MS Access.

Do zilustrowania omawianych zagadnień wykorzystamy bazę danych *zarzad_zieleni*, utworzoną w temacie 16 w ćwiczeniach 1–5. Składa się ona z dwóch tabel: *drzewa* oraz *ulice*. Zbudowaliśmy w niej również tabelę *drzewostan*, która łączy ze sobą informacje z obu tabel i pokazuje, jakie gatunki drzew rosną przy poszczególnych ulicach miasta. Wszystkie ćwiczenia w tym temacie wykonuj na tej bazie.

17.1. Zapytania zwracające dane

Zawartość tabel bazy danych możemy wyświetlić za pomocą narzędzi dostępnych w phpMyAdmin. W tym celu wystarczy przejść do zakładki **Struktura** i wybrać interesującą nas tabelę (rys. 17.1).



Rys. 17.1. Spis tabel bazy danych widoczny w konsoli phpMyAdmin

Dobra rada

W konsoli phpMyAdmin w zakładce **Zapytanie** znajdziesz narzędzie, które pomoże ci utworzyć zapytanie do bazy danych. Składowe zapytania wybierzesz z listy rozwijanej.

My jednak do wyświetlenia zawartości bazy chcemy wykorzystać język SQL. Użyjemy w tym celu zapytania rozpoczynającego się od klauzuli **SELECT** – pozwala ona wskazać dane, które chcemy zobaczyć, np.:

**Zapytanie wyświetlające
wszystkie rekordy
z tabeli drzewa**

```
1. SELECT *
2. FROM drzewa;
```

Takiego zapytania użyliśmy już w poprzednim temacie do wyświetlenia wszystkich rekordów z tabeli drzewa. Po wykonaniu zapytania przyciskiem **Wykonaj** (lub skrótem **Ctrl + Enter**) zobaczymy wynik (rys. 17.2).

IDD	Gatunek	Rodzaj
1	świerk	iglaste
2	klon	liściaste
3	topola	liściaste
4	sosna	iglaste
5	brzoza	liściaste
6	buk	liściaste
7	jesion	liściaste
8	olsza	liściaste
9	lipa	liściaste
10	modrzew	iglaste

Rys. 17.2. Wynik zapytania wyświetlającego wszystkie rekordy z tabeli drzewa

Ćwiczenie 1

Napisz zapytanie do bazy danych *zarzad_zieleni* wyświetlające wszystkie rekordy z tabeli drzewa.

Selekcja wyników zapytania

**Selekcja wyników
zapytania**

Co zrobić, gdy chcemy np. wyświetlić wszystkie drzewa iglaste? Wykorzystujemy wtedy możliwość **selekcji wyników zapytania**, czyli takiego ich filtrowania, aby zostały wyświetlone tylko te rekordy, których wybrany atrybut ma określoną wartość. Do selekcji wyników zapytania służy klauzula **WHERE**.

**Zapytania
uwzględniające
selekcję wyników
zapytania**

```
1. SELECT *
2. FROM drzewa
3. WHERE Rodzaj="iglaste";
```

Klauzula **WHERE** (linia 3) informuje o tym, jaki atrybut danej tabeli chcemy filtrować (Rodzaj) oraz jaka wartość nas interesuje (**iglaste**). Jest więc w tym miejscu sprawdzany pewien warunek, a w wynikach zapytania zostaną uwzględnione tylko te rekordy, które go spełniają (rys. 17.3).

IDD	Gatunek	Rodzaj
1	świerk	iglaste
4	sosna	iglaste
10	modrzew	iglaste

Rys. 17.3. Wynik zapytania uwzględniający kryteria filtrowania

Ćwiczenie 2

Napisz zapytanie do bazy danych *zarzadz_zieleni* wyświetlające wszystkie drzewa iglaste z tabeli *drzewa*.

Podczas selekcji wyników zapytania za pomocą klauzuli **WHERE** wykorzystaliśmy znak równości, który jest **operatorem porównania**. W instrukcji z klauzulą **WHERE** możemy zastosować również inne operatory porównania – przedstawia je tabela 17.1.

Operator	Opis
=	równy
>	wiekszy
>=	wiekszy lub równy
<	mniejszy
<=	mniejszy lub równy
<>	różny

Tabela 17.1. Operatory porównania stosowane w klauzuli **WHERE**

Dobra rada

W języku SQL wielkość liter nie ma znaczenia. Możesz więc w kryteriach napisać zarówno „iglaste”, jak i „Iglaste” lub „IGLASTE”.

Warto wiedzieć

Operatory porównania są powszechnie używane w językach programowania do konstruowania instrukcji warunkowych oraz pętli.

Istnieje także **operator IS**, który w połączeniu ze **znacznikiem NULL** można wykorzystać na dwa sposoby:

- ▶ **IS NULL** – wyświetli wszystkie rekordy, w których wybrany atrybut jest pusty (nie ma żadnej wartości),
- ▶ **IS NOT NULL** – wyświetli wszystkie rekordy, w których wybrany atrybut nie jest pusty (ma jakąkolwiek wartość).

W języku SQL dostępne są **operatory arytmetyczne**: dodawanie (+), odejmowanie (−), mnożenie (*), dzielenie (/) oraz reszta z dzielenia, czyli modulo (%). Najprostszym ich zastosowaniem jest wysłanie zapytania obliczającego wyrażenie matematyczne. Można je jednak wykorzystać także do wykonywania operacji arytmetycznych na danych.

Operator logiczny • Możemy też używać **operatorów logicznych**, aby na różne sposoby łączyć ze sobą warunki. Operatory te zebrano w tabeli 17.2.

Operator	Opis
AND	i
OR	lub
NOT	nieprawda, że
BETWEEN	poędzy

Tabela 17.2. Operatory logiczne stosowane w klauzuli `WHERE`

Warto wspomnieć, że w języku SQL w wyniku działania operatorów logicznych możemy uzyskać oprócz prawdy (`true`) lub fałszu (`false`) również wartość nieznaną (`unknown`).

Wykorzystamy operatory porównania i operatory logiczne, aby z tabeli drzewa wybrać tylko drzewa iglaste o wartości IDD między 1 i 5.

Zapytanie wyświetlające •
drzewa iglaste o IDD
między 1 i 5

```
1. SELECT *
2. FROM drzewa
3. WHERE Rodzaj="iglaste"
4. AND IDD BETWEEN 1 AND 5;
```

Dobra rada

Dzięki operatorom porównania i operatorom logicznym możesz dowolnie rozbudowywać warunki. W razie potrzeby użyj nawiasów.

W linii 1 wskazujemy, że chcemy wyświetlić wszystkie atrybuty. Zapis w linii 2 oznacza, że mają to być atrybuty z tabeli drzewa.

O tym, że w wyniku zapytania mają być uwzględnione tylko drzewa iglaste, informuje instrukcja z linii 3, w której użyliśmy operatora porównania `=`. Musi być też spełniony warunek z linii 4 (z operatorem `AND`), który określa, że wartość atrybutu IDD tabeli drzewa powinna być między (`BETWEEN`) 1 i 5.

Ćwiczenie 3

Napisz zapytania do bazy danych `zarzad_zieleni` wyświetlające:

- wszystkie drzewa iglaste, których IDD zawiera się między 1 i 5,
- wszystkie ulice, których IDU zawiera się między 30 i 50.

A to ciekawe

Archiwum internetu

The Wayback Machine to ogromna baza danych, która nieprzerwanie od 1996 r. gromadzi publicznie dostępne strony internetowe, zapisując co jakiś czas ich aktualne wersje. Według danych z 2020 r. w bazie znajduje się już blisko 500 miliardów stron.

INTERNET ARCHIVE
WayBackMachine

Sortowanie wyników zapytania

Wyniki zwracane przez zapytania wyświetlane są w takiej kolejności, w jakiej zostały umieszczone w tabeli. Możemy je jednak sortować według wybranej kolumny lub nawet według kilku kolumn. Służy do tego **klauszula** `ORDER BY`.

🔗 **Klauszula** `ORDER BY`

```
1. SELECT *
2. FROM drzewa
3. WHERE Rodzaj="iglaste"
4. AND IDD BETWEEN 1 AND 5
5. ORDER BY Gatunek;
```

🔗 **Zapytanie** uwzględniające sortowanie wyników

Powyższe zapytanie dzięki klauzuli `SELECT` (linia 1) zwróci z tabeli `drzewa` (linia 2) wszystkie drzewa iglaste (linia 3), których `IDD` jest między 1 i 5 (linia 4). Dodatkowo w linii 5 wskazaliśmy, że wyniki zapytania mają być posortowane według atrybutu `Gatunek`. Wynik tego zapytania widać na rysunku 17.4.

👍 **Dobra rada**

Po przecinku możesz wpisywać kolejne atrybuty, według których chcesz sortować dane. Kolejność atrybutów decyduje o kolejności sortowania wyników.

IDD	Gatunek	Rodzaj
4	sosna	iglaste
1	świerk	iglaste

Rys. 17.4. Wynik zapytania po sortowaniu

Ćwiczenie 4

Napisz zapytanie do bazy danych `zarzadz_zieleni` wyświetlające wszystkie drzewa iglaste, których `IDD` zawiera się między 1 i 5. Dane posortuj według gatunków.

Projekcja atrybutów

Może się zdarzyć, że będziemy chcieli wyświetlić tylko wybrane atrybuty i tylko z wybranych tabel. Na przykład z tabeli `drzewa` pokażemy jedynie kolumny `Gatunek` i `Rodzaj`.

Aby wybrać atrybuty do wyświetlenia, wypisujemy je po klauzuli `SELECT`. Jest to tzw. **projekcja atrybutów**, czyli pominięcie w wynikach zapytania atrybutów niewymienionych w instrukcji.

🔗 **Projekcja atrybutów**

```
1. SELECT Gatunek, Rodzaj
2. FROM drzewa
3. WHERE Rodzaj="iglaste"
4. AND IDD BETWEEN 1 AND 5
5. ORDER BY Gatunek;
```

🔗 **Zapytanie** uwzględniające projekcję atrybutów

W linii 1 po klauzuli `SELECT` zamiast gwiazdki `*` wpisujemy nazwy atrybutów, które chcemy wyświetlić. Wynik tego zapytania jest taki sam jak w poprzednim przykładzie (rys. 17.4, s. 309), jednak zostaną w nim ukryte atrybuty niewymienione w linii 1, czyli w naszym przypadku niewidoczna będzie kolumna `IDD` (rys. 17.5).

Gatunek	Rodzaj
sosna	iglaste
świerk	iglaste

Rys. 17.5. Projekcja atrybutów `Gatunek` i `Rodzaj`

Iloczyn kartezjański

Poznaliśmy klauzule najczęściej występujące w zapytaniu `SELECT`. Jednak na razie wszystkie zapytania dotyczyły jednej tabeli. Aby zapytanie skierować do kilku tabel, trzeba te tabele wymienić po klauzuli `FROM`, oddzielone przecinkami. Wrócimy do podstawowej formy zapytania wyświetlającej wszystkie rekordy, jednak tym razem uwzględnimy obie stworzone wcześniej tabele. W wyniku takiego zapytania uzyskamy tzw.

Iloczyn kartezjański • **iloczyn kartezjański**, czyli wszystkie kombinacje rekordów z tych tabel.

Zapytanie tworzące
iloczyn kartezjański

1. `SELECT *`
2. `FROM drzewa, ulice;`

Warto wiedzieć

Nazwa „iloczyn kartezjański” pochodzi od kartezjańskiego układu współrzędnych, w którym położenie każdego punktu jest określone przez jego dwie współrzędne.

Wynikiem powyższego zapytania jest iloczyn kartezjański rekordów obu tabel wymienionych w linii 2. Jeśli obie tabele zawierają odpowiednio m i n rekordów, iloczyn kartezjański zwróci $m \cdot n$ rekordów. Ich liczba znacznie wzrasta po dołączeniu do iloczynu kartezjańskiego kolejnych tabel. Łącznie rekordów może więc być bardzo wiele.

Ćwiczenie 5

Napisz zapytanie do bazy danych `zarzad_zieleni` wyświetlające iloczyn kartezjański tabel `drzewa` i `ulice`.

Zapamiętaj

Do wyświetlenia informacji z bazy danych wykorzystuje się zapytania, które buduje się z następujących klauzul:

- `SELECT` – wybór atrybutów do wyświetlenia,
- `FROM` – wskazanie tabel, z których wybierane są atrybuty,
- `WHERE` – selekcja (filtrowanie) wyników zapytania,
- `ORDER BY` – porządkowanie (sortowanie) wyników zapytania.

Zapytania w języku SQL

Dzięki odpowiedniemu zapytaniu skierowanemu do bazy danych możemy z niej wydobyć dokładnie takie informacje, jakich potrzebujemy. Zapytanie buduje się z klauzul, które składają się ze słów kluczowych języka SQL. Klauzule zapisujemy w określonym porządku.

Przykładowe zapytanie do bazy danych

Poniższe zapytanie zwróci nazwy wszystkich tych gatunków (linia 1) z tabeli drzewa (linia 2), które są iglaste (linia 3), i posortuje je alfabetycznie (linia 4).

Klauzula **SELECT** wskazuje atrybuty, których wartości zostaną zwrócone w wyniku zapytania.

```
1 SELECT Gatunek
2 FROM drzewa
3 WHERE Rodzaj="iglaste"
4 ORDER BY Gatunek;
```

Klauzula **FROM** wskazuje tabelę lub kilka tabel, z których pobierane są wyniki.

Dzięki klauzuli **WHERE** możemy dokonać selekcji, czyli uwzględnić w wynikach tylko te rekordy, które spełniają określone kryteria.

Klauzula **ORDER BY** sortuje rekordy według wskazanego atrybutu.

Wynik zapytania

Gatunek	1
modrzew	
sosna	
świerk	

Zasady tworzenia zapytań do bazy danych

Każde zapytanie kończymy średnikiem.

Zapytanie można zapisać w jednym wierszu lub podzielić na kilka wierszy.

Aby zapytanie zwróciło wartości kilku atrybutów, oddzielamy te atrybuty przecinkami.

Wielkość liter nie ma znaczenia, ale przyjęło się zapisywać klauzule dużymi literami.

Możemy wpisać wiele zapytań i wykonać je razem.

Podczas selekcji rekordów można podać kilka kryteriów i połączyć je operatorami logicznymi, np.: **AND**, **OR**.

17.2. Zapytania wykorzystujące związki między tabelami

Nasza baza danych zawiera też tabelę `drzewostan`, która dzięki utworzonym związkom łączy ze sobą dane z tabel `drzewa` i `ulice`. Wykorzystując ten fakt, możemy przygotować zapytania wyświetlające

Klauzula `INNER JOIN` • dokładnie to, co nas interesuje. Użyjemy do tego **klauzuli `INNER JOIN`**, **Złączenie wewnętrzne** • tworzącej **złączenie wewnętrzne** między tabelami, czyli połączenie rekordów w co najmniej dwóch tabelach.

Na początek przygotujemy wykaz nazw ulic wraz z liczbą drzew. Zastosujemy zapis z kropką `tabela.Atrybut`, aby zaznaczyć, z których tabel pochodzą poszczególne atrybuty.

Zapytania wyświetlające nazwy ulic i liczbę drzew •

```
1. SELECT ulice.Ulica, drzewostan.Liczba_drzew
2. FROM ulice INNER JOIN drzewostan
3. ON ulice.IDU=drzewostan.ID_ulicy;
```

W linii 1 wskazujemy, że interesują nas atrybuty: `Ulica` z tabeli `ulice` i `Liczba_drzew` z tabeli `drzewostan`. Tabele te zostały połączone klauzulą `INNER JOIN` (linia 2). W wyniku wyszukiwania zostaną uwzględnione tylko te rekordy z obu tabel, które spełniają warunek

Klauzula `ON` • opisany po **klauzuli `ON`** (linia 3). W naszym przykładzie będą to te rekordy z tabeli `ulice`, których wartość pola `IDU` jest taka sama jak wartość pola `ID_ulicy` z tabeli `drzewostan` (rys. 17.6).

Ulica	Liczba_drzew
Piaskowa	13
Piaskowa	12
Piaskowa	20
Piaskowa	2
Piaskowa	3
Piaskowa	7
Piaskowa	17
Kwiatowa	15
Kwiatowa	6
Kwiatowa	16
Kwiatowa	19
Kwiatowa	8

Rys. 17.6. Fragment zestawienia nazw ulic i liczby drzew uwzględniającego związek między tabelami

Ćwiczenie 6

Napisz zapytanie do bazy danych `zarzadz_zieleni` wyświetlające zestawienie nazw ulic i liczby drzew. Uwzględnij związek między tabelami.

Agregowanie danych

W ostatnim zapytaniu każdy rekord z tabeli `drzewostan` był traktowany indywidualnie. Oznacza to, że ulice z różnymi gatunkami drzew pojawiły się w wyniku wielokrotnie. My chcielibyśmy jednak dowiedzieć się, ile w sumie jest drzew przy każdej ulicy. Wykorzystamy w tym celu jedną z funkcji **agregowania danych**. Do najbardziej popularnych tego typu funkcji należą:

- ▶ **SUM** – oblicza sumę wartości pól,
- ▶ **AVG** – oblicza średnią arytmetyczną wartości pól,
- ▶ **COUNT** – zwraca liczbę pól zawierających wartości różne od **NULL**,
- ▶ **MIN, MAX** – zwraca odpowiednio minimalną lub maksymalną wartość.

Wykonamy zestawienie nazw ulic z liczbą wszystkich drzew, które przy nich rosną. Dane posortujemy malejąco według liczby drzew.

```
1. SELECT ulice.Ulica, SUM(drzewostan.Liczba_drzew)
2. AS Suma_drzew
3. FROM ulice LEFT JOIN drzewostan
4. ON ulice.IDU=drzewostan.ID_ulicy
5. GROUP BY ulice.Ulica
6. ORDER BY SUM(drzewostan.Liczba_drzew) DESC, ulice.Ulica;
```

Agregowanie danych,
s. 283 [🔗](#)

🔗 Zapytanie zwracające sumę liczby drzew przy każdej ulicy

W linii 1 wybieramy atrybut `Ulica` z tabeli `ulice` oraz wykorzystujemy **funkcję agregującą** **SUM**, która zsumuje wartości pól atrybutu `Liczba_drzew` z tabeli `drzewostan` dla powtarzających się ulic. Zsumowane wartości będą widoczne w kolumnie `Suma_drzew` – taką nazwę nadajemy jej za pomocą **klauzuli** **AS** (linia 2).

🔗 Funkcja agregująca SUM

🔗 Klauzula AS

Zapis w linii 3 łączy tabelę `ulice` z tabelą `drzewostan` za pomocą **klauzuli** **LEFT JOIN**. Jest to tzw. **złączenie zewnętrzne lewe**, które w wyniku uwzględni wszystkie rekordy z tabeli występującej po lewej stronie klauzuli – nawet te, które nie spełniają warunku widocznego w linii 3. W takiej sytuacji atrybut `Suma_drzew` będzie miał wartość **NULL**. Zostaną więc wyświetlone wszystkie ulice, nawet te, przy których nie posadzono żadnych drzew.

🔗 Klauzula LEFT JOIN, złączenie zewnętrzne lewe

Można również tworzyć tzw. **złączenie zewnętrzne prawe** poprzez zastosowanie klauzuli, która działa odwrotnie: uwzględnia wszystkie rekordy z tabeli występującej po jej prawej stronie. Służy do tego **klauzula** **RIGHT JOIN**.

🔗 Złączenie zewnętrzne prawe

🔗 Klauzula RIGHT JOIN

W linii 4 wykorzystujemy **klauzulę** **GROUP BY** do łączenia rekordów o takich samych wartościach atrybutu `Ulica`.

🔗 Klauzula GROUP BY

Wyniki zapytania sortujemy najpierw według sumy liczby drzew, używając klauzuli **ORDER BY** (linia 5). Będzie to sortowanie malejące, ponieważ zastosowaliśmy **klauzulę** **DESC**. Ulice o tej samej liczbie drzew zostaną posortowane alfabetycznie według nazw.

🔗 Klauzula DESC

Wyniki tego zapytania są widoczne na rysunku 17.7 na s. 314.

Ulica	Suma_drzew
Fabryczna	144
Kwiatowa	118
Cisowa	103
Słowackiego	101
Sosnowa	101
Wierzbowa	101
Kolejowa	99
Jagodowa	97
Strumykowa	97
Krajobrazowa	96
Moniuszki	96
Fiołkowa	95
Karmelowa	95
Owsiana	94

Rys. 17.7. Fragment zestawienia nazw ulic z sumą liczby drzew

Ćwiczenie 7

Napisz zapytanie do bazy danych *zarzad_zieleni*, które wyświetli:

- zestawienie nazw ulic z sumą liczby drzew,
- zestawienie nazw drzew liściastych z ich łączną liczbą.

Dane posortuj malejąco według liczby drzew.

Klauzula HAVING •
Selekcja wyników
zapytania,
s. 306 

Mówiąc o agregowaniu danych, warto wspomnieć także o **klauzuli HAVING**. Pozwala ona na **selekcję wyników zapytania**, w którym użyto funkcji agregujących. Działa więc podobnie jak klauzula **WHERE**, z tą różnicą, że umożliwia filtrowanie już zgrupowanych wartości.

Zastosujemy klauzulę **HAVING**, aby z naszego ostatniego wyniku (rys. 17.7) wyświetlić tylko te ulice, przy których rośnie więcej niż 100 drzew.

```
1. SELECT ulice.Ulica, SUM(drzewostan.Liczba_drzew)
2. AS Suma_drzew
3. FROM ulice LEFT JOIN drzewostan
4. ON ulice.IDU=drzewostan.ID_ulicy
5. GROUP BY ulice.Ulica
6. HAVING Suma_drzew>100
7. ORDER BY SUM(drzewostan.Liczba_drzew) DESC, ulice.Ulica;
```

Klauzulę **HAVING** dodajemy do zapytania w linii 6. Wpisujemy po niej **Suma_drzew** – tak w linii 2 nazwaliśmy wynik działania funkcji sumującej liczbę drzew przy każdej ulicy. Następnie używamy operatora **>**, aby wskazać, że w wyniku mają być uwzględnione tylko te rekordy, dla których **Suma_drzew** jest większa niż 100.

Zapytania do kilku tabel połączonych związkami

Na koniec zbudujemy zapytanie, które pokaże, ile drzew każdego gatunku występuje na każdej ulicy. Będzie to więc zapytanie uwzględniające wszystkie trzy tabele naszej bazy danych.

```
1. SELECT ulice.Ulica, drzewa.Gatunek, drzewostan.Liczba_drzew
2. FROM ulice INNER JOIN(drzewa INNER JOIN drzewostan
3. ON drzewa.IDD=drzewostan.ID_drzewa)
4. ON ulice.IDU=drzewostan.ID_ulicy
5. ORDER BY ulice.Ulica, drzewostan.Liczba_drzew DESC;
```

Zapytanie tworzące zestawienie z kilku tabel

W liniach 2–4 widać zagnieżdżenie złączenia z dwukrotnym wykorzystaniem klauzuli `INNER JOIN`. Najpierw wewnątrz nawiasu łączone są ze sobą tabele `drzewa` i `drzewostan`, a następnie do utworzonego w ten sposób złączenia dodawana jest tabela `ulice`. Wynik takiego zapytania widoczny jest na rysunku 17.8.

Dobra rada

Każde zapytanie możesz wykorzystać jako część innego zapytania – utworzysz w ten sposób podzapytanie. Więcej o podzapytaniach znajdziesz w dodatku 10 na s. 434.

Ulica	Gatunek	Liczba_drzew
Akacyjowa	sosna	20
Akacyjowa	olsza	19
Akacyjowa	klon	15
Akacyjowa	świerk	10
Akacyjowa	topola	8
Akacyjowa	brzoza	5
Akacyjowa	modrzew	2
Asnyka	świerk	17
Asnyka	sosna	15
Asnyka	brzoza	10
Asnyka	jesion	10
Asnyka	buk	10
Asnyka	topola	2
Asnyka	lipa	2
Bajkowa	jesion	19
Bajkowa	buk	18
Bajkowa	brzoza	13
Bajkowa	sosna	13
Bajkowa	olsza	12

Rys. 17.8. Fragment zestawienia danych z trzech tabel

Ćwiczenie 8

Napisz zapytanie do bazy danych *zarzad_zieleni* generujące zestawienie liczby drzew iglastych na każdej ulicy.

Zapamiętaj

W zapytaniach uwzględniających związki używa się klauzul `INNER JOIN`, `LEFT JOIN` lub `RIGHT JOIN` – określają one sposób złączenia tabel. Można też wykorzystywać funkcje agregujące, takie jak `SUM` (sumowanie) czy `AVG` (średnia), do łączenia ze sobą wartości wybranego atrybutu.

17.3. Stosowanie języka SQL w kwerendach MS Access

Jednym z trybów tworzenia kwerend w programie MS Access jest tryb widoku SQL. Jeśli znamy możliwości tworzenia zapytań w języku SQL, możemy to wykorzystać w kwerendach programu Access. Utworzymy kwerendę wyświetlającą zestawienie nazw ulic z liczbą wszystkich drzew, które przy nich rosną. Będzie to więc takie samo zestawienie jak to na rysunku 17.7 ze s. 314.

Każdą z tabel bazy danych wyeksportujemy najpierw do formatu TXT, korzystając z poniższego zapytania.

Zapytania eksportujące dane do pliku tekstowego

```
1. SELECT *
2. INTO OUTFILE 'drzewa.txt'
3. FROM drzewa;
```

Takie samo zapytanie powtarzamy dla tabel `ulice` i `drzewostan`. Wyeksportowane pliki znajdują się na dysku w folderze zawierającym bazę danych, czyli `C:\xampp\mysql\data\zarzad_zieleni`.

Tabele można też wyeksportować, korzystając z możliwości dostępnych w zakładce **Eksport**. Znajduje się tam również opcja zapisywania bazy do formatu SQL, który uwzględnia związki między tabelami i jest dobrym sposobem na tworzenie kopii zapasowej bazy danych. Warto robić takie kopie często, ponieważ wykonanych zapytań, np. usuwających dane, nie można już cofnąć.

Po wyeksportowaniu tabel z phpMyAdmin możemy je **zimportować do programu Access** i utworzyć relacje między nimi (rys. 17.9).

Importowanie danych do tabeli,
s. 298

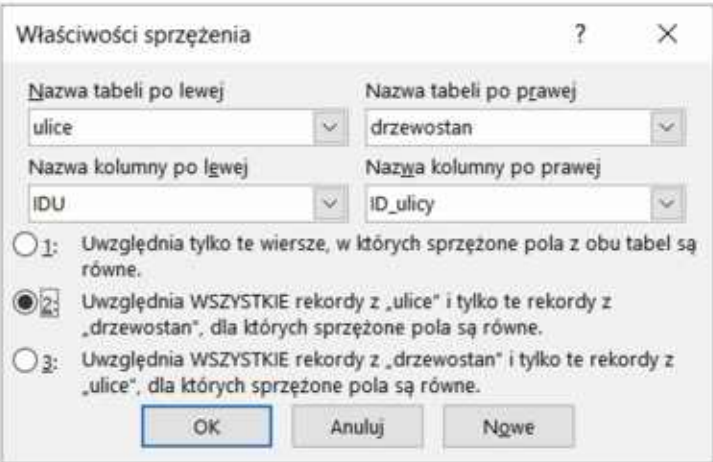
Dobra rada

Kiedy importujesz dane z pliku do programu Access, pamiętaj, aby w kreatorze importu ustawić kodowanie znaków zgodne z importowanym plikiem.



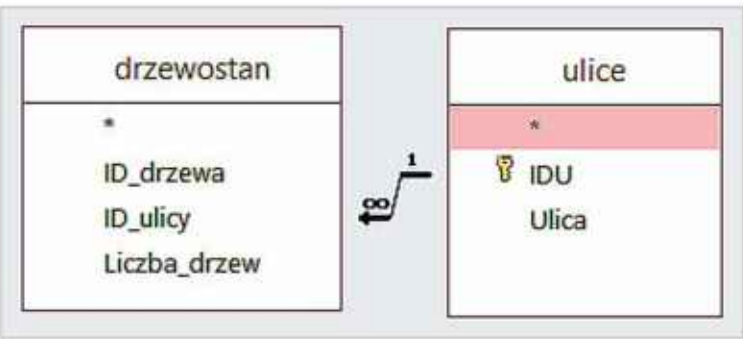
Rys. 17.9. Relacje między tabelami utworzone w programie Access

Po zapisaniu wszystkich zmian rozpoczynamy budowanie kwerendy. Wybieramy **Tworzenie** → **Kwerendy** → **Projekt kwerendy**. Dodajemy tabele ulice i drzewostan. Możemy teraz zmodyfikować relacje na potrzeby kwerendy. Relacje ustawione wcześniej nie zostaną zmienione. Klikamy dwukrotnie na linię tworzącą relację między tabelami, aby otworzyć okno **Właściwości sprzężenia** (rys. 17.10).



Rys. 17.10. Właściwości sprzężenia tabel bazy danych w programie Access

W oknie tym są dostępne trzy opcje, które odpowiadają klauzulom języka SQL: `INNER JOIN`, `LEFT JOIN` oraz `RIGHT JOIN`. My chcemy w kwerendzie uwzględnić wszystkie ulice, nawet te, przy których nie rosną drzewa, dlatego zaznaczamy drugą opcję. Zmianę sprzężenia symbolizuje niewielka strzałka na końcu linii oznaczającej relację (rys. 17.11).



Rys. 17.11. Relacja ze sprzężeniem odpowiadającym klauzuli `LEFT JOIN`

Wypełniamy siatkę projektu tak jak na rysunku 17.12.

Pole:	Ulica	Suma_drzew: Liczba_drzew	Ulica
Tabela:	ulice	drzewostan	ulice
Suma:	Grupuj według	Suma	Wyrażenie
Sortuj:		Malejąco	Rosnąco
Pokaż:	☒	☒	☐
Kryteria:			
lub:			

Rys. 17.12. Projekt kwerendy wyświetlającej sumę drzew przy każdej ulicy

W pierwszej kolumnie siatki projektu dodajemy do kwerendy pole (atrybut) *Ulica* z tabeli *ulice*. W drugiej kolumnie tworzymy pole *Suma_drzew*, które będzie sumowało wartości atrybutu *Liczba_drzew* z tabeli *drzewostan* w zależności od ulicy i sortowało zsumowane wartości malejąco. Trzecia kolumna służy do sortowania alfabetycznego według nazw ulic, jeśli przy kilku ulicach będzie tyle samo drzew. Ta kolumna nie będzie widoczna w kwerendzie.

Po uruchomieniu kwerendy widzimy wszystkie ulice oraz sumę drzew przy każdej z nich. Możemy od razu zauważyć, przy której ulicy rośnie najwięcej drzew (rys. 17.13).

Ulica	Suma_drzew
Fabryczna	144
Kwiatowa	118
Cisowa	103
Słowackiego	101
Sosnowa	101
Wierzbowa	101
Kolejowa	99
Jagodowa	97

Rys. 17.13. Fragment wyniku kwerendy wyświetlającej sumę drzew przy każdej ulicy



Dobra rada

Jeśli znasz język SQL, możesz tworzyć kwerendy w programie Access, wpisując zapytania w widoku SQL. Możesz również przygotować kwerendę w programie Access, aby sprawdzić, jak odpowiadające jej zapytanie będzie wyglądało w języku SQL.

Jeśli na karcie **Narzędzia główne** przełączymy teraz widok kwerendy na **Widok SQL**, zobaczymy zapytanie takie jak na rysunku 17.14.

Kwerenda1
<pre>SELECT ulice.Ulica, Sum(drzewostan.Liczba_drzew) AS Suma_drzew FROM ulice LEFT JOIN drzewostan ON ulice.IDU = drzewostan.ID_ulicy GROUP BY ulice.Ulica ORDER BY Sum(drzewostan.Liczba_drzew) DESC, ulice.Ulica;</pre>

Rys. 17.14. Zapytanie SQL widoczne w programie MS Access

Zauważ, że jest to takie samo zapytanie jak to ze s. 313.

Ćwiczenie 9

Wyeksportuj tabele bazy danych *zarząd_zieleni* do plików tekstowych i zaimportuj je do programu Access. Zbuduj relacje między polami tak jak na rysunku 17.9 ze s. 316, a następnie wykonaj poniższe polecenia.

- Przygotuj kwerendę wyświetlającą sumę liczby drzew na poszczególnych ulicach.
- Przejdź do widoku SQL i skopiuj z niego zapytanie. Następnie wklej je w zakładce SQL panelu phpMyAdmin i uruchom. Czy zapytanie w języku SQL zwróciło takie same dane jak kwerenda programu Access?

Podsumowanie

- Zapytania do bazy danych pozwalają wydobyć z niej określone informacje.
- Wyniki zapytań można poddawać selekcji, czyli filtrowaniu według określonych kryteriów.
- Sortowanie wyniku zapytania pozwala wyświetlić dane w ustalonym porządku.
- Dzięki projekcji atrybutów można wybrać do wyświetlenia w wyniku zapytania tylko te atrybuty, które są potrzebne.
- Agregowanie danych pozwala m.in. sumować wartości wybranego atrybutu.
- W systemach zarządzania bazą danych użytkownik może korzystać z narzędzi ułatwiających tworzenie poleceń w języku SQL (np. projektowanie kwerend w MS Access).

Zadania

- ★ **1** W bazie danych znajdują się dwie tabele:
zakupy (IDT, Towar, Cena, ID_sklepu)
sklepy (IDS, Sklep)
Zapisz w języku SQL zapytanie zwracające wszystkie dane z obu tabel.
- ★ **2** Dane o nauczycielach zapisane są w następującej tabeli:
nauczyciele (IDN, Imie, Nazwisko, Przedmiot)
Napisz zapytanie SQL, która wypisze wszystkich nauczycieli matematyki.
- ★★ **3** W bazie danych znajdują się dwie tabele:
zakupy (IDT, Towar, Cena, ID_sklepu)
sklepy (IDS, Sklep)
Atrybut ID_sklepu w tabeli zakupy jest kluczem obcym i wskazuje na atrybut IDS tabeli sklepy. Zapisz w języku SQL zapytanie zwracające informacje o nazwach wszystkich towarów z tabeli zakupy nieprzekraczających ceny 3 zł. W wynikach powinny się również znaleźć informacje o sklepach oferujących te towary. Dane posortuj według cen towarów.
- ★★ **4** W bazie danych prowadzonej przez szkołę znajdują się dane o pracy nauczycieli na wycieczkach szkolnych. Dane te zebrane są w trzech tabelach:
nauczyciele (IDN, Imie, Nazwisko)
wycieczki (IDW, Wycieczka, Liczba_dni)
organizacja (ID_nauczyciela, ID_wycieczki)
Atrybut ID_nauczyciela w tabeli organizacja wskazuje na atrybut IDN tabeli nauczyciele, a atrybut ID_wycieczki w tabeli organizacja wskazuje na atrybut IDW tabeli wycieczki. Napisz zapytanie SQL wyświetlające listę nauczycieli, którzy nie brali udziału w wycieczkach.

- ★★ **5** Utwórz nową bazę danych o nazwie *gatunki_filmowe*. Zaimportuj do niej dane otrzymane od nauczyciela (np. w plikach *filmy_5.txt*, *gatunki.txt*). Utwórz związek między tabelami tak, żeby atrybut *ID_gatunku* w tabeli *filmy* był kluczem obcym i wskazywał na atrybut *IDG* z tabeli *gatunki*. Napisz zapytanie, które wyświetli zestawienie wszystkich komedii.
- ★★★ **6** Utwórz nową bazę danych o nazwie *seanse_kinowe*. Zaimportuj do niej dane otrzymane od nauczyciela (np. w plikach *filmy_6.txt*, *kina.txt*, *seanse.txt*). Utwórz związek między tabelami tak, żeby atrybuty *ID_kina* i *ID_filmu* w tabeli *seanse* wskazywały odpowiednio na: *IDK* z tabeli *kina* oraz *IDF* z tabeli *filmy*. Napisz zapytanie wyświetlające zestawienie tytułów filmów, które będą grane w kinie Luna. Dane posortuj alfabetycznie według tytułów filmów.
- ★★★ **7** Utwórz nową bazę danych o nazwie *matura*. Ze strony Głównego Urzędu Statystycznego (stat.gov.pl) pobierz plik CSV zawierający informacje o średnich wynikach egzaminu maturalnego. Zaimportuj plik do bazy danych. Napisz zapytanie wyświetlające te egzaminy maturalne, których średni wynik zawierał się między 70% a 100%.

18. Tworzenie aplikacji korzystającej z sieciowej bazy danych

Jeśli znasz języki SQL oraz HTML, już tylko krok dzieli cię od stworzenia własnej aplikacji internetowej. Tym krokiem jest poznanie podstaw języka PHP, który połączy to, co widać w przeglądarce, z tym, co jest niewidoczne, czyli bazą danych. W tym temacie nauczysz się tworzyć aplikację, która pozwoli na gromadzenie informacji i zarządzanie nimi.

Cele lekcji

- Poznasz etapy projektowania aplikacji internetowej.
- Zaprojektujesz wygląd witryny, wykorzystując znaczniki języka HTML.
- Połączysz interfejs użytkownika z bazą danych dzięki instrukcjom języka PHP.

W praktyce zawodowej duże projekty informatyczne są przedsięwzięciem, które wymaga zaangażowania zespołu wykwalifikowanych fachowców. Wiele czasu poświęca się na dokładne przeanalizowanie potrzeb klienta, aby produkt jak najlepiej spełniał swoje funkcje. Ważnym etapem jest również testowanie i udoskonalanie stworzonej aplikacji. My jednak proces tworzenia aplikacji internetowej sprowadzimy do możliwie prostego i użytecznego przykładu, który pokaże, jak powstają tego typu programy.

18.1. Projektowanie aplikacji internetowej

Aplikacja działająca w przeglądarce internetowej jest zbudowana z wielu elementów. Dlatego jej przygotowanie wymaga ułożenia i wdrożenia planu działania. Wyróżnimy następujące etapy pracy nad aplikacją:

1. ustalenie sposobu działania aplikacji,
2. wybór technologii, konfiguracja oprogramowania, przygotowanie bazy danych,
3. wykonanie interfejsu użytkownika i stworzenie oprogramowania,
4. zapewnienie komunikacji serwisu z bazą danych.

Realizację pierwszego punktu warto rozpocząć od ustalenia szczegółów pomysłu. My wykorzystamy bazę danych z poprzednich dwóch tematów. Zawiera ona listy gatunków drzew i ulic miasta oraz informacje o tym, ile jakich drzew rośnie przy poszczególnych ulicach. Stworzymy aplikację dla pracowników zarządu zieleni miejskiej. Główną funkcją tej aplikacji będzie możliwość dodawania drzew i ulic do bazy danych oraz wyświetlania aktualnych list ulic i drzew.

Dobra rada

Kiedy projektujesz aplikację, postaraj się jak najdokładniej określić założenia projektu. Opisz możliwe funkcjonalności. Możesz nawet na początku naszkicować projekt na kartce. Dzięki temu będziesz wiedzieć, co dokładnie trzeba zrobić w kolejnych etapach pracy.

Back-end i front-end

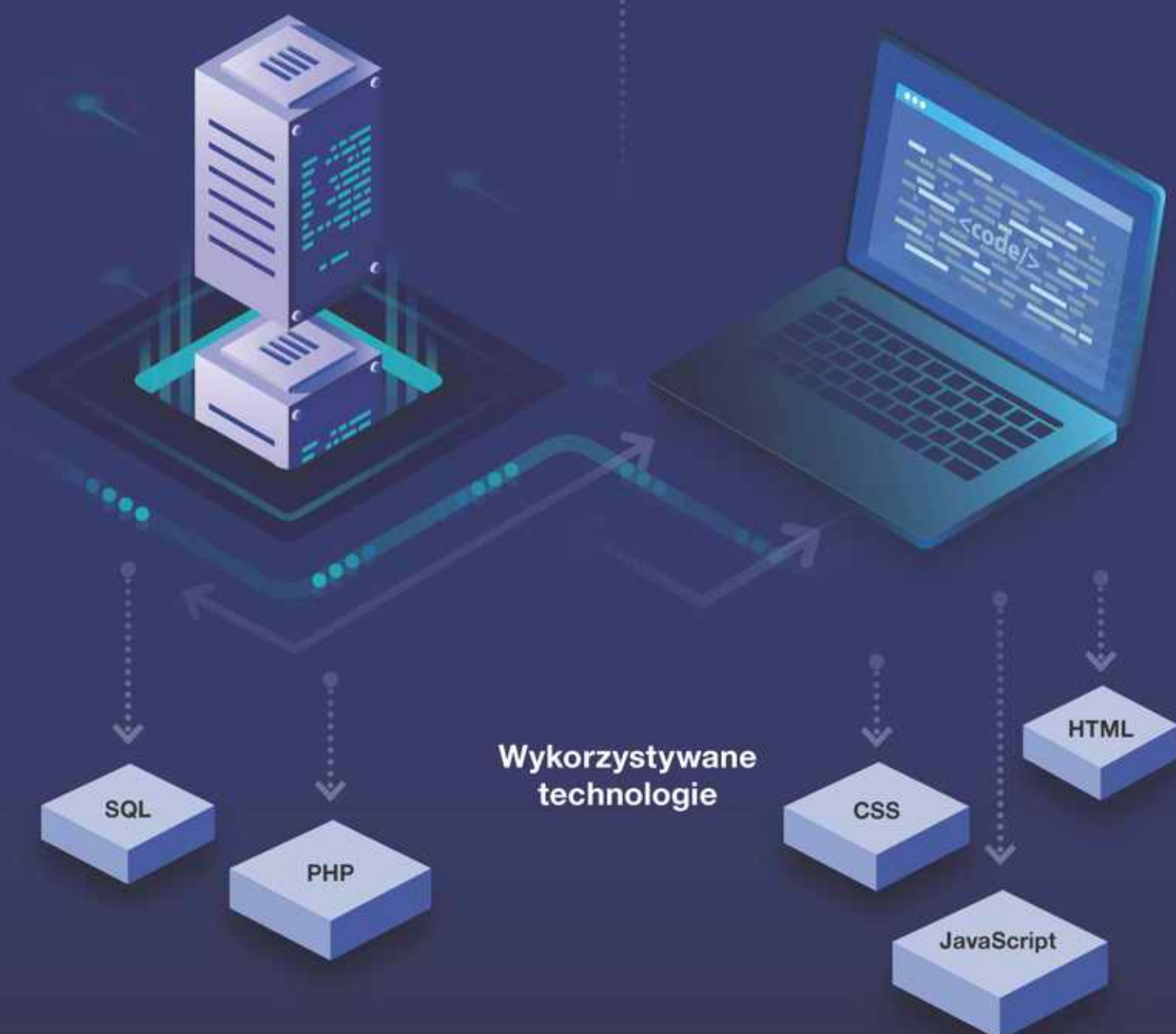
Aplikacje internetowe składają się z dwóch głównych części: back-end i front-end. Od ich prawidłowej współpracy zależy sprawne, bezpieczne i poprawne funkcjonowanie wielu serwisów, z których korzystamy na co dzień.

Back-end

Technologie, których nie widzi użytkownik. Do back-endu zalicza się system zarządzania bazą danych zainstalowany na serwerze oraz skrypty wymieniające informacje między bazą danych a użytkownikiem.

Front-end

Technologie, które widzi i z których aktywnie korzysta użytkownik. Front-end odpowiada za wygląd strony internetowej, na której pracuje aplikacja bazodanowa, oraz za funkcjonalności serwisu, z którymi bezpośrednio styka się użytkownik.



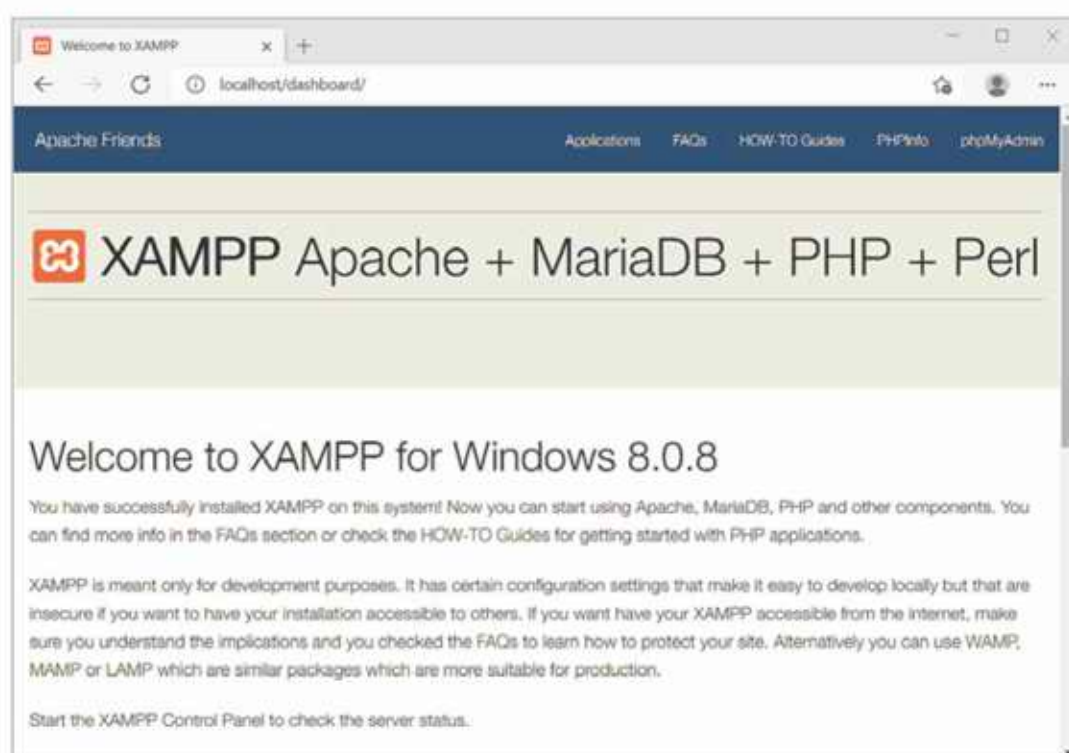
Wybór technologii i konfiguracja oprogramowania

W budowie aplikacji internetowych można wyróżnić dwie części. Pierwsza z nich to **back-end**, czyli elementy niewidoczne dla użytkownika aplikacji. Drugą częścią jest **front-end**, który obejmuje to, z czego użytkownik aktywnie korzysta.

W naszym przypadku back-end to baza danych oraz skrypty pośredniczące w komunikacji między bazą a interfejsem użytkownika. Za front-end natomiast uznajemy interfejs aplikacji i oprogramowanie odpowiedzialne za interakcję z nią.

Baza danych na potrzeby naszego projektu jest już gotowa. Do jej utworzenia wykorzystaliśmy **język SQL** oraz narzędzia dostępne w pakiecie XAMPP. Zanim przejdziemy do kolejnych etapów, utworzymy panel kontrolny tego pakietu (XAMPP Control Panel) i włączymy serwer stron internetowych (serwer webowy) Apache oraz serwer baz danych MySQL. Sprawdzimy, czy serwer internetowy jest uruchomiony, wpisując w przeglądarce internetowej adres *localhost* (rys. 18.1).

