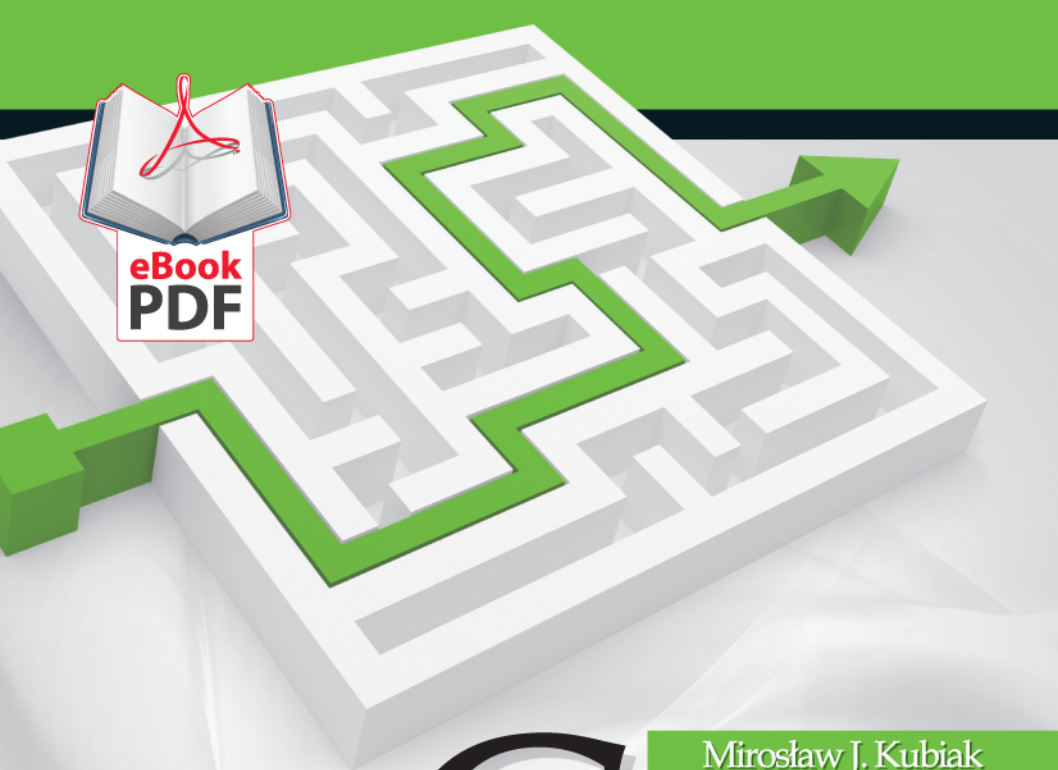


Mirosław J. Kubiak

C++

Zadania z programowania z przykładowymi rozwiązaniami

C++ w analizie konkretnych przykładów

- *Proste operacje wejścia/wyjścia*
- *Tablice, iteracje oraz podprogramy*
- *Programowanie obiektowe i pliki tekstowe*

Helion

Redaktor prowadzący: Ewelina Burska
Projekt okładki: Radosław Zbytniewski

Materiały graficzne na okładce zostały wykorzystane za zgodą iStockPhoto Inc.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie?cppzad_p
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody wykorzystane w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/cppzad.zip

ISBN: 978-83-246-3707-2
Numer katalogowy: 6900

Spis treści

Od autora

Trójzbiór *Zadania z programowania z rozwiązaniami* to pierwszy w Polsce zbiór zadań adresowany do wszystkich osób zainteresowanych programowaniem, które w krótkim czasie, poprzez analizę zaproponowanych rozwiązań, chciałyby nauczyć się solidnych podstaw programowania w trzech językach: Turbo Pascalu, C++ oraz Javie.

Składa się on z trzech zbiorów zadań:

Turbo Pascal. Zadania z programowania z przykładowymi rozwiązaniami.