Język SQL,
s. 289 [↗](#)



Rys. 18.1. Aktywny serwer internetowy Apache

Oba te serwery są niezbędne do działania aplikacji. Serwer Apache będzie odpowiadał za wyświetlanie jej interfejsu, a serwer MySQL pozwoli na obsługę **zapytań** do bazy danych wysyłanych przez serwer webowy.

Zapytanie do bazy danych,
s. 290 [↗](#)

Potrzebujemy jeszcze folderu, w którym umieścimy pliki aplikacji. Nazwiemy go *zarzad_zieleni* i zapiszemy w katalogu *htdocs*, który znajduje się w miejscu zainstalowania pakietu XAMPP. Domyślnie jest to folder *xampp* na głównym dysku twardym naszego komputera. Pełna ścieżka utworzonego folderu wygląda więc następująco: *C:\xampp\htdocs\zarzad_zieleni*.

Warto wiedzieć

Jeśli w folderze z plikami aplikacji znajduje się też plik *index.php*, to ma on wyższy priorytet niż *index.html* i jest znajdowany jako pierwszy po wpisaniu adresu w przeglądarce internetowej.

Język HTML, podręcznik *Informatyka na czasie 1. Zakres rozszerzony*, s. 230–244 [🔗](#)

Język CSS, podręcznik *Informatyka na czasie 1. Zakres rozszerzony*, s. 245–248 [🔗](#)

Jeśli teraz w pasku adresu przeglądarki internetowej wpiszymy adres *localhost/zarzad_zieleni*, przeglądarka poszuka w tym folderze pliku *index.html* i jeśli go znajdzie, wyświetli jego zawartość jako stronę główną serwisu. Dlatego w folderze *zarzad_zieleni* utworzymy również plik *index.html*.

18.2. Realizacja oprogramowania po stronie front-end

Widoczne w aplikacji teksty i obiekty wstawimy za pomocą znaczników **języka HTML**. Nie będziemy się tu skupiać na wyglądzie strony WWW, ponieważ najważniejszym celem jest zaprogramowanie elementów współpracujących z sieciową bazą danych. Można jednak dowolnie zmodyfikować interfejs, wykorzystując możliwości **języka CSS**, aby uatrakcyjnić aplikację.

Wygląd naszej aplikacji, czyli jej interfejs, przedstawia rysunek 18.2.

Zarząd zieleni

Aplikacja umożliwia:

- dodawanie drzew i ulic do bazy
- wyświetlanie listy ulic lub listy drzew

Dodawanie

Dodaj drzewo

Gatunek: Rodzaj:

Dodaj ulicę

Ulica:

Wyświetlanie

Wyświetl listę drzew lub ulic

Wybierz, co chcesz wyświetlić:

Rys. 18.2. Interfejs aplikacji do zarządzania informacjami w bazie danych zarządu zieleni

Interfejs aplikacji

Do utworzenia pliku HTML można użyć dowolnego edytora tekstu, np. systemowego Notatnika. Najlepiej jednak wykorzystać program, który będzie kolorował znaczniki języka HTML, taki jak Notepad++. Po jego uruchomieniu zapisujemy w utworzonym wcześniej folderze plik o nazwie *index.html*. Plik na razie jest pusty, jednak dzięki dodaniu rozszerzenia *html* program automatycznie włączy kolorowanie składni. Jeśli tak się nie stanie, możemy ręcznie włączyć kolorowanie, wybierając język HTML z menu **Składnia**.

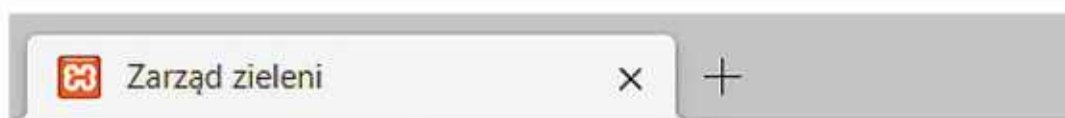
Zacniemy od utworzenia podstawowej struktury dokumentu HTML.

Dobra rada

W ustawieniach programu Notepad++ przejdź do kategorii **Autouzupełnianie** i zaznacz autowstawianie tagów zamykających języka HTML. Usprawni ci to pracę – kiedy wpiszesz jakiś znacznik otwierający, program automatycznie wstawi jego zamykający odpowiednik.

```
1. <!DOCTYPE html>
2. <html lang="pl">
3. <head>
4.     <meta charset="UTF-8" />
5.     <title>Zarząd zieleni</title>
6. </head>
7. <body>
8.
9. </body>
10. </html>
```

W liniach 3–6 znajduje się sekcja **HEAD**, która przekazuje przeglądarce dwie informacje o naszej witrynie. Pierwsza dotyczy kodowania znaków (linia 4) – jest to UTF-8, czyli takie samo kodowanie, jakiego użyliśmy w bazie *zarzad_zieleni*. Druga informacja to tytuł strony (linia 5). Będzie on widoczny na karcie przeglądarki (rys. 18.3).



Rys. 18.3. Tytuł strony widoczny na karcie przeglądarki internetowej

W sekcji **BODY** tworzymy główną treść strony. Zacniemy od wyświetlenia nazwy i opisu aplikacji.

```
7. <body>
8. <h1>Zarząd zieleni</h1>
9.     <p>Aplikacja umożliwia:</p>
10.    <ul>
11.        <li>dodawanie drzew i ulic do bazy</li>
12.        <li>wyświetlanie listy ulic lub listy drzew</li>
13.    </ul>
14.    <hr />
15. </body>
```


W linii 8 tworzymy nagłówek pierwszego poziomu, w którym wpisujemy nazwę aplikacji. W liniach 9–13 dodajemy akapit (znaczniki `<p>` i `</p>`) i listę wypunktowaną (`<ul>` i `</ul>`), aby opisać działanie aplikacji. Ten fragment kodu kończymy w linii 14 znacznikiem `<hr />`, który zamyka tę część strony poziomą linią (rys. 18.4).

Zarząd zieleni

Aplikacja umożliwia:

- dodawanie drzew i ulic do bazy
- wyświetlanie listy ulic lub listy drzew

Rys. 18.4. Pierwsza część strony z nazwą aplikacji i opisem jej działania

Formularze

Teraz dodamy pierwszy formularz, który pozwoli użytkownikowi na dodanie drzewa do bazy. Poniższy kod umieszczamy po znaczniku `<hr>`.

```
15. <h2>Dodawanie</h2>
16. <h3>Dodaj drzewo</h3>
17. <form action="dodaj_drzewo.php" method="POST">
18.     Gatunek:
19.     <input type="text" name="gatunek" maxlength="50" />
20.     Rodzaj:
21.     <select name="rodzaj">
22.         <option>liściaste</option>
23.         <option>iglaste</option>
24.     </select>
25.     <input type="submit" value="Zapisz" />
26. </form>
```

Warto wiedzieć

Oprócz pól tekstowych, list rozwijanych i przycisków formularz może zawierać inne elementy, np. pola wyboru czy przyciski opcji. Formularz może też służyć do przesyłania plików na serwer.

Przed pierwszym formularzem dodajemy dwa nagłówki: drugiego stopnia (linia 15) i trzeciego stopnia (linia 16). Oznaczamy w ten sposób, do czego służy ta część aplikacji. W linii 17 od znacznika `<form>` zaczynamy formularz.

Pierwszym elementem formularza jest pole tekstowe – za jego wstawienie odpowiada znacznik `<input>` w linii 19. Drugim elementem jest lista rozwijana (linie 21–24). Trzecim jest przycisk typu `submit`, którego naciśnięcie wyśle dane z formularza do pliku wskazanego za pomocą atrybutu `action` w znaczniku `<form>` (linia 17). Znacznik `<form>` ma jeszcze atrybut `method`, który wskazuje sposób przesyłania danych do pliku. Jego wartość ustawiona na `POST` oznacza, że dane nie zostaną dopisane do adresu URL i ich przekazanie nie będzie widoczne dla użytkownika.

Drugim sposobem przesyłania danych z formularza jest metoda **GET**, która dane te dodaje na końcu adresu podanego w atrybucie **action**. Tę metodę wykorzystuje się np. w wyszukiwarkach internetowych.

Elementy formularza mają swoje unikalne nazwy, nadane za pomocą atrybutu **name**. Zostaną one wykorzystane przez plik PHP do jednoznacznego określenia, jaka informacja została przesłana.

Stworzymy teraz kolejny formularz, którego zadaniem będzie dodawanie nowych ulic do bazy danych.

```
27. <h3>Dodaj ulicę</h3>
28. <form action="dodaj_ulice.php" method="POST">
29. Ulica:
30.     <input type="text" name="ulica" maxlength="50" />
31.     <input type="submit" value="Zapisz" />
32. </form>
33. <hr />
```

Formularz ten jest bardzo podobny do poprzedniego. Składa się z pola tekstowego i przycisku **Zapisz**. Po jego naciśnięciu dane są przesyłane do pliku *dodaj_ulice.php* (linia 28). Na końcu formularza wstawiamy poziomą linię za pomocą znacznika `<hr />` w linii 33 (rys. 18.5).

Warto wiedzieć

Pole tekstowe formularza pozwala przesłać praktycznie każdy tekst, w tym zapytanie SQL. Wprawny użytkownik może więc przesłać zapytanie usuwające rekordy, tabele lub nawet bazę danych (atak *SQL injection*). Aby się przed tym uchronić, warto zablokować niektóre uprawnienia do bazy danych i zastosować funkcję zabezpieczającą języka PHP, np. `addslashes`.

Rys. 18.5. Formularze pozwalające na dodawanie drzew i ulic do bazy danych

W ostatniej części dokumentu *index.html* utworzymy formularz pozwalający określić, jakie dane chcemy wyświetlić. Z listy rozwijanej będziemy mogli wybrać, czy mają to być drzewa czy ulice.

Rys. 18.6. Lista rozwijana umożliwiająca wyświetlenie drzew lub ulic

Następnie za pomocą przycisku **Wyświetl** wyślemy nasz wybór do pliku *wyświetl.php*.


```

34. <h2>Wyświetlanie</h2>
35. <h3>Wyświetl listę drzew lub ulic</h3>
36. <form action="wyswietl.php" method="POST">
37.     Wybierz, co chcesz wyświetlić:
38.     <select name="lista">
39.         <option>drzewa</option>
40.         <option>ulice</option>
41.     </select>
42.     <input type="submit" value="Wyświetl" />
43. </form>

```

W linii 38 definiujemy listę rozwijaną, w liniach 39 i 40 – elementy tej listy, a w linii 42 – przycisk typu `submit`, wysyłający dane (rys. 18.7).

Wyświetlanie

Wyświetl listę drzew lub ulic

Wybierz, co chcesz wyświetlić:

Rys. 18.7. Formularz odpowiedzialny za wybranie, co ma zostać wyświetlone

Dobra rada

Jeśli zechcesz utworzyć kolejną aplikację, po prostu załóż dla niej nowy folder w katalogu `C:\xampp\htdocs`. Pamiętaj, aby w tym folderze umieścić plik strony głównej `index.html` lub `index.php`.

Po zapisaniu zmian w pliku otwieramy przeglądarkę internetową i wpisujemy w niej adres `localhost/zarzad_zieleni`. Zobaczymy utworzoną właśnie stronę internetową z formularzami. Jeśli strona się nie wyświetliła, sprawdźmy, czy plik `index.html` znajduje się w folderze `zarzad_zieleni` oraz czy uruchomiony został serwer Apache.

Ćwiczenie 1

Otwórz panel kontrolny XAMPP i uruchom serwery Apache oraz MySQL. Utwórz folder `zarzad_zieleni` na dysku głównym swojego komputera w folderze `C:\xampp\htdocs`. W dowolnym edytorze tekstu za pomocą znaczników HTML utwórz interfejs aplikacji taki jak na rysunku 18.2. Plik zapisz pod nazwą `index.html` w folderze `zarzad_zieleni`.



Zapamiętaj

Współczesne strony internetowe to najczęściej aplikacje działające w przeglądarce i składające się z dwóch głównych części: technologii odpowiedzialnych za interfejs użytkownika (front-end) i technologii ukrytych przed wzrokiem użytkownika (back-end). Wygląd aplikacji projektuje się za pomocą znaczników HTML oraz CSS.

18.3. Realizacja oprogramowania po stronie back-end

Po utworzeniu interfejsu aplikacji czas zająć się jej silnikiem, czyli elementami, które odpowiadają za prawidłowe działanie programu oraz komunikację z bazą danych.

W pliku HTML utworzyliśmy trzy formularze i określiliśmy w nich pliki, do których mają być przekazywane dane. Są to kolejno:

- ▶ *dodaj_drzewo.php* – po wprowadzeniu do formularza gatunku i rodzaju nowego drzewa skrypt z tego pliku doda je jako nowy rekord w tabeli drzewa,
- ▶ *dodaj_ulice.php* – po wprowadzeniu do formularza nazwy ulicy skrypt z tego pliku doda ją jako nowy rekord w tabeli ulice,
- ▶ *wyswietl.php* – po wybraniu z listy rozwijanej słowa *drzewa* lub *ulice* skrypt z tego pliku wyświetli rekordy z odpowiedniej tabeli.

Powyższe trzy pliki utworzymy, wykorzystując możliwości **języka PHP**. Jest to skryptowy język programowania, czyli taki, którego kod źródłowy zawiera skrypty wykonywane po stronie serwera. W chwili wywołania skryptu z pliku PHP interpreter zainstalowany na serwerze internetowym na bieżąco wykonuje kod i zwraca użytkownikowi wyniki jego działania. W naszym przypadku skrypty w plikach PHP będą przyjmowały dane z formularzy umieszczonych w pliku *index.html*. Dzięki zapytaniom SQL zawartym w skryptach PHP przyjęte dane będą zapisywane w bazie lub zostaną wyświetlone rekordy z wybranej tabeli.

Podobnie jak w przypadku pliku HTML, trzy pliki PHP utworzymy w folderze *zarzadz_zieloni*. Zaczniemy od pliku *dodaj_drzewo.php*. Wykorzystamy w nim również znaczniki HTML. Plik będzie zawierał podstawową strukturę dokumentu HTML, a w niej – w sekcji **BODY** – znacznik otwierający `<?php` i zamykający `?>`. Wszystko, co znajdzie się między tymi znacznikami, serwer internetowy przekaże do interpretera PHP, aby ten wykonał kod i zwrócił wynik.

Przyjmowanie danych z formularza

Powiedzieliśmy wcześniej, że formularz przyjmuje dane od użytkownika, a po naciśnięciu przycisku typu **submit** przekazuje je do pliku docelowego PHP. W przypadku formularza służącego do dodawania drzewa były to dwie informacje: gatunek i rodzaj. Tak też nazwaliliśmy pola formularza przyjmującego te dane – odpowiada za to ich atrybut **name**.

W pliku PHP utworzymy dwie zmienne, które posłużą do przechowywania wartości przyjętych z pól formularza:

1. `$gatunek = $_POST["gatunek"];`
2. `$rodzaj = $_POST["rodzaj"];`

Warto wiedzieć

Silnik to potoczna nazwa niewidocznych elementów aplikacji. Można je porównać do silnika samochodu, który również jest ukryty, ale stanowi najważniejszy element odpowiedzialny za napęd pojazdu.

Język PHP

Dobra rada

W plikach PHP możesz używać dowolnych znaczników HTML, a także tworzyć linki do plików CSS. W ten sposób możesz zmieniać wygląd strony zawierającej skrypty.

Nazwy zmiennych w języku PHP poprzedzamy znakiem dolara. Następnie po znaku równości przypisujemy tym zmiennym wartości. W naszym przypadku wskazujemy, że pochodzą one z metody `POST` zastosowanej w formularzu HTML. W nawiasach kwadratowych i cudzysłowach wpisujemy nazwy pól formularza, z których tworzone zmienne mają pobrać wpisane wartości. Każdą instrukcję języka PHP kończymy średnikiem.

Połączenie z bazą danych

Kiedy wartości z formularza zostaną przypisane do zmiennych, należy wskazać, co interpreter ma z nimi zrobić. W naszym przypadku wartości te zostaną przekazane do bazy danych. Jednak zanim to nastąpi, skrypt musi nawiązać połączenie z bazą danych. W tym celu stworzymy kolejną zmienną, która będzie nowym **obiektem** należącym do **klasy** `mysqli`. Dzięki niej będziemy mogli korzystać z języka SQL w skrypcie PHP.

Obiekt,
s. 226 
Klasa `mysqli` 

```
3. $polaczenie = new mysqli("localhost", "admin",  
4. "@dm1n1Str@t0r", "zarzad_zieleni");
```

Atrybut,
s. 226 

W nawiasach wpisujemy **atrybuty** nowego obiektu. Są to kolejno:

- ▶ nazwa i adres serwera – w naszym przypadku jest to nasz komputer, dlatego wpisujemy `localhost`,
- ▶ użytkownik – wykorzystamy utworzone w temacie 16 konto użytkownika `admin`,
- ▶ hasło – wpisujemy hasło przypisane użytkownikowi `admin`,
- ▶ nazwa bazy danych – w naszym przypadku jest to `zarzad_zieleni`.

Dodawanie rekordu do bazy danych

Po przypisaniu do zmiennych informacji pochodzących z formularza i połączeniu skryptu z bazą danych pozostaje jeszcze wpisanie tych danych jako nowego rekordu w bazie.

A to ciekawe

System operacyjny w chmurze

Coraz więcej programów komputerowych jest przenoszonych do chmury. W 2021 r. firma Microsoft zaprezentowała system Windows działający w chmurze i obsługiwany przez przeglądarkę internetową z dowolnego miejsca na świecie. Każdy użytkownik takiego systemu może go skonfigurować według własnych preferencji i przechowywać w nim swoje zasoby.



```

5. $polaczenie->query
6. ("INSERT INTO
7. drzewa(Gatunek, Rodzaj)
8. VALUES
9. ('$gatunek', '$rodzaj')");

```

W linii 5 wywołujemy **metodę** query obiektu \$polaczenie. Jej zadaniem jest wykonanie **zapytania do bazy danych**. Zapytanie umieszczamy jako parametr metody query w nawiasach okrągłych i w cudzysłowie. Nasze zapytanie wykorzystuje **klauzulę** INSERT INTO (linia 6), aby atrybutom Gatunek i Rodzaj w tabeli drzewa przypisać wartości (linia 8) znajdujące się w zmiennych \$gatunek i \$rodzaj (linia 9).

• **Metoda** query

Zapytanie do bazy danych, s. 290 [↗](#)

Klauzula, s. 291 [↗](#)

Zamykanie połączenia i wyświetlenie komunikatu

Po wykonaniu zapytania należy zamknąć aktywne połączenie z bazą danych. W tym celu w 10 linii pliku wywołujemy **metodę** close obiektu \$polaczenie.

• **Metoda** close

```
10. $polaczenie->close();
```

Metoda close nie przyjmuje żadnych parametrów.

Skrypt PHP powinien już działać prawidłowo, czyli dodawać nowy rekord do bazy danych. Jednak aby aplikacja była kompletna, dopiszemy jeszcze komunikat dla użytkownika i wstawimy hiperłącze umożliwiające powrót do strony głównej.

```

11. echo "Dodano drzewo $rodzaj $gatunek<br />";
12. echo "<a href='index.html'>Wróć</a>";

```

Do wyświetlania napisów w przeglądarce używamy w języku PHP **słowa kluczowego** echo. Po nim w cudzysłowie wpisujemy frazę, która ma zostać wyświetlona. Zauważ, że możemy również wyświetlać aktualne wartości zmiennych – w tym celu po prostu podajemy ich nazwy (linia 11) – oraz wykorzystywać znaczniki HTML, np. tworzące hiperłącza (linia 12).

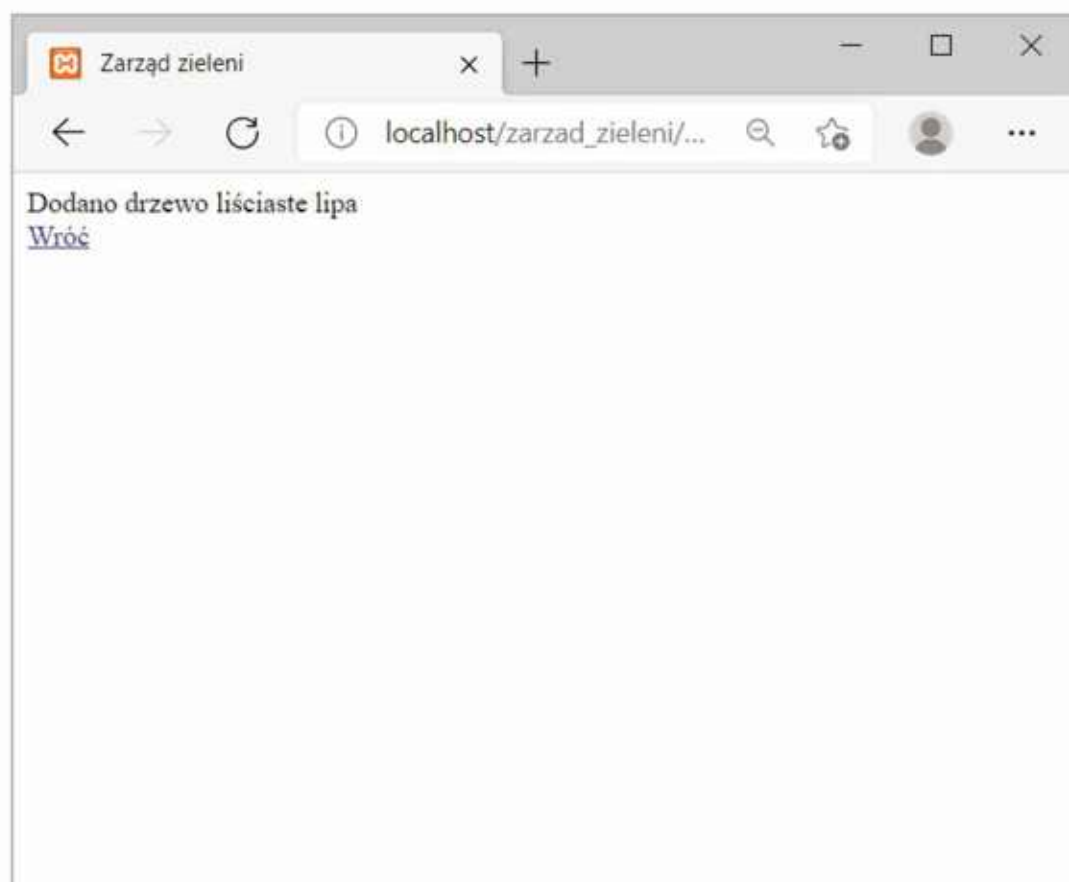
• **Słowo kluczowe** echo

Testowanie aplikacji

Po zapisaniu zmian w pliku *dodaj_drzewo.php* możemy już przetestować działanie gotowej części aplikacji. W przeglądarce internetowej wpisujemy adres *localhost/zarzad_zieleni*, aby uruchomić aplikację. W pierwszym formularzu widocznym na stronie wprowadzamy nazwę nowego drzewa i wybieramy, jakiego jest rodzaju. Zatwierdzamy przyciskiem **Zapisz** i zostajemy przeniesieni do pliku *dodaj_drzewo.php*. Skrypt zawarty w tym pliku doda nowe drzewo do tabeli bazy danych i wyświetli odpowiedni komunikat (rys. 18.8, s. 332).

Dobra rada

Zanim przejdziesz do kolejnego etapu tworzenia aplikacji, możesz sprawdzić, czy skrypt na pewno dodał drzewo do bazy. W tym celu wykorzystaj zapytanie do bazy danych – napisz je w phpMyAdmin lub użyj Wiersza poleceń.



Rys. 18.8. Okno przeglądarki z komunikatem potwierdzającym dodanie nowego drzewa

Warto wiedzieć

Programiści umieszczają w pliku PHP również kod, który ma zadziałać w razie błędu (np. braku połączenia z bazą danych). Najczęściej taki kod przenosi na inną stronę w serwisie.

Ćwiczenie 2

W dowolnym edytorze tekstu, korzystając z instrukcji języka PHP, utwórz skrypt:

- a. dodający drzewo do bazy danych (plik zapisz pod nazwą *dodaj_drzewo.php* w folderze *zarząd_zieleni*),
- b. dodający ulicę do bazy danych (plik zapisz pod nazwą *dodaj_ulice.php* w folderze *zarząd_zieleni*).



Zapamiętaj

Za pomocą instrukcji języka PHP można tworzyć skrypty przekazujące informacje do bazy danych. Skrypty są wykonywane po stronie serwera.

Wyświetlanie danych

Skrypt dodający nowy rekord do tabeli ulice jest bardzo podobny do skryptu dodającego rekord do tabeli drzewa, dlatego nie będziemy go omawiać. Przejdziemy do ostatniego pliku PHP, czyli *wyswietl.php*.

Plik przyjmuje z formularza HTML jedną wartość, zapisaną pod nazwą *lista*. W znany już sposób przypiszemy tę wartość do zmiennej i utworzymy połączenie z bazą danych.

```

1. $lista = $_POST["lista"];
2. $polaczenie = new mysqli("localhost",
3. "admin", "@dm1n1Str@t0r", "zarzad_zieleni");

```

Zmienna `$lista` może przyjąć jedną z dwóch wartości: drzewa lub ulice. Dzieje się tak, ponieważ w formularzu źródłowym użyliśmy listy rozwijanej zawierającej tylko te dwie opcje. W zależności od tego, jaka to będzie wartość, powinny zostać wyświetlone inne dane. Do wyboru ścieżki działania wykorzystamy **instrukcję warunkową if**.

🔗 Instrukcja warunkowa if

```

4. if($lista == "drzewa"){
5.     $wyswietl = $polaczenie->query("SELECT
6.     Gatunek, Rodzaj FROM drzewa");
7.     while($wiersz = $wyswietl->fetch_array()){
8.         echo "Gatunek: ".$wiersz["Gatunek"]." Rodzaj: ".
9.         $wiersz["Rodzaj"]."<br />";
10.    }
11. }

```

W nawiasie po instrukcji **if** sprawdzamy, czy zmienna `$lista` ma wartość drzewa. Jeśli warunek jest spełniony, wykonywane są instrukcje w nawiasach klamrowych.

W linii 5 dla zmiennej (obiektu) `$polaczenie` wywoływana jest metoda `query`, wysyłająca zapytanie do bazy danych. Zapytanie ma zwrócić wartości atrybutów `Gatunek` i `Rodzaj` z tabeli `drzewa`, a wynik ma zostać przypisany do zmiennej tablicowej `$wyswietl`.

W linii 7 wykorzystujemy **pętlę while**, aby wyświetlić kolejne rekordy z tabeli `drzewa`. W warunku działania pętli widzimy przypisanie do zmiennej `$wiersz` wyniku działania **metody** `fetch_array`, wywołanej dla zmiennej tablicowej `$wyswietl`. Metoda `fetch_array` pobiera kolejne wartości ze zmiennej `$wyswietl` i dzięki instrukcji zawartej w pętli wyświetla wartości indeksów `Gatunek` i `Rodzaj`. Instrukcja w pętli będzie powtarzana, dopóki istnieją jakieś nieodczytane wartości.

Zmienna `$wiersz` jest interpretowana jako zmienna tablicowa o dwóch polach, ponieważ może przyjmować dwie wartości. Indeksy zmiennej tablicowej mają nazwy odpowiadające nazwom atrybutów tabeli (`Gatunek`, `Rodzaj`).

Prześledźmy jeszcze działanie instrukcji w pętli. Słowo kluczowe **echo** spowoduje wyświetlenie kolejnych wartości, które są połączone ze sobą za pomocą **operatora konkatencji**. W języku PHP operatorem tym jest kropka. Pierwsza pojawi się wartość pola z indeksem `Gatunek` zmiennej tablicowej `$wiersz`. Potem zostanie wyświetlona spacja, a następnie wartość pola z indeksem `Rodzaj` zmiennej tablicowej `$wiersz`. Na końcu zostanie wstawiony znacznik końca wiersza `<br />`.

📄 Warto wiedzieć

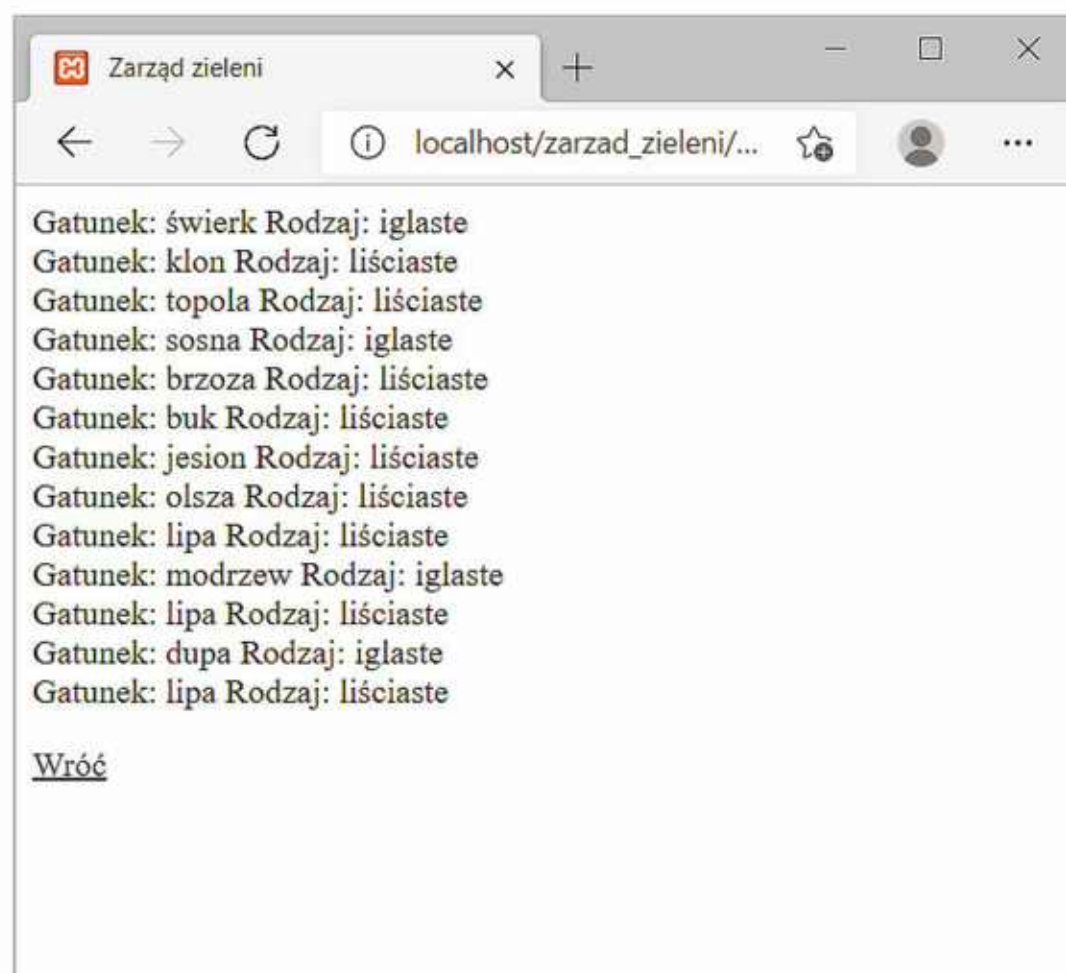
Język PHP jest bardzo podobny do języka C++. Korzysta z takich samych operatorów i obowiązują w nim podobne reguły, np. średnik na końcu każdej instrukcji.

🔗 Pętla while

🔗 Metoda `fetch_array`

🔗 Operator konkatencji

Podobną instrukcję warunkową konstruujemy, aby wyświetlić nazwy ulic. Na końcu pliku należy zamknąć połączenie z bazą danych i wyświetlić hiperłącze pozwalające na powrót do strony głównej. Efekt działania formularza po wybraniu wartości „drzewa” jest widoczny na rysunku 18.9.



Rys. 18.9. Wynik działania skryptu z pliku *wyswietl.php*

Ćwiczenie 3

W dowolnym edytorze tekstu, korzystając z instrukcji języka PHP, utwórz skrypt pozwalający na wyświetlenie listy drzew lub listy ulic z bazy danych *zarząd_zieleni*. Plik zapisz pod nazwą *wyswietl.php* w folderze *zarząd_zieleni*.

Zapamiętaj

Język PHP jest podobny do innych interpretowanych języków programowania. Można w nim wykorzystywać instrukcje warunkowe oraz pętle. Pozwala on również na przyjmowanie danych z formularzy HTML oraz przekazywanie ich do bazy danych dzięki zapytaniom języka SQL.

Podsumowanie

- Przed wykonaniem aplikacji sieciowej należy wybrać środowisko do jej uruchomienia (serwery WWW i bazodanowe) oraz technologie do tworzenia strony WWW i skryptów (HTML, CSS, PHP).
- Za wygląd strony i oprogramowanie zadań wykonywanych przez przeglądarkę odpowiedzialna jest część aplikacji po stronie front-end.
- Za zadania polegające na komunikacji z bazą danych odpowiedzialna jest część aplikacji po stronie back-end.
- Dane do aplikacji internetowej są przesyłane za pośrednictwem formularzy metodą **POST** lub metodą **GET**.
- Skryptowy język programowania PHP wykorzystuje instrukcje znane z wielu innych języków programowania, m.in. instrukcje warunkowe i pętle.
- Jednym z kluczowych zadań w procesie tworzenia aplikacji jest jej testowanie oraz wprowadzanie poprawek i ulepszeń.

Zadania

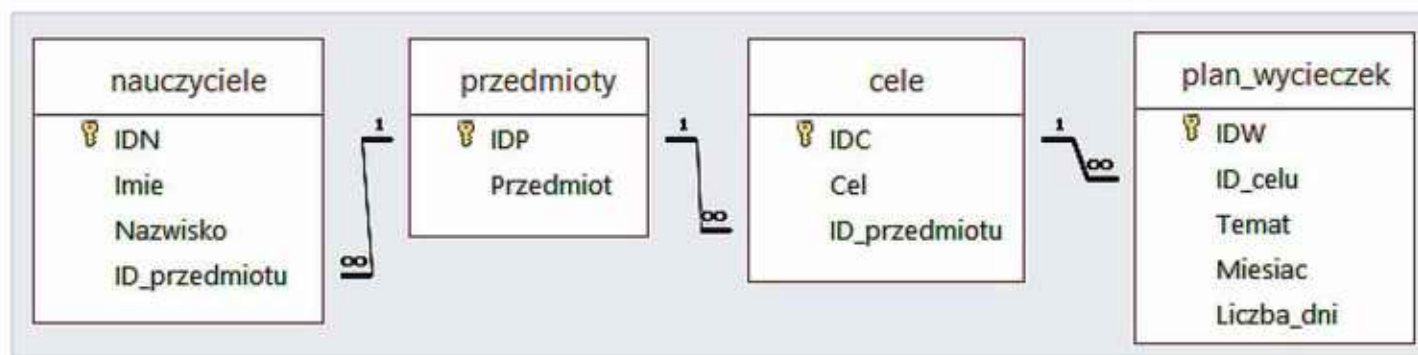
- * **1** Wykorzystaj znaczniki HTML oraz CSS, aby uatrakcyjnić wygląd utworzonej w temacie aplikacji. Dodaj kolory, zmień krój czcionki, wyrównaj elementy na stronie.
- ** **2** Do aplikacji utworzonej w temacie dodaj skrypty umożliwiające przypisanie gatunków drzew do ulic oraz określenie liczby drzew przy każdej ulicy.
- ** **3** Do aplikacji utworzonej w temacie dodaj skrypty pozwalające na wyświetlanie informacji o tym, jakie gatunki drzew i w jakiej liczbie rosną przy wybranej ulicy.
- ** **4** Zmodyfikuj aplikację utworzoną w temacie, rozdzielając jej poszczególne funkcjonalności na oddzielne podstrony. Na stronie głównej umieść nazwę aplikacji i jej opis oraz menu, które pozwoli na poruszanie się między podstronami.
- *** **5** Utwórz aplikację, która pozwoli ci na zapisywanie swoich ocen z poszczególnych przedmiotów. Przygotuj bazę danych złożoną z przynajmniej dwóch tabel, zaprojektuj interfejs aplikacji oraz napisz skrypty umożliwiające komunikację z bazą danych.
- *** **6** Do serwera SQL zaimportuj bazę danych przekazaną przez nauczyciela (np. w pliku *artykuly_autorzy.accdb*). Baza zawiera informacje o artykułach na blog i ich autorach. Utwórz aplikację pozwalającą na dodawanie informacji do bazy danych oraz wyświetlanie listy artykułów i ich autorów.

Wskazówka: Data w programie Access domyślnie zapisywana jest w formacie 01.09.2021. Z kolei język SQL wymaga daty w formacie 2021-09-01. Aby po wyeksportowaniu danych daty w pliku tekstowym były w odpowiednim formacie, w kreatorze eksportu kliknij przycisk **Zaawansowane** i w otwartym oknie zmień **Kolejność dat** na **RMD** oraz **Ogranicznik daty** na **–**. Więcej informacji o funkcjach daty i czasu w języku SQL znajdziesz w dodatku 11 na s. 435.

Sposób na zadania

Zadanie 1

W szkole sporządzono plan wycieczek. Częścią tego planu jest określenie celu każdej wycieczki oraz powiązanego z nim przedmiotu. Informacje te umieszczono w tabeli `cele` w postaci: identyfikator celu, nazwa celu, identyfikator przedmiotu. Listę przedmiotów zawiera tabela `przedmioty`, a jej układ jest następujący: identyfikator przedmiotu, nazwa przedmiotu. W tabeli `nauczyciele` znajduje się lista nauczycieli zatrudnionych w szkole wraz z identyfikatorami przedmiotów, których uczą. W tabeli `plan_wycieczek` zapisano: identyfikator wycieczki, identyfikator celu, temat, miesiąc, w którym ma się odbyć wyjazd, oraz planowaną liczbę dni wycieczki. Tabele są połączone następującymi relacjami (związkami):



Rozwiąż poniższe zadania. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.1 (0–1)

Napisz zapytanie w języku SQL, które wyświetli zestawienie wycieczek i celów realizowanych w ramach każdej wycieczki.

Rozwiązanie

Tagi: zapytanie, związek, klauzula, złączenie

Krok 1

Wykorzystujemy klauzulę `SELECT`, aby wskazać, że chcemy wyświetlić wartości atrybutów `IDW` i `Temat` z tabeli `plan_wycieczek` oraz atrybutu `Cel` z tabeli `cele`.

Krok 2

Za pomocą klauzuli `FROM` wskazujemy tabele, z których będą pobierane dane. Z użyciem klauzuli `INNER JOIN` tworzymy wewnętrzne złączenie między tymi tabelami.

Krok 3

Używamy klauzuli `ON`, dzięki której w wyniku wyszukiwania zostaną uwzględnione tylko rekordy spełniające wpisany warunek.

Odpowiedź: Zapytanie w języku SQL zwracające jako wynik zestawienie wycieczek i związanych z nimi celów wygląda następująco:

1. `SELECT plan_wycieczek.IDW, plan_wycieczek.Temat, cele.Cel`
2. `FROM cele INNER JOIN plan_wycieczek`
3. `ON cele.IDC=plan_wycieczek.ID_celu;`

Zadanie 1.2 (0–1)

Napisz zapytanie w języku SQL, które wyświetli wszystkie wycieczki zaplanowane na listopad trwające dłużej niż jeden dzień.

Rozwiązanie

Tagi: zapytanie, selekcja wyników zapytania, operator porównania

Krok 1

Za pomocą klauzuli **SELECT** wskazujemy, że chcemy wyświetlić wartości atrybutów **IDW**, **Temat**, **Miesiąc** oraz **Liczba_dni**.

Krok 2

Wpisujemy klauzulę **FROM**, aby wybrać tabelę, z której będą pobierane dane. W naszym przypadku jest to tabela **plan_wycieczek**.

Krok 3

Dokonujemy selekcji wyników zapytania za pomocą klauzuli **WHERE**, po której wpisujemy dwa warunki filtrowania. Oba muszą być spełnione jednocześnie, dlatego łączymy je operatorem **AND**. Pierwszy warunek wybierze wszystkie rekordy, których atrybut **Miesiąc** ma wartość **listopad**. Drugi wybierze z tych rekordów tylko te, których atrybut **Liczba_dni** ma wartość większą niż 1.

Odpowiedź: Zapytanie w języku SQL zwracające jako wynik listę wycieczek zaplanowanych na listopad i trwających dłużej niż jeden dzień wygląda następująco:

```
1. SELECT IDW, Temat, Miesiac, Liczba_dni
2. FROM plan_wycieczek
3. WHERE Miesiac="listopad"
4. AND Liczba_dni>1;
```

Informacja do zadań 1.3 i 1.4

W plikach, które otrzymasz od nauczyciela (np. *WUZ4_zad1_cele.txt*, *WUZ4_zad1_przedmioty.txt*, *WUZ4_zad1_nauczyciele.txt*, *WUZ4_zad1_plan_wycieczek.txt*), znajdują się przykładowe dane z tabel bazy danych.

Oto 5 pierwszych wierszy każdego pliku:

WUZ4_zad1_cele.txt

IDC	Cel	ID_przedmiotu
1	Badanie elementów przyrody	1
2	Poznanie historii Polski i Europy	5
3	Używanie języka obcego w praktyce	7
4	Poznanie literatury polskiej	6

WUZ4_zad1_przedmioty.txt

IDP	Przedmiot
1	Biologia
2	Chemia
3	Fizyka
4	Geografia

WUZ4_zad1_nauczyciele.txt

IDN	Imie	Nazwisko	ID_przedmiotu
1	Jan	Pospieszalski	1
2	Przemysław	Starościński	2
3	Maria	Niewiadomska	3
4	Marcin	Trzeciakowski	4

WUZ4_zad1_plan_wycieczek.txt

IDW	ID_celu	Temat	Miesiac	Liczba_dni
1	1	Ewolucja biologiczna...	czerwiec	1
2	2	Życie ludzi w średniowieczu	czerwiec	1
3	3	Język angielski w kulturze...	czerwiec	1
4	4	Literatura polska na przełomie...	czerwiec	2

Wykorzystaj dane z plików do rozwiązywania poniższych zadań. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.3 (0–1)

Podaj imiona i nazwiska wszystkich nauczycieli języków obcych. Zwróć uwagę, że w tabeli przedmioty są dwa języki obce: angielski i niemiecki.

Rozwiązanie

Tagi: kwerenda wybierająca, alternatywa logiczna

Krok 1

Dane z plików importujemy do oddzielnych tabel bazy danych w programie Access. Tabele łączymy relacjami. Tworzymy nową kwerendę w widoku projektu. Do kwerendy dodajemy tabele nauczyciele i przedmioty.

Krok 2

Do siatki widoku projektu dodajemy pola Imie i Nazwisko z tabeli nauczyciele oraz Przedmiot z tabeli przedmioty.

Krok 3

W wierszu **Kryteria** dla pola Przedmiot wpisujemy „język angielski”. Jako drugie kryterium (w wierszu **lub**) wpisujemy „język niemiecki”. Możemy też wpisać oba kryteria w jednym wierszu w postaci: „język angielski” Or „język niemiecki”. Wykorzystamy w ten sposób operator logiczny Or, który oznacza „lub” (jest to więc alternatywa logiczna). Odznaczamy opcję **Pokaż** dla pola Przedmiot.

Pole:	Imie	Nazwisko	Przedmiot
Tabela:	nauczyciele	nauczyciele	przedmioty
Sortuj:			
Pokaż:	☒	☒	☐
Kryteria:			"język niemiecki"
lub:			"język angielski"

Zapytanie w języku SQL:

1. `SELECT nauczyciele.Imie, nauczyciele.Nazwisko`
2. `FROM przedmioty INNER JOIN nauczyciele`
3. `ON przedmioty.IDP=nauczyciele.ID_przedmiotu`
4. `WHERE przedmioty.Przedmiot="język angielski"`
5. `OR przedmioty.Przedmiot="język niemiecki";`

Odpowiedź: Lista nauczycieli uczących języków obcych wygląda następująco:

Imie	Nazwisko
Daria	Spokojna
Ewa	Gotek
Janusz	Sosnowski
Arkadiusz	Miałkowski
Michał	Lipiec
Monika	Wierzbowiecka
Gniewomir	Bystry
Zbigniew	Mądry
Piotr	Wiodnicki
Alicja	Witek

Zadanie 1.4 (0–1)

Które cele i ile razy były realizowane w czerwcu?

Rozwiązanie

Tagi: kwerenda wybierająca, agregowanie treści, eksportowanie danych

Krok 1

Tworzymy nową kwerendę w widoku projektu. Dodajemy tabele `cele` i `plan_wycieczek`. Wybieramy **Narzędzia kwerend** → **Projektowanie** → **Pokazywanie/ukrywanie** → **Sumy**, aby włączyć wiersz **Suma**.

Krok 2

Dodajemy pola `Cel` i `IDC` z tabeli `cele`. Dla pola `IDC` ustawiamy **Suma** na **Policz** i kolumnie kwerendy nadajemy nazwę `Liczba_realizacji`. Dodajemy też pole `Miesiac` z tabeli `plan_wycieczek`, wyłączamy widoczność tego pola i ustawiamy **Suma** na **Gdzie**. Dopisujemy w kryteriach słowo „czerwiec”.

Pole:	Cel	Liczba_realizacji: IDC	Miesiac
Tabela:	cele	cele	plan_wycieczek
Suma:	Grupuj według	Suma	Gdzie
Sortuj:			
Pokaż:	☒	☒	☐
Kryteria:			"czerwiec"

Zapytanie w języku SQL:

```
1. SELECT cele.Cel, COUNT(cele.IDC)
2. AS Liczba_realizacji
3. FROM cele INNER JOIN plan_wycieczek
4. ON cele.IDC=plan_wycieczek.ID_celu
5. WHERE plan_wycieczek.Miesiac="czerwiec"
6. GROUP BY cele.Cel;
```

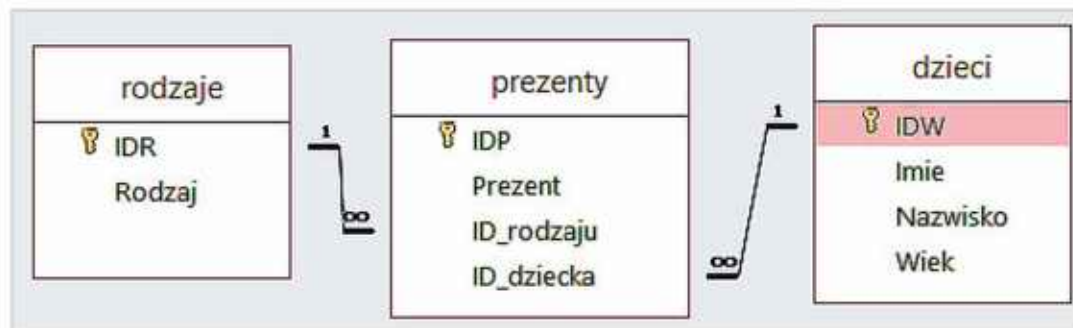
Odpowiedź: Lista zawierająca informacje o tym, ile razy poszczególne cele zrealizowano w czerwcu, wygląda następująco:

Cel	Liczba_realizacji
Badanie elementów przyrody	3
Nauka o zastosowaniach komputerów	1
Opanowanie zasad bezpieczeństwa w życiu codziennym	1
Poznanie historii Polski i Europy	1
Poznanie literatury polskiej	1
Praktyczne wykorzystywanie praw fizyki	1
Rozumienie zjawisk społecznych	1
Rozwijanie sprawności fizycznej	1
Rozwijanie umiejętności matematycznych	1
Używanie języka obcego w praktyce	1

Trening

Zadanie 1

Pewna fundacja zakupiła prezenty gwiazdkowe dla dzieci z domu dziecka. Upominki rozdzielono na dwie partie. Pierwsza partia dotarła na miejsce i została już zapakowana. Podpisano również, który prezent ma trafić do którego dziecka. Informacje o zapakowanych prezentach zawarte są w tabeli prezenty, a jej układ jest następujący: identyfikator prezentu, nazwa prezentu, identyfikator rodzaju prezentu, identyfikator dziecka. W tabeli dzieci znajdują się informacje o podopiecznych: identyfikator, imię, nazwisko, wiek. Tabela rodzaje składa się z identyfikatora rodzaju prezentu oraz jego nazwy. Relacje (związki) między tabelami wyglądają następująco:



Rozwiąż poniższe zadania. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.1 (0–1) **ROZWIĄŻ BEZ KOMPUTERA**

Napisz zapytanie w języku SQL, które wyświetli listę wszystkich dzieci oraz przypisanych do nich prezentów. Listę posortuj alfabetycznie według nazwisk.

Zadanie 1.2 (0–1) **ROZWIĄŻ BEZ KOMPUTERA**

Napisz zapytanie w języku SQL, które wyświetli listę wszystkich rodzajów prezentów wraz ze średnią wieku dzieci, dla których przeznaczono prezenty danego rodzaju. Posortuj listę według średniej wieku dzieci.

Informacje do zadań 1.3 i 1.4

W plikach, które otrzymasz od nauczyciela (np. *WUZ4_zad1_prezenty_trening.txt*, *WUZ4_zad1_rodzaje_trening.txt*, *WUZ4_zad1_dzieci_trening.txt*), znajdują się przykładowe dane z tabel bazy danych.

Oto 5 pierwszych wierszy każdego pliku:

WUZ4_zad1_prezenty_trening.txt

IDP	Prezent	ID_rodzaju	ID_dziecka
1	Lokomotywa	3	1
2	Zestaw kreatywny	9	3
3	Samochód Carbro	2	4
4	Laleczka Betty	5	5

WUZ4_zad1_rodzaje_trening.txt

IDR	Rodzaj
1	Przytulanka
2	Pojazd kołowy bez napędu
3	Pojazd kołowy z napędem
4	Samolocik

WUZ4_zad1_dzieci_trening.txt

IDW	Imie	Nazwisko	Wiek
1	Michał	Antonowski	8
3	Daria	Drzymała	10
4	Marcin	Zwierzyński	6
5	Julia	Niewidzicka	7

Wykorzystaj dane z plików do rozwiązywania poniższych zadań. Odpowiedzi zapisz w sposób wskazany przez nauczyciela.

Zadanie 1.3 (0–1) **ROZWIĄŻ NA KOMPUTERZE**

Podaj imiona i nazwiska wszystkich dzieci w wieku poniżej 10 lat, które otrzymają prezenty z kategorii (rodzaju) „Przybory” lub „Książka”.

Zadanie 1.4 (0–1) **ROZWIĄŻ NA KOMPUTERZE**

Do których dzieci nie zostały jeszcze przypisane prezenty?

Relacyjne bazy danych

Wyobraź sobie, że...

...jesteś analitykiem biznesowym w firmie zajmującej się sprzedażą podzespołów elektronicznych. Dysponujesz dużą ilością danych zapisanych w wielu **tabelach** zebranych w **systemie zarządzania bazą danych** i połączonych za pomocą **relacji**. Dzięki dostępnym narzędziom, głównie wykorzystaniu **kwerend**, tworzysz i przekazujesz zarządowi szczegółowe **raporty**, co pozwala podejmować decyzje zwiększające przychody firmy, a także zadowolenie klientów.



...zostałaś wolontariuszką w schronisku dla zwierząt. Ze zdziwieniem stwierdzasz, że schronisko gromadzi informacje o szczepieniach podopiecznych i inne ważne dane w papierowych zeszytach. Postanawiasz przenieść te informacje do **relacyjnej bazy danych**. Tworzysz **formularze**, aby inni wolontariusze mogli łatwo wprowadzać dane. Twoje zaangażowanie zostaje docenione i dostajesz propozycję stałej pracy w schronisku.

...jesteś trenerem personalnym i w krótkim czasie zdobyłeś wielu klientów. Chcesz, aby ich treningi były efektywne, dlatego tworzysz **relacyjną bazę danych** i łączysz jej tabele za pomocą **związków**. Ponieważ uwzględniasz **selekcję wyników zapytania** oraz **projekcję atrybutów**, uzyskujesz szczegółowe dane o rezultatach treningów swoich klientów. Dzięki temu dostosowujesz intensywność ćwiczeń do indywidualnych możliwości, a twoi klienci osiągają lepsze wyniki.



...prowadzisz sklep z własnoręcznie robionymi zabawkami i chcesz przenieść swoją działalność do internetu. Dzięki znajomości **języka PHP** tworzysz aplikację do składania zamówień. Łączysz ją z bazą danych o produktach, klientach i zamówieniach, którą przygotowałaś, wykorzystując **język SQL**. Za pomocą **zapytań do bazy danych** uzyskujesz wiele cennych informacji. Na ich podstawie przygotowujesz spersonalizowane newslettery z nowymi produktami. Dzięki temu grono klientów szybko się powiększa, a sprzedaż rośnie.



Pułapki cyfrowego świata

Rozwój społeczeństw, a szczególnie gospodarki, edukacji i usług publicznych, jest obecnie w dużej mierze możliwy dzięki sprawnemu przepływowi danych w internecie. Dlatego tak ważny stał się problem bezpieczeństwa w sieci. Wiele się o nim mówi zarówno w związku z przesyłaniem danych, jak i zachowaniami użytkowników internetu. W ramach tego projektu zespołowego zbierzecie informacje na wybrany temat, który będzie dotyczył zagrożeń w środowisku usług sieciowych i umiejętności radzenia sobie z nimi. Następnie zorganizujecie dyskusję panelową na ten temat.



Cel projektu

Przygotowanie się do roli ekspertów z wybranego zagadnienia dotyczącego zagrożeń i pułapek pojawiających się podczas korzystania z usług internetowych, a następnie zorganizowanie i przeprowadzenie dyskusji panelowej na ten temat.

To trzeba wiedzieć

Czym jest dyskusja panelowa?

Dyskusja panelowa (zwana też panelem) jest uważana za skuteczny sposób uczenia się, ponieważ łączy przygotowane wypowiedzi ekspertów z możliwością zadawania pytań przez słuchaczy. Najpierw każdy z ekspertów przedstawia swój punkt widzenia na zadany temat w stosunkowo krótkim wystąpieniu. Może ono zawierać także przekaz multimedialny. Po wysłuchaniu wszystkich prezentacji uczestnicy panelu mogą zadawać ekspertom pytania. Całą dyskusję prowadzi moderator.

Specjalistyczne serwisy

Wiarygodne informacje na temat cyberbezpieczeństwa można znaleźć w artykułach i raportach zespołu CERT, działającego w strukturach Naukowej i Akademickiej Sieci Komputerowej. Cennym źródłem są też serwisy specjalistyczne takie jak niebezpiecznik.pl czy zaufanatrzeciastrona.pl. Ich twórcy omawiają konkretne przypadki naruszenia bezpieczeństwa, luki w systemach i oprogramowaniu, opracowują także tematyczne podcasty. Warto również sięgnąć po serwisy administracji rządowej oraz bazy aktów prawnych.

Propozycje tematów

1. Jak bezpiecznie kupować w internecie?
2. Czy prywatność jest dobrem luksusowym? Jak ją chronić?
3. Jak bezpiecznie prowadzić edukację zdalną i hybrydową?
4. Biały wywiad. Czy jest się czego obawiać?
5. Jak bezpiecznie kształtować cyfrową tożsamość?
6. Co o mnie wie mój telefon i jak go zabezpieczyć?
7. Jak skutecznie niszczyć dane (nie tylko cyfrowe)?
8. Jak bezpiecznie posługiwać się kartami płatniczymi w Polsce i za granicą?

Etapy projektu

1. Wybierzcie jeden z tematów i podzielcie się na zespoły, w których będziecie go opracowywać.
2. Poszukajcie dodatkowych informacji o tym, czym jest dyskusja panelowa, po co i w jakich sytuacjach się ją stosuje. Wysłuchajcie fragmentu takiej dyskusji na dowolny temat.
3. Ustalcie z nauczycielem, do kogo będzie skierowany panel dyskusyjny. Mogą to być np. wasi rówieśnicy, uczniowie klas pierwszych, nauczyciele lub uczniowie szkoły podstawowej.
4. Przeanalizujcie temat i podzielcie go na mniejsze zagadnienia, a następnie opracujcie je w zespole. Przygotujcie się jak najlepiej, ponieważ jedna osoba z zespołu zostanie ekspertem, któremu słuchacze będą zadawać pytania. Temat możecie opracować w formie mapy myśli.
5. Przygotujcie opisy realnych przykładów naruszeń bezpieczeństwa i przedstawcie kilka różnych scenariuszy rozwoju każdej z takich sytuacji.
6. Wybierzcie wraz z nauczycielem osobę, która będzie moderatorem dyskusji, a z każdego zespołu wyłóćcie jednego eksperta. Moderator ustali z nauczycielem zagadnienia oraz czas wystąpień ekspertów.
7. Przygotujcie wskazanego eksperta do wystąpienia na podstawie mapy myśli.
8. Opracujcie materiały, które wzmocnią przekaz podczas panelu, np.:
 - prezentację multimedialną,
 - krótki, prosty film,
 - animację,
 - zestaw dodatkowych rekwizytów.
9. Zorganizujcie w zespole co najmniej jedno spotkanie w trybie zdalnym, które będzie próbnym wystąpieniem eksperta z wykorzystaniem prezentacji multimedialnej lub rekwizytów.
10. Przeprowadźcie spotkanie organizacyjne między ekspertami i moderatorem, aby ustalić zasady panelu.
11. Zastanówcie się, jak możecie udokumentować dyskusję (np. w formie artykułu lub wideo), oraz czy możecie ją odbyć w trybie zdalnym.
12. Przeprowadźcie dyskusję panelową wraz z częścią poświęconą pytaniom od publiczności.
13. Podsumujcie efekty dyskusji.

Przykładowe narzędzia

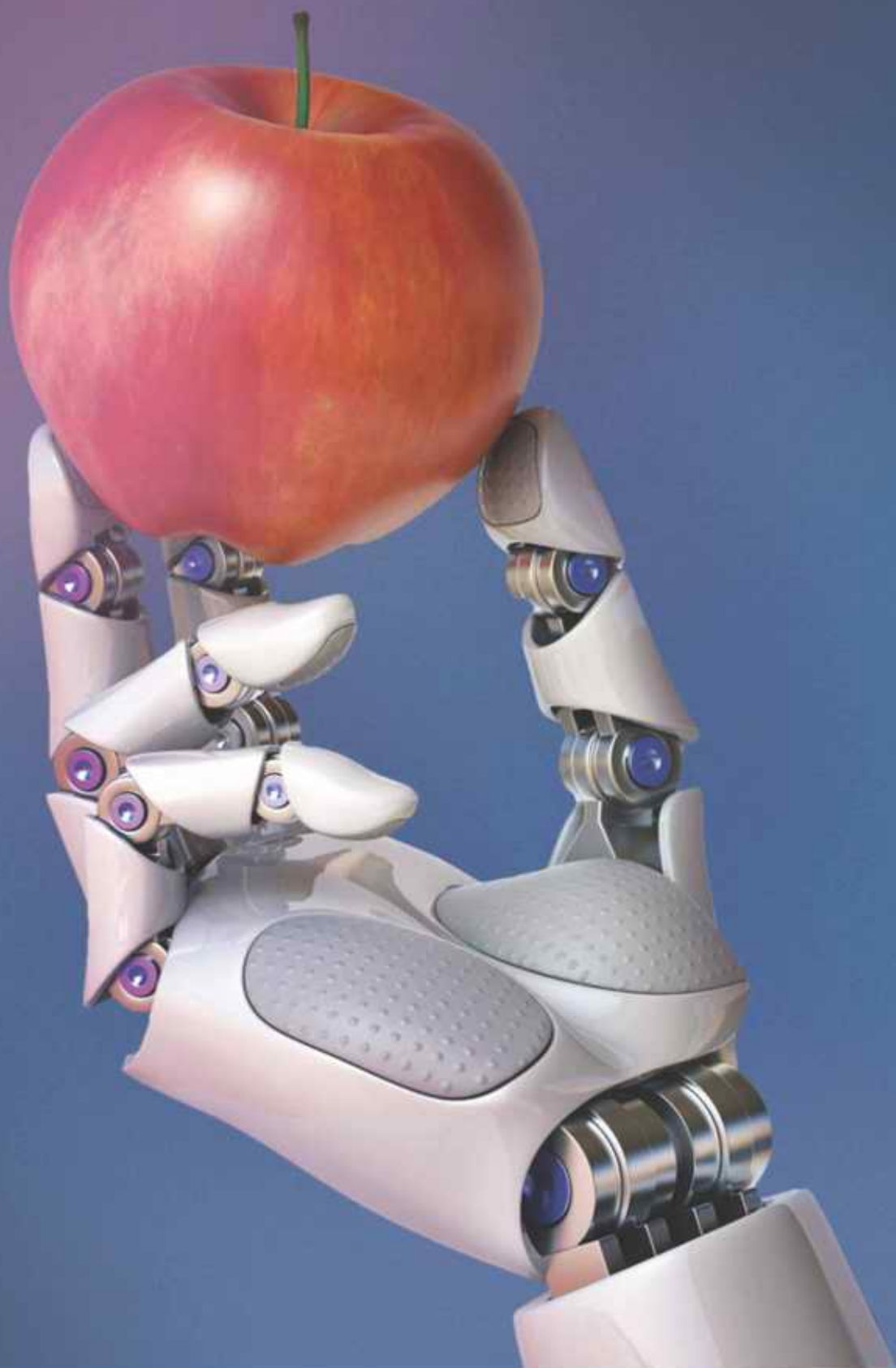
FreeMind, Freeplane – narzędzia do budowy mapy myśli.

Sway – narzędzie do szybkiego tworzenia serwisów WWW oraz prezentacji multimedialnych, ułatwiające łączenie zasobów własnych i dostępnych w sieci.

Microsoft Teams, Zoom, Google Meet – narzędzia do prowadzenia spotkań zdalnych.

Microsoft PowerPoint, Keynote – programy do tworzenia prezentacji w formie slajdów.

OneDrive, iCloud, Google Drive – dyski wirtualne do przechowywania materiałów.



5

Rozwiązywanie różnych problemów z wykorzystaniem komputera

- 19. Sterujemy robotem
- 20. Sztuka publikowania w sieci
- 21. Grafiki informacyjne

Rozwiązywanie różnych problemów z wykorzystaniem komputera

Pierwsze mechaniczne zabawki pojawiły się już w starożytności. Dziś roboty są ważną częścią codziennego życia – nie tylko zastępują ludzi przy zajęciach monotonnych i niebezpiecznych, lecz także są źródłem rozrywki i pozwalają zdobywać nowe umiejętności.



Robotyce nieodłącznie towarzyszy **mechanika**. Za jej twórcę uznawany jest Archytas z Tarentu, który ok. 420 r. p.n.e. skonstruował drewnianego gołębia, zasilanego sprężonym powietrzem i mogącego przelecieć nawet do 200 metrów. Mechaniczne marionetki były także znane i używane w Chinach. W III w. n.e. Ma Jun skonstruował cały mechaniczny teatr lalek napędzany kołem wodnym.

420 r. p.n.e.



Pierwszą **taśmę montażową** uruchomiono 1913 r. w fabryce samochodów Forda w amerykańskim Detroit. Zmiana organizacji pracy pozwoliła skrócić czas montażu auta z 12,5 do 1,5 godziny. Kluczowym elementem było zastąpienie wysoko wykwalifikowanych pracowników osobami wyszkolonymi do montażu konkretnych elementów samochodu na każdym z 84 stanowisk. Dzięki temu znacząco spadła cena samochodu.

1913 r.

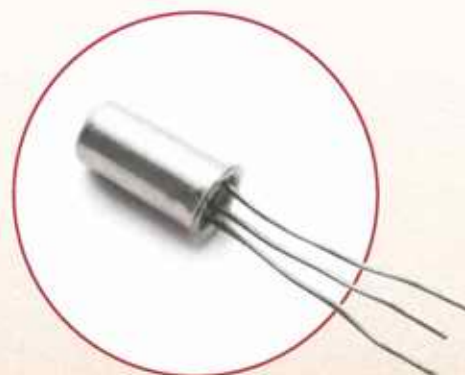
1774 r.



Szwajcarski zegarmistrz i mechanik Pierre Jaquet-Droz skonstruował razem z synami trzy roboty humanoidalne, które nazwał **automatami**. Najbardziej złożonym był Pisarz – składał się z ok. 6 tys. ruchomych elementów. Robot ten nie tylko się poruszał, lecz także pisał gęsim piórem na papierze. Był wyposażony w urządzenie wejściowe, za pomocą którego można było zdefiniować pisane przez niego litery. Ten element uznaje się za jedną z pierwszych programowalnych pamięci.

1947 r.

Pierwszy działający **tranzystor** skonstruowali w 1947 r. fizycy pracujący w Bell Laboratories – John Bardeen, Walter Houser Brattain i William Bradford Shockley. Wynalezienie tranzystora, a następnie układów scalonych zapoczątkowało rozwój elektroniki na szeroką skalę, umożliwiło powstanie znanych dzisiaj komputerów oraz udoskonalanie automatów i robotów.





W 1950 r. brytyjski matematyk i kryptolog Alan Turing opublikował pionierską pracę dotyczącą sztucznej inteligencji – *Computing Machinery and Intelligence* (*Maszyna licząca a inteligencja*). Turing zaproponował w niej test sprawdzający, czy maszyna może przejawiać inteligentne zachowania tak jak człowiek. Test uważano za zdany, jeśli człowiek nie był w stanie określić, czy rozmawia z innym człowiekiem czy z maszyną, przy czym obie strony miały się zachowywać w sposób najbardziej zbliżony do ludzkiego. Dziś ten test jest nazywany **testem Turinga**.

1950 r.

W 2012 r. stan Nevada w USA zezwolił na poruszanie się **samochodów autonomicznych** po drogach publicznych. Zezwolenie to rozszerzono na autonomiczne samochody ciężarowe w roku 2015. Dopuszczanie pojazdów autonomicznych do jazdy w ruchu publicznym nie jest powszechne, a prace nad wprowadzeniem odpowiedniego prawa wciąż trwają.



2012 r.

1997 r.



4 lipca 1997 r. na powierzchni Marsa wylądowała amerykańska sonda kosmiczna Pathfinder. Znajdujący się na jej pokładzie **łazik Sojourner** był pierwszym pojazdem poruszającym się po powierzchni obcej planety, którym zdalnie sterowano z Ziemi. Jego misja zakończyła się sukcesem – przesłał na Ziemię 550 zdjęć i przeprowadził analizę składu chemicznego skał oraz otworzył drogę kolejnym, coraz bardziej złożonym robotom badawczym.

2017 r.

Sophia, humanoidalny robot firmy Hanson Robotics z Hongkongu, to pierwszy **robot**, który uzyskał **obywatelstwo** – w 2017 r. nadała mu je Arabia Saudyjska. Sophia jest wyposażona w sztuczną inteligencję, dzięki czemu może się uczyć, dostosowywać do ludzkich zachowań i sprawniej komunikować. Ma zaimplementowane systemy rozpoznawania mowy oraz obrazu. Sophia udziela wywiadów na całym świecie, w 2018 r. można było jej wysłuchać w Krakowie.



19. Sterujemy robotem

Od wielu lat tematy związane z automatyką i robotyką zyskują na popularności. Uczelnie wyższe wykorzystują roboty na różnych kierunkach studiów, przedsiębiorstwa używają ich m.in. do przyspieszenia pracy, zwiększenia jej precyzji i wydajności oraz do jej automatyzacji. Roboty już od ponad 100 lat nie tylko towarzyszą nam w literaturze i filmie, lecz także pod różnymi postaciami są obecne w naszym życiu codziennym. Ułatwiają wiele prac, a niekiedy wręcz nas wyręczają. W tym temacie nauczysz się programować roboty edukacyjne oraz symulować ich działanie.

Cel lekcji

- Dowiesz się, jak się definiuje roboty.
- Poznasz budowę oraz parametry wybranych robotów edukacyjnych.
- Nauczysz się programować roboty z wykorzystaniem różnych środowisk dostępnych online.
- Przeprowadzisz symulacje zaprogramowanych robotów w środowisku wirtualnym.

19.1. Czym są roboty?

Warto wiedzieć

Pod koniec lat 50. XX w. polski pisarz i futurysta Stanisław Lem napisał opowiadanie *Czy pan istnieje, Mr. Johns?*, które dotyczy problemu cyborgizacji, czyli zastępowania narządów człowieka urządzeniami mechanicznymi.

Na podstawie opowiadania Andrzej Wajda nakręcił film pt. *Przekładaniec*.

Warto wiedzieć

Ze względu na powszechne użycie robotów w przemyśle powstała norma PN-EN ISO 8373:2001, która definiuje robota przemysłowego.

Słowo „robot” upowszechniło się w latach 20. XX w. po publikacji sztuki *R.U.R.* autorstwa Karela Čapka – czeskiego pisarza, jednego z pionierów fantastyki naukowej (rysunek 19.1 przedstawia robota inspirowanego tą sztuką). *R.U.R.* to skrót od *Rossumovi Univerzální Roboti*, co można przetłumaczyć jako „Roboty Uniwersalne Rossuma”. Sztuka ta miała swoją polską prapremierę w 1922 r. w warszawskim Teatrze Rozmaitości.

Zaledwie kilka lat później roboty pojawiły się w filmie *Metropolis* w reżyserii Fritza Langa i towarzyszą nam do dziś na ekranach kin i telewizorów. Do najpopularniejszych należą: para C-3PO i R2-D2 z *Gwiezdnych wojen*, roboty serii T (T-800, T-X) z filmów o Terminatorze czy też tytułowy bohater animacji *WALL-E*.

Samo pojęcie robota definiowane jest różnie w zależności od przeznaczenia. Według Encyklopedii Powszechnej PWN robot to „cybernetyczna (sterowana) maszyna lokomocyjno-manipulacyjna przeznaczona do wspomagania lub realizacji czynności energetyczno-ruchowych, sensualnych i intelektualnych człowieka”.



Rys. 19.1. Robot inspirowany sztuką *R.U.R.*

Przyjmijmy nieco uproszczoną definicję. **Robotem** będziemy nazywać urządzenie możliwe do zaprogramowania lub sterowania, które jest zdolne do poruszania się (wykonywania różnych ruchów w przestrzeni), a jego zadaniem jest ułatwienie człowiekowi pracy lub zastąpienie go w skomplikowanych, powtarzalnych lub monotonnych czynnościach.

19.2. Roboty w edukacji

Dzięki różnorodnym robotom edukacyjnym (rys. 19.2) programowania robotów może się dziś nauczyć niemal każdy, niezależnie od wieku czy etapu edukacji szkolnej.



Rys. 19.2. Przykłady robotów edukacyjnych

Wraz z robotem otrzymuje się dostęp do środowiska, w którym można zaprogramować jego działanie. Istnieją też środowiska programistyczne obsługujące roboty różnych producentów.

Brak robota nie stanowi problemu, ponieważ mogą go zastąpić **symulatory robotów**. Są to środowiska, które umożliwiają programowanie oraz obserwację zachowania wirtualnych robotów (często cyfrowych odpowiedników rzeczywistych robotów).

Robotów edukacyjnych jest wiele, regularnie pojawiają się też nowe modele. Dlatego omówimy tylko wybrane roboty, które spełniają następujące kryteria ważne w edukacji:

- ▶ popularność w literaturze i zasobach internetowych – zapewnia to dostęp do obszernej bazy wiedzy na temat robota, jego budowy, programowania, zastosowań i przykładów realizacji,
- ▶ różnorodność dostępnych środowisk programowania – pozwoli to wybrać odpowiedni język i środowisko programowania, niezależnie od posiadanego systemu operacyjnego, z możliwością użycia narzędzi online i offline, języka tekstowego lub wizualnego,
- ▶ możliwość symulacji robota – dzięki temu można uczyć się robotyki na ekranie komputera, symulując działanie robota dokładnie tak, jak zachowywałby się w rzeczywistości.

• Symulator robota

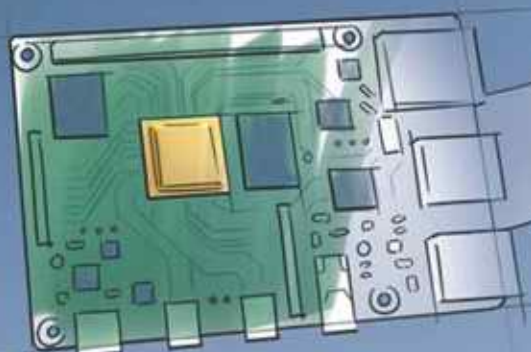
Warto wiedzieć

Popularnymi robotami edukacyjnymi są:

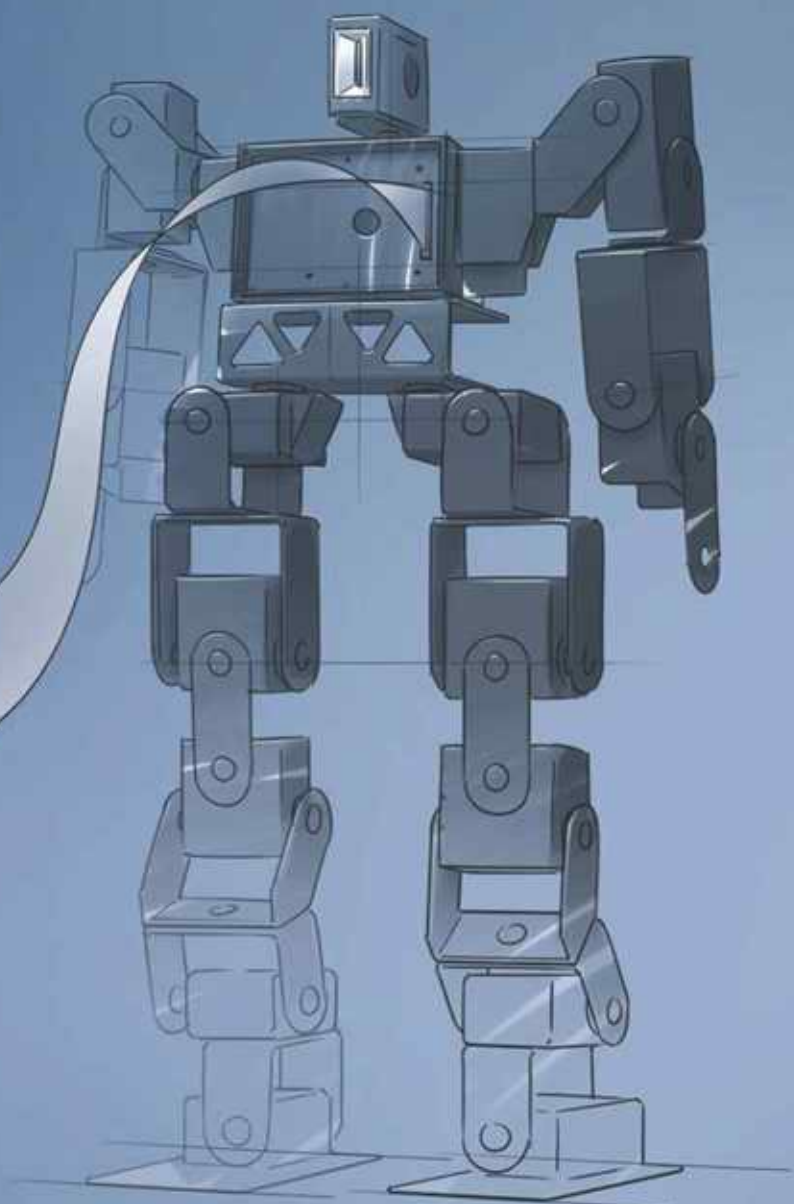
- ▶ Edison,
- ▶ Dash i Dot,
- ▶ mBot,
- ▶ Photon,
- ▶ Ozobot,
- ▶ LEGO MINDSTORMS,
- ▶ micro:Maqueen.

Co warto wiedzieć o robotach i robotyce?

Nauka robotyki jest świetnym sposobem na rozwijanie umiejętności z obszaru nauk ścisłych. Daje także dużo satysfakcji i jest zajęciem bardzo perspektywicznym. Roboty mogą się między sobą znacząco różnić, dlatego warto przyjrzeć im się bliżej.



Robotem steruje komputer wyposażony w procesor. W nim wykonywane są programy, które mogą wykorzystywać dane pobrane z czujników umieszczonych w robocie.



Roboty rozrywkowe

Niektóre roboty dostarczają człowiekowi rozrywki. Mogą mieć różną postać, np. imitować ludzi lub zwierzęta.



Robot zabawka

- Waga: 1,6 kg
- Czas działania: 1,5 h
- Liczba kamer: 1
- Czujniki: mikrofon, kamera, czujnik temperatury

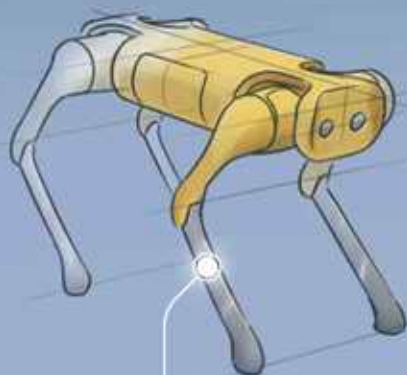


Robot humanoidalny

- Waga: 18 kg
- Uchwyt: 0,6 kg (chwyt dłonią)
- Liczba kamer: do 4
- Czujniki: widzenie głębi 3D, kamera oka, mikrofon, kompas
- Sztuczna skóra twarzy i szyi
- System rozpoznawania i wyrażania emocji

Roboty specjalistyczne

Konstrukcja robotów specjalistycznych jest zróżnicowana. Dron musi być lekki i szybki. Robot saperski ma rozbudowany system kamer i precyzyjne ramię. Robot penetracyjny powinien sprawnie docierać do trudno dostępnych miejsc.



Robot penetracyjny

- Waga: 32 kg
- Maks. prędkość: 5 km/h
- Maks. udźwig: 14 kg
- Czas działania: 1,5 h
- Zakres temperatur: od -20°C do 45°C

Robot saperski

- Waga: 220 kg
- Maks. prędkość: 4,8 km/h
- Maks. udźwig: 60 kg
- Czas działania: 3–4 h
- Liczba kamer: 5



Dron

- Waga: 4,5 kg
- Maks. prędkość: 82 km/h
- Maks. pułap: 3 km
- Maks. czas lotu: 32 min
- Liczba kamer: 1

Roboty pomagające człowiekowi

Zadaniem niektórych robotów jest pomoc człowiekowi w codziennych czynnościach, np. opiece nad osobami starszymi lub w przemieszczaniu się.

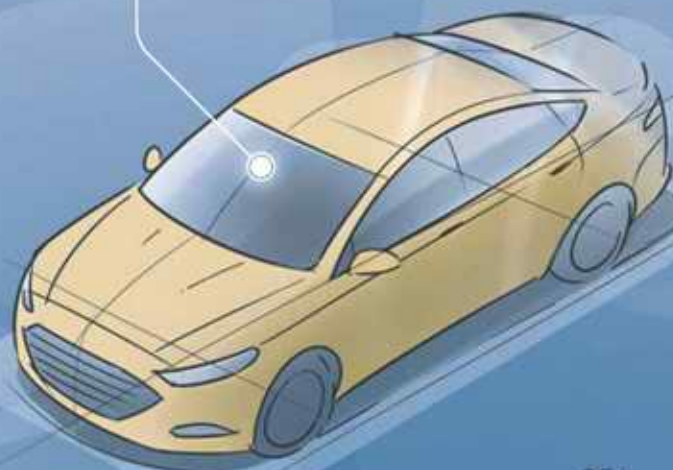


Robot opiekun

- Waga: 3 kg
- Czujniki: kamera, mikrofon, głośnik
- Funkcje: rozrywka (np. karaoke, filmy, zabawy), ciśnieniomierz, rozmowy wideo, stymulowanie pamięci, alarmy (np. na żądanie lub po dłuższej nieobecności podopiecznego)

Samochód autonomiczny

- Waga: 2000 kg
- Liczba kamer: 8
- Czujniki: 12 czujników ultradźwiękowych, radar, sonar
- Systemy: GPS, mapowanie 3D, chmura obliczeniowa, system komunikacji, uczenie maszynowe



19.3. Robot LEGO MINDSTORMS EV3

Warto wiedzieć

Robot EV3 jest następcą zestawu NXT 2.0.

Jednym z popularnych na rynku edukacyjnym robotów jest zestaw LEGO MINDSTORMS EV3, który daje bardzo duże możliwości twórcze. Trzonem tego zestawu jest inteligentna kostka (widoczna na rys. 19.3) – programowalny komputer wyposażony w procesor ARM9.

Kostka EV3 zawiera:

- ▶ podświetlany interfejs,
- ▶ przyciski sterujące,
- ▶ ekran,
- ▶ głośnik,
- ▶ port USB,
- ▶ czytnik kart miniSD,
- ▶ cztery porty wejściowe,
- ▶ cztery porty wyjściowe.

Zestaw EV3 wspiera takie technologie jak Bluetooth oraz wi-fi. Pozwala również na programowanie zarówno z wykorzystaniem narzędzi komputerowych (specjalnych środowisk programistycznych), jak i bezpośrednio z poziomu samej kostki.



Rys. 19.3. Kostka EV3 wraz z podłączonymi do niej silnikami i czujnikami

Serwomotor ➤ W zestawie EV3 wykorzystywane są **serwomotory**, czyli silniki służące do poruszania elementów konstrukcji robota. Do jednej kostki EV3 można podłączyć cztery niezależne silniki. EV3 można również wyposażać w następujące czujniki:

Czujnik kolorów ➤ **czujnik kolorów** – mierzy barwę oraz natężenie światła,

Czujnik podczerwieni ➤ **czujnik podczerwieni** – wykrywa źródła podczerwieni, np. z nadajnika sterującego robotem,

Czujnik żyroskopowy ➤ **czujnik żyroskopowy** – wykrywa ruch obrotowy robota i zmianę jego położenia,

Czujnik dotykowy ➤ **czujnik dotykowy** – wykrywa naciśnięcie lub zwolnienie przycisku,

Czujnik ultradźwiękowy ➤ **czujnik ultradźwiękowy** – mierzy odległość od obiektów,

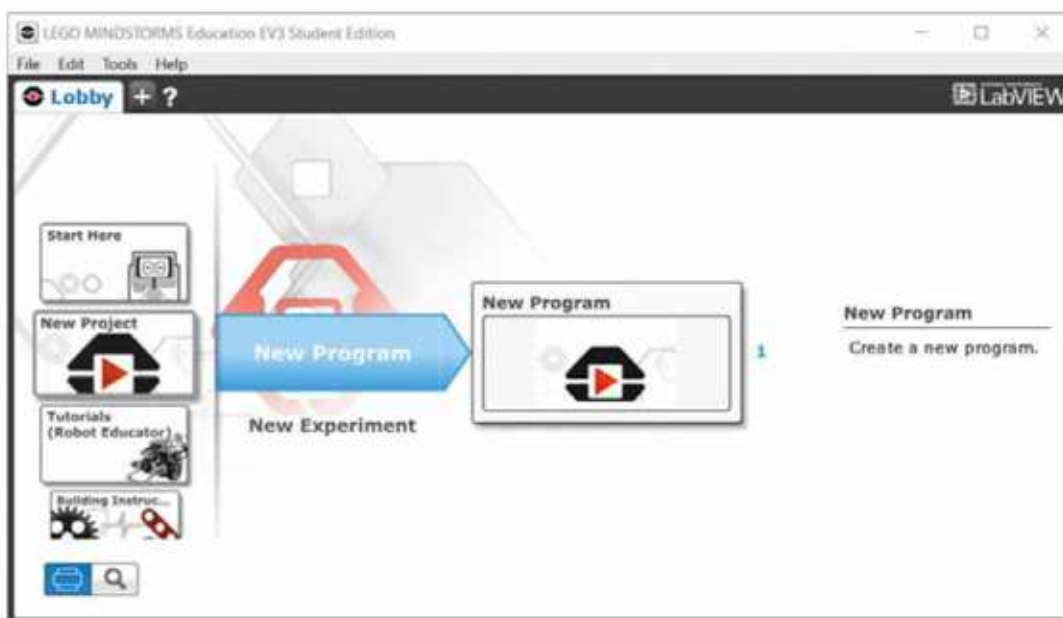
Czujnik temperatury ➤ **czujnik temperatury** – mierzy temperaturę w zakresie od -20°C do $+120^{\circ}\text{C}$.

Dzięki możliwości łączenia kostki EV3 z innymi klockami producenta zestaw ten pozwala konstruować wiele różnorodnych robotów edukacyjnych.

Środowisko LEGO MINDSTORMS

Podstawowym narzędziem programowania kostki EV3 i podłączonych do niej silników oraz czujników jest oprogramowanie firmy LEGO przeznaczone dla zestawu MINDSTORMS. Jest ono darmowe i można je pobrać ze strony producenta.

Po uruchomieniu środowiska w systemie Windows i wybraniu opcji **New Program** (rys. 19.4) pojawi się zakładka, w której za pomocą bloków symbolizujących elementy zestawu tworzymy program dla naszego robota. Dodatkowo możemy również zbudować jednego z zaprojektowanych wcześniej robotów według przygotowanej instrukcji (narzędzie **Building Instructions**).



Rys. 19.4. Środowisko LEGO MINDSTORMS

Programowanie robota EV3 w środowisku LEGO MINDSTORMS

Przyjmijmy, że robot ma nam zakomunikować początek wykonywania programu, a następnie poruszać się do przodu, dopóki nie naciśniemy przycisku w podłączonym do robota czujniku dotyku.

Pierwsza instrukcja, z której skorzystamy, będzie odpowiadała za wypowiadanie komunikatu dźwiękowego, np. „Hello”. Następnie użyjemy pętli, aby robot poruszał się do momentu wciśnięcia odpowiedniego czujnika.

Rysunek 19.5 przedstawia przykładowy program.



Rys. 19.5. Program zapisany w środowisku LEGO MINDSTORMS

👍 Dobra rada

Jeśli jesteś użytkownikiem systemu macOS, do zaprogramowania robota EV3 możesz wykorzystać również oprogramowanie LEGO MINDSTORMS. Przypomina ono nieco środowisko Scratch.

👍 Dobra rada

Jeśli jesteś użytkownikiem systemu Linux, robota EV3 możesz zaprogramować za pomocą systemu operacyjnego ev3dev (opartego na dystrybucji Debian). Robota możesz uruchomić z wykorzystaniem karty pamięci microSD i programować m.in. w językach C++ lub Python.

Program składa się z bloku początku i bloku pętli (kolor pomarańczowy) oraz dwóch bloków instrukcji (kolor zielony). Kolejne instrukcje odczytujemy od lewej do prawej. Po przesłaniu programu do kostki EV3 i uruchomieniu go robot przywita się (pierwsza zielona instrukcja), a następnie będzie się poruszał. W pętli wykonywane będą obroty silników (z mocą 20%) podłączonych do portów B i C kostki EV3. Pętla będzie wykonywana, dopóki nie wciśniemy czujnika dotyku podłączonego do portu 1 kostki EV3.

19.4. Mikrokontroler BBC micro:bit

Warto wiedzieć

Kolejna wersja płytki micro:bit o nazwie BBC micro:bit 2 zawiera dodatkowo głośnik i mikrofon oraz przycisk dotykowy. Wyposażona jest w tryb oszczędzania energii i ma większą moc obliczeniową.

Konstruowanie robotów można połączyć z nauką elektroniki. Przykładem takiego rozwiązania jest mikrokontroler BBC micro:bit (rys. 19.6), który miał premierę w 2015 r.

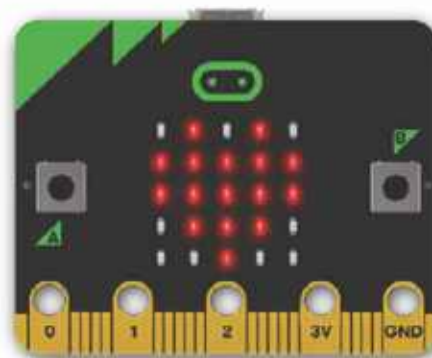
Jest to niewielka płytka elektroniczna wyposażona w procesor ARM Cortex-M0, moduły komunikacji Bluetooth i USB, dwa przyciski oraz dwa wbudowane czujniki:

- Magnetometr** ▶ **magnetometr** – czujnik, który działa na zasadzie kompasu, można dzięki niemu wskazać kierunek północny,
- Akcelerometr** ▶ **akcelerometr** – czujnik mierzący ruch wzdłuż trzech osi: X (przechylenie w lewo/w prawo), Y (przechylenie w przód/w tył), Z (poruszanie się w górę/w dół).

Charakterystycznym elementem płytki micro:bit jest umieszczony na niej ekran składający się z 25 diod LED (5×5), na którym można wyświetlać proste komunikaty graficzne lub tekstowe. Płytka wyposażona jest również w złącze rozszerzeń pozwalające rozbudować ją o dodatkowe akcesoria.

Płytka micro:bit pojawia się w wielu zestawach edukacyjnych z zakresu robotyki. Można dzięki nim budować roboty lub realizować ciekawe projekty rozwijające m.in. umiejętności programistyczne.

Przykładem takiego zastosowania jest robot micro:Maqueen (rys. 19.7), który wykorzystuje właśnie płytke micro:bit oraz czujniki: ultradźwiękowy, wykrywania linii oraz podczerwieni (umożliwia on sterowanie robotem za pomocą pilota).



Rys. 19.6. Mikrokontroler BBC micro:bit



Rys. 19.7. Przykład robota micro:Maqueen, wykonanego z użyciem mikrokontrolera micro:bit

Programowanie płytki micro:bit w języku Python

Do programowania płytki micro:bit wykorzystuje się język Python, a właściwie jego odmianę – **język MicroPython**, który został zoptymalizowany pod kątem stosowania go w mikrokontrolerach.

Kod źródłowy można tworzyć w środowisku online opartym na edytorze Mu, dostępnym na stronie python.microbit.org/v/2.0.

Aby zobaczyć działanie napisanego kodu, należy przesłać go na płytkę i uruchomić. Przykładowy kod źródłowy programu wyświetlającego napis „Witaj!” oraz symbol serca przedstawiono na rysunku 19.8.

Dokumentację i dokładny opis poleceń języka MicroPython w języku polskim można znaleźć na stronie:

microbit-micropython.readthedocs.io/pl/latest/index.html

Dobra rada

Kod źródłowy możesz tworzyć również w samym edytorze Mu. Aby to zrobić, wybierz **Tryb** → **BBC micro:bit**.



Rys. 19.8. Programowanie płytki micro:bit w języku MicroPython

19.5. Programowanie robotów i symulatory

Jeśli nie mamy dostępu do fizycznego robota, warto wykorzystać narzędzia do programowania i symulacji działania robotów. Dzięki nim można zrealizować projekty o takim samym stopniu skomplikowania i mieć z nich taką samą satysfakcję jak w wypadku pracy z rzeczywistym robotem.

Omówimy środowiska online, które pozwalają programować roboty oraz symulować ich działanie: Open Roberta Lab, Microsoft Make-Code oraz VEXcode VR.

Warto wiedzieć

Symulatory pozwalają programować i testować roboty w bezpiecznych warunkach, bez obaw o przypadkowe błędy, które mogłyby spowodować uszkodzenia sprzętu.

A to ciekawe

Roboty w muzyce

Niemiecki zespół muzyczny Kraftwerk jest uznawany za prekursora muzyki elektronicznej. Muzycy od lat 70. konstruowali własne elektroniczne instrumenty i komponowali na nich swoje utwory. W roku 1978 wydali album *Die Mensch-Maschine*, który odwoływał się do relacji człowiek–maszyna. W utworze *Die Roboter* robotyczny głos z syntezatora dźwięku wypowiada po rosyjsku „Я твой слуга, я твой работник” („Jestem twoim sługą, jestem twoim pracownikiem”). Na koncertach zespołu stale pojawiają się robotyczne sylwetki muzyków.



19.6. Symulacja robota EV3 w środowisku Open Roberta Lab

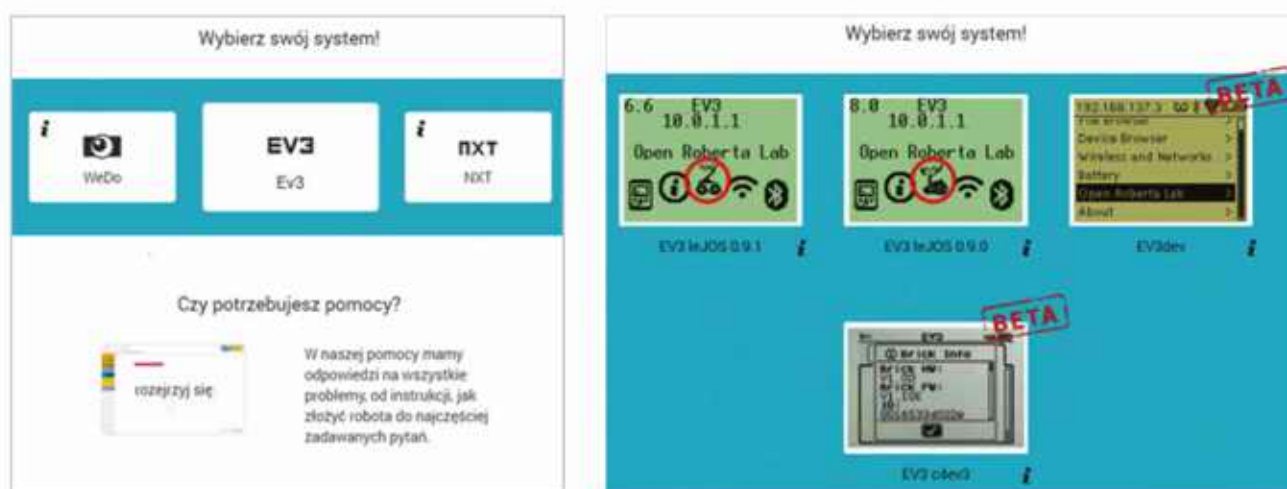
Warto wiedzieć

Okna symulacji robotów są dostępne tylko dla niektórych modeli dostępnych w środowisku Open Roberta Lab.

Open Roberta Lab to darmowe środowisko programowania i symulacji kilku popularnych robotów, w tym LEGO MINDSTORMS EV3. Wykorzystuje ono rozwiązania chmurowe, więc nie wymaga instalacji. Założenie konta na stronie projektu umożliwia zapisanie online wszystkich utworzonych programów.

Programowanie odbywa się za pomocą graficznego interfejsu, w którym program układa się z dostępnych bloków. W zależności od wybranego robota można również podejrzeć kod przetłumaczony m.in. na język Python. Utworzony program można następnie zaimplementować w robocie lub przetestować w oknie symulatora. Serwis Open Roberta Lab jest dostępny w języku polskim.

Po uruchomieniu strony projektu lab.open-roberta.org wybieramy robota oraz system operacyjny. W naszym wypadku wybierzmy robota EV3 i system EV3 c4ev3 (rys. 19.9).



Rys. 19.9. Wybór robota EV3 w środowisku Open Roberta Lab

Rysunek 19.10 przedstawia okno z obszarem roboczym (zrzut z lewej) oraz okno z podglądem konfiguracji robota EV3 (zrzut z prawej).



Rys. 19.10. Obszar roboczy Open Roberta Lab i konfiguracja robota EV3

Aby włączyć podgląd robota, wystarczy wybrać przycisk edycji i opcję **otwórz/zamknij symulator**. Podgląd przedstawia: robota na dwóch kołach, promień czujnika ultradźwiękowego wychwytyujące przeszkodę (niebieski kwadrat), kolorowe prostokąty oraz ścieżkę, po której robot może jeździć, jeśli zostanie tak zaprogramowany. W tym widoku symuluje się działanie robota (rys. 19.11).



Rys. 19.11. Włączanie podglądu, w którym można symulować działanie robota

W oknie symulatora znajduje się kilka przycisków – na rysunku 19.11 oznaczone są one kolejnymi numerami od 1 do 6. Przyciski pozwalają na:

- ▶ uruchomienie programu (1),
- ▶ zmianę tła sceny (2),
- ▶ podgląd programowalnej kostki zestawu EV3 (3),
- ▶ podgląd wartości poszczególnych czujników (4),
- ▶ umieszczenie robota w punkcie początkowym (5),
- ▶ wgranie własnego tła (6).

Program *Zatrzymanie przed przeszkodą*

Zaprogramujemy teraz reakcję robota na spotkanie z przeszkodą: robot zatrzyma się, jeśli wykryje, że przeszkoda znajduje się w odległości 10 cm.

Czujnik ultradźwiękowy,
s. 352 

Aby to zrobić, zmienimy najpierw tło w oknie symulatora (przycisk 2 z rys. 19.11, s. 357) na takie, w którym robot jest ustawiony w pewnej odległości przed przeszkodą (rys. 19.12). Linie przerywane obrazują działanie czujnika ultradźwiękowego.



Rys. 19.12. Robot ustawiony przed przeszkodą

Robotem będzie sterowała pętla, wykonywana do momentu zwrócenia przez czujnik ultradźwiękowy informacji o tym, że odległość od przeszkody jest mniejsza lub równa 10 cm. Poruszanie się robota do przodu uzyskamy przez ustawienie jednakowej prędkości w lewym i prawym silniku.

Kod źródłowy programu *Zatrzymanie przed przeszkodą* przedstawiono na rysunku 19.13.

Kod źródłowy programu  *Zatrzymanie przed przeszkodą*



Rys. 19.13. Kod źródłowy programu *Zatrzymanie przed przeszkodą*

Na rysunku 19.14 przedstawiono wynik symulacji – pogrubiona linia pokazuje tor ruchu robota, a linie wychodzące od robota obrazują działanie sensora ultradźwiękowego.



Rys. 19.14. Efekt symulacji działania robota w programie *Zatrzymanie przed przeszkodą*

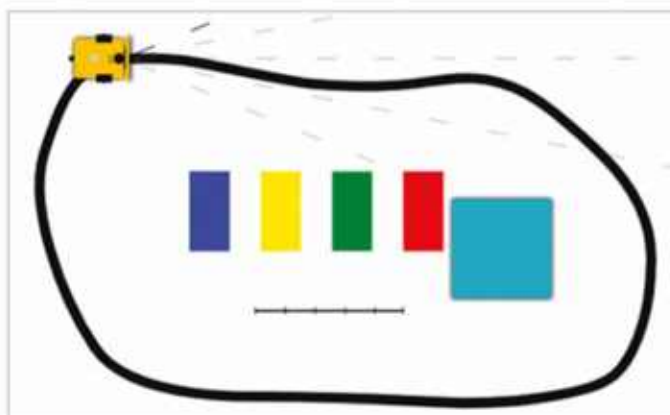
Ćwiczenie 1

- a. Zapisz w środowisku Open Roberta Lab kod źródłowy programu *Zatrzymanie przed przeszkodą* i przeprowadź symulację robota.
- b. Zmodyfikuj program *Zatrzymanie przed przeszkodą* tak, aby po zatrzymaniu robot dwukrotnie wypowiedział słowa „Poprawne zatrzymanie”.

Program *Jazda po ścieżce*

Wykonamy kolejne zadanie: robot będzie jeździł po wyznaczonej ścieżce narysowanej linią ciągłą.

Ustawiamy odpowiednie tło (rys. 19.15) i przesuwamy przeszkodę tak, aby nie kolidowała z robotem podczas jazdy po linii. Robota umieszczamy na czarnej linii.



Rys. 19.15. Ustawienie robota na ścieżce

W rozwiązaniu możemy wykorzystać **czujnik kolorów**, który pozwoli skorygować tor ruchu robota, gdyby wyjeżdżał poza czarną linię. Rysunek 19.16 przedstawia zapis kodu źródłowego programu *Jazda po ścieżce*.



Rys. 19.16. Kod źródłowy programu *Jazda po ścieżce*

W pętli sprawdzany jest kolor, wykrywany przez czujnik koloru. Jedno koło (port B) obraca się do przodu z prędkością 25, drugie zaś (port C) w przeciwnym kierunku z prędkością 3 (wartość -3). Kiedy czujnik koloru nie rozpoznaje czarnego koloru ścieżki, silniki poruszają się w przeciwnie strony. W konsekwencji robot jeździ zygzakiem po czarnej linii. Blok **port silnika** można znaleźć na liście instrukcji oznaczonych gwiazdką i numerem 2 w części **Działania** → **Ruch**.

Ćwiczenie 2

- Zapisz w środowisku Open Roberta Lab kod źródłowy programu *Jazda po ścieżce* i przeprowadź symulację działania robota.
- Zmodyfikuj kod programu *Jazda po ścieżce* tak, aby robot pokonał trasę, jadąc tyłem.

Warto wiedzieć

Podczas jazdy robota po ścieżce czujnik koloru może działać w jednym z trzech trybów: rozpoznawania kolorów, określania natężenia światła odbitego i określania natężenia światła otoczenia. W polskim tłumaczeniu opisu tego czujnika dwa razy pojawia się „światło otoczenia”. Należy pamiętać, że pierwsza opcja dotyczy natężenia światła odbitego, a druga – światła otoczenia.

Czujnik kolorów, s. 352 [↗](#)

Kod źródłowy programu *Jazda po ścieżce*

Dobra rada

Open Roberta Lab udostępnia dla robota EV3 dwie listy instrukcji – jedna jest oznaczona kwadratem, a druga gwiazdką. Na tej drugiej znajdziesz bardziej zaawansowane instrukcje sterujące.

PROGRAM NEPOprog



Zapamiętaj

Producenci robotów poza samymi robotami udostępniają również środowiska pozwalające na ich programowanie. Można korzystać także z innych serwisów – często oferują one wygodne narzędzia do symulacji.

19.7. Symulacja płytki micro:bit w środowisku Microsoft MakeCode

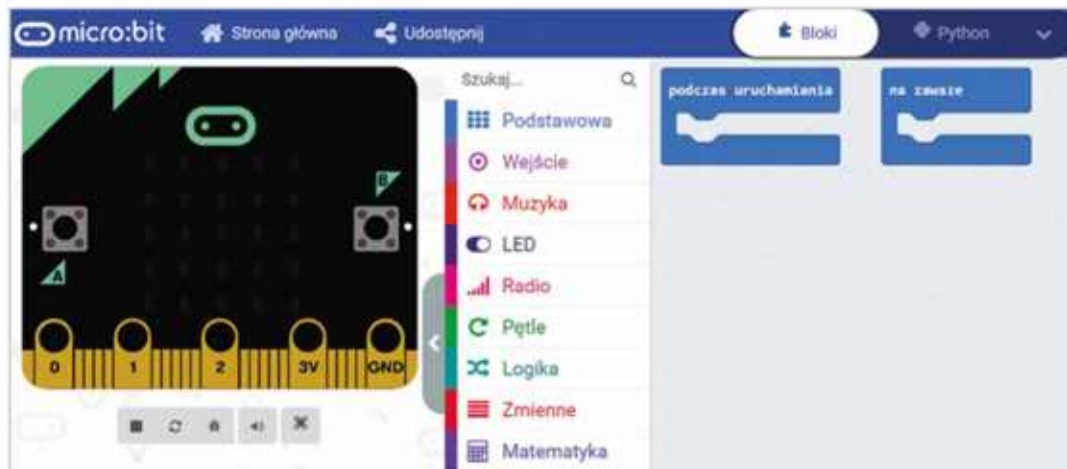
Warto wiedzieć

Środowisko MakeCode obsługuje wiele robotów i innych projektów edukacyjnych, w tym:

- ▶ LEGO MINDSTORMS EV3,
- ▶ Circuit Playground Express,
- ▶ Minecraft,
- ▶ Arcade,
- ▶ Cue,
- ▶ Chibi Chip.

Płytką micro:bit może być programowana również z wykorzystaniem interfejsu graficznego. Przykładem takiego środowiska jest serwis Microsoft MakeCode (makecode.microbit.org). Na tej stronie poza samym środowiskiem programowania znajdziemy również dużą bazę przykładów, lekcji i filmów instruktażowych z wykorzystaniem tego mikrokontrolera.

Rysunek 19.17 pokazuje fragment okna środowiska Microsoft MakeCode uruchomionego dla płytki micro:bit. Po lewej stronie okna znajduje się obraz reprezentujący wygląd jednej strony płytki micro:bit z charakterystycznymi diodami LED i dwoma przyciskami oznaczonymi literami A i B.



Rys. 19.17. Środowisko programistyczne Microsoft MakeCode dla płytki micro:bit

Warto wiedzieć

Programowanie za pomocą elementów graficznych jest charakterystyczne nie tylko dla robotów edukacyjnych. Takie rozwiązanie występuje również w zastosowaniach profesjonalnych i komercyjnych. Przykładem może być środowisko Simulink.

W tym środowisku można wygodnie tworzyć kod źródłowy za pomocą bloków, które zawierają gotowe instrukcje. Pozwalają one wyświetlać m.in. zdefiniowane wcześniej symbole lub napisy, które powstają poprzez zaświecenie się wybranych diod. Można również samodzielnie tworzyć symbole według własnych projektów.

Program Rzut kostką

Zaprogramujemy płytkę tak, aby na początku zaświecone diody utworzyły symbol serca, a po potrząśnięciu płytki został zasymulowany rzut kostką do gry – diody powinny zaświecić się tak, aby pojawiła się liczba od 1 do 6.

Rysunek 19.18 przedstawia kod źródłowy programu *Rzut kostką*.

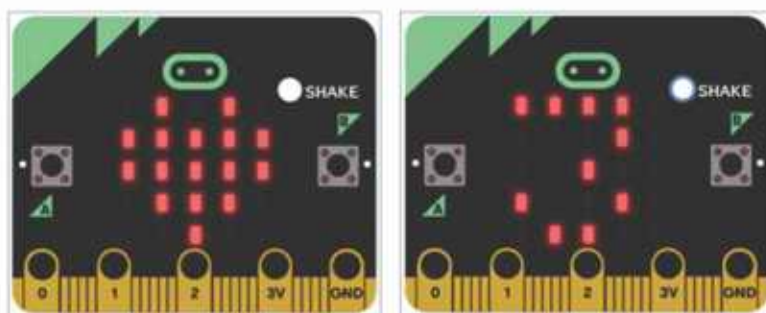


❶ Kod źródłowy programu *Rzut kostką*

Rys. 19.18. Kod źródłowy programu *Rzut kostką*

Aby wyświetlić symbol serca, w bloku *podczas uruchamiania* wykorzystamy instrukcję *pokaż ikonę*. Do zaprogramowania działania płytki po potrząśnięciu nią użyjemy bloku z instrukcją *na*.

Po uruchomieniu programu na ekranie mikrokontrolera pojawia się ikona serca, a kiedy nim potrząsamy (lub naciskamy przycisk **SHAKE**), wyświetla się losowa liczba z zadanego zakresu (rys. 19.19).



Rys. 19.19. Symulacja działania programu *Rzut kostką*

Ćwiczenie 3

Zapisz w środowisku Microsoft MakeCode kod źródłowy programu *Rzut kostką* i przeprowadź symulację działania płytki micro:bit.

A to ciekawe

Imię dla robota

Amerykańska agencja kosmiczna NASA co jakiś czas ogłasza konkursy na nazwy marsjańskich łazików.

Dotychczasowe nazwy zostały wymyślane przez uczniów.

Łazik Sojourner z 1997 r. został nazwany przez 12-letnią Vallery Ambroise,

która upamiętniła w ten sposób Sojourner Truth, afroamerykańską działaczkę

na rzecz praw kobiet i zniesienia niewolnictwa. Łaziki z 2003 r. – Spirit i Opportunity –

nazwała 9-letnia Sofi Collins, a Curiosity z 2011 r. – 12-letnia Clara Ma.



19.8. Środowisko do symulacji VEXcode VR

VEXcode VR (vr.vex.com) jest darmowym środowiskiem do symulacji robota przeznaczonym do celów edukacyjnych. Pozwala ono programować wirtualnego robota z wykorzystaniem bloków i przypomina środowisko Scratch. Bardziej zaawansowani użytkownicy mogą również programować tego wirtualnego robota w języku Python.

Robot, którego programujemy, jest wyposażony m.in. w:

- ▶ elektromagnes (do podnoszenia przedmiotów),
- ▶ wirtualne oczy (np. do rozpoznawania przeszkód),
- ▶ czujniki dotykowe (lewy i prawy),
- ▶ czujnik odległości,
- ▶ żyroskop.

W środowisku VEXcode VR na głównej belce (rys. 19.20) umieszczono następujące narzędzia: zmiana języka, menu **Plik**, samouczki, zmiana tła (narzędzie **Boisko**) oraz narzędzia do sterowania wykonywaniem programu. Poniżej znajdują się: opcja przełączania kodu blokowego na kod języka Python oraz kontrola wartości zmiennych i czujników.



Rys. 19.20. Narzędzia środowiska VEXcode VR

W prawej części okna znajduje się panel **Kod**. Zawiera on bloki instrukcji, z których można tworzyć program sterujący robotem. Fragment okna przedstawia rysunek 19.21.

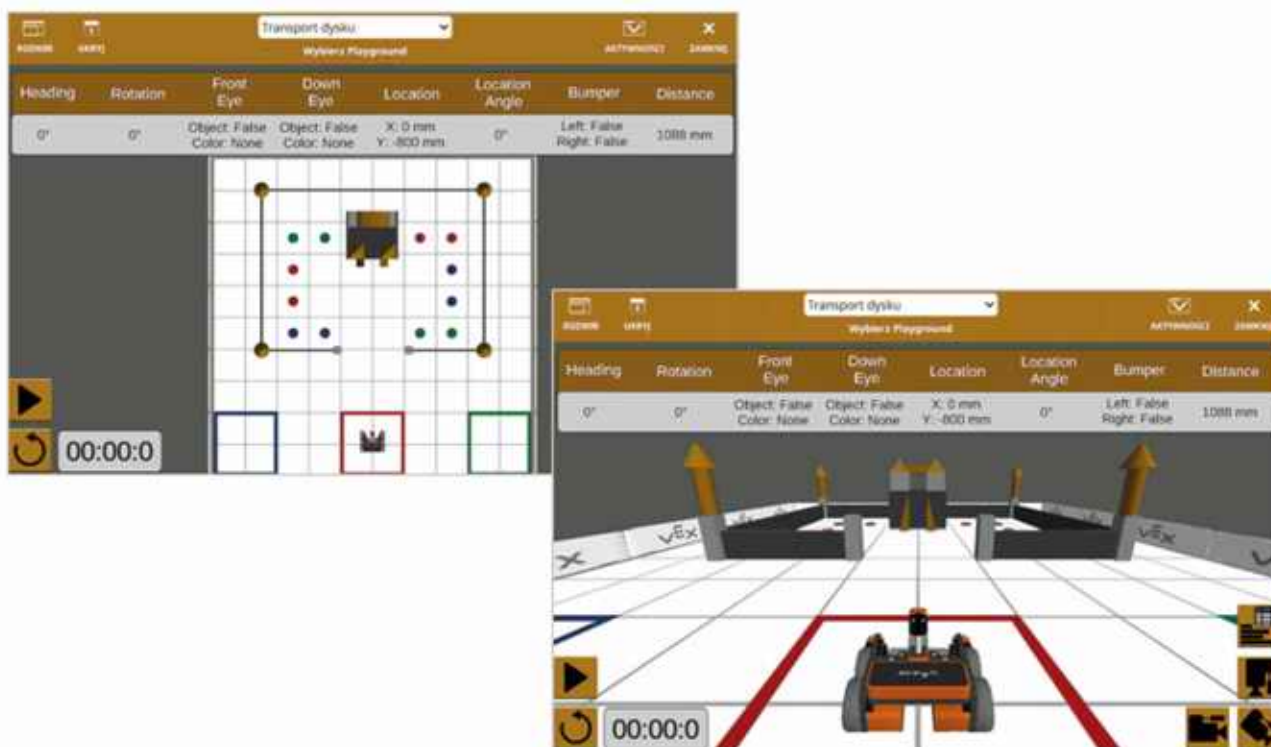


Rys. 19.21. Panel Kod zawierający instrukcje sterujące robotem

Warto wiedzieć

Na niektórych planszach (tłach w narzędziu **Boisko**) dostępna jest jeszcze trzecia kamera, pokazująca obraz, który widzi robot.

W środowisku VEXcode VR możemy obserwować zachowanie robota z różnych perspektyw – mamy do dyspozycji m.in. widok z góry oraz widok trójwymiarowy z kamery podążającej za robotem (rys. 19.22).

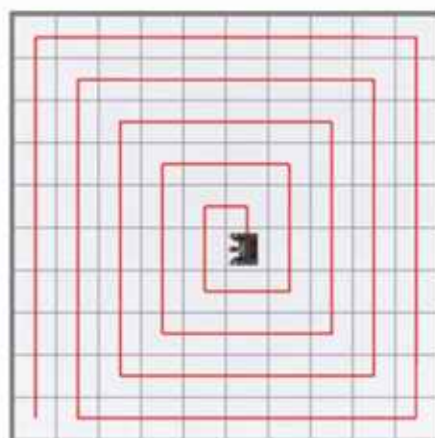


Rys. 19.22. Okno symulacji robota

Narzędzie **Boisko** (ang. *Playground*) służy do ustawiania różnych plansz, po których porusza się robot, oraz do kontrolowania zachowania robota realizującego zapisany program. W wybranym widoku możemy uruchomić program sterujący robotem i w wygodny sposób zobaczyć, czy wykonuje on zakładane zadanie.

Program *Wszystkie pola planszy*

Zaprogramujemy teraz robota tak, aby przeszedł przez wszystkie pola na planszy o rozmiarach 10×10 i oznaczył swoją drogę czerwoną linią. Zakładamy, że robot startuje z lewego dolnego pola, a kończy na środku planszy – tak jak na rysunku 19.23.



Rys. 19.23. Plansza, po której porusza się robot

Robot wielokrotnie będzie jechał prosto przez określoną liczbę pól oraz będzie zawsze skręcał w prawo. Dlatego warto użyć pętli do zautomatyzowania i uproszczenia kodu źródłowego.

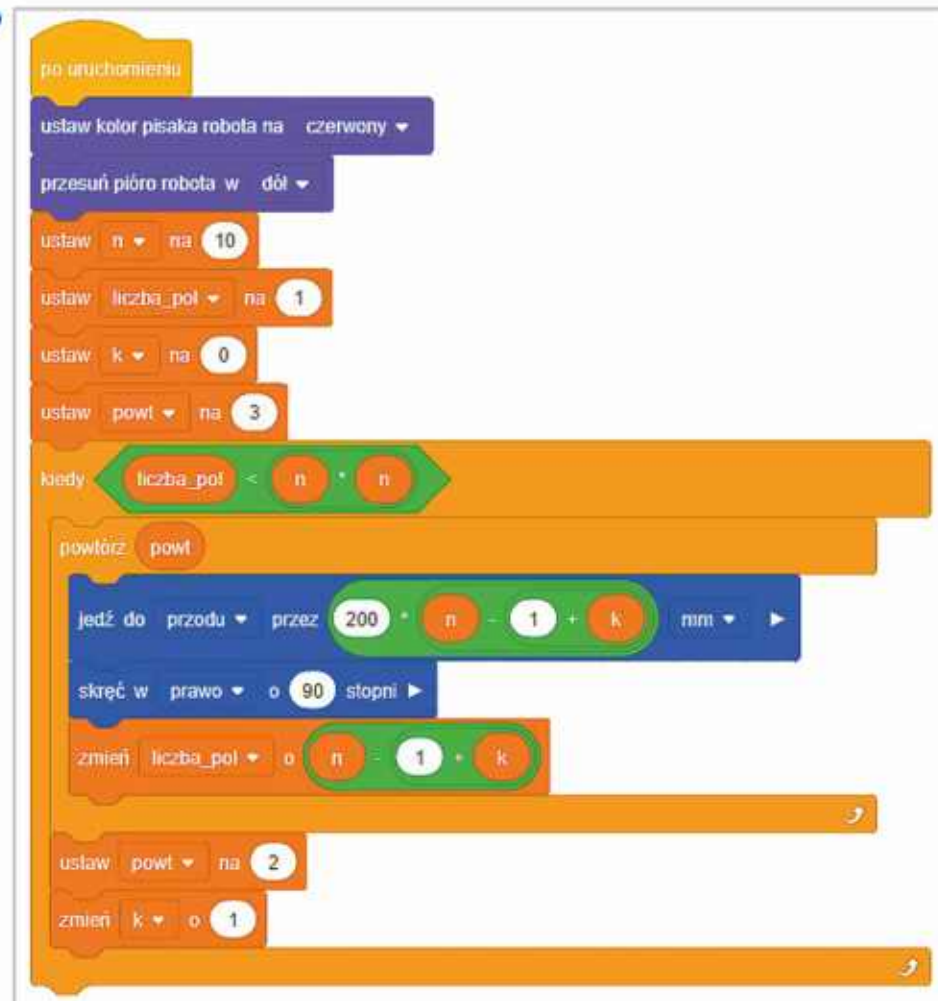
Na rysunku 19.24 przedstawiono trasę robota. Przez pierwsze 3 odcinki robot pokona 9 pól, następnie długość dwóch kolejnych odcinków zmniejszy się do 8 pól, następnych dwóch odcinków do 7 pól itd.



Rys. 19.24. Zmiany długości przemierzanych odcinków

Instrukcje sterujące robotem można zapisać jako pętle zagnieżdżone: wewnętrzna pętla sterująca jazdą na wprost i skrętem w prawo najpierw wykona się 3 razy, a następnie wielokrotnie wykona się 2 razy. Liczbę jej powtórzeń kontroluje zewnętrzna pętla, która będzie się wykonywała, dopóki robot nie odwiedzi wszystkich pól. Rysunek 19.25 przedstawia kod źródłowy programu *Wszystkie pola planszy*.

Kod źródłowy programu *Wszystkie pola planszy*



Rys. 19.25. Kod źródłowy programu *Wszystkie pola planszy*

Program rozpoczyna instrukcje dotyczące zaznaczania trasy kolorem czerwonym. Następnie definiujemy zmienne: n – oznaczającą długość boku kwadratu, $liczba_pol$ – zliczającą liczbę odwiedzonych przez robota pól na planszy, k – wskazującą liczbę krawędzi kwadratu, które należy wykluczyć na danym etapie, $powt$ – określającą liczbę powtórzeń wewnętrznej pętli.

Wartość zmiennej $powt$ po pierwszym obrocie pętli zmienia się z 3 na 2, aby po każdym dwóch odcinkach zmniejszyła się długość kolejnego odcinka. Długość pojedynczego kwadratu wynosi 200 mm, dlatego wyrażenie $200 * (n - (1 + k))$ poprawnie opisuje długość trasy na każdym etapie.

Ćwiczenie 4

- Zapisz kod źródłowy programu *Wszystkie pola planszy* i przeprowadź symulację ruchu robota.
- Wywołaj program dla planszy o boku 6.

Program *Wyjście z labiryntu*

Jednym z klasycznych przykładów zadań dla robota jest znajdowanie wyjścia z labiryntu. Problem ten można rozwiązać na wiele sposobów, wykorzystując wiele różnych algorytmów.

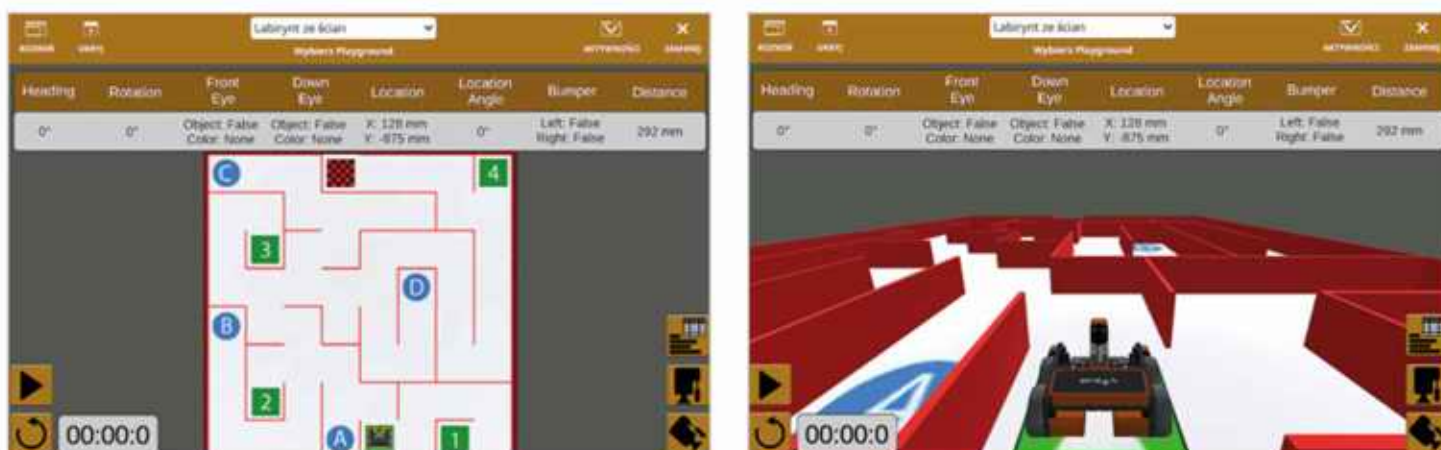
Jednym z możliwych rozwiązań jest takie, w którym robot porusza się do przodu o pewną ustaloną odległość (w naszym przypadku jest to długość pojedynczej ściany w labiryncie), a następnie obraca się w prawo i próbuje jechać prosto. Jeśli napotka przed sobą ścianę, obraca się w lewo do momentu, w którym nie będzie przed nim ściany i znów będzie mógł jechać prosto.

W środowisku VEXcode VR można skorzystać z kilku zdefiniowanych rodzajów plansz, które pozwalają realizować różnorodne algorytmy. Jedną z nich jest plansza **Labirynt ze ścian**.

Rysunek 19.26 przedstawia tę planszę w dwóch widokach: z kamery znajdującej się za robotem oraz z kamery znajdującej się nad planszą.

Dobra rada

Zauważ, że jeśli po pierwszym skrócie w prawo robot napotka na ścianę, to pierwszy obrót w lewo oznacza powrót do poprzedniego ustawienia. Robot będzie wtedy kontynuował jazdę prosto.

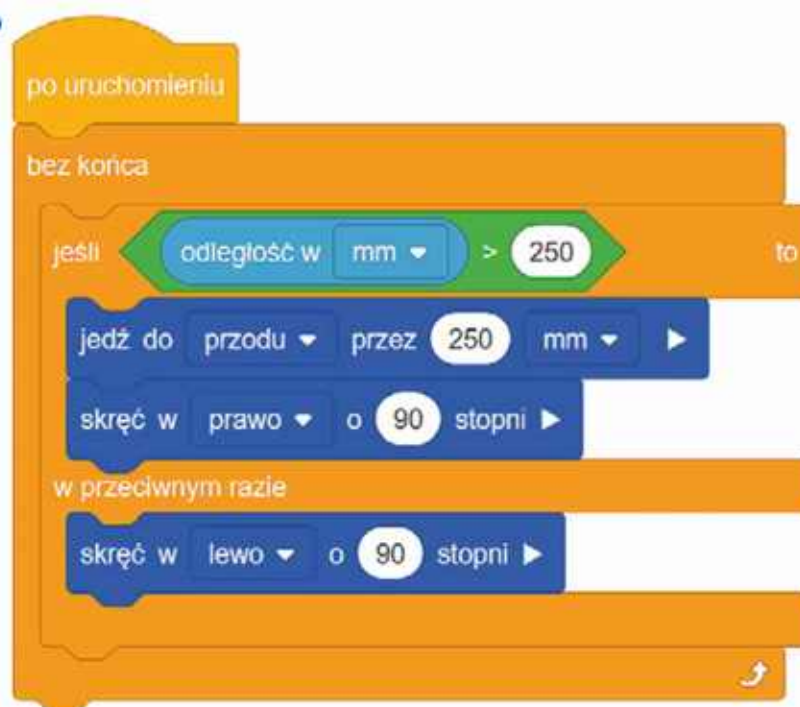


Rys. 19.26. Plansza Labirynt ze ścian – widok z góry i zza robota

Z rysunku 19.26 na s. 365 wiemy, że plansza zbudowana jest z kwadratowych pól. Każde z nich ma długość boku równą 250 mm. Ściany labiryntu są ułożone wzdłuż krawędzi tej siatki, a więc pod kątem 90 stopni.

Na rysunku 19.27 przedstawiono zapis kodu źródłowego programu *Wyjście z labiryntu*.

Kod źródłowy programu **Wyjście z labiryntu**



Rys. 19.27. Kod źródłowy programu *Wyjście z labiryntu*

W programie robot porusza się do przodu o jedno pole i skręca w prawo o 90 stopni, jeżeli odległość od przeszkody jest większa niż 250 mm, w przeciwnym razie skręca w lewo. Powtarza tę czynność aż do opuszczenia labiryntu.

Przedstawiona wcześniej metoda sprawdzi się tylko dla prostych labiryntów, w których wyjście jest połączone z krawędzią labiryntu. Nie jest jednak optymalna.

Warto wiedzieć

Problem poszukiwania wyjścia z labiryntu można rozwiązać między innymi z wykorzystaniem algorytmów: Minty'ego, Bellmana–Forda, Dijkstry oraz Dantziga.

Ćwiczenie 5

- Zapisz kod źródłowy programu *Wyjście z labiryntu* i przeprowadź jego symulację.
- Zmodyfikuj kod źródłowy programu *Wyjście z labiryntu* tak, aby robot w pierwszej kolejności szukał możliwości skrętu w lewo.

Zapamiętaj

Symulatory robotów mogą skutecznie zastępować fizyczne urządzenia. Dają możliwość realizacji ciekawych projektów oraz korzystania z gotowych plansz, po których poruszają się roboty. Wiele środowisk symulacyjnych działa w trybie online.



Podsumowanie

- Robot to urządzenie możliwe do zaprogramowania lub sterowania, które jest zdolne do poruszania się (wykonywania różnych ruchów w przestrzeni), a jego zadaniem jest ułatwienie człowiekowi pracy lub zastąpienie go w skomplikowanych, powtarzalnych lub monotonnych czynnościach.
- Roboty można programować w wielu językach, np. za pomocą bloków oraz języków Python i C++.
- Środowiska symulacyjne (np. VEXcode VR, Microsoft MakeCode lub Open Roberta Lab) pozwalają na poznawanie robotów, programowanie i obserwowanie ich działania. Niektóre oferują widok trójwymiarowy, dzięki któremu można dokładnie zarejestrować pracę robota.



Zadania

- ★ **1** W środowisku VEXcode VR zaprogramuj robota tak, aby odwiedził wszystkie pola planszy o rozmiarze 10 na 10 w inny sposób niż opisany w tym temacie na s. 363–365.
- ★ **2** Zaproponuj, opisz i wykonaj prezentację na temat wybranego zagadnienia, które można rozwiązać za pomocą jednego lub wielu robotów.
- ★ **3** Zademonstruj możliwości wybranego robota. Opisz i zaprezentuj zastosowania dostępnych czujników i silników.
- ★★ **4** W kilkusobowej grupie zaprojektujcie proste ramię robota służące do wykonywania jakiejś czynności, np. chwytak przenoszący przedmiot z lewej strony ramienia na prawą. Zaprogramujcie kostkę EV3 tak, aby sterowała tym ramieniem. Zademonstrujcie w środowisku Microsoft MakeCode działanie programu i kostki EV3.
- ★★ **5** Zaprogramuj robota muzycznego, który będzie tworzył własną muzykę, np. z wykorzystaniem akcelerometru.
- ★★ **6** Zaprogramuj robota, który będzie służył jako kontroler do sterowania w grze komputerowej.
- ★★ **7** W środowisku VEXcode VR wybierz planszę o nazwie **Siatka liczb**. Zaprogramuj robota tak, aby odwiedził wszystkie pola planszy oznaczone liczbami nieparzystymi przy możliwie najmniejszej liczbie najazdów na pola parzyste.
- ★★★ **8** Zaprogramuj robota (lub mikrokontroler) tak, aby za jego pomocą pokazać działanie wybranych praw fizyki, np. siły odśrodkowej.
- ★★★ **9** Zaprogramuj robota tak, aby pokonał dowolny labirynt.
- ★★★ **10** W kilkusobowej grupie zaprogramujcie wybranego robota poruszającego się po wyznaczonej ścieżce z możliwością omijania przeszkód na drodze.

20. Sztuka publikowania w sieci

Nasze działania w różnych dziedzinach życia coraz częściej mają miejsce w sieci, a czasem muszą wręcz ograniczyć się tylko do niej. Dlatego szczególnie ważna staje się umiejętność szeroko rozumianego publikowania w internecie – od komunikacji z przyjaciółmi po przygotowywanie materiałów na potrzeby nauki, pracy czy hobby. Z tego tematu dowiesz się, jak tworzyć zrozumiałe, atrakcyjne i profesjonalne treści, które mają szansę zaistnieć w sieci.

Cele lekcji

- Nauczysz się opracowywać interesujące dla odbiorców treści internetowe.
- Poznasz narzędzia, które mogą pomóc w przygotowaniu publikacji wzbogaconych o grafikę, nagrania audio lub filmy.
- Zrozumiesz pojęcia: webcast, webinarium, screencast i podcast.
- Nauczysz się przygotowywać do wystąpienia przed kamerą i mikrofonem.
- Dowiesz się, jakie narzędzia pozwalają monitorować i analizować ruch w internecie.

Warto wiedzieć

Nad zróżnicowaniem formy przekazu warto się zastanowić w przypadku serii publikacji, które będą się pojawiać cyklicznie. Wówczas formę można dopasować do aktualnego tematu.

Twórcy internetowi najczęściej wybierają jedną główną formę publikowania, np. teksty na blogu, filmy w serwisie wideo lub audycje tylko do słuchania. Niezależnie jednak od tego, jaką podstawową formę przekazu wybierzemy, przyda się umiejętność tworzenia pozostałych. Publikacje filmowe wymagają bowiem np. przygotowania tekstów zapowiadających ich premierę, a także komunikatów (tekstu lub grafiki) do zamieszczenia w sieciach społecznościowych.

Ćwiczenie 1

Zastanów się, co przyciąga twoją uwagę w sieci. Omówcie to w grupie i wymieńcie najważniejsze czynniki. Następnie zastanówcie się, co sprawia, że:

- a. wracacie do tego samego miejsca w sieci – witryny lub kanału,
- b. przestajecie odwiedzać daną witrynę lub kanał.

20.1. Jak zainteresować odbiorców swoimi publikacjami?

Gdy przeglądamy informacje w sieci, pewne czynniki decydują, że przy jednych treściach zatrzymujemy się na dłużej, by poznać szczegóły, a w wypadku innych przeciwnie – przechodzimy dalej. Podobnie jest z całym serwisami internetowymi.

Zastanówmy się, które elementy sprawiają, że publikowane w internecie treści są atrakcyjne i przyciągają uwagę odbiorców.

Treści dopasowane do odbiorców

Przygotowanie treści dopasowanych do odbiorcy, jego potrzeb i oczekiwań, jest w dzisiejszych czasach koniecznością – bez tego nie uda się wzbudzić szerszego zainteresowania.

Zauważ, że inaczej należy przygotować np. zaproszenie do obejrzenia filmu dla rówieśników, a inaczej dla nauczycieli czy specjalistów w danej dziedzinie. Trzeba również odpowiednio dobrać konstrukcję samej publikacji, język przekazu oraz warstwę graficzną. Nie bez znaczenia jest wiek i wykształcenie odbiorcy.

Ćwiczenie 2

Opisz krótko swoją ulubioną grę, książkę lub film w sposób dostosowany do odbiorców w wieku 70 lat oraz 10 lat.

Warto wiedzieć

Tworzenie prasy skierowanej do czytelników z niższym wykształceniem wymaga odpowiedniego dostosowania tekstu. Kiedyś redakcje zatrudniały osoby o takim profilu w roli testerów, którzy oznaczali niezrozumiałe dla nich fragmenty tekstu. Z czasem jednak redaktorzy uczyli się pisać wystarczająco przystępnie, a testerzy tracili pracę.

Co przyciąga uwagę użytkowników?

Oto wybrane elementy, które przyciągają uwagę odbiorców podczas przeglądania informacji w internecie.

- ▶ Tytuły (nagłówki) oraz **lidy** (ang. *lead*), czyli krótkie zapowiedzi treści. **Lid**

Wzbudzające emocje i intrygujące tytuły zachęcają do przeczytania, obejrzenia lub odsłuchania całości materiału.

- ▶ Dobrze skonstruowane teksty informacyjne – wprowadzenia, streszczenia, zaproszenia oraz zapowiedzi, np. w mediach społecznościowych.
- ▶ Ilustracje – wszelkiego rodzaju grafiki związane z danym tematem, włącznie z banerami, oraz pełniące rolę informacyjną, np. odpowiedniki plakatów filmowych.
- ▶ Właściwy dobór słów, które przybliżą użytkownikowi zagadnienie, ale również będą odpowiadały zapytaniom w wyszukiwarkach internetowych. W ten sposób można zwiększać szansę na to, że naszą publikację znajdą nowi użytkownicy, bo pojawi się ona wysoko w **organicznych wynikach wyszukiwania** (ang. *organic search*), czyli takich, które nie są wynikiem opłaconych kampanii.

- ▶ **Hashtagi**, czyli krótkie teksty poprzedzone znakiem #, które pełnią funkcję znacznika treści. Hashtagi pozwalają uporządkować pokrewne publikacje, a niekiedy pomagają różnym inicjatywom zyskać rozgłos.

Ważnym elementem, który może zwrócić uwagę internautów na publikację, są również reklamy. Najczęściej mają one postać graficzną lub tekstową. Warto pamiętać o dwóch ważnych zasadach. Po pierwsze – reklama nie może być zbyt nachalna. Nikt nie lubi często wyskakujących banerów. Po drugie – reklama nie może oszukiwać.

Tworzenie tytułów zachęcających do kliknięcia, ale mylących co do zawartości jest niedopuszczalne. Takie działanie nazywane jest często **clickbaitem** (od ang. *to click* – klikać, *bait* – przynęta) i jest bardzo negatywnie odbierane przez użytkowników.

Warto wiedzieć

Tytuł publikacji, który ma przyciągnąć uwagę, nie musi ściśle odpowiadać jej tematowi. Na przykład artykuł o inteligencji szczurów może mieć tytuł: *Dlaczego tak trudno pozbyć się szczurów?*

Organiczne wyniki wyszukiwania

Hashtag

Clickbait

Jak przygotować treść publikacji

Aby sprawnie przygotować publikację, która zainteresuje wiele osób, warto odpowiednio zorganizować sobie pracę.

1

Zrób plan

Znajdź pomysł na temat publikacji i zaplanuj jej zawartość.

Publikacja tekstowa

- konspekt z punktami do opracowania
- opis grafik ilustrujących tekst

Nagranie wideo lub występ na żywo

- scenariusz lub scenorys z opisem scen i ujęć
- tekst lektora
- opis potrzebnych grafik

2

Podдай plan krytycznej analizie

Przygotowany plan warto przeanalizować i odpowiedzieć sobie na kilka pytań:



Odpowiedzi na te pytania powinny skłonić do wprowadzenia niezbędnych poprawek.

3

Ustal objętość publikacji

- Im dłuższy materiał, tym więcej czasu zajmie jego przygotowanie.
- Pamiętaj, że wraz ze wzrostem objętości publikacji spada uwaga odbiorców. Publikacja powinna omawiać wszystkie istotne zagadnienia, ale nie powinna być zbyt obszerna.



4

Wymyśl tytuł

- Dobry tytuł powinien przyciągać uwagę i zachęcać do zapoznania się z treścią.
- Sformułuj tytuł tak, by łatwo było go używać w różnych celach, m.in.: w zapowiedziach, na banerach, plakatach, gadżetach, planszach początkowych i końcowych.
- Tytuł powinien brzmieć tak samo niezależnie od miejsca użycia – nawet drobne odstępstwa mogą dezorientować odbiorców.

5

Wykonaj plan

- Podczas tworzenia publikacji trzymaj się przygotowanego konspektu lub scenariusza.
- Wprowadź modyfikacje, jeśli ulepszą one materiał i sprawią, że będzie lepiej odpowiadał potrzebom odbiorców.
- Opublikuj materiał zgodnie z zapowiedziami. Możesz skorzystać z systemów, które automatycznie publikują treści o określonej porze.

Unikaj błędów

- Przed publikacją sprawdź: imiona i nazwiska, nazwy, adresy, daty, dane teleadresowe itp.
- Nie używaj wyrażen typu „dziś”, „wczoraj”, „w zeszłym tygodniu”, bo nie wiesz, kiedy odbiorca zapozna się z publikacją.

Śledź reakcje i wprowadzaj poprawki

- Czytaj komentarze i utrzymuj kontakt z odbiorcami.
- Staraj się zrozumieć grono swoich odbiorców – pamiętaj, że może się ono zmieniać.
- Analizuj trendy w sieci.
- Dostosowuj nowe publikacje do oczekiwań odbiorców i aktualizuj starsze treści.



Jak utrzymać uwagę odbiorców?

Zastosowanie kilku wymienionych poniżej zasad może sprawić, że odbiorcy nie tylko zapoznają się z przygotowanym przez nas materiałem, lecz także będą wracać do naszego serwisu.

- ▶ Twórz interesujące i użyteczne treści, dostosowane do potrzeb i oczekiwań odbiorcy pod względem zawartości i języka przekazu.
- ▶ Aktualizuj treści – dbaj o to, aby nawet starsze materiały zawierały aktualne odnośniki, sprawdzaj, czy wszystko działa prawidłowo, oznaczaj nieaktualne treści jako archiwalne lub usuwaj je z serwisu.
- ▶ Bierz odpowiedzialność za słowo – publikuj informacje sprawdzone w rzetelnych źródłach, konsekwentnie realizuj zapowiadane działania, zrezygnuj z publikowania treści, których nie rozumiesz.
- ▶ Publikuj systematycznie, zwłaszcza na blogach i w serwisach z poradami. Jest to obowiązek, jeśli tworzysz coś w formie serii złożonej z wielu odcinków (części).
- ▶ Monitoruj ruch w sieci i **poznawaj własnych odbiorców** – pozwoli to na lepsze dopasowanie treści do ich potrzeb.
- ▶ Komunikuj się z odbiorcami – odpowiadaj na ich pytania i uwagi, zachęcaj do wykonania określonych działań (np. udostępniania lub zakupu), badaj preferencje użytkowników (np. za pomocą ankiet), umożliwiał dyskusję, organizuj konkursy.

Dobra rada

Ponieważ filmy trudniej aktualizować niż teksty, możesz w zamian przygotować odpowiednie adnotacje, np. w opisie filmu.

Jak poznać (lepiej) swojego odbiorcę,
s. 385 

Storytelling,
podręcznik *Informatyka*
na czasie 1. Zakres
rozszerzony,
s. 157 

Ćwiczenie 3

Przypomnij sobie, na czym polega **storytelling**. Zastanów się, czy i jak można go zastosować do treści publikowanych w internecie.



Zapamiętaj

Publikacja powinna uwzględniać potrzeby i zainteresowania odbiorców. Należy zadbać o to, by materiał był interesujący, przejrzysty, kompletny oraz treściwy, a informacje w nim zawarte były sprawdzone w rzetelnych źródłach.

A to ciekawe

Czym jest content marketing?

Content marketing to jedna z popularnych i efektywnych form marketingu, którą powszechnie wykorzystuje się w internecie. Publikowanie interesujących i przydatnych treści (np. artykułów, webinarów, infografik) przyciąga klienta na stronę firmy i potwierdza, że pracują w niej eksperci w danej dziedzinie. Dzięki temu łatwiej jest przekonać użytkownika np. do zakupów.



20.2. Teksty informacyjne

Teksty opracowuje się nie tylko na potrzeby publikacji ściśle tekstowych, lecz także podczas tworzenia filmów, podcastów czy webinarów. Najczęściej są to **teksty informacyjne**, czyli takie, których głównym zadaniem jest skuteczne przekazanie informacji. Z tego powodu zazwyczaj są pozbawione wszelkich środków artystycznych, cechujących np. teksty literackie.

• **Tekst informacyjny**

Przygotowanie do pisania

Przed rozpoczęciem pisania należy się zastanowić, jakiego rodzaju tekst informacyjny będzie nam potrzebny. Może to być m.in.:

- ▶ zaproszenie, np. do obejrzenia filmu lub wysłuchania **podcastu**,
- ▶ zapowiedź wydarzenia,
- ▶ opis audycji,
- ▶ relacja z wydarzenia,
- ▶ tekst do działu „o nas” na stronę internetową lub blog.

Podcast,
s. 377 [↗](#)

Każdy z wymienionych typów ma inną charakterystykę i zazwyczaj zawiera nieco inne informacje. Zanim zaczniemy pisać, musimy ustalić, jaka jest najważniejsza informacja do przekazania, oraz przemyśleć konstrukcję tekstu.

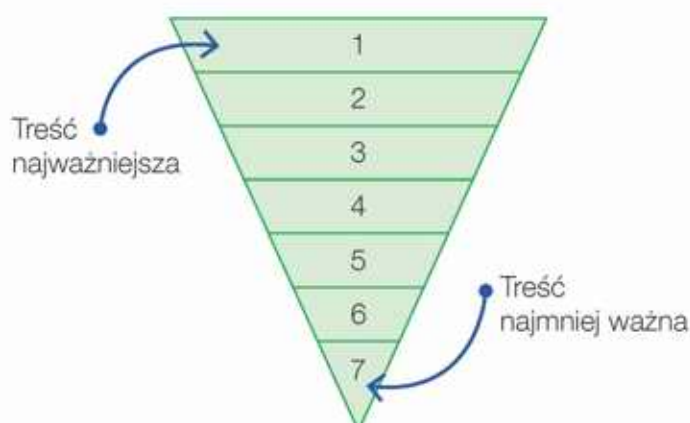
Nawet w wypadku krótkiego tekstu warto przygotować **konspekt**, czyli zarys przedstawiony w punktach. Powinny się w nim znaleźć zagadnienia, które chcemy uwzględnić. Dzięki temu podczas pisania o niczym nie zapomnimy.

• **Konspekt**

Punkty w konspekcie warto na bieżąco ustawiać w formie odwróconej piramidy (rys. 20.1). Najważniejsze treści, które opiszemy najszerzej, umieszczamy na początku, a najmniej ważne, którym poświęcimy mniej miejsca – na końcu.

Dobra rada

Stworzenie konspektu da ci jeszcze jedną korzyść – liczba pozostałych do opracowania punktów będzie przypominała o ilości pracy do wykonania oraz skłaniała do zwięzłego opisywania punktów.



Rys. 20.1. Konstrukcja konspektu w formie odwróconej piramidy

Dzięki zapisaniu haseł z konspektu w formie odwróconej piramidy już w trakcie tworzenia tekstu zorientujemy się, jaka będzie jego objętość i z jakich treści możemy zrezygnować, aby nadal przekazać wszystkie najważniejsze informacje.

Tworzenie komunikatu tekstowego

Zasada 5W ● W napisaniu tekstu może pomóc nam tzw. **zasada 5W**. Jej nazwa pochodzi od pięciu pytań w języku angielskim zaczynających się na literę *w*. Należy je sobie postawić w odniesieniu do informacji, którą chcemy przekazać:

Warto wiedzieć

Zasada 5W jest powszechnie stosowana podczas przygotowywania informacji dziennikarskiej.

1. *Who?* – Kto?
2. *What?* – Co?
3. *Where?* – Gdzie?
4. *When?* – Kiedy?
5. *Why?* – Dlaczego (po co)?

Jeśli odpowiemy w tekście na powyższe pytania, w wielu przypadkach to wystarczy, aby zaciekać czytelnika i przekazać mu wszystko, co najważniejsze. Koniecznie natomiast trzeba uwzględnić te informacje, które będą szczególnie istotne dla naszego odbiorcy. Pomoże w tym również przygotowanie **konspektu** w formie odwróconej piramidy.

Konspekt,
s. 373 

Jeśli tekst zaprasza do obejrzenia filmu, w którym występuje znana i ceniona przez użytkowników osoba, warto podać jej nazwisko w pierwszych zdaniach i dodatkowo wyróżnić je w tekście.

Jeśli jednak dopiero budujemy swoją markę, na pierwszym miejscu umieścimy interesujący tytuł oraz opis, który przyciągnie odbiorców. Własne nazwisko należy podać w dalszej części. Jeśli publikacja się spodoba, nazwisko (pseudonim lub logo) autora zacznie być prędzej czy później rozpoznawalne i z czasem może samo w sobie stać się zachętą dla użytkowników.

Ćwiczenie 4

Zastosuj zasadę 5W, aby:

- a. przygotować tekst zaproszenia na zbliżające się webinarium z udziałem eksperta,
- b. zmodyfikować przygotowany tekst tak, aby zapraszał do obejrzenia nagrania webinarium, które już się odbyło.

A to ciekawe

Klucz do sukcesu

O tym, jak duże znaczenie ma tytuł oraz użyte w nim słownictwo, przekonała się kiedyś firma Microsoft. Ten sam artykuł z poradami dotyczącymi edytora prezentacji został opublikowany pod dwoma bardzo podobnymi tytułami: *10 smart tips for using PowerPoint* oraz *10 key tips for using PowerPoint*. Okazało się, że artykuł opatrzony drugim tytułem przeczytało kilka razy więcej osób.



20.3. Grafika i identyfikacja wizualna

Przygotowanie grafiki, podobnie jak opracowanie tekstów informacyjnych, jest elementem towarzyszącym każdej formie publikacji sieciowej. Do elementów graficznych możemy zaliczyć:

- ▶ ilustracje w tekście – zdjęcia, **grafiki informacyjne (infografiki)**, wykresy, schematy,
- ▶ plansze tytułowe i końcowe w filmach lub wystąpieniach na żywo,
- ▶ podpisy osób w filmie lub w czasie wystąpień na żywo,
- ▶ plakaty i banery informacyjne zapraszające na wydarzenia oraz zachęcające do zapoznania się z publikacjami,
- ▶ logotypy,
- ▶ materiały prezentacyjne i animacje wykorzystywane w filmach lub podczas wystąpień na żywo.

Język wizualny stał się bardzo popularnym sposobem komunikacji w internecie, ponieważ jest prosty i atrakcyjny oraz przyciąga uwagę odbiorców. Warto poświęcić czas na przemyślenie i przygotowanie odpowiednich grafik towarzyszących publikacji sieciowej.

Oprogramowanie i serwisy ułatwiające przygotowanie grafik do publikacji

Istnieje wiele aplikacji i serwisów, dzięki którym można przygotować atrakcyjną grafikę. Niektóre z nich omówiliśmy w poprzednich częściach podręcznika. Warto jednak kilka aplikacji i serwisów wyróżnić. Przedstawia je tabela 20.1.

Program lub serwis	Przeznaczenie
Canva	Serwis oferuje wiele narzędzi, szablonów oraz grafik, które można wykorzystać w projektach do stworzenia m.in. plakatów, banerów, prezentacji i plansz tytułowych.
Edytor prezentacji (np. PowerPoint, Impress)	Aplikacje do tworzenia prezentacji mają wbudowane narzędzia graficzne. Dzięki nim można wygodnie przygotować plakat, baner lub planszę tytułową.
Wycinek i szkic (Windows 10)	Narzędzie to pozwala na łatwe wykonywanie i przetwarzanie zrzutów ekranowych, które mogą być przydatne przy tworzeniu samouczków. Można je uruchomić skrótem klawiszowym Windows + Shift + S .
Pixabay, Pixels, Unsplash	Te serwisy internetowe udostępniają zdjęcia na otwartych licencjach (często do użytku komercyjnego). Przed użyciem zdjęcia trzeba sprawdzić warunki licencyjne, ponieważ mogą one narzucać sposób oznaczania autora zdjęcia.
Programy graficzne (np. Inkscape, GIMP)	Program do grafiki wektorowej (np. Inkscape) można wykorzystać np. do stworzenia logo lub baneru. Programy do grafiki rastrowej pomogą m.in. odpowiednio wykadrować zdjęcia.

Tabela 20.1. Programy i serwisy ułatwiające przygotowanie grafik do publikacji

Grafiki informacyjne,
s. 387–402 [🔗](#)

Dobra rada

Wskazówki dotyczące grafiki możesz znaleźć w tematach *Sztuka prezentacji* oraz *Grafika 2D i 3D* z podręcznika *Informatyka na czasie 1*.

Warto wiedzieć

W internecie można znaleźć również serwisy oferujące darmowe zasoby muzyczne, np. Free Music Archive lub Bensound.

Identyfikacja wizualna

Identyfikacja wizualna • **identyfikację wizualną**, czyli rozpoznawalność przygotowywanych przez siebie materiałów. Mogą ją zapewnić stałe, powtarzane w każdej publikacji elementy graficzne, nazywane czasem motywem lub szablonem. Należą do nich:

Warto wiedzieć

W publikacjach multimedialnych stałymi elementami są intra oraz dżingle.

- ▶ czcionki,
- ▶ tła,
- ▶ kolorystyka,
- ▶ zdjęcie,
- ▶ układ treści.

Stale elementy pozwalają szybko skojarzyć treści z daną firmą, serią publikacji lub autorem. Warto także zadbać o ich spójność nie tylko w samych publikacjach, lecz także na stronie WWW, blogu, w mediach społecznościowych i wszelkich zapowiedziach.

Jednym z elementów graficznych ściśle związanych z identyfikacją wizualną jest **logo**. Może się ono składać z samego symbolu graficz-

Sygnet, logotyp • nego, zwanego **sygnetem**, z sygnetu i **logotypu** (nazwy) lub samego logotypu. Czasem zawiera też dodatkowy tekst, np. slogan.

Zazwyczaj dąży się do tego, by logotyp był oryginalny, ale równocześnie prosty i łatwy do użycia w różnych publikacjach. Przykładowe logo składające się z sygnetu i logotypu przedstawia rysunek 20.2.

Warto wiedzieć

Oryginalne rozwiązania przyciągają uwagę odbiorców, ale nie można z nimi przesadzić. Gdy jedna z firm technologicznych wprowadziła na swoich stronach przewijanie treści lewo–prawy, szybko musiała z pomysłu zrezygnować, bo użytkownicy nie potrafili z tego korzystać.



Rys. 20.2. Logo Olimpiady Informatycznej, składające się z sygnetu i logotypu

Ćwiczenie 5

Wskaż wizualne i (o ile to możliwe) dźwiękowe elementy identyfikujące:

- a. wybrany serwis VOD (video na życzenie),
- b. Wielką Orkiestrę Świątecznej Pomocy,
- c. miasta: Rybnik, Katowice, Melbourne,
- d. wybranego blogera lub kanał internetowy.

20.4. Nagrania audio i wideo oraz wystąpienia na żywo

Wiele treści w internecie ma postać różnego rodzaju filmów, audycji lub relacji na żywo. Tabela 20.2 przedstawia najpopularniejsze rodzaje publikacji związane z wideo i dźwiękiem.

Rodzaj publikacji	Opis	
Webinarium (webinar)	Rodzaj seminarium organizowanego na żywo w internecie. Może być prowadzone przez jedną osobę lub wiele osób znajdujących się w oddalonych miejscach. Podczas webinarium wykorzystuje się transmisje audio-wideo z kamer, udostępnianie prezentacji lub zawartości ekranu komputera. Możliwa jest interakcja z publicznością, np. dyskusja z uczestnikami lub odpowiedzi na pytania (zadane np. na czacie). Webinaria wykorzystuje się głównie jako narzędzia szkoleniowe i marketingowe.	🔗 Webinarium (webinar)
Webcast	Audycja internetowa w formie wideo (przypominająca audycję telewizyjną), najczęściej realizowana na żywo, choć może być również odtwarzana z przygotowanego wcześniej materiału. Zazwyczaj nie wykorzystuje możliwości bezpośredniej komunikacji między odbiorcami a prowadzącymi. Nie oznacza to jednak całkowitego braku interakcji – widzowie mogą np. głosować w przeprowadzanych podczas programu ankietach.	🔗 Webcast
Podcast	Rodzaj audycji internetowej (najczęściej audio) publikowanej regularnie jako seria odcinków.	🔗 Podcast
Screencast	Film będący zapisem czynności wykonywanych na ekranie komputera uzupełniony o komentarz lektora. Wykorzystuje się go jako materiał szkoleniowy lub demonstracyjny. Nagrywanie i publikowanie tego typu filmu określane jest często jako screencasting, a film nazywany jest tutorialiem.	🔗 Screencast

Tabela 20.2. Popularne typy internetowych publikacji audiowizualnych

Znaczenie przedstawionych wyżej pojęć może się zmieniać wraz z rozwojem technologii. Wszystkie te działania wymagają jednak odpowiedniej pracy z kamerą oraz mikrofonem.

Praca z kamerą

W zastosowaniach amatorskich i półprofesjonalnych do rejestrowania filmów wykorzystuje się nie tylko kamery, lecz także smartfony. Na przykład w czasie nadawania na żywo użycie smartfonu pozwoli uzyskać o wiele lepszy efekt niż korzystanie z kamery wbudowanej w laptop.

Większość telefonów po zainstalowaniu dodatkowej aplikacji pozwala używać aparatu jak zewnętrznej kamery USB lub kamery bezprzewodowej (z wykorzystaniem wi-fi). Dzięki temu otrzymujemy lepszy obraz i więcej możliwości kadrowania.

W czasie przygotowań do nagrania warto pamiętać o kilku prostych zasadach, które pomogą uzyskać materiał lepszej jakości.

📄 Warto wiedzieć

Wiele aplikacji pozwala ustabilizować zapisany film, czyli pozbawić go drgań. Warto z tego korzystać, ale trzeba pamiętać, że stabilizacja odbywa się w tym przypadku kosztem rozdzielczości filmu.

Praca z kamerą

Podczas nagrywania materiału wideo warto pamiętać o wskazówkach, które pomogą wykorzystać możliwości dostępnych urządzeń i podnieść jakość nagrań.

Oświetlenie

Zadbaj o to, by nagrywany obiekt lub osoba były odpowiednio oświetlone. Uważaj na cień rzucany przez czapkę lub kapelusz.

Naturalne światło

Najlepsze ujęcia powstaną o poranku, gdy słońce jeszcze jest nisko, albo po południu, gdy światło nie jest już tak ostre.



Mikrofon

Korzystaj z mikrofonu krawatowego lub ręcznego. Zamontuj też osłonę chroniącą przed szumem.

Nagranie na zewnątrz

Stabilizacja kamery

Przy ujęciach w plenerze sprawdzi się tzw. gimbal, czyli urządzenie, które stabilizuje kamerę.

Kadrowanie



Unikaj kadrowania z funkcją zoomu, ponieważ może to pogorszyć jakość nagrania. Wykorzystuj optyczne możliwości kamery lub przestaw ją tak, aby dopasować kadr.

Podczas nagrań na żywo zabezpiecz przestrzeń przed niepowołanymi osobami. Poproś kogoś, by informował przechodniów o trwającym nagraniu i prosił ich o omijanie filmowanego obszaru.

Jeśli dużo gestykulujesz lub korzystasz z rekwizytów, lepiej wypadniesz w dalszym planie, tzw. planie średnim. Nie musisz też być w centrum kadru.

Sztuczne oświetlenie

Światło powinno być rozproszone, miękkie i równomierne. Pozwoli to filmowanej osobie uniknąć cieni pod oczami i refleksów na twarzy.

Nagranie w pomieszczeniu



Mikrofon

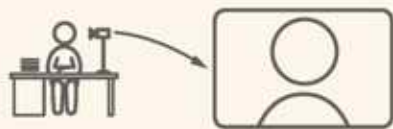
Korzystaj z dodatkowego, zewnętrznego mikrofonu. Polepszy on jakość nagrania.

Tylna kamera

Staraj się korzystać z tylnej kamery smartfonu. Zazwyczaj ma ona lepsze parametry od kamery przedniej.

Nagranie z wielu kamer

Jeśli używasz jednocześnie wielu kamer, włącz w każdej z nich nagrywanie obrazu oraz dźwięku. Ułatwiś sobie synchronizację fragmentów.



Przy ujęciach z bliska kamera powinna być na wysokości twoich oczu. Uważaj z gestykulacją, aby dłonie nie pojawiły się w kadrze.



Zadbaj o przestrzeń wokół siebie. Usuń wszystkie niepotrzebne rzeczy, które mogłyby odwracać uwagę od tego, co najważniejsze.



Jeśli nie masz wpływu na swoje otoczenie, ustaw wirtualne tło. Pamiętaj, żeby postać na tle była dobrze oświetlona.

Zachowanie przed kamerą

Aby dobrze wypaść przed kamerą, warto stosować zasady podobne do tych, które obowiązują podczas występów publicznych.

Utrzymuj kontakt wzrokowy



Podczas wywiadu patrz na osobę, z którą rozmawiasz, a nie w obiektyw kamery. W innych okolicznościach zapytaj operatora kamery, jaka pozycja będzie bardziej naturalna, lub zrób próbę i oceń to samodzielnie.

Dobierz odpowiedni strój



Kamera lubi szarości i błękity. Unikaj drobnych wzorów, pasków i zieleni – zwłaszcza gdy występujesz w studiu telewizyjnym. Załóż dłuższą spódnicę lub dłuższe spodnie, aby w pozycji siedzącej niczego niepotrzebnie nie odsłonić.

Przyjmij właściwą postawę



Wyprostuj się, stań pewnie na lekko rozstawionych nogach ze stopami skierowanymi na zewnątrz. Jeśli siedzisz, siedź prosto, ale swobodnie. Nie rozstawiaj nóg i nie zapadaj się w głębokich fotelach. Unikaj kręcenia i bujania się na krześle.

Gestykuluj z umiarem



Wyznacz przestrzeń, w której możesz się poruszać podczas nagrania, by nie wyjść z kadru, gdy kamera jest ustawiona nieruchomo (lub poproś operatora o wyznaczenie tej przestrzeni).

Unikaj biżuterii i daj się pomalować



Mieniąca się biżuteria może powodować refleksy, a duże ozdoby mogą odwracać uwagę od twojej wypowiedzi. Dzięki makijażowi oświetlenie nie będzie wywoływać refleksów na twojej skórze.

Przekazuj właściwe emocje



Uśmiech i pozytywne nastawienie prelegenta udzielają się słuchaczom. Jednak nie okazuj radości, gdy mówisz o rzeczach poważnych lub smutnych – nie będzie to naturalne ani wiarygodne.

Zanim wystąpi się przed kamerą (np. w studiu telewizyjnym lub w sondzie ulicznej), warto wcześniej porozmawiać zarówno z osobą prowadzącą rozmowę, jak i z operatorem kamery. To, jak wypadniemy na nagraniu, będzie zależało również od pracy operatora.

O zasadach zachowania przed kamerą warto pamiętać również podczas przygotowywania np. poradnika kucharskiego czy pokazu tresury ulubionego zwierzęcia. Wtedy oprócz kucharza czy tresera ważne są potrawa lub zwierzę. To one będą głównymi obiektami nagrania – należy zadbać o odpowiednie ich sfilmowanie, a samemu pozostać nieco w tle.

Na planie filmowym mogą się zdarzyć nieprzewidziane sytuacje. Ujęcia z wielu kamer (np. kilku smartfonów lub tabletów) ułatwią pozbycie się niepotrzebnych ujęć na etapie montażu, a także pozwolą zwiększyć dynamikę ostatecznej wersji filmu.

Dobra rada

Jeśli nagrywasz film z kilku kamer, poproś kogoś o pomoc. Możesz także ustawić kilka urządzeń na stałe, a samodzielnie nagrywać ujęcia z ręki.

Ćwiczenie 6

Przeanalizuj filmy dotyczące wybranego zagadnienia, które mają dużą oglądalność w internecie. Zastanów się:

- w jaki sposób zostały nagrane i jakie techniki w nich zastosowano,
- jak zachowują się osoby występujące na nagraniach.

Praca z mikrofonem

Tworzenie publikacji dźwiękowych oraz filmowych wymaga odpowiedniej pracy z mikrofonem. Aby uzyskać dobre efekty, warto stosować się do kilku podstawowych zasad.

- ▶ **Przećwicz pracę z mikrofonem.** Wypróbuj różne ustawienia parametrów mikrofonu oraz różne jego położenie. Wiele mikrofonów pozwala wybrać sposób nagrywania (np. dźwięk dookołny lub kierunkowy) oraz czułość. Część parametrów można ustawić w programie do rejestrowania dźwięku. Zazwyczaj powinno się zachować odległość kilkunastu centymetrów między twarzą a mikrofonem. Unikaj zbyt głośnego dźwięku powodującego zniekształcenie nagrania. W razie potrzeby zmniejsz czułość mikrofonu.
- ▶ **Zadbaj o wyciszenie pomieszczenia.** Większość mikrofonów rejestruje tzw. **szum tła** (ang. *background noise*), czyli dźwięki pochodzące z otoczenia. Sprawdź, czy tam, gdzie planujesz nagrywać, występują takie dźwięki. Zapewnij sobie spokojne miejsce: zamknij okno, drzwi i poinformuj innych o nagraniu (np. przyklej na drzwiach kartkę z odpowiednim komunikatem). Unikaj echa.
- ▶ **Przygotuj skrypt wypowiedzi.** Nie musi to być dokładny zapis tego, co chcesz powiedzieć, lecz ogólny plan, który pozwoli uniknąć pomyłek i przestojów. Wypowiedź będzie żywsza niż w przypadku odczytania przygotowanego tekstu.
- ▶ **Zadbaj o dykcję i natężenie głosu.** Mów do mikrofonu tak jak do osoby siedzącej naprzeciwko ciebie przy stole, w głośnym otoczeniu.
- ▶ **Odpowiednio zareaguj na pomyłkę.** Zrób chwilę przerwy i dopiero potem wznów nagranie. Przerwa ułatwi później montaż.

Warto wiedzieć

Mikrofony rejestrują nawet dźwięki występujące w trakcie oddychania. Mogą one stanowić istotne zakłócenia dla odbiorcy.

Szum tła

Warto wiedzieć

W telewizji często używa się tzw. telepromptera, czyli ekranu, który wyświetla tekst do odczytania przez osobę występującą przed kamerą. Widz odnosi wtedy wrażenie, że tekst jest wygłaszany z pamięci.

Jak mówić do mikrofonu?

Niezależnie od tego, czy wypowiadamy się na filmie czy w materiale audio, pomóc może wyobrażenie sobie osób, które będą nas słuchać. Oprócz tego warto zastosować kilka prostych zasad.

Mów naturalnie



Mów podobnie jak podczas rozmowy: zmieniaj tempo, natężenie i brzmienie głosu. Nie używaj stylu urzędowego i zbyt oficjalnego.

Mów wyraźnie



Lepiej mówić wolniej i ciszej niż zbyt szybko i zbyt głośno. Unikaj zjadania końcówek i wydawania zbędnych dźwięków.

Mów prosto



Dopasuj język do odbiorcy. Jeśli nie jest on sprecyzowany, używaj prostego języka, aby każdy mógł cię zrozumieć.

Mów konkretnie



Używaj krótkich zdań oraz unikaj dygresji. Korzystaj z przygotowanego skryptu.

Nagrywanie i udostępnianie ekranu

Podczas nagrywania operacji wykonywanych na ekranie komputera również warto stosować się do kilku zasad.

Przygotuj ekran komputera



Usuń z pulpitu, paska zadań oraz zasobnika systemowego zbędne dokumenty i ikony. Pozamykaj niepotrzebne okna aplikacji oraz zakładki w przeglądarce internetowej. Ustaw jako tapetę tło w jednolitym, stonowanym kolorze.

Zablokuj powiadomienia



Ustaw tryb pracy komputera tak, aby zablokowane były wszystkie powiadomienia (np. od programu antywirusowego). Zamknij komunikatory, wyłącz automatyczne aktualizacje. Zablokuj wyskakujące okna w przeglądarce internetowej.

Stosuj zasady zachowania się przed kamerą



Jeśli chcesz w materiale uwzględnić siebie, zastosuj się do wskazówek dotyczących wystąpień przed kamerą oraz pamiętaj, aby czasem spojrzeć w obiektyw. Dobrze to zrobić wtedy, kiedy zapowiadasz, co za chwilę zaprezentujesz.

Skup się na wskaźniku myszy



Poruszaj kursorem myszy precyzyjnie – wolnymi, płynnymi ruchami wskazuj wykonywane operacje. Skorzystaj z narzędzi do wygładzania ruchów kursora w programie rejestrującym ekran komputera.

Podziel materiał na mniejsze fragmenty



Materiał zarejestruj jako serię krótkich nagrań. Zaczynij nagrywać kilka sekund przed właściwą częścią filmu i zakończ kilka sekund po niej. W kolejnych nagraniach staraj się zachować te same ustawienia ekranu i mikrofonu.

Nie spiesz się



Zarezerwuj sobie odpowiednio dużo czasu, zwłaszcza jeśli masz małe doświadczenie w nagrywaniu. Stworzenie jednogodzinowego samouczka może zająć nawet ponad godzinę.

Dobra rada

Stosuj z umiarem efekty dodatkowe – animacje napisów i przejść między ujęciami oraz dżingle.

Dobra rada

Na urządzeniach mobilnych możesz użyć aplikacji Hokusai Audio Editor (iOS/iPadOS) lub Super Sound (Android).

ScreenCast,
s. 377 

Montaż materiału

Przed opublikowaniem materiału trzeba wykonać jego korektę i montaż. Warto wyregulować dźwięk, usunąć szum i ustabilizować obraz. Do wideo można dodać plansze, napisy początkowe i końcowe, a do audio – zapowiedzi, dżingle i zakończenie.

Do montażu prostych projektów filmowych można wykorzystać oprogramowanie dostarczone wraz z systemem, np. aplikację Zdjęcia w Windows, iMovie w macOS czy iPadOS. Przy bardziej złożonych projektach warto sięgać po programy oferujące więcej opcji, np. używany w produkcjach filmowych DaVinci Resolve. Do pracy nad dźwiękiem może się przydać program Audacity, bezpłatny i dający duże możliwości.

Do nagrywania ekranu oraz podstawowego montażu **screencastu** można wykorzystać usługę Stream (wymagane jest szkolne konto Office 365). Wbudowana w nią funkcja **Zarejestruj ekran** pozwala nagrać czynności wykonywane na pulpicie lub w wybranej aplikacji, narrację oraz obraz z kamery. Czas nagrania nie może jednak w tym wypadku przekroczyć 15 minut.

Innym sposobem jest wykorzystanie wbudowanej w system operacyjny funkcjonalności ukrytej pod skrótem klawiszowym **Windows + G**.

Zapamiętaj

Zapewnienie odbiorcom ciekawej rozrywki lub wartościowej wiedzy może przysłonić niedoskonałości techniczne przygotowanego materiału. Dlatego przed opracowaniem publikacji należy się przede wszystkim zastanowić, jaki jest jej cel.

Warto wiedzieć

W systemach macOS i iPadOS również znajdują się wbudowane narzędzia do nagrywania ekranu.

20.5. Przygotowanie materiału do publikacji

Po opracowaniu materiału musimy wybrać format, w którym go opublikujemy. Dotyczy to przede wszystkim plików audio, wideo oraz grafiki. Mimo że większość serwisów, w których możemy publikować, przyjmie niemal dowolny format materiału, warto rozważyć, który będzie najlepszy dla naszego projektu. O jakości publikacji decydują bowiem nie tylko oczywiste parametry, takie jak rozdzielczość czy liczba klatek na sekundę, lecz także sposób zapisu pliku oraz rodzaj kompresji.

Kompresję danych stosuje się przede wszystkim w celu zmniejszenia objętości pliku. Dzięki kompresji te same lub prawie te same informacje można zapisać za pomocą mniejszej liczby bitów, zajmą więc mniej miejsca na dysku komputera, a także będą szybciej i efektywniej przesyłane przez internet. Stosunek rozmiaru danych przed kompresją do rozmiaru po jej zastosowaniu nazywamy **stopniem kompresji**.

Warto wiedzieć

Serwisy internetowe muszą działać sprawnie również przy wolniejszych łączach internetowych. Pomaga w tym m.in. kompresja danych.

Kompresja danych

Stopień kompresji

Ćwiczenie 7

- Za pomocą dostępnej w systemie operacyjnym funkcji skompresuj plik, który przekaże ci nauczyciel (np. *kompresja.txt*). Porównaj wielkość pliku przed kompresją i po jej wykonaniu. Jaki stopień kompresji udało się osiągnąć?
- Udostępniony przez nauczyciela plik, np. *kompresja.bmp*, zapisz w różnych formatach. Porównaj wielkość oraz jakość utworzonych plików. Przedyskutujcie w klasie, jak można zastosować wybrane formaty.
- Udostępnione przez nauczyciela zdjęcie jest zapisane w dwóch plikach, np. *zdjecie_1.bmp* i *zdjecie_1.jpg*. Skompresuj oba pliki. Sprawdź, który z nich ma wyższy stopień kompresji.

Rodzaje kompresji danych i format ich zapisu

Aby zwiększyć zasięg i dostępność publikowanych materiałów, należy zadbać o to, by pliki nie miały zbyt dużego rozmiaru. Z drugiej jednak strony materiał powinien być jak najlepszej jakości. Dlatego format publikacji i zastosowany rodzaj kompresji mają duże znaczenie.

Warto wiedzieć

Algorytm kompresji tekstu omówiono w podręczniku *Informatyka na czasie 2. Zakres rozszerzony*, s. 310–316.

Metody kompresji możemy podzielić ze względu na możliwość odtworzenia pierwotnej informacji (każdego jej bita) po przeprowadzeniu **dekompresji**, czyli procesu odwrotnego do kompresji.

Kompresja bezstratna • **Kompresja bezstratna** w uproszczeniu polega na zastępowaniu powtarzających się informacji znacznikami, które zajmują mniej miejsca, ale umożliwiają odtworzenie pierwotnego zapisu. Tę metodę stosuje się m.in. w kompresji tekstów, grafik i dźwięków. Można ją porównać do zastępowania w tekście pełnych sformułowań skrótami – np. zamiast pisać „między innymi”, możemy użyć skrótu „m.in.”.

Kompresja stratna • **Kompresja stratna** oznacza utratę części danych niemających dużego wpływu na odczytanie znaczenia całości. Dekompresja nie pozwoli więc odtworzyć pierwotnych informacji, ale pozwoli uzyskać ich satysfakcjonujące przybliżenie. Ten typ kompresji jest stosowany w formatach **graficznych**, **audio** oraz audio-video. Utrata części barw w obrazie lub usunięcie z utworu muzycznego dźwięków najmniej słyszalnych przez ludzkie ucho mogą być akceptowalne dla użytkownika.

W przypadku grafiki przykładami formatów wykorzystujących kompresję stratną są: JPEG i DjVu, a bezstratną PNG i GIF. Format TIFF obsługuje oba typy kompresji. Formatami bez kompresji są: PDS, XCF oraz BMP. Format BMP może również wykorzystywać kompresję bezstratną.

Samo rozszerzenie pliku nie określa wszystkich parametrów formatu, w tym sposobu kompresji. Niektóre formaty pozwalają nie tylko korzystać z różnych rodzajów kompresji, lecz także określać jej **stopień**.

Tabela 20.3 opisuje wybrane popularne formaty plików wideo.

Format	Opis formatu pliku
MP4	Jest to jeden z najpopularniejszych formatów pliku wideo. Wykorzystuje algorytm kodowania MPEG-4, który zapewnia niższą jakość niż niektóre konkurencyjne standardy, ale dobrze nadaje się do filmów zamieszczanych w serwisach YouTube, Facebook, Twitter i Instagram.
MOV	Format QuickTime Movie został opracowany przez Apple. Pozwala na zachowanie wysokiej jakości wideo, ale pliki zazwyczaj mają duże rozmiary. Jest obsługiwany w serwisach Facebook i YouTube.
WMV	Format Windows Media Viewer został opracowany przez Microsoft. Pliki WMV zapewniają dobrą jakość wideo, ale mają duże rozmiary i nie umożliwiają wyboru własnych proporcji obrazu. Format jest obsługiwany w serwisie YouTube.
AVI	Format Audio Video Interleave został opracowany przez Microsoft. Działa w większości przeglądarek i zapewnia wysoką jakość, ale charakteryzuje się znacznym rozmiarem plików. Jest obsługiwany w serwisie YouTube.
MKV	Format Matroska Multimedia Container należy do kategorii rozwiązań <i>open source</i> . Może zawierać dodatkowe informacje, takie jak napisy w różnych językach lub wiele ścieżek dźwiękowych. Nie jest obsługiwany przez większość serwisów społecznościowych.

Tabela 20.3. Opis popularnych formatów plików wideo

Reprezentacja obrazów, dźwięków i animacji,
podręcznik
Informatyka na czasie 2.
Zakres rozszerzony,
s. 43–44

Stopień kompresji,
s. 383

Dobra rada

Aby zaoszczędzić miejsce na dysku komputera, możesz użyć narzędzi do archiwizacji, które skompresują wybrane pliki lub katalogi. Najlepsze efekty uzyskasz, jeśli skompresujesz pliki zapisane w formacie bez kompresji.

Warto wiedzieć

Oprogramowanie typu *open source* jest dostępne dla wszystkich wraz z kodem źródłowym. Często jest rozwijane w ramach społeczności.

20.6. Jak poznać (lepiej) swojego odbiorcę?

Przed przygotowaniem publikacji internetowej przyjmujemy wiele założeń na jej temat. Jednym z najważniejszych jest pytanie o głównego odbiorcę. Nie zawsze nasze założenia i wynikające z nich działania odniosą zamierzony skutek. Dlatego warto śledzić efekty i wyciągać wnioski – poprawiać zamieszczone treści i uwzględniać nową wiedzę podczas przygotowywania kolejnych materiałów.

Do zbadania tego, w jaki sposób nasza publikacja oddziałuje na odbiorców, możemy wykorzystać gotowe narzędzia. Przedstawia je tabela 20.4.

Narzędzie	Przeznaczenie
Google Trends	Pozwala sprawdzać, które hasła są najczęściej wyszukiwane w sieci, oraz badać zmiany ich popularności w czasie. Na podstawie tych wyników można zidentyfikować poszukiwane przez internautów słowa i używać ich we własnych tekstach, np. w opisach, tytułach, lidach czy w głównej treści publikacji.
Narzędzia analityczne (np. Google Analytics, stat.4u.pl)	Służą do zbierania informacji o osobach odwiedzających dane serwisy. Można się z nich dowiedzieć więcej o swoich odbiorcach, m.in. poznać ich wiek, narodowość, miejsce, z którego łączą się z serwisem, sposób dotarcia do naszych treści (przez wyszukiwanie, bezpośrednie wpisanie adresu czy link z innego serwisu), czas spędzony w serwisie, odwiedzane podstrony.
Narzędzia badające internet (np. Mediapanel)	Warto korzystać z informacji i raportów przekazywanych przez firmy badające internet, przepływy odbiorców oraz trendy. Informacje te są skierowane głównie do reklamodawców, ale mogą być również przydatne przy rozwijaniu własnego serwisu.

Tabela 20.4. Narzędzia analizujące zasoby internetu

Łączenie informacji z różnych źródeł oraz śledzenie trendów wyszukiwania pomoże przygotowywać treści nie tylko lepiej odpowiadające oczekiwaniom odbiorców, lecz także łatwiejsze do znalezienia w gęstocie informacji.

Dobra rada

Do narzędzi analitycznych możesz sięgnąć również na etapie planowania publikacji. Pozwolą one m.in. odnaleźć słowa często wyszukiwane przez internautów.

Lid,
s. 369 

Warto wiedzieć

Wiele serwisów (np. tych, które pozwalają publikować filmy) oferuje swoje własne, wbudowane narzędzia analityczne.

A to ciekawe

Jak poznać przyszłych klientów?

Niektóre firmy analityczne i doradcze publikują raporty dotyczące zachowań konsumenckich w internecie. Część takich raportów jest dostępna dla każdego użytkownika. Pozwalają one zdobyć wiedzę na temat m.in. rynku e-commerce, wykorzystania urządzeń mobilnych podczas zakupów online czy preferowanych metod płatności. Często raporty są wzbogacone o fachowe komentarze i opinie ekspertów.

Podsumowanie

- Aby zainteresować odbiorców przygotowaną treścią, należy nadać jej przyciągający uwagę tytuł, wykorzystać wysokiej jakości ilustracje oraz teksty zawierające słowa, które interakcji łatwo odnajdą dzięki wyszukiwarkom.
- Treść publikacji należy dostosować do odbiorcy, m.in. jego wieku oraz doświadczeń. Trzeba zadbać o odpowiedni język przekazu.
- W konstruowaniu tekstów informacyjnych może pomóc konspekt zapisany w postaci odwróconej piramidy – najważniejsze treści, najszerzej opisane, umieszczamy na początku komunikatu, a najmniej ważne – na końcu.
- Identyfikacja wizualna oraz dźwiękowa nadaje publikacjom jednolitą, uporządkowaną formę i pozwala skojarzyć je z autorem.
- W pracy z kamerą i mikrofonem warto stosować podstawowe zasady, które pomogą uniknąć niepotrzebnego powtarzania nagrań oraz skrócić czas montażu.
- Narzędzia analityczne pozwalają lepiej poznać odbiorców treści oraz dopasować publikacje do ich potrzeb.

Zadania

- ★ **1** Poszukaj serwisów oferujących gotowe dżingle, które można wykorzystać we własnych nagraniach. Określ dostępne rozszerzenia plików.
- ★ **2** Sprawdź, czy i w jaki sposób możesz spowodować, by publikowane przez ciebie nagrania w różnych serwisach wideo (np. YouTube, Vimeo) były odtwarzane bez reklam.
- ★ **3** Znajdź informacje o tym, jakie tematy i hasła były najczęściej wyszukiwane przez ostatnich 12 miesięcy w Polsce w kategorii edukacja. Zastanów się, jak możesz wykorzystać tego typu analizy do przygotowania własnych treści na określony temat.
- ★★ **4** Przygotuj grafikę do wykorzystania w sieciach społecznościowych reklamującą serię webinarów lub blog na interesujący cię temat. Załóż, że grafika będzie kierowana do twoich kolegów. Wybierz odpowiedni format pliku.
- ★★ **5** Przygotuj tekst, który będzie relacją z wybranego przez ciebie webinarium lub podcastu. Tekst powinien zachęcić twoich rówieśników do odsłuchania nagrania.
- ★★★ **6** Przygotuj podcast na wybrany temat. Opracuj towarzyszące mu materiały promocyjne: zapowiedź tekstową i grafiki do zamieszczenia w sieciach społecznościowych lub na stronach WWW.
- ★★★ **7** Przygotuj publikację wideo (webcast, screencast lub webinarium) na wybrany temat związany z edukacją. Opracuj towarzyszące publikacji materiały promocyjne (zapowiedź tekstową, grafiki do zamieszczenia w sieciach społecznościowych lub na stronach WWW).

21. Grafiki informacyjne

Wraz z rozwojem technologii zwiększyła się rola komunikacji wizualnej. Ma ona długą tradycję, gdyż najstarsze zachowane środki przekazu to powstałe przed tysiącami lat malowidła naskalne. Ukazywały one wizerunki zwierząt oraz sceny polowań i nie były tylko dekoracjami – dostarczały także informacji. Współcześnie grafiki informacyjne uznaje się za gatunek dziennikarski, a ich głównym zadaniem jest skuteczne przekazanie wiadomości. W tym temacie dowiesz się, jak pracować nad takimi grafikami.

Cele lekcji

- Dowiesz się, czym jest grafika informacyjna, oraz poznasz przykłady różnych rodzajów takich grafik.
- Nauczysz się projektować proste infografiki.
- Rozwiniesz umiejętność pracy z grafiką komputerową.

Ćwiczenie 1

Opisz swoje pasje i zainteresowania, używając wyłącznie:

- a. edytora tekstu i 150–200 słów,
- b. edytora grafiki i narzędzia **Pióro**,
- c. edytora grafiki i dwóch narzędzi: **Pióro** oraz **Tekst**.

21.1. Sposoby przekazywania informacji

Informacje można przekazywać różnymi metodami. Poniżej przedstawiono trzy sposoby podania danych dotyczących marnowania żywności.

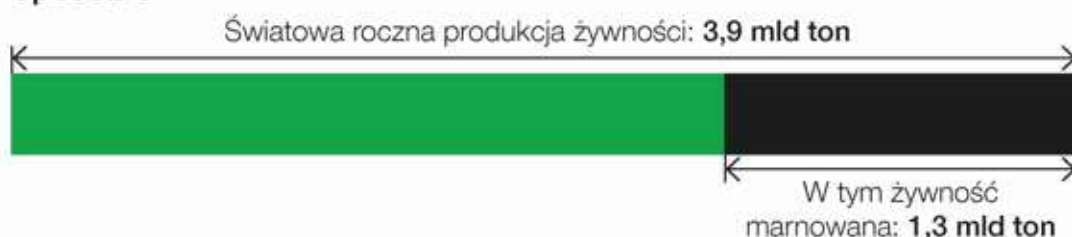
Sposób 1

Organizacja Narodów Zjednoczonych ds. Wyżywienia i Rolnictwa, nazywana w skrócie FAO, zajmuje się redystrybucją żywności i rozwojem obszarów wiejskich. Opracowuje również raporty dotyczące żywności. Z raportu opublikowanego w 2011 r. wynika, że w ciągu roku na świecie produkuje się ok. 3,9 mld ton żywności, z czego marnuje się 1,3 mld ton jej jadalnych części.

Sposób 2

Według raportu *Global food losses and food waste – Extent, causes and prevention*, przygotowanego w 2011 r. przez Organizację Narodów Zjednoczonych ds. Wyżywienia i Rolnictwa, w ciągu roku na świecie marnuje się 1,3 mld ton żywności, co stanowi 1/3 całej produkcji (ok. 3,9 mld ton).

Sposób 3



Źródło: *Global food losses and food waste – Extent, causes and prevention*, FAO, 2011.

Dobra rada

Aby graficznie porównać wartości, możesz użyć prostokątów, których pola są w odpowiednich proporcjach. Taki sposób wizualizacji ma najmniejszy tzw. współczynnik kłamstwa (ang. *lie factor*). Unikaj natomiast wykresów kołowych. Mają one wysoki współczynnik kłamstwa, ponieważ porównywanie wycinków koła jest trudne dla ludzkiego oka.

Jeśli zależy nam na porównaniu dwóch wartości, to rozbudowany opis (sposób 1) nie jest efektywny. Lepiej ten cel realizuje krótszy tekst (sposób 2). Najlepiej jednak sprawdzi się prosta grafika (sposób 3), która do minimum ogranicza wszystkie dodatkowe informacje.

Jednym ze skutecznych sposobów przekazywania informacji są **grafiki informacyjne**, nazywane również **infografikami** (ang. *infographics*). Przedstawiają one treść w sposób wizualny tak, aby była ona możliwie prosta i dostosowana do odbiorcy.

Grafika informacyjna
(infografika)

Uporządkowanie informacji

Infografiki powinny zawierać informacje już uporządkowane. Jest to konieczne, aby odbiorca mógł je łatwo zrozumieć i przyswoić. Wyróż-

Sposoby porządkowania informacji mamy 5 głównych **sposobów porządkowania informacji**:

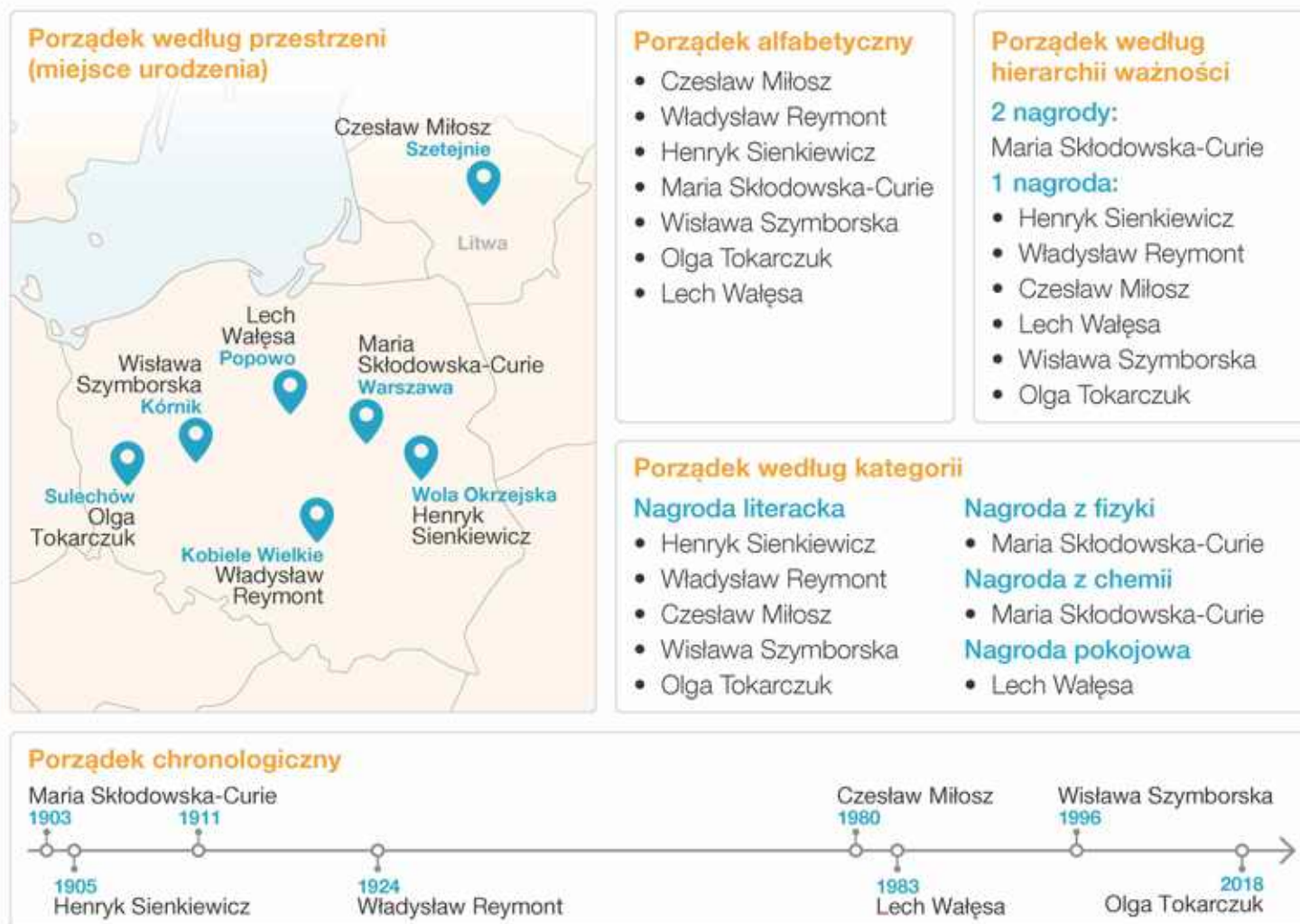
- ▶ według alfabetu,
- ▶ według czasu,
- ▶ według hierarchii,
- ▶ według kategorii,
- ▶ według przestrzeni.

Wszystkie wymienione sposoby zastosowano do uporządkowania informacji o polskich noblistach.

Dobra rada

Pamiętaj, że zebranie informacji na dany temat to dopiero połowa sukcesu. Zanim zaprezentujesz dane odbiorcy, odpowiednio je uporządkuj.

Sposoby porządkowania informacji



Jeśli dobrze wybierzemy sposób uporządkowania informacji, możemy przekazać odbiorcy dodatkową wiedzę – coś, czego nie byłoby widać przy ustawieniu tych samych (lub podobnych) danych w innym porządku.

Na przykład umieszczenie noblistów na mapie pozwala zauważyć, że nie wszyscy urodzili się w obecnych granicach Polski, a uporządkowanie chronologiczne pokazuje, jak długo Polacy czekali na nagrodę po roku 1924.

Dobra rada

Sposoby porządkowania informacji możesz łączyć. Na przykład na osi czasu oprócz dat i nazwisk laureatów nagrody Nobla możesz dodatkowo umieścić informację o kategoriach, w których tę nagrodę otrzymali.

Ćwiczenie 2

- Jakie informacje o polskich noblistach nie są widoczne w porządku alfabetycznym, a jakie – w porządku według kategorii?
- Pomyśl o 10 przedmiotach codziennego użytku, z których korzystasz. Uporządkuj je na dwa wybrane przez siebie sposoby.

21.2. Piktogramy

Z przekazywaniem informacji za pomocą obrazu spotykamy się dziś w różnych dziedzinach życia. Na co dzień korzystamy choćby ze znaków drogowych, tabliczek informacyjnych czy emotikonów i emoji. Są to przykłady **piktogramów** (ang. *pictograms*). Dobry piktogram cechują: prostota, uniwersalność i niezależność od kultury. Na przykład znaki drogowe (rys. 21.1) są uniwersalnym sposobem zapisu informacji, zrozumiałym dla kierowców na całym świecie.

Piktogram



Rys. 21.1. Wybrane znaki ostrzegawcze

Warto wiedzieć

W znakach ostrzegawczych nieprzypadkowo zostały użyte kolory: żółty, czerwony i czarny. Kolor żółty łatwo dostrzec z dużej odległości. Najbardziej kontrastowy do żółtego jest kolor czarny. Natomiast czerwień obramowania podkreśla ważność znaku.

Piktogramy wykorzystuje się również w czasie igrzysk olimpijskich. Oznacza się nimi obiekty sportowe, pojawiają się na mapach, biletach oraz podczas transmisji telewizyjnych. Na igrzyska w Monachium w 1972 r. niemiecki projektant Otto Aicher opracował piktogramy przedstawiające różne dyscypliny sportu (rys. 21.2).



Rys. 21.2. Wybrane piktogramy olimpijskie z 1972 r.

Piktogramy pełnią ważną funkcję w komunikacji – niwelują barierę językową oraz pomagają osobom z niepełnosprawnościami (np. osobom niemówiącym).

Ćwiczenie 3

- Zastanówcie się w grupie, w jakich sytuacjach korzysta się z piktogramów. Podajcie kilka przykładów z różnych dziedzin.
- Odszukaj piktogramy zastosowane w tym podręczniku i określ ich rolę.

21.3. Przykłady infografik

Infografiki są rodzajem komunikacji wizualnej. Ze względu na cel przekazu można je podzielić na dwie kategorie:

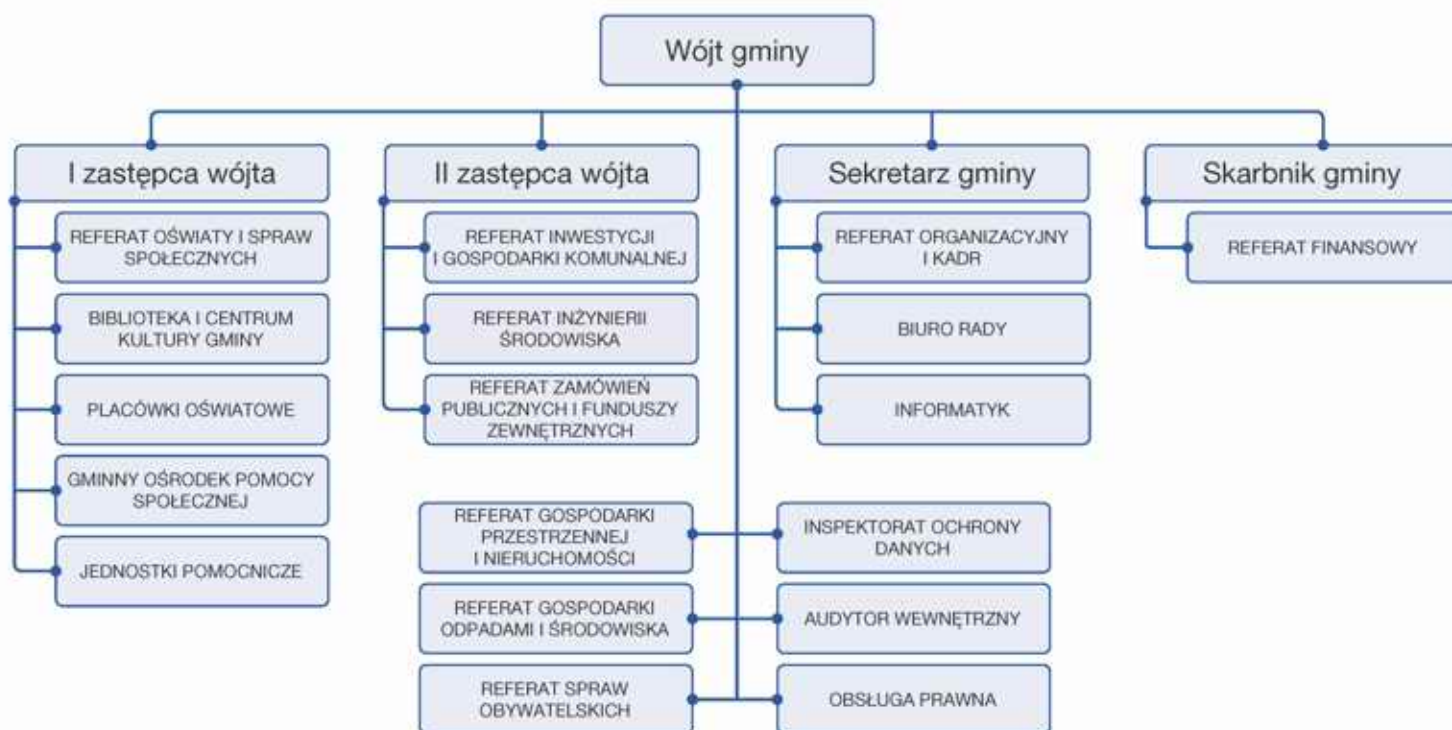
Grafika narracyjna ➤ **grafikę narracyjną** – należą do niej: diagramy przepływowe (opisujące procesy), osie czasu, harmonogramy, schematy organizacyjne, mapy myśli,

Wizualizacja danych ➤ **wizualizację danych** – należą do niej: grafiki liczbowe, diagramy statystyczne, grafiki porównawcze (np. do porównywania kilku cech, postaw), mapy, kartogramy i kartodiagramy.

Wiele o tworzeniu infografik można się nauczyć, analizując ich przykłady.

Schemat organizacyjny

Informacje uporządkowane według hierarchii można przedstawić za pomocą **schematu organizacyjnego** (rys. 21.3). Cechują go prostota oraz precyzja przekazu. Schemat taki pozwala jednoznacznie opisać nawet skomplikowaną strukturę.



Rys. 21.3. Przykładowy schemat organizacyjny

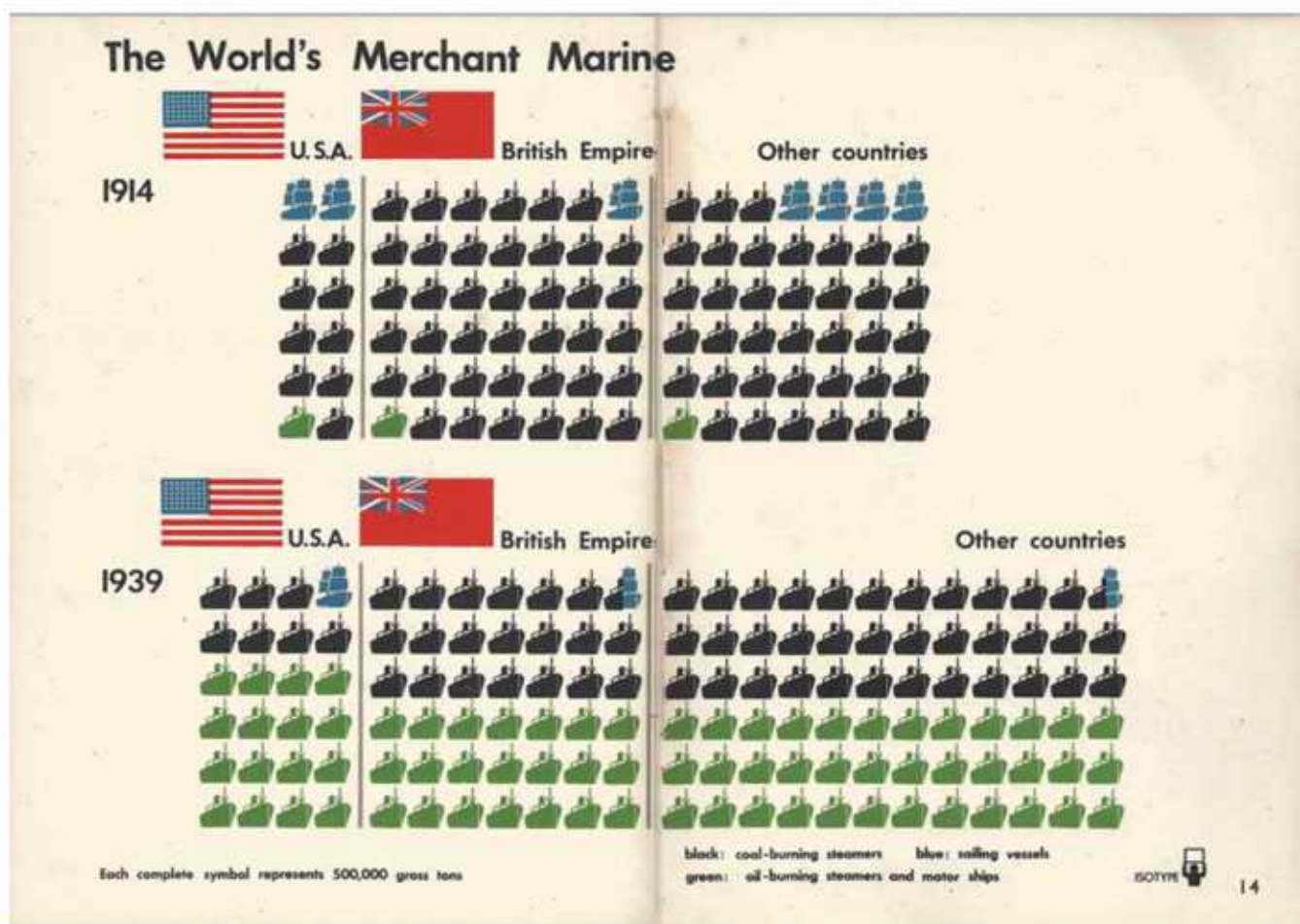
Ćwiczenie 5

Znajdź schemat linii komunikacyjnych wybranego miasta w Polsce. Czy jest inspirowany pomysłem Becka? Zapoznaj się z legendą. Jakiej kolorystyki użyto?

Język piktogramów Isotype

Około 1925 r. zespół ekspertów pracujący pod kierunkiem Ottona Neuratha, socjologa i specjalisty od komunikacji, opracował obrazko-

Język Isotype • wy język nazywany **Isotype** (ang. *International System of Typographic Picture Education*). Jego podstawę stanowiły jednoznaczne piktogramy, które miały za zadanie w uniwersalny i prosty sposób ilustrować złożone treści (rys. 21.5).



Warto wiedzieć

Piktogramy Isotype uzupełnia się o niezbędny tekst. Na infografice o marynarce handlowej są to m.in. tytuł wykresu i informacja o podziale okrętów na należące do USA, Imperium Brytyjskiego oraz reszty świata. Ponadto kolorem oznaczono sposób zasilania okrętu.

Rys. 21.5. Infografika opisująca liczebność marynarki handlowej na świecie w 1914 i 1939 r.

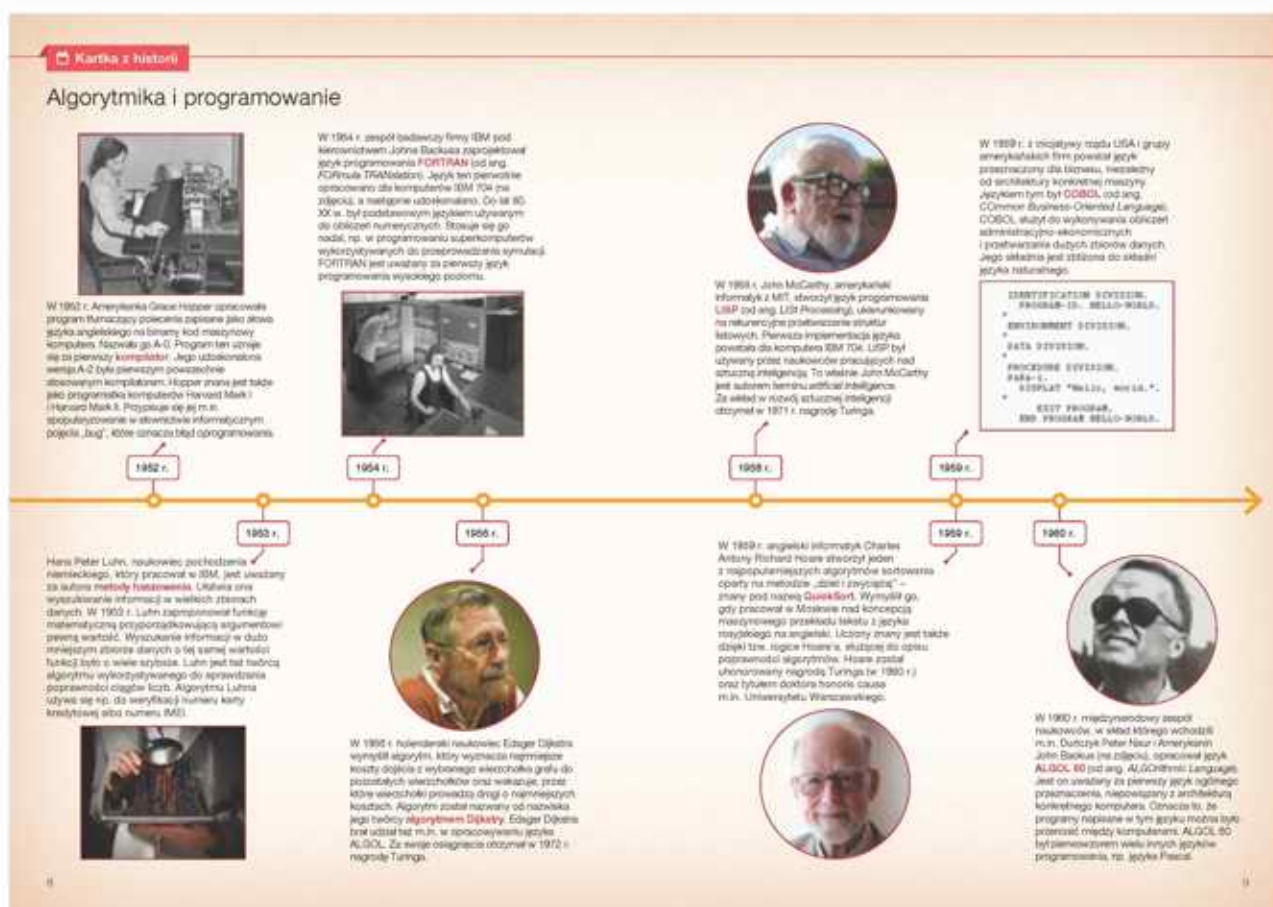
Jak pokazuje powyższy przykład, informację o ilości uzyskiwano nie przez zmianę rozmiaru znaku, lecz przez jego zwielokrotnienie.

Ćwiczenie 6

Przedstaw na infografice liczbę ton jabłek zebranych przez sadowników w Polsce oraz w innych krajach świata (łącznie) w 2000 r. oraz w 2020 r.

Oś czasu

Jeśli chcemy uporządkować informacje chronologicznie, dobrze jest zastosować **oś czasu**. Pokazuje ona przebieg wydarzeń na linii upływającego czasu. Pozwala to odbiorcy jednym spojrzeniem objąć całość zagadnienia, a wprowadzony porządek ułatwia naukę (rys. 21.6).



Rys. 21.6. Oś czasu pochodząca z podręcznika *Informatyka na czasie 3*

Ćwiczenie 7

Przedstaw graficznie, w postaci równoległych osi czasu, sobotnie godziny pracy wybranych urzędów i instytucji użyteczności publicznej (np. główny urząd pocztowy, muzea, ogród zoologiczny, park miejski) w dowolnym mieście w Polsce. Możesz dodać informacje o dziennej i nocnej komunikacji miejskiej.

21.4. Jak stworzyć poprawną infografikę?

Na infografice tekst i obraz powinny się wzajemnie uzupełniać. Przekazywaną treść należy przedstawić w sposób jawny, bez zbędnych metafor. Obrazowi powinien towarzyszyć tylko konieczny tekst objaśniający.

Poprawnie przygotowana infografika przedstawia tylko jeden problem lub temat. Należy zaprezentować go w sposób prosty, przystępny i dostosowany do docelowego odbiorcy.

Warto wiedzieć

Atrakcyjność wizualna nie musi być nieodłączną cechą infografiki. Ważniejsze jest skuteczne przekazanie informacji. Na przykład schematy organizacyjne dobrze realizują to zadanie, choć najczęściej nie są zbyt efektowne.

Do cech dobrej infografiki możemy zaliczyć:

- ▶ odpowiednie uporządkowanie informacji,
- ▶ odwołanie się do wiedzy i doświadczeń odbiorcy,
- ▶ prostotę i harmonię kompozycji,
- ▶ właściwy dobór kolorów,
- ▶ odpowiednie rozłożenie akcentów.

Aby odbiorcy łatwiej było przyswajać informacje we właściwej kolejności, warto różnicować wielkość elementów (co podkreśla ich hierarchię) oraz układać je względem siebie według ustalonego porządku. Znaczącą rolę odgrywają również kolory.

Funkcje koloru

Kolory na infografice należy dobierać uważnie, ponieważ są one również nośnikami treści. Można wyróżnić kilka **funkcji koloru**. Przedstawiono je w tabeli 21.1.

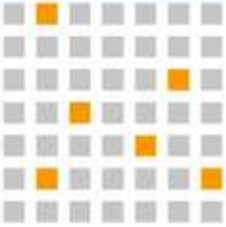
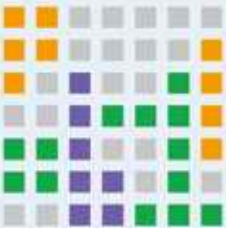
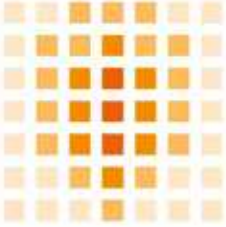
	Funkcja koloru	Interpretacja graficzna
Akcentująca  funkcja koloru	Funkcja akcentująca pozwala zwrócić uwagę na wybrany element.	
Rozróżniająca  funkcja koloru	Funkcja rozróżniająca pozwala odróżnić elementy od siebie (i w niektórych przypadkach podzielić infografikę na sekcje).	
Wartościująca  funkcja koloru	Funkcja wartościująca pozwala porównać ze sobą wybrane elementy.	

Tabela 21.1. Funkcje koloru

Warto wiedzieć

Kolory mają również określoną wagę i przeznaczenie, np. do oznaczania zakazu (m.in. na znakach drogowych) używamy koloru czerwonego.

Kolory mają też określoną symbolikę, ściśle związaną z kulturą. Na przykład w kulturze europejskiej czerni oznacza zazwyczaj żałobę lub śmierć, czerwień zaś dynamizm, wysoką temperaturę lub krew.

Ćwiczenie 8

Zastanów się, jakie skojarzenia kulturowe są związane z kolorem:

- a. żółtym,
- b. zielonym.

Dobór kolorów

W trakcie prac nad infografiką należy zastanowić się nie tylko nad funkcją kolorów, lecz także nad ogólną kolorystyką pracy. W jej wyborze może pomóc tzw. **koło barw** (ang. *color wheel*), na którym umieszcza się kolory podzielone na trzy podstawowe grupy:

- ▶ **barwy podstawowe** – czerwony, niebieski, żółty,
- ▶ **barwy pochodne** – powstałe w wyniku wymieszania dwóch barw podstawowych: pomarańczowy (czerwony z żółtym), zielony (żółty z niebieskim) oraz fioletowy (niebieski z czerwonym),
- ▶ **barwy trzeciego rzędu** – powstałe w wyniku połączenia koloru podstawowego z sąsiadującym z nim kolorem pochodnym.

Rysunek 21.7 przedstawia tradycyjne koło barw, utworzone w opisanym wyżej sposób dla 12 barw o różnym **nasyceniu**, czyli o różnej zawartości bieli. Kolory ułożone są w kolejności od najbardziej nasyconego, najciemniejszego na brzegu koła do najmniej nasyconego, najjaśniejszego bliżej środka koła.



Rys. 21.7. Koło barw

● Koło barw

● Barwy podstawowe

● Barwy pochodne

● Barwy trzeciego rzędu

● Nasycenie koloru

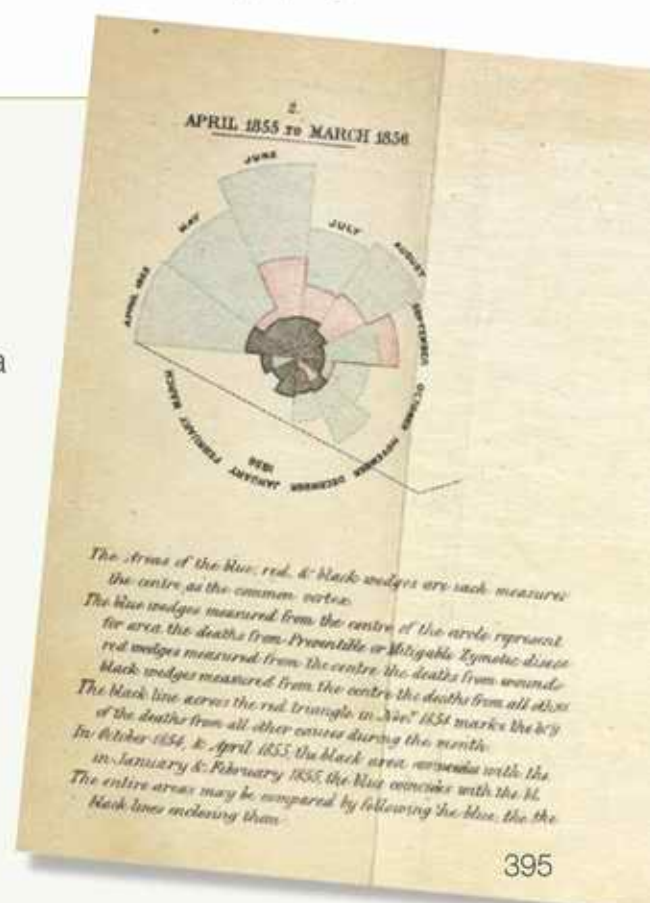
Warto wiedzieć

Założenie, że podstawowymi kolorami są czerwony, niebieski i żółty, jest uwarunkowane historycznie i wywodzi się ze sztuki malarskiej. Współcześnie wiadomo, że dla receptorów ludzkiego oka (czopków) podstawowe barwy to: czerwony, niebieski i zielony. Jeśli postrzegamy wybrany obiekt jako żółty, to znaczy, że do oka dotarło światło czerwone i zielone w proporcjach 1 : 1.

A to ciekawe

Diagram ratujący życie

Podczas wojny krymskiej (1853–1856) angielska pielęgniarka Florence Nightingale opiekowała się rannymi żołnierzami i analizowała dane o ich zgonach. Przygotowała raport, z którego wynikało, że więcej żołnierzy umierało z powodu niedożywienia i fatalnych warunków sanitarnych niż na skutek odniesionych ran. Przełożeni Nightingale nie byli zainteresowani wnioskami płynącymi z raportu, jednak ona pozostawała nieugięta. Ostatecznie swoją analizę zaprezentowała samej królowej Wiktorii. Przygotowała na tę okazję specjalną grafikę, nazywaną dzisiaj diagramem centrycznym albo biegunowym.



Koło barw pozwala dobierać kolory na kilka sposobów. Najczęściej stosowane przedstawiono w tabeli 21.2.

Barwy sąsiadujące

Dobra rada

Jeśli stosujesz barwy sąsiadujące, to powinny one mieć takie samo nasycenie.

Barwy monochromatyczne

Barwy trójkątne

Barwy dopełniające

Układ barw	Wybór z koła barw	Opis
Barwy sąsiadujące		Kolory sąsiadujące nazywa się też analogicznymi. Pozwalają uzyskać wrażenie harmonii, gdyż nasz wzrok dostrzega, że mają ze sobą coś wspólnego.
Barwy monochromatyczne		Jeden kolor w różnych odcieniach i o różnym nasyceniu (ciemne lub jasne, czyli pastelowe). Z reguły projekty tworzone z wykorzystaniem tego schematu są dość spokojne. Unikamy ryzyka błędnego zestawienia różnych kolorów.
Barwy trójkątne		Wyznacza się je w kole barw za pomocą trójkąta równobocznego. Jeden z kolorów powinien dominować nad pozostałymi.
Barwy dopełniające		Kolory umieszczone w kole barw naprzeciw siebie. Trzeba ostrożnie je dobierać, aby nie uzyskać zbyt dużego kontrastu, ponieważ może on wywoływać niepokój.

Tabela 21.2. Dobór kolorów z wykorzystaniem koła barw

Warto wiedzieć

Internetowe narzędzie palettable.io pozwala stworzyć kompozycję 5 kolorów, z których każdy kolejny jest dobierany do wybranych poprzednio.

Podczas dobierania kolorów dobrze sprawdzają się narzędzia online. Wiele z nich umożliwia wykorzystanie koła barw i sposobów opisanych w tabeli 21.2. Przykładami darmowych narzędzi są: Adobe Color (color.adobe.com) oraz canva.com/colors/color-wheel/. Często można w nich też skorzystać z gotowych schematów kolorystycznych.

Ćwiczenie 9

W pliku, który otrzymasz od nauczyciela (np. *firma_transportowa.bmp*), znajduje się schemat organizacyjny pewnej firmy transportowej. Otwórz go w wybranym edytorze grafiki i dobierz do niego wielobarwny schemat kolorystyczny tak, aby kolor był dodatkowym nośnikiem informacji.

Zapamiętaj

Podstawową zaletą infografiki jest to, że ułatwia ona zrozumienie danego zagadnienia. Za dobrze zaprojektowaną infografikę należy uznać taką, która jest czytelna i przystępna dla odbiorcy.

Akcentowanie informacji

Najważniejsze informacje znajdujące się na infografice powinny być wyraźnie zaznaczone i odpowiednio zaakcentowane. Dzięki temu uzyskuje się kontrast między informacjami ważnymi i mniej ważnymi.

Istnieje kilka **sposobów akcentowania najważniejszych informacji** (tekstu lub grafiki) – przedstawia je rysunek 21.8. **Sposoby akcentowania informacji**



Rys. 21.8. Sposoby akcentowania informacji na infografice

Warto wiedzieć

W projektowaniu graficznym oraz składzie tekstu światłem nazywa się odstęp, np. między literami, wyrazami lub elementami graficznymi.

Ćwiczenie 10

W pliku, który otrzymasz od nauczyciela (np. *akcentowanie.pdf*), znajdują się odnośniki do kilku infografik. Wybierz dwie i określ, jakie sposoby akcentowania informacji na nich zastosowano.

Typowe błędy

Źródłem chaosu na infografice może być m.in. niestosowanie się do nawyków wzrokowych czytelników. Przywykliśmy czytać od lewej do prawej i z góry na dół. Jeśli wzrok odbiorcy ma podążać inną ścieżką, to elementy graficzne i tekstowe muszą taką ścieżkę wyraźnie tworzyć.

Do najczęściej popełnianych błędów podczas tworzenia infografik można zaliczyć:

- ▶ nielogiczny układ treści,
- ▶ błędny dobór środków ilustrujących przekazywane informacje,
- ▶ przesadną ozdobność i zbędne szczegóły,
- ▶ źle dobrany typ wykresu lub diagramu,
- ▶ powtórzenia informacji,
- ▶ nieczytelne metafory,
- ▶ błędne użycie koloru.

Przedstawimy teraz przykład poprawnej infografiki, która łączy w sobie wiele różnych technik.

Warto wiedzieć

Nienaturalnym sposobem układania treści jest używanie układu odwrotnego do ruchu wskazówek zegara.

Zastosowanie różnych technik na infografice

Często infografiki są przygotowywane jako samodzielne źródła informacji, w których tekst i obraz wzajemnie się przenikają.

Tytuł infografiki i krótki tekst wprowadzający pomagają zrozumieć przedstawiany temat.

Legendę objaśniającą znaczenie kolorów i piktogramów umieszczono w neutralnym miejscu. Mimo że symbole zegara i krzyża są bardzo sugestywne, wyjaśniono je w legendzie.

Konflikty na świecie

Aktualnie na świecie toczy się kilkadziesiąt konfliktów – zarówno pomiędzy państwami, jak i wewnętrznych. Część z nich znacząco wpływa na dany region oraz świat i jest nagłaśniana przez media, a inne mają wymiar lokalny, ale wszystkie są tragedią ludności cywilnej. Zapoznaj się z opisami wybranych konfliktów, ilustrującymi różnorodność ich przyczyn i przebieg.

LEGENDA

- konflikty między państwami
- konflikty polityczne (m.in. walka o władzę, zbrojna opozycja, konflikty ideologiczne)
- konflikty etniczne (w tym dążenia do niepodległości)
- konflikty religijne
- † liczba ofiar
- 🕒 czas trwania konfliktu

LEGENDA

- konflikty między państwami
- konflikty polityczne (m.in. walka o władzę, zbrojna opozycja, konflikty ideologiczne)
- konflikty etniczne (w tym dążenia do niepodległości)
- konflikty religijne
- † liczba ofiar
- 🕒 czas trwania konfliktu

SYK

006 r. † 100–150 tys.
u trwają wojny narkotykowe.

Policja i wojsko walczą z kartelami narkotykowymi – potężnymi grupami przestępczymi, które kontrolują część kraju. Krwawe konflikty wybuchają również pomiędzy samymi kartelami.

KOLUMBIA
od 1964 r.
† 220 tys.

Tło infografiki jest stonowane. Użyto w nim odcieni tylko jednego koloru, który koresponduje z wymową jednego zdjęcia. Wzmacnia to przekaz o dramacie setek tysięcy ofiar. Na tle wyraźnie wyeksponowano obszary objęte konfliktem.



Co piąte dziecko na świecie żyje w strefie wojny – często nie ma gdzie mieszkać i co jeść, nie może się uczyć, nie ma dostępu do lekarza. Z powodu konfliktów w połowie 2020 r. ze swoich domów wysiedlonych zostało 80 milionów ludzi.

W ramkach znajduje się krótka charakterystyka wybranych konfliktów. Podstawowe informacje – czas trwania konfliktu oraz liczbę ofiar – oznaczono piktogramami.

ROSJA – UKRAINA (DONBAS)

od 2014 r. ↑ 13 tys.

Rosja wykorzystała trudną sytuację wewnętrzną na Ukrainie i zajęła Półwysep Krymski, a następnie wywołała konflikt na wschodzie Ukrainy, udzielając zbrojnego wsparcia zwolennikom niezależności tych terenów.

ponad 30 tys.
konfliktu
ędzy
kami a Żydami
te przez Izrael
po II wojnie światowej, na
których Palestyńczycy chcą
utworzyć własne państwo.

ARMENIA – AZERBEJDŻAN (GÓRSKI KARABACH)

od 1991 r. ↑ 30 tys.

Konflikt toczy się o Górski Karabach. Teren leży w granicach Azerbejdżanu, jednak większość mieszkańców tego obszaru stanowią Ormianie, którzy kontrolują go dzięki wsparciu wojsk Armenii.

TURCJA (KURDOWIE)

od 1984 r.
↑ 34–46 tys.

LIBIA

od 2011 r.
↑ 33–38 tys.

IRAK

od 2003 r.
↑ 450 tys.

AFGANISTAN

od 2001 r.
↑ 150–160 tys.

INDIE – PAKISTAN (KASZMIR)

od 1947 r.
↑ 47–100 tys.

KOREA PÓLNOČNA – KOREA POŁUDNIOWA

od 1953 r.
około 1 tys.

MALI

od 2012 r.
↑ 4–11 tys.

JEMEN

od 1994 r.
ponad 100 tys.

SOMALIA

od 1988 r.
↑ 300–550 tys.

SUDAN (DARFUR)

od 2003 r. ↑ 300–350 tys.

SUDAN POŁUDNIOWY

od 2011 r. ↑ 400 tys.

DEMOKRATYCZNA REPUBLIKA KONGA

od 1996 r. ↑ 2,5–4 mln

REPUBLIKA ŚRODKOWOAFRYKAŃSKA

od 2013 r. ↑ ponad 8 tys.

NIGERIA

od 2009 r. ↑ 27–38 tys.

W kraju zamieszkanym przez chrześcijan i muzułmanów trwa wojna z islamskim ugrupowaniem terrorystycznym Boko Haram. Dąży ono do przekształcenia Nigerii w państwo wyznaniowe i prześladowuje chrześcijan – dopuszcza się porwań, mordów.

SYRIA

od 2011 r. ↑ 470–560 tys.

W kraju toczy się wojna domowa między wojskami prezydenta Baszara Al-Asada a opozycyjnymi ugrupowaniami zbrojnymi. Strona rządowa jest wspierana militarnie przez Rosję i Iran, natomiast jej przeciwnicy – przez USA i Turcję.

Informacje uporządkowane są według przestrzeni (mapa) oraz kategorii (rodzaje konfliktów), którym przyporządkowano kolory.

Na infografice zastosowano następujące funkcje kolorów:

- akcentującą – na monochromatycznym tle mocne kolory zwracają uwagę czytelnika,



- różnicującą – różne barwy odpowiadają różnym rodzajom konfliktów.



21.5. Tworzymy własną infografikę

Warto wiedzieć

Do najważniejszych twórców infografik na świecie należy Fernando Baptista. Jest on autorem wielu niezapomnianych prac, m.in. dla National Geographic. Część jego infografik można odnaleźć w internecie.

Praca nad infografiką to w dużej mierze proces twórczy, wymykający się twardym regułom. Określimy jednak kilka kluczowych czynności, które należałoby wykonać podczas tworzenia dowolnej grafiki informacyjnej. Przedstawimy je w postaci listy kroków:

- 1 Wybierz problem, który infografika ma objaśniać.
- 2 Określ grupę odbiorców i sformułuj najważniejszy cel infografiki.
- 3 Zbierz informacje na wybrany temat i zredaguj zwięzły tekst.
- 4 Zamień zredagowany tekst na język wizualny.
- 5 Określ podstawowe parametry infografiki i opracuj jej szkic.
- 6 Wykonaj infografikę zgodnie ze szkicem.
- 7 Wprowadź poprawki i dopracuj kompozycję.

Pokażemy na przykładzie, jak można zrealizować powyższe kroki.

Infografika *Recykling papieru*

Założmy, że mamy za zadanie opracować infografikę omawiającą proces recyklingu papieru. Chcemy, aby uczniowie szkół podstawowych i ponadpodstawowych (grupa docelowa) rozumieli, jak efektywnie można przetwarzać papier, jeśli tylko zadbamy o odpowiednią segregację odpadów (cel). Wykonaliśmy więc kroki 1 i 2.

Podczas opracowania tematu (krok 3) korzystamy tylko z rzetelnych źródeł, aby nie wprowadzić odbiorców w błąd. Jeśli interesującej nas informacji nie możemy potwierdzić w sprawdzonych źródłach, lepiej z niej zrezygnować.

Zebrane informacje najwygodniej przedstawić w formie tekstu. Już na tym etapie powinien on być zwięzły i dobrze zredagowany.

Oto przykładowy tekst:

Warto wiedzieć

Dzięki selektywnej zbiórce makulatury powstają m.in. banknoty, bandaże, opakowania na jajka, fartuchy szpitalne i papier ksero.

Recykling papieru

Co roku ścina się tysiące drzew, aby wyprodukować z nich papier na gazety, książki i opakowania. Tymczasem znaczną część makulatury można by ponownie wykorzystać. Zużyty papier wyrzucamy do niebieskich pojemników na śmieci. Odpady papierowe muszą zostać rozdrobnione na małe kawałeczki. Skrawki papieru miesza się z wodą i gotuje, aż powstanie z nich płynna miazga (pulpą). Tę mokrą pulpę rozsmarowuje się bardzo cienko na metalowej siatce. Następnie pulpa jest ściskana i suszona – tak powstaje nowy arkusz papieru. Papier pochodzący z recyklingu jest ciemniejszy, ponieważ zawiera resztki farby drukarskiej. Makulaturę można przerobić na papier od 5 do 7 razy.

W kroku 4 przetwarzamy tekst na formę wizualną:

- ▶ dzielimy tekst na bloki znaczeniowe,
- ▶ zaznaczamy słowa oznaczające obiekty (rzeczowniki z określeniami) istotne dla przekazu (wyróżnimy je kolorem pomarańczowym),
- ▶ zaznaczamy powiązania między obiektami (na zielono).

Tytuł: Recykling papieru

Wprowadzenie: Co roku ścina się tysiące drzew, aby wyprodukować z nich papier na gazety, książki i opakowania. Tymczasem znaczną część makulatury można by ponownie wykorzystać.

1. Zużyty papier wyrzucamy do niebieskich pojemników na śmieci.
2. Odpady papierowe muszą zostać rozdrobnione na małe kawałeczki.
3. Skrawki papieru miesza się z wodą i gotuje, aż powstanie płynna miazga (pulpa).
4. Tę moką pulpę rozsmarowuje się bardzo cienko na metalowej siatce.
5. Następnie pulpa jest ściskana i suszona – tak powstaje nowy arkusz papieru.

Dodatkowe informacje: Papier pochodzący z recyklingu jest ciemniejszy, ponieważ zawiera resztki farby drukarskiej. Makulaturę można przerobić na papier od 5 do 7 razy.

Dzięki przetworzeniu tekstu łatwiej będzie sporządzić szkic (krok 5): pomarańczowe elementy można zwizualizować wprost, zielone zaś oznaczają fazy procesu – należy rozważyć, czy ich zobrazowanie pomoże odbiorcy ten proces zrozumieć.

Po przemyśleniu tematu oraz analizie przygotowanego tekstu możemy zdecydować się na następujące rozwiązania:

- ▶ Recykling papieru jest procesem, dlatego uporządkujemy informacje według czasu.
- ▶ Ponieważ mamy do czynienia z procesem powtarzalnym, zastosujemy okrąg, na którym osadzimy kolejne etapy.
- ▶ Ponumerujemy etapy procesu zgodnie z ruchem wskazówek zegara.
- ▶ Informację o kolejności etapów recyklingu wzmocnimy przez dodanie strzałek na okręgu. Będą one szczególnie ważne dla odbiorcy podczas przechodzenia z etapu ostatniego (gotowy arkusz papieru) do pierwszego (zużyty papier).
- ▶ Infografika nie wymaga użycia wielu kolorów. Jako główny wybierzemy niebieski, ponieważ jest to kolor pojemnika na makulaturę.
- ▶ Użyty kolor będzie miał funkcję akcentującą – zwróci uwagę odbiorcy na kolejne etapy procesu oraz utrwali skojarzenie z kolorem pojemnika, do którego wrzuca się odpady papierowe.

Dobra rada

Szkic możesz wykonać odręcznie na kartce. Zadbaj o ogólną kompozycję, nie skupiaj się na zbyt wielu detalach. Samą infografikę możesz stworzyć w programie do grafiki wektorowej, np. Inkscape. W wypadku prostych infografik dobrze sprawdzą się również narzędzia graficzne edytorów prezentacji, np. Microsoft PowerPoint.

Sposoby porządkowania informacji,
s. 388 

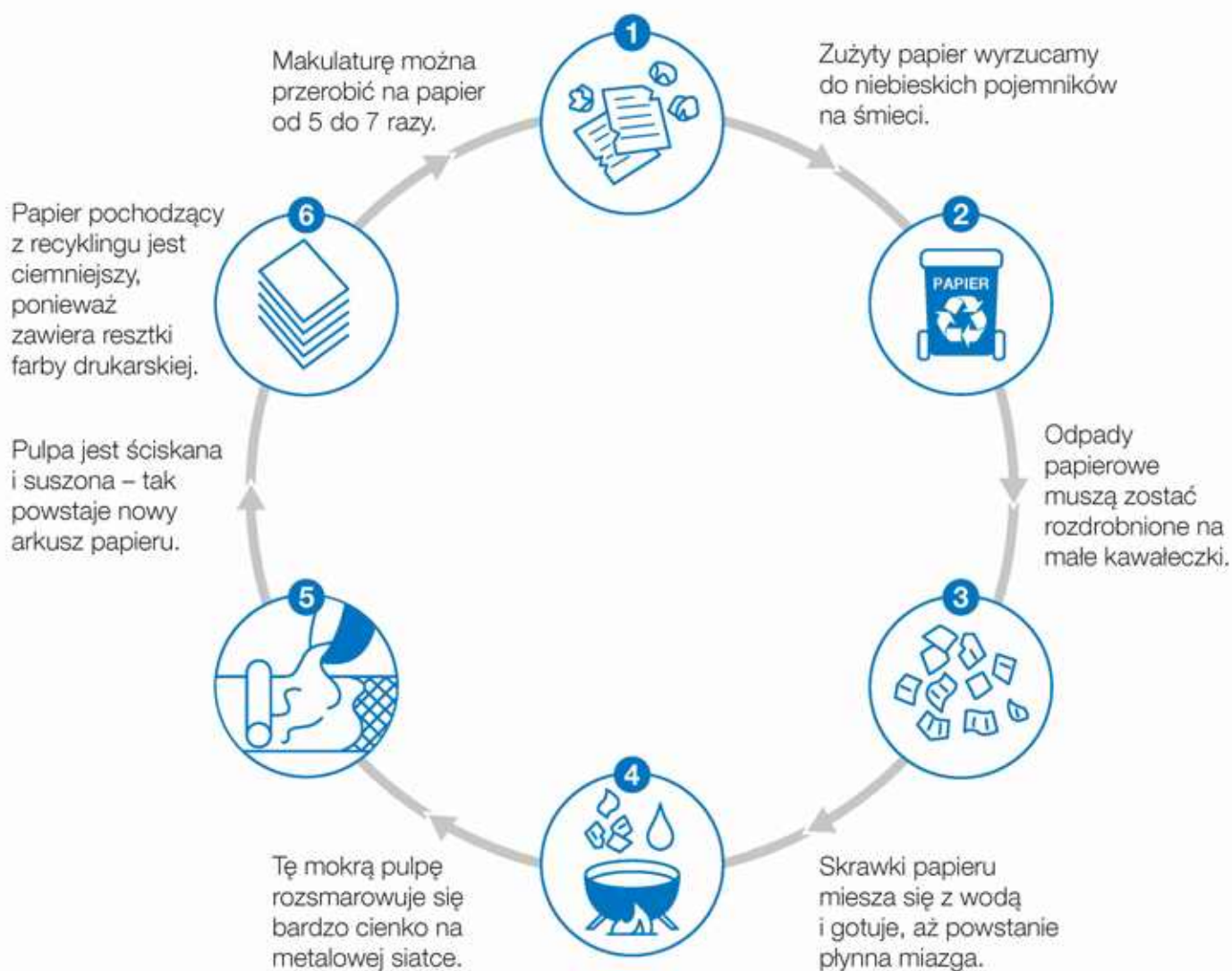
Akcentująca funkcja koloru,
s. 394 

Następnie należy przygotować infografikę, poddać ją krytycznej analizie i wprowadzić poprawki (kroki 6 i 7). Jeśli mamy taką możliwość, warto pokazać projekt osobie, która reprezentuje docelową grupę odbiorców. Można się w ten sposób dowiedzieć, czy infografika realizuje założony cel.

Oto przykładowy efekt końcowy.

Recykling papieru

Co roku ścina się tysiące drzew, aby wyprodukować z nich papier na gazety, książki i opakowania. Tymczasem znaczną część makulatury można by ponownie wykorzystać. Co się dzieje z makulaturą, gdy poprawnie segregujemy śmieci?



Ćwiczenie 11

Stwórz szkic infografiki, która omówi:

- zmiany oświetlenia Ziemi wynikające z ruchu obrotowego,
- obieg wody w przyrodzie,
- pojęcie łańcucha pokarmowego na dwóch różnych przykładach.

Podsumowanie

- Informacje można przedstawiać na różne sposoby. Jednym z nich jest połączenie tekstu z obrazem w formie grafiki informacyjnej (infografiki).
- Informacje na infografice powinny być uporządkowane. Wyróżniamy uporządkowanie: alfabetyczne, chronologiczne, według kategorii, przestrzenne i hierarchiczne.
- Przygotowanie atrakcyjnych wizualnie infografik wymaga znajomości zasad doboru barw oraz funkcji, jakie mogą pełnić kolory.
- W projektowaniu infografiki należy szczególną uwagę zwracać na to, czy będzie ona zrozumiała dla potencjalnego odbiorcy. Skuteczne przekazanie informacji jest ważniejsze niż atrakcyjność wizualna.

Zadania

- ★ **1** Poszukaj reprodukcji infografiki z 1869 r. autorstwa francuskiego inżyniera Charles'a Minarda. Ukazuje ona marsz armii Napoleona na Moskwę z 1812 r. oraz jej dramatyczny odwrót – wyścig z czasem podczas mroźnej zimy. Postaraj się odczytać z infografiki jak najwięcej informacji.
- ★ **2** Przygotuj prostą grafikę porównawczą, na której wskażesz podobieństwa i różnice:
 - a. dwóch wybranych dyscyplin sportowych (np. tenisa ziemnego i stołowego),
 - b. dwóch form nauki – zdalnej i stacjonarnej,
 - c. dwóch wybranych stylów w modzie (np. sportowego i boho).
- ★ **3** Omów techniki zastosowane na wybranej infografice. Możesz skorzystać z serwisu <http://visualoop.com>, w którym publikowane są prace autorów z całego świata.
- ★ **4** Przygotuj opis procesu smażenia naleśników w postaci infografiki.
- ★ **5** Odszukaj piktogramy wykorzystywane w twojej szkole. Określ ich funkcję.
- ★★ **6** Wyszukaj niezbędne informacje, a następnie opracuj infografikę, na której przedstawisz swoją ulubioną aktorkę oraz najważniejsze filmy, w których występowała. Użyj co najmniej dwóch sposobów porządkowania informacji.
- ★★ **7** Wykonaj infografikę przedstawiającą roczne wydatki gminy, w której mieszkasz. Uwzględnij m.in. nakłady na drogi, gospodarkę komunalną, przedszkola i szkoły, kulturę i biblioteki, gospodarkę gruntami, opiekę społeczną, kulturę fizyczną.
- ★★★ **8** Przygotuj infografikę przedstawiającą proces publikacji wywiadu w czasopiśmie (np. w tygodniku) – od przygotowań przed rozmową do ukazania się numeru.
- ★★★ **9** Opracuj infografikę, która porówna fabułę przynajmniej 6 klasycznych baśni dla dzieci. Pokaż podobieństwa i różnice.

Rozwiązywanie różnych problemów z wykorzystaniem komputera

Wyobraź sobie, że...

...po studiach medycznych zamierzasz zostać chirurgiem. Trafiasz na rezydenturę do szpitala, w którym operuje się również za pomocą **specjalistycznych robotów** chirurgicznych. Jesteś pod wrażeniem tego, jak szybko pacjenci wracają do zdrowia po takich operacjach, oraz jak dużą zręczność i precyzję robot zapewnia lekarzowi. Masz okazję uczestniczyć w szkoleniu, na którym uczysz się obsługi robota na **symulatorze**. Z czasem zaczynasz operować pacjentów.



...pracujesz w dziale promocji miasta. Kierujesz zespołem przygotowującym kampanię, która ma przyciągnąć turystów oraz zachęcić mieszkańców do odkrywania miasta na nowo. Z pomocą grafików opracowujecie **identyfikację wizualną**, m.in. rozpoznawalne **logo**, które od tej pory będzie widnieć na wszystkich materiałach informacyjnych. Kręcicie także spot promocyjny do mediów społecznościowych i wymyślcie oryginalne hasło, które rozpowszechniacie w formie **hashtagu**.

...jesteś grafikiem i pracujesz jako freelancer. Od portalu informacyjnego dostajesz zlecenie na **infografikę** o zmianach demograficznych w Polsce. Zapoznajesz się z otrzymanymi materiałami i wpadasz na pomysł **wizualizacji danych**. Najpierw **porządkujesz informacje**, a następnie przygotowujesz **szkic**. Odpowiednio **dobierasz kolory** i **akcentujesz** to, co najważniejsze. Twoja praca podoba się zarówno redaktorom, jak i czytelnikom portalu, dlatego otrzymujesz kolejne zlecenie.



...skończyłaś studia przyrodnicze i chcesz popularyzować wiedzę o kryzysie klimatycznym. Na blogu publikujesz rzetelne i przystępne **teksty informacyjne**. Z czasem otwierasz kanał wideo, na którym organizujesz m.in. **webinaria**, a także tworzysz tematyczny **podcast**. Stosujesz zasady **pracy z kamerą** i **mikrofonem** poznane w szkole, dlatego twoje materiały od początku są na wysokim poziomie. Za pomocą **narzędzi analitycznych** regularnie śledzisz ruch w sieci, aby jak najlepiej dopasowywać treści do swoich odbiorców.



Analiza postępu technologicznego w ostatnich latach

Technologia, z której korzystamy na co dzień, rozwija się właściwie nieustająco. Postęp ten najczęściej jest ewolucyjny, więc nie zawsze go zauważamy. Warto się jednak zastanowić, co sprawia, że niektóre technologie się przyjmują, a inne nie. Jeśli nauczymy się identyfikować takie czynniki, łatwiej nam będzie podejmować życiowe decyzje, np. o wyborze studiów czy założeniu firmy. Choć przewidywanie przyszłości w wypadku rozwoju technologii jest zadaniem niezmiernie trudnym, warto spróbować się z nim zmierzyć.



Cel projektu

Analiza czynników, które spowodowały, że dana technologia odniosła sukces lub poniosła porażkę w stosunku do wcześniejszych przewidywań. Wyniki należy przedstawić w wybranej formie, np. raportu, filmu, prezentacji z nagraniem narracją, podcastu.

To trzeba wiedzieć

Analiza trendów

O zmianach w popularności danej technologii można wiele powiedzieć na podstawie tego, jak często hasła z nią związane są wyszukiwane w internecie. Trendy te można prześledzić np. w Google Trends. Analiza archiwalnych artykułów w serwisach informacyjnych pomoże poznać przyczyny zmian i zrozumieć, co wpływało na zainteresowanie daną technologią – jakie wydarzenia, ludzkie potrzeby, sytuacja na świecie.

Badania ankietowe

Więcej informacji można zdobyć, przeprowadzając ankiety wśród rodziny i znajomych w różnym wieku, z różnym wykształceniem i mieszkających w różnych miejscach. Można do tego wykorzystać Formularze Google lub Microsoft Forms. Warto w nich uwzględnić pytania otwarte, aby umożliwić respondentom dołączenie własnych przemyśleń. Ankietę można też przeprowadzić w formie rozmowy – osobiście, telefonicznie lub za pomocą komunikatora internetowego.

Propozycje tematów

1. Jedna z dziedzin opracowanych w ramach projektu w klasie 1: wirtualna oraz rozszerzona rzeczywistość, automatyka i robotyka, druk 3D, sztuczna inteligencja, autonomiczne pojazdy, inteligentne miasta
2. Inna technologia wybrana w ramach projektu w klasie 1
3. Wideo na żądanie (VOD)
4. Filmy 360 stopni oraz filmy interaktywne

Etapy projektu

1. Wybierzcie w zespole technologię (oraz związany z nią produkt lub usługę), której rozwój będziecie analizować.
2. Poszukajcie materiałów opisujących początki wybranej technologii oraz analiz przewidujących jej rozwój i wpływ na ludzi – wybrane grupy społeczne lub całe społeczeństwo (w kraju, na świecie). Dane te powinny być starsze niż 3 lata.
3. Znajdźcie bieżące dane dotyczące wybranej przez was technologii. Mogą to być raporty i artykuły prasowe zawierające informacje o prawdopodobnych przyczynach jej sukcesu i szybkiego rozwoju oraz jej wpływie na różne aspekty życia ludzi. Odnieście się również do własnych doświadczeń i przeprowadźcie badania wśród rodziny i znajomych (ankiety, rozmowy, wywiady).
4. Porównajcie przewidywania dotyczące rozwoju wybranej usługi i jej wpływu na życie ludzi z tym, jak rzeczywiście jest dzisiaj. Przeprowadźcie burzę mózgów, aby zidentyfikować przyczyny takiego rozwoju wydarzeń.
5. Zdecydujcie, w jakiej formie przedstawicie swoje wnioski.
6. Podzielcie się zadaniami, wybierzcie narzędzia odpowiednie do pracy zespołowej oraz wybranej przez was formy przekazu (tekst, film).
7. W opracowaniu uwzględnijcie, jaki wpływ (pozytywny, negatywny lub żaden) miała wybrana technologia na różne dziedziny życia, np. edukację, opiekę zdrowotną, przemysł czy rolnictwo. Podajcie konkretne przykłady, które pokazują, czy przewidywania z przeszłości były trafne czy nie.
8. Do opracowania i zaprezentowania wniosków wykorzystajcie techniki poznane w trakcie nauki dotyczące różnych form publikowania (np. tworzenia prezentacji, animacji czy stron WWW).
9. Gotową pracę opublikujcie w sieci w taki sposób, by dostęp do niej miały jedynie pozostałe osoby w klasie oraz nauczyciel.

Przykładowe narzędzia

Microsoft Teams, Google Suite – do pracy zespołowej, wymiany plików, komunikacji oraz udostępniania przygotowanych treści.

Microsoft Forms, Formularze Google – do przygotowania i przeprowadzenia ankiet.

Microsoft Excel, Arkusze Google – do zebrania i analizy wyników oraz przedstawienia ich na wykresach.

Microsoft PowerPoint, Prezentacje Google – do opracowania prezentacji multimedialnej przedstawiającej najważniejsze fakty o danej technologii.

Microsoft Stream – do opublikowania filmu dla zamkniętej grupy odbiorców.

Dodatki

1. Wybrane typy danych w języku C++

W języku C++ każda zmienna, w zależności od tego, czy jest liczbą całkowitą, liczbą rzeczywistą, znakiem, napisem czy wartością logiczną, musi mieć jawnie przypisany odpowiedni typ danych. Określa on sposób zapisu zmiennej w pamięci komputera oraz zbiór operacji, które można wykonać na tej zmiennej.

Typy liczbowe

Wśród typów liczbowych wyróżnia się typy całkowite oraz zmiennoprzecinkowe (zmiennopozycyjne), czyli takie, które pozwalają zapisać również liczby rzeczywiste. W programowaniu wartości zapisane jako 2 i 2.0 to liczby różnych typów. Pierwsza z nich jest typu całkowitego, a druga jest typu zmiennoprzecinkowego.

Nazwa typu	Liczba bajtów	Przeznaczenie i zakres
<code>short</code>	2	Liczby całkowite ze znakiem od -2^{15} do $2^{15} - 1$, czyli od -32 768 do 32 767
<code>long</code> lub <code>int</code>	4	Liczby całkowite ze znakiem od -2^{31} do $2^{31} - 1$, czyli od -2 147 483 648 do 2 147 483 647
<code>long long</code>	8	Liczby całkowite ze znakiem od -2^{63} do $2^{63} - 1$, czyli od -9 223 372 036 854 775 808 do 9 223 372 036 854 775 807
<code>unsigned short</code>	2	Liczby całkowite bez znaku od 0 do $2^{16} - 1$, czyli od 0 do 65 535
<code>unsigned long</code> lub <code>unsigned int</code>	4	Liczby całkowite bez znaku od 0 do $2^{32} - 1$, czyli od 0 do 4 294 967 295
<code>unsigned long long</code>	8	Liczby całkowite bez znaku od 0 do $2^{64} - 1$, czyli od 0 do 18 446 744 073 709 551 615
<code>float</code>	4	Liczby rzeczywiste z zakresu od $1.17549\text{e-}38$ do $3.40282\text{e}+38$, w których liczba cyfr znaczących (dokładność) jest nie większa niż 7
<code>double</code>	8	Liczby rzeczywiste z zakresu od $2.22507\text{e-}308$ do $1.79769\text{e}+308$, w których liczba cyfr znaczących (dokładność) jest nie większa niż 15

Uwaga: Zapisy typu $\text{e-}38$ oraz $\text{e}+38$ oznaczają notację wykładniczą (tj. odpowiednio czynniki 10^{-38} oraz 10^{38}).

Tabela zawiera informacje o wybranych typach danych wykorzystywanych w języku C++ do zapisu liczb całkowitych i rzeczywistych dostępnych w kompilatorach 32-bitowych. Należy zwrócić uwagę, że w zależności od kompilatora C++ oraz wersji systemu (32-bitowy lub 64-bitowy) niektóre zakresy (liczba bajtów przeznaczonych do reprezentacji poszczególnych typów) mogą się różnić.

Liczbę bajtów przeznaczonych dla zmiennych danego typu można sprawdzić, korzystając z funkcji **sizeof**. Na przykład wynikiem **sizeof(int)** będzie najczęściej 4.

Typ znakowy i typ napisowy

Pojedyncze znaki można zapisywać w zmiennych typu **char**. Na przykład deklaracja **char litera='a'**; to deklaracja zmiennej **litera**, której następnie przypisano literę **a**. Napisy, czyli tzw. łańcuchy znaków, zapisuje się z użyciem zmiennych typu **string**. Typ ten w sensie ścisłym nie jest typem prostym C++, ale ma charakter obiektowy, tzn. po deklaracji zmiennej uzyskujemy dostęp do wbudowanych funkcji (tzw. metod), które pozwalają łatwo wykonywać operacje na napisach.

Metody najczęściej używane dla klasy **string**

Nazwa metody	Parametry	Wynik	Przykład zastosowania metody dla zmiennej s typu string s="Ala ma kota"
find (szukane)	szukane – fraza lub znak	Pozycja pierwszego wystąpienia szukanej frazy w napisie. Jeśli dany fragment nie występuje, wynikiem funkcji jest liczba spoza zakresu 0..size()-1	Instrukcja: cout<<s.find("ma"); Wynik: 4
substr (pozycja, ile)	pozycja – indeks znaku w napisie, ile – liczba znaków	Fragment ciągu znaków zapisanych od indeksu pozycja . Liczbę znaków określa parametr ile	Instrukcja: cout<<s.substr(4,2); Wynik: ma
erase (pozycja, ile)	pozycja – indeks znaku w napisie, ile – liczba znaków	Napis bez ciągu znaków o długości ile liczonego od indeksu pozycja	Instrukcje: s.erase(4,3); cout<<s; Wynik: Ala kota
size ()	–	Długość ciągu znaków	Instrukcja: cout<<s.size(); Wynik: 11

Oprócz metod wymienionych w tabeli często korzysta się także ze stałej **npos**. Metoda **find** przyjmuje wartość równą tej stałej, gdy szukana fraza nie zostanie odnaleziona w tekście. Na przykład dla napisu **s="Ala ma kota"**, **s.find("mak")==s.npos**. Stałą **npos** można także wykorzystać do sprawdzenia, czy znak jest elementem określonego zestawu znaków, np. czy litera jest samogłoską. W poniższym przykładzie znak wczytujemy tak długo, aż podamy samogłoskę.


```

1. const string sam = "aeiouy";
2. char znak;
3. do
4.     cin>>znak;
5. while (sam.find(znak)!=sam.npos);

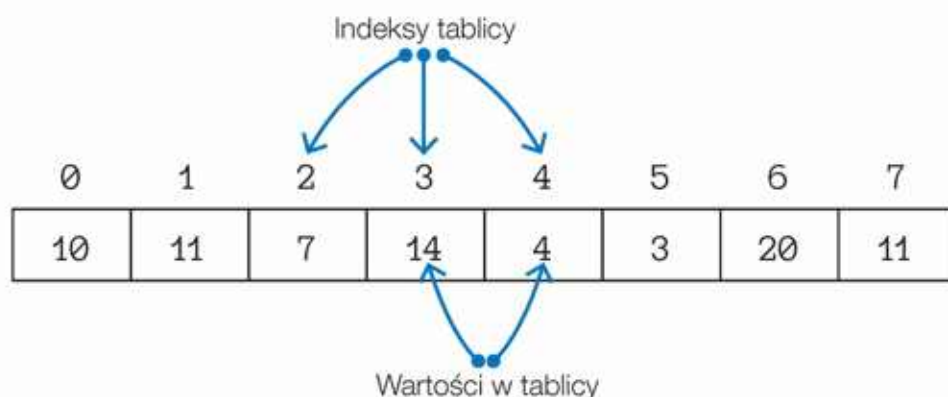
```

Typ logiczny

Do deklarowania zmiennych logicznych używa się typu **bool**, który przyjmuje wartości **true** i **false**. Są to słowa kluczowe języka C++. Wartość **false** reprezentowana jest w pamięci komputera przez liczbę 0, a wartość **true** – przez liczbę 1.

Tablice

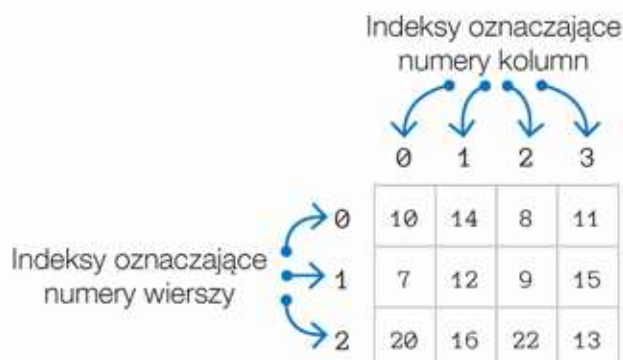
Tablica to struktura danych grupująca wiele elementów tego samego typu. Na przykład deklaracja **int tablica[4];** zastępuje deklarację czterech zmiennych typu **int**.



Tablice upraszczają proces zapisywania algorytmów w języku programowania, gdyż można odwołać się do wielu wartości za pomocą jednej nazwy. Dostęp do poszczególnych elementów zapisanych w tablicy uzyskuje się za pomocą mechanizmu indeksowania, tj. podania pozycji elementu w tablicy, licząc od 0, np. **tablica[0]**, **tablica[1]**, **tablica[2]** i **tablica[3]**.

Zapis	Wyjaśnienie
int tablica[4];	Deklaracja tablicy czterech liczb typu int
const int n = 4; int tablica[n];	Deklaracja tablicy, której liczbę elementów określa stała n
int tablica[3] = {1,2,3};	Deklaracja tablicy trzech liczb typu int połączona z przypisaniem im wartości
int tablica[] = {1,2,3};	Deklaracja tablicy liczb typu int połączona z przypisaniem im wartości – rozmiar tablicy określony jest na podstawie liczby przypisanych wartości
int tablica[4] = {0};	Deklaracja tablicy czterech liczb typu int połączona z przypisaniem im wszystkim wartości 0
int tablica[4] = {1,2};	Deklaracja tablicy czterech liczb typu int połączona z przypisaniem im wartości: 1, 2, 0 i 0

Tablica może mieć więcej niż jeden wymiar. W praktyce zastosowanie mają tablice dwuwymiarowe.



Do poszczególnych elementów tablicy dwuwymiarowej odwołujemy się, podając dwa indeksy – numer wiersza i numer kolumny. Na przykład `tablica[0][3]` oznacza liczbę 11.

Przykłady deklaracji tablic dwuwymiarowych:

Zapis	Wyjaśnienie
<code>int tablica[3][4];</code>	Deklaracja tablicy o 3 wierszach i 4 kolumnach. Elementy tablicy są typu <code>int</code> .
<code>const int N = 3, M = 4;</code> <code>int tablica[N][M];</code>	Deklaracja tablicy, której liczbę wierszy i kolumn określają stałe <code>N</code> i <code>M</code> . Elementy tablicy są typu <code>int</code> .
<code>int tablica[2][3] = {{10, 14, 8},</code> <code> {7, 12, 19}};</code>	Deklaracja tablicy o 2 wierszach i 3 kolumnach, której elementami są liczby całkowite, z przypisaniem wartości poszczególnym elementom.
<code>int tablica[][2] = {{10, 14},</code> <code> {17, 12},</code> <code> {20, 16}};</code>	Deklaracja tablicy dwuwymiarowej z przypisaniem wartości elementom. Liczba wierszy jest określona na podstawie przypisywanych wartości – w ten sposób można określić tylko pierwszy wymiar tablicy.

Struktury

Struktura jest złożonym typem danych, który pozwala zgrupować wiele zmiennych różnych typów pod jedną nazwą, przy czym każda ze zmiennych ma własną nazwę. Oto przykład definicji struktury o nazwie `uczen`, grupującej wybrane dane o uczniu.

```

1. struct uczen
2. {
3.     string imie;
4.     string nazwisko;
5.     int numer;    // numer w dzienniku
6.     float srednia; // średnia ocen
7.     bool dysleksja;
8. };

```


Po zdefiniowaniu typu danych można deklarować zmienne tego typu. Poniżej przykład deklaracji zmiennej `uczen1` typu `uczen`.

```
1. uczen uczen1;
```

Aby odwołać się do wartości przechowywanych w strukturze, korzystamy z notacji z kropką. Oto przykład przypisania wartości poszczególnym składowym zmiennej `uczen1`.

```
1. uczen1.nazwisko="Kowalski";
2. uczen1.imie="Jan";
3. uczen1.numer=10;
4. uczen1.srednia=4.5;
5. uczen1.dysleksja=false;
```

Można także tworzyć tablice struktur. Wówczas wykorzystujemy pętle do przeglądania elementów tablicy. Korzystając z funkcji `sort` z biblioteki `algorithm`, można posortować dane umieszczone w tablicy według określonego kryterium. W tym celu należy zdefiniować własną funkcję typu logicznego porównującą dwa elementy.

```
1. uczen uczniowie[10];
2.
3. // definicja funkcji porównującej dwóch uczniów
4. bool porownaj (uczen uczen1, uczen uczen2)
5. {
6.     return (uczen1.srednia > uczen2.srednia);
7. }
8.
9. sort(uczniowie, uczniowie+10, porownaj);
10.
11. for (int i=0; i<10; i++)
12.     cout<<uczniowie[i].nazwisko<<" "<<uczniowie[i].srednia;
```

2. Instrukcje sterujące w języku C++

Instrukcja złożona

Blok kilku instrukcji możemy ująć w nawiasy klamrowe. Są one wówczas traktowane jak jedna instrukcja. Taką instrukcję nazywamy instrukcją złożoną.

```
{
    instrukcja_1;
    instrukcja_2;
    instrukcja_3;
    ...
    instrukcja_n;
}
```

Instrukcja warunkowa

Składnia instrukcji warunkowej jest następująca:

```
if (warunek)
    instrukcja na tak;
else
    instrukcja na nie;
```

Jeśli warunek przyjmuje wartość **prawda**, wykonywana jest instrukcja na tak, w przeciwnym przypadku instrukcja na nie.

```
1. cin>>n;
2. if (n%2==0)
3.     cout<<"Parzysta";
4. else
5.     cout<<"Nieparzysta";
```

Druga część instrukcji warunkowej (**else**) jest opcjonalna. Wówczas instrukcja warunkowa przyjmuje postać:

```
if (warunek) instrukcja;
```

Po słowach kluczowych **if** lub **else** może wystąpić instrukcja złożona, czyli blok instrukcji ujętych w nawiasy klamrowe.

Instrukcja powtarzania – pętla while

Składnia pętli **while** jest następująca:

```
while (warunek) instrukcja;
```

Instrukcja w pętli jest powtarzana, dopóki warunek przyjmuje wartość logiczną **prawda**. Warunek jest sprawdzany przed wykonaniem instrukcji. Oznacza to, że pętla nie wykona się ani razu, gdy na początku warunek ma wartość logiczną **fałsz**.

Instrukcja wykonywana w pętli może być instrukcją złożoną, czyli blokiem instrukcji ujętych w nawiasy klamrowe.


```

1. cin>>n;
2. while (n>0)
3. {
4.     cout<<n%10<<endl;
5.     n=n/10;
6. }

```

Instrukcja powtarzania – pętla for

Składnia pętli **for** jest następująca:

for (instrukcja inicjująca; warunek powtarzania; instrukcja modyfikująca)
instrukcja;

Instrukcja inicjująca to najczęściej instrukcja przypisująca wartość początkową zmiennej, która pełni funkcję licznika. Warunek powtarzania to warunek logiczny, który powinien spełniać licznik, aby instrukcja zawarta w pętli została wykonana. Instrukcja modyfikująca to najczęściej instrukcja zmieniająca wartość licznika. Instrukcja wykonywana w pętli może być instrukcją złożoną, a więc blokiem instrukcji ujętych w nawiasy klamrowe.

```

1. cin>>n;
2. suma=0;
3. for (d=1;d<=n/2;d++)
4.     if (n%d==0) suma+=d;
5. if (suma==n) cout<<"Liczba doskonała";

```

Instrukcję **for** można zapisać za pomocą pętli **while**:

```

instrukcja inicjująca;
while (warunek powtarzania)
{
    instrukcja;
    instrukcja modyfikująca;
}

```

Instrukcja powtarzania – pętla do while

Składnia pętli **do while** jest następująca:

```

do
    instrukcja
while (warunek);

```

Najpierw wykonywana jest instrukcja, a potem sprawdzany jest warunek. Ponieważ sprawdzenie warunku następuje po wykonaniu instrukcji, pętla wykona się co najmniej raz. Kolejne powtórzenia pętli są wykonywane, dopóki warunek przyjmuje wartość logiczną **prawda**. Instrukcja wykonywana w pętli może być instrukcją złożoną, czyli blokiem instrukcji ujętych w nawiasy klamrowe.

```

1. do
2.     cin>>n;
3. while (n<=0);

```

Instrukcja wyboru switch

Składnia instrukcji **switch** jest następująca:

```
switch (zmienna)
{
    case wartość_1:
        // instrukcje, gdy zmienna==wartość_1
        break;
    case wartość_2:
        // instrukcje, gdy zmienna==wartość_2
        break;
    ...
    case wartość_n:
        // instrukcje, gdy zmienna==wartość_n
        break;
    default:
        // instrukcje, gdy zmienna nie przyjmuje
        // żadnej z wymienionych wcześniej wartości
}
```

Wartości, jakie może przyjąć zmienna, muszą być typu całkowitoliczbowego. W zależności od wartości zmiennej wykonywana jest jedna z części kodu zaczynających się od słowa kluczowego **case**. Na przykład gdy zmienna przyjmie wartość `wartosc_2`, program wykona instrukcje od części rozpoczynającej się od **case wartosc_2**. Część **default** jest opcjonalna (może nie wystąpić) – podobnie jak część **else** w instrukcji warunkowej **if**. Dzięki instrukcji **break** można przerwać wykonywanie kolejnych instrukcji zapisanych w **switch**. Gdyby zabrakło **break**, po wykonaniu instrukcji dla danej wartości zmiennej program przeszedłby do kolejnych linii w instrukcji **switch**. Jeśli nie wystąpi część **default**, nie ma potrzeby używania **break** w części z ostatnim **case**.

```
1. switch (dzialanie)
2. {
3.     case 1:
4.         c = a + b
5.         break;
6.     case 2:
7.         c = a - b
8.         break;
9.     case 3:
10.        c = a * b
11.        break;
12.    case 4:
13.        c = a/b
14. }
```


Funkcja

Funkcja to podprogram, który odpowiada za realizację określonej części programu. Funkcje definiuje się przed definicją funkcji `main`. Definicja funkcji składa się z nagłówka funkcji i bloku instrukcji. Nagłówek zawiera typ wyniku funkcji, jej nazwę oraz listę parametrów określającą typ i nazwę każdego z parametrów.

```
typ nazwa_funkcji(typ parametr_1, typ parametr_2, ...)  
{  
    instrukcja_1;  
    instrukcja_2;  
    ...  
    instrukcja_n;  
}
```

Jeśli funkcja ma zwrócić wynik, w jej treści musi zostać użyta instrukcja **return**. Jej parametrem jest wartość zwracana przez funkcję i musi być typu określonego w nagłówku funkcji. Na przykład instrukcja **return a**; oznacza, że wynikiem funkcji będzie wartość zmiennej `a`. Instrukcja **return** kończy działanie funkcji, może być wielokrotnie użyta w bloku instrukcji funkcji.

```
1. int SumaCyfr(int n)  
2. {  
3.     int suma=0;  
4.     while (n>0)  
5.     {  
6.         suma=suma+n%10; n=n/10;  
7.     }  
8.     return suma;  
9. }
```

Jeśli typem funkcji w nagłówku jest **void**, to funkcja wykonuje tylko określone zadania i nie zwraca żadnego wyniku.

```
1. void Cyfry(int n)  
2. {  
3.     while (n>0)  
4.     {  
5.         cout<<n%10;  
6.         n=n/10;  
7.     }  
8. }
```

Dzięki funkcjom program jest bardziej przejrzysty i łatwiej go zmodyfikować. Pozwalają one unikać powtórzeń, ponieważ tę samą funkcję można wywołać wielokrotnie dla różnych wartości parametrów.

3. Obsługa plików tekstowych w języku C++

Korzystanie z plików tekstowych wymaga dopisania w kodzie źródłowym programu dyrektywy `#include <fstream>`. Komunikacja z plikami z poziomu programu w języku C++ odbywa się za pośrednictwem zmiennych plikowych. Najczęściej korzystamy ze zmiennych następujących typów:

ifstream – zmienne tego typu służą do obsługi plików wejściowych (do odczytu danych),

ofstream – zmienne tego typu służą do obsługi plików wyjściowych (do zapisu danych),

fstream – zmienne tego typu służą do obsługi plików przeznaczonych zarówno do odczytu, jak i zapisu danych.

Praca z plikami tekstowymi ma kilka etapów.

1. Otwarcie pliku.
2. Wczytywanie danych z pliku oraz zapisywanie danych w pliku.
3. Zamknięcie pliku.

Otwarcie pliku

Plik można otworzyć od razu podczas deklaracji zmiennej plikowej lub później, korzystając z metody `open`. Aby otworzyć pliki skojarzone ze zmiennymi typu **fstream**, podajemy parametr określający sposób otwarcia pliku.

Działanie	Zapis	Wyjaśnienie
Otwarcie pliku do odczytu	<pre>ifstream wejscie("dane.txt");</pre>	Otwarcie do odczytu pliku <i>dane.txt</i> skojarzonego ze zmienną plikową <i>wejscie</i> typu ifstream podczas deklaracji zmiennej.
	<pre>fstream plik; plik.open("dane.txt", ios::in);</pre>	Deklaracja zmiennej plikowej <i>plik</i> typu fstream , a następnie skojarzenie z tą zmienną pliku <i>dane.txt</i> i otwarcie go. Parametr <code>ios::in</code> (ang. <i>input</i> – wejście) sygnalizuje, że plik ma być otwarty do odczytu.
Otwarcie pliku do dopisywania	<pre>fstream plik("dane.txt", ios::app);</pre>	Otwarcie pliku do dopisywania informacji na końcu. Parametr <code>ios::app</code> (ang. <i>append</i> – dodać) sygnalizuje, że plik ma być otwarty do dopisywania informacji.
Otwarcie pliku do odczytu lub zapisu	<pre>fstream plik("dane.txt", ios::in ios::out);</pre>	Otwarcie pliku <i>dane.txt</i> skojarzonego ze zmienną plikową <i>plik</i> typu fstream . Atrybuty otwarcia pliku można łączyć. Z pliku otwartego w ten sposób można zarówno odczytywać dane, jak i je do niego zapisywać.

Otwarcie pliku do zapisu	<code>ofstream wyjście("wyniki.txt");</code>	Otwarcie do zapisu pliku <i>wyniki.txt</i> skojarzonego ze zmienną plikową <i>wyjście</i> typu <code>ofstream</code> podczas deklaracji zmiennej.
	<code>fstream plik; plik.open("dane.txt", ios::out);</code>	Deklaracja zmiennej plikowej <i>plik</i> typu <code>fstream</code> , a następnie skojarzenie z tą zmienną i otwarcie pliku do zapisu <i>dane.txt</i> . Parametr <code>ios::out</code> (ang. <i>output</i> – wyjście) sygnalizuje, że plik ma być otwarty do zapisu.

Plik, z którego możemy odczytywać informacje lub je do niego dopisywać, musi istnieć. Jeśli podana jest tylko nazwa pliku (bez ścieżki dostępu), musi on być zapisany w tym samym folderze, w którym znajduje się plik wykonywalny programu. To, czy plik został prawidłowo otwarty, można sprawdzić, korzystając z metody `is_open`. Jej wartością jest **prawda**, jeśli otwarcie przebiegło pomyślnie, a jeśli nie – **fałsz**.

```

1. ifstream wejście;
2. wejście.open("dane.txt");
3. if (wejście.is_open())
4. {
5.     // wczytywanie i przetwarzanie danych z pliku
6. }
7. else cout << "Błąd otwarcia pliku";

```

Do sprawdzenia poprawności otwarcia pliku możemy też wykorzystać metodę `good`.

Pliki reprezentowane przez zmienną typu `ofstream` lub otwarte tylko z parametrem `ios::out` są tworzone jako nowe puste pliki w tym samym folderze, w którym zapisany jest plik wykonywalny programu (o ile nie została podana ścieżka dostępu). Jeśli plik o danej nazwie już istnieje, zostanie on skasowany i zastąpiony nowym.

Wczytywanie danych z pliku oraz zapisywanie danych w pliku

Wczytywanie i zapisywanie danych odbywa się podobnie jak w przypadku korzystania ze standardowego wejścia (klawiatury) i wyjścia (ekran). Jednak zamiast polecenia `cin` lub `cout` podajemy nazwę zmiennej plikowej reprezentującej dany plik.

Działanie	Zapis	Wyjaśnienie
Wczytywanie danych z pliku	<code>wejście>>s;</code>	Wczytanie do zmiennej <i>s</i> danych z pliku reprezentowanego przez zmienną <i>wejście</i> .
	<code>wejście>>s>>t;</code>	Wczytanie danych z pliku do dwóch zmiennych, z których pierwsza może być np. typu <code>string</code> , a druga np. typu <code>int</code> .
Wczytanie wiersza danych z pliku	<code>getline(wejście,s);</code>	Wczytanie wiersza danych z pliku reprezentowanego przez zmienną <i>wejście</i> i zapisanie go w zmiennej <i>s</i> .
Zapisanie danych w pliku	<code>wyjście<<s;</code>	Zapisanie wartości zmiennej <i>s</i> w pliku reprezentowanym przez zmienną <i>wyjście</i> .

Wczytanie danej typu **string** z pliku z użyciem operatora **>>** powoduje (podobnie jak przy wczytywaniu z klawiatury) pobranie napisu do napotkania spacji lub końca wiersza. Jeśli chcemy pobrać cały wiersz danych wraz ze spacjami, należy użyć funkcji `getline`. Na przykład dla wiersza „Ala ma kota” po wykonaniu instrukcji `wejście>>s`; zmienna `s` będzie miała wartość „Ala”. Natomiast po wykonaniu instrukcji `getline(wejście,s)`; zmienna `s` będzie miała wartość „Ala ma kota”.

Metodą przydatną podczas pracy z plikami, z których wczytujemy dane, jest metoda `eof` (ang. *end of file* – koniec pliku). Przyjmuje ona wartość **prawda**, kiedy zostanie napotkany koniec pliku (nie można już wczytywać dalej danych), a w przeciwnym wypadku – **fałsz**.

```
1. ifstream wejście("dane.txt");
2. while (!wejście.eof())
3. {
4.     // instrukcje wczytywania i przetwarzania danych z pliku
5. }
```

Wygodną formą pobierania danych z pliku jest użycie instrukcji wczytywania w warunku pętli **while**.

```
while (wejście>>dane)
{
    // instrukcje przetwarzania danych
}
```

Jeśli wykonanie instrukcji `wejście>>dane` zakończy się niepowodzeniem, wówczas pętla zwraca wartość `0` oznaczającą **fałsz**, w przeciwnym przypadku – wartość różną od zera interpretowaną jako **prawda**.

Zamknięcie pliku

Zamknięcie pliku realizuje metoda `close`. Pliki zawsze powinny zostać zamknięte po zakończeniu wszystkich operacji zapisu i odczytu. Na przykład:

```
1. wyjście.close();
```

Jeśli nie zamkniemy pliku do zapisu, część danych może zostać utracona.

4. Wybrane funkcje matematyczne z biblioteki `cmath`

Działanie	Wyrażenie	Wynik	Uwagi
Potęgowanie	<code>pow(2.0; -2)</code>	0.25	Drugi argument funkcji to wykładnik potęgi. Może być liczbą niecałkowitą.
Pierwiastek kwadratowy	<code>sqrt(2.0)</code>	1.41421	Zwracana jest wartość przybliżenia.
Zaokrąglenie w górę (sufit)	<code>ceil(2.5)</code> <code>ceil(-2.5)</code>	3 -2	–
Zaokrąglenie w dół (podłoga)	<code>floor(2.5)</code> <code>floor(-2.5)</code>	2 -3	–
Zaokrąglenie (tradycyjne)	<code>round(2.5)</code> <code>round(-2.5)</code>	3 -3	–
Obcięcie części ułamkowej	<code>trunc(2.5)</code> <code>trunc(-2.5)</code>	2 -2	–
Wartość bezwzględna	<code>abs(2)</code> <code>abs(-2)</code>	2 2	Można też stosować funkcję <code>fabs</code> z biblioteki <code>math.h</code>
Funkcja sinus	<code>sin(M_PI/2)</code>	1	Parametrem jest kąt wyrażony w radianach. <code>M_PI/2</code> zwraca wartość $\pi/2$.
Funkcja cosinus	<code>cos(M_PI/2)</code>	0.5	Parametrem jest kąt wyrażony w radianach. <code>M_PI/2</code> zwraca wartość $\pi/2$.
Funkcja tangens	<code>tan(M_PI/2)</code>	1	Parametrem jest kąt wyrażony w radianach. <code>M_PI/2</code> zwraca wartość $\pi/2$.

5. Wybrane szablony z biblioteki STL

Biblioteka STL (ang. *Standard Template Library* – standardowa biblioteka szablonów) jest zbiorem bibliotek języka C++ zawierających wzorce (szablony) typów struktur danych oraz algorytmów, które działają niezależnie od typu przechowywanych informacji. Poniżej opisane są tylko wybrane struktury danych oraz algorytmy wykorzystywane w podręczniku.

Typ `pair` (para)

Typ `pair` jest typem danych służącym do reprezentowania dwóch wartości (pary), które mogą być różnych typów. Deklaracja zmiennej typu `pair` jest następująca:

`pair<typ pierwszego elementu, typ drugiego elementu> nazwa_zmiennej;`

Przykładowa deklaracja zmiennej typu `pair` reprezentującej dwie liczby całkowite, np. współrzędne punktu na płaszczyźnie:

```
1. pair<int,int> p;
```

Typ `pair` jest typem obiekowym. Dostęp do poszczególnych wartości pary jest możliwy dzięki atrybutom obiektu `first` i `second`. Wartości elementom tej pary najłatwiej przypisać, korzystając z funkcji `make_pair`.

```
1. p=make_pair(3,5);
2. cout<<p.first<<" "<<p.second; // wypisanie elementów pary
```

Wartości elementom pary można nadać od razu podczas deklaracji, np.:

```
1. pair<int,int> p=make_pair(3,5);
```

lub

```
1. pair<int,int> p(3,5);
```

Element pary może być dowolnego typu, w tym również kolejną parą. Deklarując taką parę, warto pamiętać, że dwa znaki `>` nie powinny wtedy wystąpić obok siebie. Należy je oddzielić spacją.

```
1. pair<pair<int,int>,pair<int,int> > // para dwóch par
```

Typ `vector`

Typ `vector` jest w pewnym sensie odpowiednikiem tablicy dynamicznej (ang. *dynamic array*), czyli takiej, której rozmiar (liczbę elementów) można określać i zmieniać podczas działania programu. Żeby korzystać z typu `vector`, należy za pomocą dyrektywy `#include <vector>` dołączyć do programu bibliotekę `vector`.

Ogólna deklaracja zmiennej typu **vector** jest następująca:

```
vector<typ elementów wektora> nazwa_wektora;
```

Przykładowa deklaracja zmiennej A typu **vector** o elementach całkowitych:

```
vector<int> A;
```

Rozmiar zmiennej typu **vector** można ustalić za pomocą metody **resize**, której parametrem jest liczba elementów. W programie można jej użyć wielokrotnie i w ten sposób zwiększać lub zmniejszać liczbę elementów. Metoda **resize** może mieć drugi parametr, który określa wartość początkową elementów zmiennej typu **vector**. Przykład użycia metody **resize**:

```
A.resize(10);
```

Do elementu typu **vector** odwołujemy się tak samo jak do elementu w tablicy – podając w nawiasach kwadratowych indeks elementu.

Aktualny rozmiar zmiennej typu **vector** zwraca bezparametrowa metoda **size**.

Poniższy fragment kodu źródłowego programu tworzy zmienną A typu **vector** o elementach całkowitych, wczytuje z klawiatury rozmiar tej zmiennej, wypełnia zmienną A losowymi liczbami i wypisuje te liczby.

```
1. vector<int> A;  
2. int n;  
3. cin>>n;  
4. A.resize(n);  
5. for (int i=0;i<A.size();i++) A[i]=rand()%100;  
6. for (int i=0;i<A.size();i++) cout<<A[i]<<" ";
```

Elementem zmiennej typu **vector** może być kolejna zmienna typu **vector**. Otrzymamy w ten sposób odpowiednik tablicy dwuwymiarowej, w której każdy wiersz może mieć różną liczbę elementów. Na przykład poniższy fragment kodu źródłowego programu utworzy tablicę o wymiarach $n \times m$.

```
1. vector<vector<int> > A;  
2. int n, m;  
3. cin>>n; cin>>m;  
4. A.resize(n);  
5. for (int i=0;i<A.size();i++) A[i].resize(m);
```

Do elementów zmiennej typu **vector**, której elementy także są typu **vector**, odwołujemy się podobnie jak do elementów tablicy dwuwymiarowej – podajemy w nawiasach kwadratowych numery wiersza i kolumny, np. `A[i][j]`.

Należy pamiętać, że dwa znaki `>` zapisane obok siebie kompilator może uznać za operator wczytywania danych. Dlatego przy deklaracji zmiennych z wykorzystaniem szablonów typów z biblioteki STL najlepiej oddzielić te znaki spacją.

Metody najczęściej używane dla klasy `vector`

Nazwa metody	Opis działania
<code>resize(ile)</code>	Ustala rozmiar zmiennej na <code>ile</code> elementów.
<code>resize(ile, pocz)</code>	Ustala rozmiar zmiennej na <code>ile</code> elementów. Parametr <code>pocz</code> określa wartość początkową elementów zmiennej.
<code>size()</code>	Zwraca aktualny rozmiar zmiennej.
<code>push_back(wart)</code>	Dodaje na końcu zmiennej nowy element o wartości <code>wart</code> .
<code>clear()</code>	Usuwa wszystkie elementy zmiennej. Metoda zwraca taki sam wynik jak instrukcja <code>resize(0)</code> .
<code>empty()</code>	Zwraca informację, czy wektor jest pusty. Metoda zwraca taki sam wynik jak instrukcja <code>size()==0</code> .

Typ `stack` (stos)

Typ `stack` jest szablonem dynamicznej struktury danych nazywanej stosem. Stos to struktura danych, w której nowy element dopisuje się do wierzchołka, a usunąć można tylko element będący wierzchołkiem. Taką strategię dostępu nazywa się LIFO (od ang. *last in, first out* – ostatni wchodzi, pierwszy wychodzi). Na stosie można przechowywać wyłącznie dane tego samego typu.

Aby skorzystać z szablonu, na początku programu należy dołączyć bibliotekę `stack` – dyrektywa `#include <stack>`. Ogólna deklaracja stosu jest następująca:

```
stack<typ elementów stosu> nazwa_stosu;
```

Przykład deklaracji stosu liczb całkowitych:

```
stack<int> stos;
```

Można wyróżnić następujące operacje na stosie:

- ▶ *push* – umieszczenie elementu na stosie;
- ▶ *pop* – usunięcie elementu będącego wierzchołkiem stosu;
- ▶ *top* – odczytanie wartości elementu będącego wierzchołkiem stosu;
- ▶ *empty* – sprawdzenie, czy stos jest pusty.

Metody najczęściej używane dla klasy `stack`

Nazwa metody	Opis działania
<code>push(wart)</code>	Umieszcza na wierzchołku stosu nowy element o wartości <code>wart</code> .
<code>pop()</code>	Usuwa element z wierzchołka stosu. Jeśli stos jest pusty, spowoduje błąd programu.
<code>top()</code>	Zwraca wartość elementu będącego wierzchołkiem stosu. Jeśli stos jest pusty, spowoduje błąd programu.
<code>empty()</code>	Zwraca informację, czy stos jest pusty.

Poniższy kod źródłowy deklaruje stos liczb całkowitych, dodaje do niego kilka liczb, następnie je wypisuje i usuwa ze stosu. Liczby będą wypisane w odwrotnej kolejności, niż były dodawane do stosu.

```
1. stack<int> stos;
2. int n=rand()%10;
3. for (int i=0;i<n;i++) stos.push(rand()%100);
4. while (!stos.empty())
5. {
6.     cout<<stos.top();
7.     stos.pop();
8. }
```

Typ queue (kolejka)

Typ **queue** jest szablonem dynamicznej struktury danych nazywanej kolejką. Kolejka to struktura danych, w której nowy element można dodać tylko na końcu kolejki, a usunąć można tylko element znajdujący się na początku kolejki. Taką strategię dostępu nazywa się FIFO (od ang. *first in, first out* – pierwszy wchodzi, pierwszy wychodzi). W kolejce można przechowywać wyłącznie dane tego samego typu.

Aby móc skorzystać z szablonu, na początku programu należy dołączyć bibliotekę **queue** – dyrektywa `#include <queue>`. Ogólna deklaracja kolejki jest następująca:

queue<typ elementów kolejki> nazwa_kolejki;

Przykład deklaracji kolejki liczb całkowitych:

queue<int> kolejka;

Można wyróżnić następujące operacje na kolejce:

- ▶ *push* – umieszczenie elementu na końcu kolejki;
- ▶ *pop* – usunięcie elementu z początku kolejki;
- ▶ *front* – pobranie wartości elementu z początku kolejki;
- ▶ *empty* – sprawdzenie, czy kolejka jest pusta.

Metody najczęściej używane dla klasy **queue**

Nazwa metody	Opis działania
<code>push(wart)</code>	Dodaje na końcu kolejki nowy element o wartości <code>wart</code> .
<code>pop()</code>	Usuwa element z początku kolejki. Jeśli kolejka jest pusta, spowoduje błąd programu.
<code>front()</code>	Zwraca wartość elementu znajdującego się na początku kolejki. Jeśli kolejka jest pusta, spowoduje błąd programu.
<code>empty()</code>	Zwraca informację, czy kolejka jest pusta.

Poniższy kod źródłowy deklaruje kolejkę liczb całkowitych, dodaje do niej kilka liczb, następnie je wypisuje i usuwa z kolejki. Liczby będą wypisane w tej samej kolejności, w jakiej były dodane do kolejki.

```
1. queue<int> kolejka;
2. int n=rand()%10;
3. for (int i=0;i<n;i++) kolejka.push(rand()%100);
4. while (!kolejka.empty())
5. {
6.     cout<<kolejka.front();
7.     kolejka.pop();
8. }
```

Typ list (lista)

Typ **list** jest szablonem dynamicznej struktury danych – listy. Lista to struktura danych, w której dostęp do elementów jest sekwencyjny. Oznacza to, że od danego elementu listy można przejść bezpośrednio do elementu sąsiedniego. Aby dostać się do elementu znajdującego się na *i*-tym miejscu listy, trzeba przejść przez wszystkie elementy, które nas od niego oddzielają. Nowy element można wstawić w dowolne miejsce listy, podobnie można usunąć z listy dowolny element. W liście przechowuje się dane tego samego typu. Stos i kolejka są szczególnymi przypadkami listy. Wyróżniamy:

- ▶ listy jednokierunkowe – przejście do sąsiedniego elementu jest możliwe tylko w jedną stronę,
- ▶ dwukierunkowe – przejście między sąsiednimi elementami jest możliwe w dwie strony,
- ▶ cykliczne – w jednokierunkowej liście cyklicznej od ostatniego elementu można przejść do pierwszego, a jeśli lista jest dwukierunkowa – można też przejść od pierwszego elementu do ostatniego.

Aby skorzystać z szablonu, na początku programu należy dołączyć bibliotekę **list** – dyrektywa **#include <list>**. Biblioteka STL implementuje szablon listy dwukierunkowej. Ogólna deklaracja listy jest następująca:

```
list<typ elementów listy> nazwa_listy;
```

Przykład deklaracji listy złożonej z liczb całkowitych:

```
list<int> lista;
```

Dostęp do elementów listy jest możliwy dzięki iteratorom. Iterator jest zmienną pozwalającą na sekwencyjne przeglądanie zawartości struktury danych z biblioteki STL. Najczęściej zawiera wskaźnik (adres w pamięci) danego elementu struktury danych. Dopuszczalne są operacje powiększenia oraz pomniejszenia iteratora (inkrementacja i dekrementacja iteratora). W przypadku typu **list** powiększenie iteratora oznacza przejście do następnego elementu listy, a pomniejszenie – przejście do poprzedniego elementu. Ogólna postać deklaracji iteratora dla typu **list** jest następująca:

```
list<typ elementów listy>::iterator nazwa_iteratora;
```


Dla listy liczb całkowitych deklaracja iteratora o nazwie `it` jest następująca:

```
list<int>::iterator it;
```

Żeby odwołać się do wartości elementu wskazywanego przez iterator, należy przed jego nazwą umieścić znak `*` (np. `*it`).

Metody najczęściej używane dla klasy `list`

Nazwa metody	Opis działania
<code>push_back(wart)</code>	Dodaje na końcu listy nowy element o wartości <code>wart</code> .
<code>push_front(wart)</code>	Dodaje na początku listy nowy element o wartości <code>wart</code> .
<code>pop_back()</code>	Usuwa ostatni element listy. Jeśli lista jest pusta, spowoduje błąd programu.
<code>pop_front()</code>	Usuwa pierwszy element listy. Jeśli lista jest pusta, spowoduje błąd programu.
<code>insert(it, wart)</code>	Wstawia nowy element o wartości <code>wart</code> przed elementem wskazywanym przez iterator <code>it</code> .
<code>erase(it)</code>	Usuwa element wskazywany przez iterator <code>it</code> .
<code>remove(wart)</code>	Usuwa wszystkie elementy listy o wartości <code>wart</code> .
<code>begin()</code>	Zwraca iterator wskazujący na pierwszy element listy.
<code>end()</code>	Zwraca iterator wskazujący miejsce za ostatnim elementem listy.
<code>clear()</code>	Usuwa wszystkie elementy listy.
<code>empty()</code>	Zwraca informację, czy lista jest pusta.

Poniższy kod źródłowy deklaruje listę liczb całkowitych, dodaje do niej kilka liczb, następnie je przegląda od początku do końca i od końca do początku.

```
1. list<int> lista;
2. list<int>::iterator it;
3. int n=rand()%10;
4. for (int i=0;i<n;i++) lista.push_back(rand()%100);
5. for (it=lista.begin();it!=lista.end();it++) cout<<*it<<" ";
6. it=lista.end();
7. do
8. {
9.     it--; cout<<*it<<" ";
10. }
11. while (it!=lista.begin());
```

Funkcja `sort`

Funkcja `sort` umożliwia sortowanie elementów w złożoności obliczeniowej $O(n \cdot \log n)$ i jest jednym z najczęściej wykorzystywanych algorytmów z biblioteki STL. Za pomocą tej funkcji można sortować zarówno tablice statyczne, jak i wektory. Sortowane elementy mogą być dowolnego typu, którego elementy można porównać relacją mniejszości `<`.

Alternatywnie można zdefiniować własną funkcję porównującą dwa elementy danego typu. Można też sortować tylko fragment tablicy lub wektora. Parametrami funkcji są wskaźniki (adresy) na pierwszy sortowany element oraz za ostatni sortowany element. W przypadku tablicy statycznej wskaźnikiem jest nazwa tablicy wraz z ewentualnym przesunięciem – od którego i do którego elementu sortujemy tablicę. Poniższy przykład pokazuje, jak za pomocą funkcji `sort` posortować N-elementową tablicę liczb całkowitych.

```
const int N=10;
int tablica[N];
....
sort(tablica, tablica+N);
```

W przypadku zmiennej typu `vector` parametrami funkcji `sort` są wartości iteratorów wskazujących początek i koniec sortowania. Do posortowania zmiennej należy skorzystać z iteratorów `begin` i `end`. Poniżej pokazano, jak posortować elementy zmiennej typu `vector` złożonej z liczb całkowitych.

```
vector<int> wektor;
....
sort(wektor.begin(), wektor.end());
```

Funkcja `sort` może mieć także trzeci parametr: nazwę funkcji zwracającej wartość logiczną – wynik porównania dwóch elementów tablicy lub zmiennej typu `vector`.

Poniższy przykład ilustruje, jak posortować tablicę liczb całkowitych w odwrotnej kolejności:

```
bool Porownaj(int a, int b)
{
    return (a>b);
}
sort(tablica, tablica+N, Porownaj);
```

Można także zdefiniować funkcję porównującą dwa elementy, których nie można porównać relacją mniejszości `<`, np. liczby wymierne zdefiniowane jako struktura złożona z dwóch liczb całkowitych: licznika i mianownika ułamka. Zakładamy, że znak ułamka jest pamiętany w liczniku, czyli mianownik jest większy lub równy 1.

```
struct Wymierna
{
    int licz, mian;
}

bool Porownaj(Wymierna a, Wymierna b)
{
    int n=NWD(a.mian, b.mian);
    return (a.licz*(b.mian/n)<b.licz*(a.mian/n));
}

vector<Wymierna> ulamki;
....
sort(ulamki.begin(), ulamki.end());
```


6. Słowa kluczowe w języku C++

Słowa kluczowe nie mogą być nazwami zmiennych, funkcji ani innych obiektów w kodzie źródłowym.

Słowo kluczowe	Opis
<code>bool</code>	Nazwa typu logicznego
<code>break</code>	Przerwanie wykonywania instrukcji sterującej (warunkowej, pętli, wyboru)
<code>case</code>	Używane w instrukcji wyboru
<code>char</code>	Nazwa typu danych dla znaków oraz liczb całkowitych jednobajtowych
<code>class</code>	Używane do definicji typu obiektowego
<code>const</code>	Używane przy definiowaniu stałej
<code>default</code>	Używane w instrukcji wyboru
<code>double</code>	Nazwa typu danych, które stanowią liczby zmiennoprzecinkowe, o tzw. podwójnej precyzji
<code>do</code>	Używane w instrukcji iteracyjnej <code>do while</code>
<code>else</code>	Używane w instrukcji warunkowej do wyodrębnienia kodu wykonywanego, gdy warunek po <code>if</code> nie jest spełniony
<code>false</code>	Stała oznaczająca wartość logiczną falsz
<code>float</code>	Nazwa typu danych, które stanowią liczby zmiennoprzecinkowe
<code>for</code>	Używane w instrukcji iteracyjnej
<code>if</code>	Używane w instrukcji warunkowej
<code>int</code>	Nazwa typu danych dla liczb całkowitych
<code>long</code>	Nazwa typu danych dla liczb całkowitych
<code>namespace</code>	Używane przy określaniu tzw. przestrzeni nazw
<code>private</code>	Specyfikator dostępu
<code>protected</code>	Specyfikator dostępu
<code>public</code>	Specyfikator dostępu
<code>return</code>	Instrukcja, która kończy działanie funkcji i zwraca wartość do miejsca wywołania tej funkcji

<code>short</code>	Nazwa typu danych dla liczb całkowitych
<code>struct</code>	Używane przy definicji złożonego typu danych – struktury
<code>switch</code>	Używane w instrukcji wyboru
<code>typedef</code>	Używane do określenia własnej nazwy typu zmiennych
<code>true</code>	Stała oznaczająca wartość logiczną prawda
<code>unsigned</code>	Używane przy deklaracji danych będących liczbami całkowitymi bez znaku
<code>using</code>	Używane przy określaniu tzw. przestrzeni nazw
<code>void</code>	Używane w nagłówku funkcji, która nie zwraca wartości
<code>virtual</code>	Używane w deklaracji metody wirtualnej
<code>while</code>	Używane w instrukcji iteracyjnej

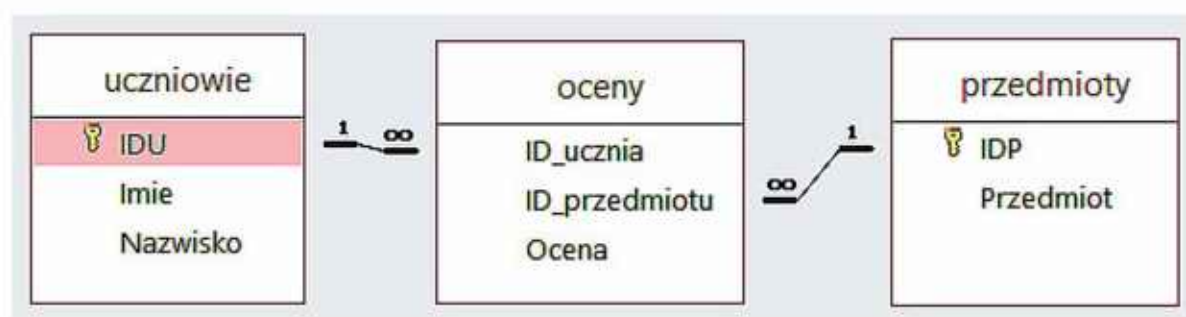
Inne słowa kluczowe

<code>asm</code>	<code>operator</code>
<code>auto</code>	<code>register</code>
<code>catch</code>	<code>reinterpret_cast</code>
<code>const_cast</code>	<code>signed</code>
<code>continue</code>	<code>sizeof</code>
<code>delete</code>	<code>static</code>
<code>dynamic_cast</code>	<code>static_cast</code>
<code>enum</code>	<code>template</code>
<code>explicit</code>	<code>this</code>
<code>export</code>	<code>throw</code>
<code>extern</code>	<code>try</code>
<code>friend</code>	<code>typeid</code>
<code>goto</code>	<code>typename</code>
<code>inline</code>	<code>union</code>
<code>mutable</code>	<code>volatile</code>
<code>new</code>	<code>wchar_t</code>

7. Kwerendy modyfikujące w programie Microsoft Access

Kwerendy w programie Microsoft Access mogą służyć nie tylko do tworzenia zestawień informacji zebranych w tabelach bazy danych. Za ich pomocą można też dodawać nowe tabele oraz rekordy do tabel, aktualizować lub usuwać istniejące rekordy, a nawet łączyć ze sobą wyniki z innych kwerend. Typ kwerendy można zmienić w widoku projektu po wybraniu **Narzędzia kwerend** → **Projektowanie** → **Typ kwerendy**. Do wykonania działań ujętych w kwerendzie służy przycisk **Uruchom** widoczny na karcie **Projektowanie**.

W bazie danych, której tabele, atrybuty tabel i związki między nimi widać na poniższym rysunku, znajdują się informacje o uczniach pewnej szkoły oraz ich ocenach końcowych z poszczególnych przedmiotów.



Na przykładzie powyższej bazy danych opiszemy kwerendy modyfikujące, które są dostępne w programie Microsoft Access.

Kwerenda dołączająca rekordy

Aby dodać nowych uczniów do tabeli uczniowie, najpierw należy ich wpisać do nowej tabeli (np. tabeli nowi) i w widoku projektu kwerendy zmienić typ na **Dołączająca**. W wyskakującym oknie dialogowym można teraz wybrać jedną z tabel bazy danych lub nawet utworzyć zupełnie nową tabelę. Można tu również wskazać inną bazę danych i do niej dołączyć nowe dane. Po uzupełnieniu siatki widoku projektu w poniższy sposób i uruchomieniu kwerendy wszystkie rekordy z tabeli nowi zostaną dodane do tabeli uczniowie.

Pole:	IDU	Imie	Nazwisko
Tabela:	nowi	nowi	nowi
Sortuj:			
Dołączanie do:	IDU	Imie	Nazwisko
Kryteria:			
lub:			

Kwerenda aktualizująca rekordy

Aby zaktualizować wartości wybranych atrybutów, można wykorzystać kwerendę aktualizującą. Kwerenda pokazana na s. 431 wszystkie oceny z języka polskiego podniesie o 1 stopień. Dodatkowe kryterium pozwoli uniknąć podniesienia ocen celujących.

Pole:	Przedmiot	Ocena
Tabela:	przedmioty	oceny
Aktualizacja do:		+1
Kryteria:	"Język polski"	<6
lub:		

Kwerenda usuwająca rekordy

Aby usunąć rekordy z tabeli za pomocą kwerendy, zmieniamy typ kwerendy na **Usuująca** i w siatce widoku projektu ustawiamy kryteria, według których mają być wybrane rekordy do usunięcia. Poniższa kwerenda usunie z tabeli oceny wszystkie jedynki.

Pole:	Ocena
Tabela:	oceny
Usuwanie:	Gdzie
Kryteria:	=1
lub:	

Kwerenda tworząca tabelę

Aby utworzyć nową tabelę z danych istniejących w innych tabelach, zmieniamy typ kwerendy na **Utwórz tabelę**, w oknie dialogowym wpisujemy nazwę nowej tabeli i w siatce widoku projektu wybieramy, jakie atrybuty ma zawierać nowa tabela. Możemy też ustalić kryteria, dzięki którym w nowej tabeli znajdą się tylko rekordy spełniające określone warunki. Poniższa kwerenda wybierze dane z tabel uczniowie oraz oceny i utworzy z nich nową tabelę, zawierającą dane uczniów, których średnia ocen jest wyższa niż 4,5.

Pole:	IDU	Imie	Nazwisko	Ocena
Tabela:	uczniowie	uczniowie	uczniowie	oceny
Suma:	Grupuj według	Grupuj według	Grupuj według	Średnia
Sortuj:				
Pokaż:	☒	☒	☒	☒
Kryteria:				>4,5

Kwerenda składająca tablele

Istnieje również kwerenda łącząca w jednej tabeli wybrane atrybuty z dwóch lub więcej tabel. Tworzy się ją z wykorzystaniem języka SQL. Poniższe zapytanie wybierze wszystkie wartości atrybutów Imie i Nazwisko z tabeli uczniowie oraz IDP i Przedmiot z tabeli przedmioty, a następnie wyświetli je razem w dwóch kolumnach.

1. `SELECT Imie, Nazwisko`
2. `FROM uczniowie`
3. `UNION`
4. `SELECT IDP, Przedmiot`
5. `FROM przedmioty;`

8. Typy danych i maski w programie Microsoft Access

Wybrane typy danych

Typ danych	Opis
Krótki tekst	Pozwala na przechowywanie 255 znaków alfanumerycznych (np. liter, liczb, znaków przestankowych). Wykorzystywany jest do zapisywania imion, nazwisk, danych teleadresowych i liczb, na których nie wykonuje się obliczeń (np. numerów telefonów).
Długi tekst	Podobnie jak Krótki tekst pozwala zapisać teksty, może jednak zawierać do 64 tysięcy znaków.
Liczba	Przechowuje dane liczbowe, na których można wykonywać obliczenia, z wyjątkiem obliczeń walutowych (patrz: typ Waluta).
Data/Godzina	Przechowuje informacje na temat czasu: daty i godziny.
Waluta	Przechowuje dane liczbowe z dokładnością do 4 miejsc dziesiętnych i zapobiega zaokrągleniom podczas obliczeń.
Autonumerowanie	Przechowuje unikatową wartość generowaną przez program dla każdego nowego rekordu.
Tak/Nie	Przechowuje wartość logiczną typu prawda (1) lub fałsz (0).
Załącznik	Pozwala dołączać pliki takie jak obrazy, dokumenty tekstowe, arkusze kalkulacyjne lub wykresy. Każde pole typu Załącznik może zawierać dowolną liczbę załączników na rekord.
Hiperłącze	Przechowuje adres (ścieżkę) do zasobów znajdujących się w sieci Internet, sieci lokalnej lub na dysku komputera.
Kreator odnośników	Nie jest faktycznym typem danych, pozwala jednak uruchomić kreator, za pomocą którego można wybrać wartości z innej tabeli lub listy.

Wybrane znaki stosowane do definiowania masek wprowadzania

Znak	Opis
0	Użytkownik musi wprowadzić cyfrę.
9	Użytkownik może wprowadzić cyfrę.
L	Użytkownik musi wprowadzić literę.
A	Użytkownik musi wprowadzić cyfrę lub literę.
?	Użytkownik może wprowadzić literę.
#	Użytkownik może wprowadzić cyfrę, spację, znak plus lub minus.
>	Wszystkie znaki po tym symbolu wyświetlane są jako duże litery.
<	Wszystkie znaki po tym symbolu wyświetlane są jako małe litery.
&	Użytkownik musi wprowadzić dowolny znak lub spację.

9. Klauzule języka SQL

Aby wykonać operacje na danych zawartych w tabelach bazy danych, wykorzystuje się zapytania w języku SQL. Tworzy się je za pomocą klauzul, które z kolei budowane są ze słów kluczowych tego języka. Oto lista klauzul najczęściej wykorzystywanych w zapytaniach języka SQL.

Klauzula	Opis
SELECT	Wybór atrybutów do wyświetlenia
FROM	Wybór tabel, z których rekordy mają być wyświetlone
WHERE	Selekcja wyników zapytania
ORDER BY	Sortowanie wyników zapytania
GROUP BY	Grupowanie wyników po użyciu funkcji agregującej
HAVING	Selekcja wyników zapytania (po użyciu funkcji agregującej)
INNER JOIN	Tworzenie złączenia wewnętrznego między tabelami
LEFT JOIN	Tworzenie złączenia zewnętrznego lewego między tabelami
RIGHT JOIN	Tworzenie złączenia zewnętrznego prawego między tabelami
ON	Określenie kryteriów złączenia tabel
DISTINCT	Usuwanie duplikatów z wyniku zapytania
DESC	Wybór sposobu sortowania: malejąco
ASC	Wybór sposobu sortowania: rosnąco
AS	Przypisanie atrybutowi aliasu, czyli innej nazwy
SUM	Funkcja agregująca: sumowanie wartości atrybutu
AVG	Funkcja agregująca: średnia wartości atrybutu
COUNT	Funkcja agregująca: zliczanie wystąpień atrybutu
MIN	Minimalna wartość atrybutu
MAX	Maksymalna wartość atrybutu
CREATE DATABASE	Tworzenie bazy danych
CREATE TABLE	Tworzenie tabeli
INSERT INTO	Dodawanie rekordu do tabeli
VALUES	Wpisanie wartości rekordu dodawanego do tabeli
AUTO_INCREMENT	Automatyczne przypisanie wartości atrybutowi według ustalonej reguły
DELETE	Usuwanie rekordu z tabeli
DROP TABLE	Usuwanie tabeli
DROP DATABASE	Usuwanie bazy danych
USE	Wybór aktywnej tabeli
INFILE	Określenie nazwy pliku, z którego importowane są dane do bazy
OUTFILE	Określenie nazwy pliku, do którego eksportowane są dane z bazy

10. Podzapytania w języku SQL

Podstawą zarządzania bazą danych w języku SQL jest konstruowanie zapytań. Wydobycie konkretnych informacji czasami wymaga wykorzystania wyniku jednego zapytania w drugim zapytaniu. Tworzy się w ten sposób podzapytanie.

W bazie danych, której tabele, atrybuty tabel i związki między nimi widać na poniższym rysunku, znajdują się informacje o uczniach pewnej szkoły oraz ich ocenach końcowych z poszczególnych przedmiotów.



Aby dowiedzieć się, jakie średnie ocen uzyskali uczniowie, można zastosować następujące zapytanie:

```
1. SELECT uczniowie.Imie, uczniowie.Nazwisko,
2. AVG(oceny.Ocena) AS Srednia_ucznia
3. FROM uczniowie INNER JOIN oceny
4. ON uczniowie.IDU=oceny.ID_ucznia
5. GROUP BY uczniowie.Imie, uczniowie.Nazwisko;
```

Do obliczenia średniej wszystkich ocen służy zapytanie:

```
1. SELECT AVG(oceny.Ocena)
2. FROM oceny;
```

Wynikiem drugiego zapytania jest jedna wartość – liczba rzeczywista opisująca średnią wszystkich ocen. Można to zapytanie wykorzystać jako część pierwszego zapytania, tworząc podzapytanie:

```
1. SELECT uczniowie.Imie, uczniowie.Nazwisko,
2. AVG(oceny.Ocena) AS Srednia_ucznia
3. FROM uczniowie INNER JOIN oceny
4. ON uczniowie.IDU=oceny.ID_ucznia
5. GROUP BY uczniowie.Imie, uczniowie.Nazwisko
6. HAVING Srednia_ucznia > (SELECT AVG(oceny.Ocena) FROM oceny);
```

Wykorzystana została klauzula **HAVING** (linia 6), która filtruje wyniki po użyciu funkcji agregującej (w tym przypadku **AVG**). Za pomocą tej klauzuli wskazano, że w wyniku mają się znaleźć tylko ci uczniowie, których średnia (**Srednia_ucznia**) jest większa niż średnia wszystkich ocen. Do obliczenia średniej wszystkich ocen użyto zapytania umieszczonego w nawiasach, które stało się w ten sposób podzapytaniem.

11. Funkcje daty i czasu w języku SQL

Podczas zarządzania informacjami zawartymi w bazie danych zachodzi czasami potrzeba wykonania operacji związanych z datą lub godziną. Stosuje się wtedy poniższe funkcje daty i czasu w połączeniu z klauzulą `SELECT`. W nawiasach po nazwie funkcji wpisujemy jej parametry. Parametrami mogą być: data w formacie '2021-09-01', godzina w formacie '09:25:28' lub połączenie daty i godziny '2021-09-01 09:25:28'. Funkcja może też nie mieć żadnych parametrów. Funkcje daty i czasu zostały zebrane w poniższej tabeli.

Funkcja	Opis
<code>CURRENT_DATE()</code>	Zwraca aktualną datę.
<code>CURRENT_TIME()</code>	Zwraca aktualną godzinę.
<code>CURRENT_TIMESTAMP()</code>	Zwraca aktualną datę i godzinę.
<code>DAYNAME()</code>	Zwraca nazwę dnia tygodnia.
<code>DAYOFMONTH()</code>	Zwraca dzień miesiąca.
<code>DAYOFWEEK()</code>	Zwraca numer dnia tygodnia.
<code>YEAR()</code>	Zwraca rok.
<code>MONTH()</code>	Zwraca numer miesiąca.
<code>MONTHNAME()</code>	Zwraca nazwę miesiąca.
<code>hour()</code>	Zwraca godzinę.
<code>minute()</code>	Zwraca minutę.
<code>ADDTIME()</code>	Dodaje czasy do siebie.
<code>DATE_ADD()</code>	Dodaje daty do siebie.
<code>DATE_SUB()</code>	Odejmuje liczbę dni od daty.
<code>NOW()</code>	Zwraca aktualną datę i godzinę.
<code>DATEDIFF()</code>	Odejmuje daty od siebie.
<code>TIMEDIFF()</code>	Odejmuje czasy od siebie.
<code>WEEKOFYEAR()</code>	Zwraca numer tygodnia w roku.

12. Typy danych w języku SQL

Podczas tworzenia tabeli bazy danych określa się nazwy jej atrybutów (kolumn) oraz typy danych, jakie mogą one przyjmować. Poniższa tabela zawiera spis tych typów.

Nazwa typu	Opis
int, integer	Liczba całkowita z zakresu od -2 147 483 648 do 2 147 483 647
mediumint	Liczba całkowita z zakresu od -8 388 608 do 8 388 607
smallint	Liczba całkowita z zakresu od -32 768 do 32 767
tinyint	Liczba całkowita z zakresu od -128 do 127
decimal(p,s), numeric(p,s)	Liczba rzeczywista (p – liczba wszystkich cyfr w liczbie, s – liczba cyfr po przecinku)
real, float(p)	Liczba zmiennoprzecinkowa o precyzji p; real odpowiada typowi float(24)
double	Liczba zmiennoprzecinkowa z zakresu od -1.7976931348623157E+308 do -2.2250738585072014E-308
decimal	Pole zmiennoprzecinkowe z zakresu takiego jak double, jednak liczba jest zapamiętana w formacie znakowym
char(n), character(n)	Typ znakowy o długości n
varchar(n)	Łańcuch znaków o długości n
date	Data w formacie 'rok-miesiąc-dzień'
time	Czas w formacie 'godzina-minuta-sekunda'
datetime	Data i czas w formacie 'rok-miesiąc-dzień godzina-minuta-sekunda'
year	Rok z przedziału 1901-2155
timestamp	Data i czas liczone od 1970-01-01 00:00:00:00

Indeks

A

agregowanie treści 283
 akcelerometr 354
 akcentująca funkcja koloru 394
 algorytm badający położenie punktów względem prostej 149
 ----- odcinka 152
 -- przecinanie się dwóch odcinków 156
 -- przynależność punktu do trójkąta 159
 ----- wielokąta wypukłego 160
 -- Dijkstry 73
 -- Karpa–Rabina 207
 -- naiwny wyszukiwania wzorca w tekście 202
 -- niestabilny 117
 -- obliczania pierwiastka kwadratowego metodą bisekcji 134
 ----- Newtona–Raphsona 134
 -- pola obszaru zamkniętego metodą prostokątów 138
 ----- trapezów 140
 -- przybliżenia liczby π metodą Monte Carlo 142
 ----- prostokątów 142
 ----- trapezów 142
 -- wartości wielomianu (schemat Hornera) 124
 ---- wyrażenia arytmetycznego w ONP, rekurencyjny 20
 ----- z wykorzystaniem stosu 23
 -- przeszukiwania w głąb (DFS) 69
 -- wszerek (BFS) 71
 -- z nawrotami 38
 -- RSA 215
 -- sortowania leksykograficznego 53
 -- kubelkowego 53
 -- stabilny 118
 -- zamiany wyrażenia z notacji tradycyjnej na ONP 12
 -- znajdowania miejsc zerowych funkcji metodą bisekcji 129
 -- pierwiastków równania kwadratowego, niestabilny 116
 ----- stabilny 118
 -- rozwinięcia binarnego ułamka właściwego 98
 -- wyjścia z labiryntu, iteracyjny z wykorzystaniem kolejki 40
 ----- rekurencyjny 31
 algorytmy rysowania fraktali rekurencyjne 165
 ---- metodą IFS 183
 architektura klient–serwer 292

arytmetyka modularna 208
 atrybut (w bazach danych) 256
 -- (pole) (w programowaniu obiektowym) 226

B

back-end 323
 barwy dopełniające 396
 -- monochromatyczne 396
 -- pochodne 395
 -- podstawowe 395
 -- sąsiadujące 396
 -- trójkątne 396
 -- trzeciego rzędu 395
 baza danych 254
 biały znak 220
 biblioteka p5.js 168
 -- STL 16
 błąd bezwzględny 115
 -- obcięcia 114
 -- przybliżenia 113
 -- reprezentacji 112
 -- względny 115
 -- zaokrąglania 112

C

cecha 106
 clickbait 369
 czujnik dotykowy 352
 -- kolorów 352
 -- podczerwieni 352
 -- temperatury 352
 -- ultradźwiękowy 352
 -- żyroskopowy 352

D

definicja klasy 226
 -- własnej nazwy typu zmiennych 67
 deklaracja iteratora dla typu `list` 51
 -- kolejki 42
 -- listy 50
 -- stosu 16
 -- zmiennej typu `vector` 66
 dekompresja 384
 destruktor 231
 dokładność obliczeń 113
 drzewo binarne 178
 -- wyrażenia algebraicznego 11
 dynamiczna struktura danych 12

dywan Sierpińskiego 180
dziedziczenie atrybutów i metod 233

F

faktoryzacja liczby 215
FIFO 39
formularze 268
fraktal 165
front-end 323
funkcja abs 114
— agregująca `SUM` 313
— `background` 169
— `createCanvas` 169
— `draw` 169
— `fill` 183
— haszująca (funkcja skrótu) 206
— `line` 169
— `max` 155
— `min` 155
— `noLoop` 170
— `setprecision` 113
— `setup` 169
— `setw` 30
— `sqrt` 117
— `square` 183
funkcja koloru 394
— — akcentująca 394
— — rozróżniająca 394
— — wartościująca 394

G

graf 62
— nieskierowany 63
— prosty 62
— skierowany 63
— spójny 63
— ważony 62
grafika informacyjna (infografika) 388
— narracyjna 390

H

hashtag 369
hasz 206
hermetyzacja danych 227
hierarchia klas 232

I

identyfikacja wizualna 376
iloczyn kartezjański 310
infografika (grafika informacyjna) 388
instrukcja `break` 19
— `continue` 18
— warunkowa `if` (język PHP) 333

— wyboru `switch` 18
integralność referencyjna danych 302
iterator 51

J

język Isotype 392
— MicroPython 355
— PHP 329
— SQL 289

K

klasa 226
— abstrakcyjna 237
— bazowa 233
— `mysqli` 330
— pochodna 233
klauszula 291
— `AS` 313
— `AUTO_INCREMENT` 297
— `CHARACTER SET` 298
— `COLLATE` 294
— `CREATE DATABASE` 294
— `CREATE TABLE` 296
— `CREATE USER` 303
— `DELETE FROM` 299
— `DESC` 313
— `DROP DATABASE` 300
— `DROP TABLE` 300
— `FOREIGN KEY` 301
— `FROM` 298
— `GRANT ALL PRIVILEGES` 303
— `GROUP BY` 313
— `HAVING` 314
— `IDENTIFIED BY` 303
— `IGNORE LINES` 298
— `INNER JOIN` 312
— `INSERT INTO` 297
— `INTO TABLE` 298
— `LEFT JOIN` 313
— `LINES TERMINATED BY` 298
— `LOAD DATA INFILE` 298
— `ON` 312
— `ORDER BY` 309
— `PRIMARY KEY` 297
— `REFERENCES` 301
— `RIGHT JOIN` 313
— `SELECT` 298
— `SET` 299
— `SET PASSWORD` 303
— `UPDATE` 299
— `USE` 295
— `VALUES` 297

- `WHERE` 306
- klucz (dla funkcji haszującej) 214
 - (w kryptografii) 206
 - główny (klucz podstawowy) 257
 - obcy 257
 - prywatny 214
 - publiczny 214
 - złożony 278
- kod z nadmiarem 107
- kolejka 39
 - priorytetowa 80
- kolizja 206
- koło barw 395
- kompresja bezstratna 384
 - danych 383
 - stratna 384
- konspekt 373
- konstruktor 229
- kreator odnośników 265
- kryptografia asymetryczna (klucza publicznego) 214
 - symetryczna 214
- kryptogram (szyfrogram) 214
- krzywa Kocha 173
- kwerenda 270
 - krzyżowa 282
 - wybierająca 279

L

- liczba podwójnej precyzji 107
 - pojedynczej precyzji 107
- lid 369
- LIFO 11
- lista 47
 - dwukierunkowa 48
 - — cykliczna 48
 - jednokierunkowa 47
 - — cykliczna 48
- listy sąsiedztwa 64
- logo 376
- logotyp 376

M

- macierz 152
 - kwadratowa 152
 - sąsiedztwa 64
- magnetometr 354
- manipulator noskipws 221
 - strumienia 31
- mantysa 106
- maska wprowadzania 265
- metoda 226
 - `begin` dla klasy `list` 51

- bisekcji (połowienia) 129
- `close()` 331
- `c_str` dla klasy `string` 232
- czysto wirtualna 237
- `empty` dla klasy `queue` 42
- — — — `stack` 16
- `end` dla klasy `list` 51
- `erase` dla klasy `list` 52
- `fetch_array()` 333
- `front` dla klasy `queue` 42
- IFS 184
- Monte Carlo 142
- Newtona–Raphsona 134
- `pop` dla klasy `queue` 42
- — — — `stack` 16
- prostokątów 138
- `push` dla klasy `queue` 42
- — — — `stack` 16
- `push_back` dla klasy `list` 52
- — — — `vector` 68
- `query()` 331
- `resize` dla klasy `vector` 66
- `size` dla klasy `vector` 66
- `top` dla klasy `stack` 16
- trapezów 140
- wirtualna 236
- wstępująca 225
- zstępująca 225

N

- nasycenie koloru 395
- notacja infiksowa 10
 - prefiksowa (notacja polska) 10
 - sufiksowa (odwrotna notacja polska – ONP) 10

O

- obiekt 226
- odwrotna notacja polska (ONP, notacja sufiksowa) 10
- operacje na kolejce 39
 - na stosie 12
- operator `IS`, znacznik `NULL` 307
 - konkatencji (język PHP) 333
 - logiczny 308
 - porównania 307
 - zasięgu 228
- operatory arytmetyczne 307
- organiczne wyniki wyszukiwania 369
- oś czasu 393

P

- pętla `while` (język PHP) 333
- pierwiastki równania kwadratowego 116

piktogram 389
 plątek Kocha 176
 podcast 377
 pole (atrybut) (w programowaniu obiektowym) 226
 — rekordu 256
 polimorfizm (wielopostaciowość) 236
 położenie punktu względem prostej 149
 porządek leksykograficzny 53
 postać znormalizowana 106
 problem Józefa Flawiusza 49
 — wyszukiwania wzorca w tekście 202
 Processing 171
 programowanie obiektowe 225
 — strukturalne 225
 projekcja atrybutów 309
 przeciążanie metod 231
 przekształcenie afiniczne 183
 przestawianie metod 233
 przynależność punktu do odcinka 153

R

raport 272
 redundancja 255
 reguła Sarrusa 152
 rekord 256
 relacja 257
 — typu jeden do jednego 258
 — — — — — wielu 257
 — — wiele do wielu 258
 — w modelu relacyjnym 290
 relacyjna baza danych 290
 reprezentacja stałoprzecinkowa (stałopozycyjna) liczb 104
 — zmiennoprzecinkowa (zmiennopozycyjna) liczb 106
 robot 349
 rozróżniająca funkcja koloru 394
 równanie ogólne prostej 149
 — prostej przechodzącej przez dwa punkty 152
 — — — — — zapis z wykorzystaniem wyznacznika macierzy 153
 ruchy Browna 145

S

sąsiednie wierzchołki grafu 62
 schemat Hornera 125
 — linii komunikacyjnych 391
 — organizacyjny 390
 — relacyjnej bazy danych 258
 — tabeli 256
 screencast 377
 selekcja wyników zapytania 306

serwomotor 352
 słowo kluczowe `echo` 331
 sortowanie kubelkowe 53
 — leksykograficzne 53
 — stabilne 59
 specyfikatory dostępu 227
 sposoby akcentowania informacji 397
 — porządkowania informacji 388
 stała `RAND_MAX` 144
 stopień kompresji 383
 stos 11
 sygnet 376
 symulator robota 349
 system zarządzania bazą danych (SZBD) 258
 szum tła 381
 szyfrogram (kryptogram) 214

T

tabela bazy danych 256
 — pomostowa (tabela łącząca, tabela łączenia) 277
 — programu Access 259
 tablica dynamiczna 66
 tekst informacyjny 373
 trójkąt Sierpińskiego 184
 typ `unsigned char` 220

W

waga krawędzi grafu 62
 wartościująca funkcja koloru 394
 wartownik 16
 webcast 377
 webinarium (webinar) 377
 wektor 155
 widok projektu 262
 wielokąt wypukły 159
 wielomian 122
 wielopostaciowość (polimorfizm) 236
 wizualizacja danych 390
 wyznacznik macierzy 152

Z

zapisywanie zmian w tabeli 264
 zapytanie do bazy danych 290
 zasada 5W 374
 zbiór Cantora 165
 złączenie wewnętrzne 312
 — zewnętrzne prawe 313
 — — lewe 313
 znacznik `NULL` 307
 związek 290

Zdjęcie na okładce: Shutterstock/Albisoima.

Zdjęcia: Archiwum NE: s. 389 (znaki drogowe); BE&W: Alamy Stock Photo – Addictive Stock Creatives s. 379, Classic Image s. 49, GK Images s. 221, John Frost Newspapers s. 391, Kristoffer Tripplaar s. 347 (samochód autonomiczny), NASA Image Collection s. 115, Science History Images s. 346 (fabryka Forda), s. 8 i 393 (Grace Hopper), s. 96 (Margaret Hamilton), The Picture Art Collection s. 97 (Dennis Ritchie), Wartimep s. 8 i 393 (komputer IBM 704), EXCLUSIVE PIX Exclusivepix Media Ltd/Exclusivepix s. 346 (Pierre Jaquet-Droz), LANDMARK MEDIA/Supplied by LMK s. 167 (kadr z filmu „Gwiezdne wojny” Epizod 3), Mary Evans Picture Library s. 348; East News: AFP/Andrew Caballero-Reynolds s. 201 (Katherine Bouman); Forum: Archiwum RSW s. 77, Interfoto/Friedrich s. 389 (piktogramy olimpijskie Otto Aichera); Getty Images: 500px Prime/Satellite Earth Art s. 167 (Himalaje), Corbis Entertainment/Rick Friedman s. 96 (Seymour Papert), Corbis Historical – Kim Kulish s. 97 (algorytm RSA), Leemage s. 252 (Biblioteka Aleksandryjska), DigitalVision – Ariel Skelley s. 250 (kobieta na tle monitora), Luis Alvarez s. 343, E+ – hudiem s. 11, ivstiv s. 166-167 (mróz na szybie – tło), JacobH s. 167 (wykres giełdowy), martin-dm s. 378, Mediterranean s. 250 (astronom), miodrag ignjatovic s. 177, Moyo Studio s. 250 (kobieta z jabłonią), Tarik Kizilkaya s. 205, Vertigo3d s. 7, Hulton Archive/Getty Images s. 347 (Izzy Sojourner), Image Source/Igor Emmerich s. 250 (spotkanie zespołu), iStock/Getty Images Plus – AdrianHancu s. 95, Balavan s. 167 (telefon), Davizro s. 82, Kateryna Kukota s. 342 (ręcznie robione zabawki), koya79 s. 345, Kwarkot s. 251, putilich s. 199, iStockphoto/Cristian Storto Fotografia s. 346 (tranzystor), Moment – by IAI s. 167 (kalafior romanesco), dowell s. 37, MR.Cole_Photographer s. 253 (mózg), Photodisc/John Lund s. 283, Science Photo Library/Laguna Design s. 166 (żuk Mandelbrota), The India Today Group s. 201 (James Gosling), VCG/Visual China Group s. 404 (chirurg); Indigo: Alamy Stock Photo – agefotostock s. 200 (Radia Perlman), Foto Arena LTDA s. 125, Gonzales Photo s. 355, Pictorial Press Ltd s. 346 (Archytas z Tarentu), UPI s. 252 (Charles Bachman), Science Photo Library s. 9 i 393 (John Backus), s. 347 (Alan M. Turing), British Library s. 395, Clouds Hill Imaging LTD s. 167 (kolonia bakterii *Paenibacillus*), Hagley Museum And Archive s. 252 (CODASYL); Shutterstock: Africa Studio s. 405, akramalrasny s. 398, Andrey_Popov s. 404 (grafik komputerowy), Anton Gvozdkov s. 347 (robot humanoidalny Sophia), April stock s. 374, Artmim s. 201 (Algorytm BERT – search), Astibuag s. 301, BAIVECTOR s. 74-75, Emilija Miljkovic s. 330, EvalCo s. 253 (SEQUEL), feeling lucky s. 370 (dziewczyna przy komputerze), FGC s. 284-285, freestyle images s. 361, fullvector s. 322 (serwerownia – ilustracja), hedgehog94 s. 342 (dziewczyna w schronisku dla zwierząt), Kaikoro s. 135, Kijja P44 s. 181, Kindersps s. 54-55, Mr.Thanakorn Kotpootorn s. 322 (komputer – ilustracja), Nancy Beijersbergen s. 146, NDAB Creativity s. 404 (zebranie zespołu), New Africa s. 106, NicoElNino s. 267, s. 342 (analitik danych), ORION PRODUCTION s. 404 (kobieta z kamerą), Picsfive s. 370 (kartka papieru), Rawpixel.com s. 371 (ludzie z telefonami), s. 372 (laptop), Rymden s. 385, SAPhotog s. 144, Sergey Chirkov s. 370 (lampa fotograficzna), Sergey Nivens s. 8 i 393 (haszowanie), solarseven s. 271, Stefano Garau s. 370 (aparat na statywie), tanatat s. 371 (lupa), TippaPatt s. 212, Vmaster s. 372 (rakieta – ilustracja), wavebreakmedia s. 342 (siłownia); Wikipedia: Wikimedia Commons/manuscripts of Edsger W. Dijkstra, University Texas at Austin, Hamilton Richards (CC BY-SA 3.0) s. 8 i 393 (Edsger Dijkstra), Wikimedia Commons (CC-BY-SA-2.0) s. 9 i 393 (John McCarty), Wikimedia Commons/Photograph by Rama/Cc-by-sa-2.0-fr s. 9 i 393 (Tony Hoare), Wikimedia s. 9 i 393 (COBOL), Wikimedia/Commons s. 23 (Friedrich Bauer), Wikimedia/Tyomitch s. 97 (Niklaus Wirth), Wikimedia Commons/Mister rf (CC BY-SA 4.0) s. 97 (kalkulator HP35), Wikimedia Commons/GaryWarth(CC BY-SA 3.0) s. 200 (Yvonne Marie Andres), Wikimedia Commons/Jeremy Kratz s. 200 (logo C++), Wikimedia Commons/Komitet Główny Olimpiady Informatycznej link: <https://oi.edu.pl/l/logo> s. 376 (logo Olimpiady Informatycznej); Report on the Loglan 82 Programming Language, Institute of Informatics University of Warsaw/Wiesława Maria Bartol, Paweł Gburzyński, Piotr Findeisen, Antoni Kreczmar, Marek Lao, Andrzej I. Litwiniuk, Tomasz Müldner, Wojciech Nykowski, Hanna Oktaba, Danuta Szczepańska, Andrzej Salwicki s. 200 (fragment kodu Loglan 82); link: www.python.org s. 201 (logo Python); fragment z pracy <https://ee.stanford.edu/~hellman/publications/24.pdf> s. 216 (public key system); Norwegian Computing Center, Ole-Johan Dahl & Kristen Nygaard – https://commons.wikimedia.org/wiki/File:Simula_-_logo.svg s. 96 i 228 (Simula 67 – logo); link: <https://www.php.net/images/logos/new-php-logo.png> s. 253 (logo PHP); link: <https://archive.org/web/> s. 308 (logo Internet Archive Wayback Machine); ISOTYPE link: <https://medium.com/nightingale/exploring-isotype-charts-only-an-ocean-between-part-1-399f227e1c69> link: <http://isotyperevisited.org/2012/08/introduction.html> s. 392 (infografika Ottoma Neuratha wykorzystująca piktogramy ISOTYPE); Janusz Mazur s. 349, 352, 354 (robot micro: Maqueen).

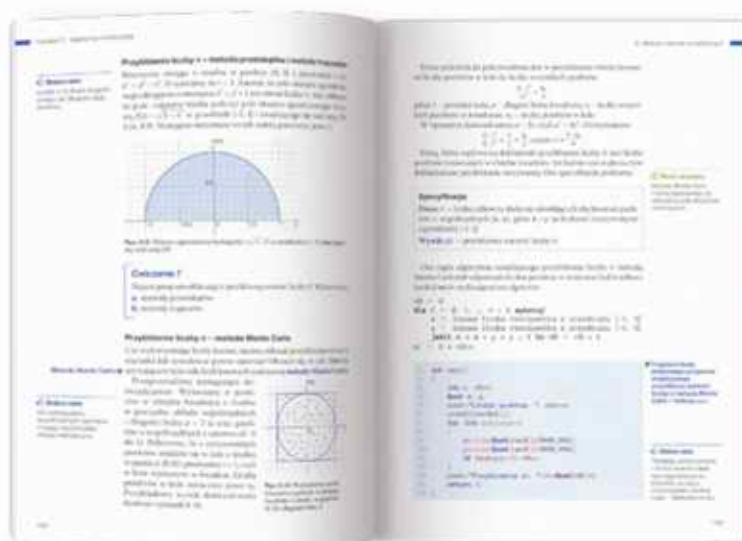
Ilustracje: Marcin Ptak (s. 21, 28-29, 33-34, 39 – układ, 41 – labirynt, 47-51 – schematy, 62-77 – grafy, schematy, 130-160 – wykresy funkcji, 146 – symulacja, 159 – raster, 165, 173-174, 177, 179-184, 190, 289, 292 – układ, 350-351, 373, 380-382 – ikonki), Joanna Ptak (s. 394, 396-397, 402), Artur Polakowski (s. 10-11, 64, 105, 107-108, 161, 203, 235-238, 376, 390), Sławomir Włodarczyk (s. 39 – awatar, 292 – komputery), Zuzanna Dudzic (s. 395).

Na stronach 398-399 wykorzystano infografikę „Konflikty na świecie” zamieszczoną w podręczniku *Dziś i jutro. Podręcznik do wiedzy o społeczeństwie dla szkoły podstawowej* autorstwa Iwony Janickiej, Arkadiusza Janickiego, Aleksandry Kuci-Mańkowskiej, Tomasza Mańkowskiego, s. 212-213.

Wydawnictwo dołożyło wszelkich starań, aby odnaleźć posiadaczy praw autorskich do wszystkich utworów zamieszczonych w podręczniku. Pozostałe osoby prosimy o kontakt z Wydawnictwem.

Wydawnictwo Nowa Era nie ponosi odpowiedzialności za treści i zasoby zamieszczone na cytowanych w tej publikacji stronach internetowych. Adresy stron internetowych zweryfikowano 26.10.2021 r.

Podręcznik *Informatyka na czasie 3* do zakresu rozszerzonego zawiera treści dotyczące algorytmiki, programowania w języku C++ i tworzenia fraktali w języku JavaScript, a także baz danych, robotyki, publikowania w sieci i tworzenia infografik. Opanowanie tych zagadnień ułatwi aktywny margines z informacjami uzupełniającymi wiadomości z lekcji. W nauce pomogą również atrakcyjne infografiki oraz liczne rysunki i kody źródłowe.



Aktywny margines

Zawiera siatkę pojęć, odsyłacze oraz praktyczne wskazówki ułatwiające opanowanie niezbędnych wiadomości.

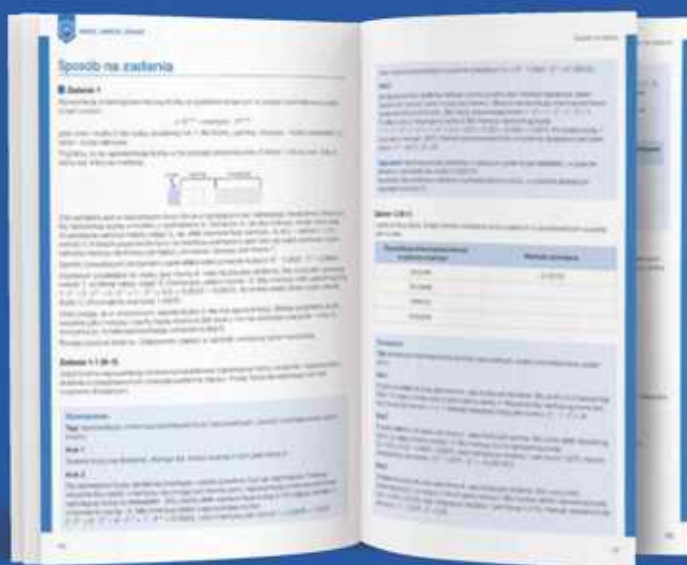
Różnorodne formy przekazu

Rysunki, infografiki, a także kody źródłowe z kolorową składnią pomagają zrozumieć i zapamiętać kluczowe informacje.



WIESZ, UMIESZ, ZDASZ

Podręcznik *Informatyka na czasie 3* wspiera kształcenie kluczowych umiejętności zawartych w podstawie programowej. Blok *Wiesz, umiesz, zdasz* składa się z dwóch części: *Sposób na zadania* to zadania rozwiązane krok po kroku, a *Trening* to zestaw zadań do samodzielnego rozwiązania. Dzięki nim można utrwalić nowo zdobytą wiedzę oraz wyćwiczyć umiejętność rozwiązywania zadań różnego typu.



Nowa Era Sp. z o.o.



www.nowaera.pl



nowaera@nowaera.pl



Centrum Kontaktu: 801 88 10 10, 58 721 48 00

ISBN 978-83-267-4295-8



9 788326 742958