C++. Zadania z programowania z przykładowymi rozwiązaniami.

Java. Zadania z programowania z przykładowymi rozwiązaniami.

Chociaż każdy z tych zbiorów **stanowi odrębną całość**, to zostały one napisane w taki sposób, aby ten sam lub bardzo podobny problem programistyczny (np. napisz program, który oblicza pole prostokąta) został rozwiązany w **trzech** językach programowania: Turbo Pascalu, C++ i Javie, strukturalnie i obiektowo. Tak skonstruowany trójzbiór *Zadania z programowania* zyskuje zupełnie nowy wymiar dydaktyczny w nauce tych trzech języków.

Zadania z programowania można również wykorzystać jako uzupełnienie wiedzy zaczerpniętej z innych książek do nauki programowania. Zakres i stopień trudności zadań pokrywa się z tradycyjnym procesem nauczania wymienionych języków. Zbiór ten może też pełnić rolę podręcznej pomocy dla początkujących programistów, w której szybko znajdą oni potrzebne im rozwiązanie.

Trójzbiór adresowany jest również do maturzystów, studentów, nauczycieli informatyki oraz osób zainteresowanych programowaniem lub rozpoczynających naukę programowania w języku C++.

Uczniowie techników informatycznych mogą zbiory zadań wykorzystać do szybkiej powtórki przed egzaminem zawodowym.

W trakcie pisania tej książki korzystałem z tzw. aplikacji konsolowych (ang. *console application*) i kompilatora firmy Borland (C++).

Mirosław J. Kubiak

1

Proste operacje wejścia-wyjścia

W tym rozdziale zamieszczono proste zadania z przykładowymi rozwiązaniami ilustrujące, w jaki sposób komputer komunikuje się z użytkownikiem w języku C++.

Każda aplikacja powinna posiadać możliwość komunikowania się z użytkownikiem. Wykorzystując proste przykłady, pokażemy, w jaki sposób program napisany w języku C++ komunikuje się z nim poprzez standardowe operacje wejścia-wyjścia.

Plik nagłówkowy z instrukcji

```
#include <iostream.h>
```

zawiera definicje klas[1] umożliwiających wykonywanie operacji wejścia-wyjścia na strumieniach. Do wyprowadzania danych na ekran służy standardowy strumień wyjściowy `cout`, który w języku C++ domyślnie przypisuje ekran do standardowego urządzenia wyjściowego systemu operacyjnego. Aby wyświetlić komunikat lub dane, trzeba do strumienia wyjściowego `cout` zastosować symbol podwójnego znaku mniejszości << (operacja wstawiania). Dwa znaki mniejszości należy wprowadzić z klawiatury.

[1] Więcej informacji na temat klas czytelnik znajdzie w rozdziale 6.

Do wprowadzania danych do programu służy standardowy strumień wejściowy `cin` oraz operator `>>` (dwa znaki większości, które również wprowadzamy z klawiatury), np. `cin >> a;`.

Do formatowania strumienia wyjściowego będziemy używali flagi formatującej `fixed` i manipulatora `setprecision(n)`. Flaga `fixed` używa do liczb zmiennoprzecinkowych ustalonej kropki dziesiętnej, natomiast manipulator `setprecision(n)` ustala ich precyzję na *n* — np. zapis `cout << setprecision(2);` oznacza, że liczby zmiennoprzecinkowe będą wyświetlane z dokładnością dwóch miejsc po kropce.

Zastosowanie manipulatora `setprecision(n)` wymaga włączenia do programu pliku nagłówkowego:

```
#include <iomanip.h>
```

Opisane powyżej podejście do operacji wejścia-wyjścia nazywa się obiektowym[2].

ZADANIE

1.1 Napisz program, który oblicza pole prostokąta. Wartości boków `a` i `b` wprowadzamy z klawiatury. W programie należy przyjąć, że zmienne `a` i `b` oraz `pole` są typu `float` (rzeczywistego). Przyjmujemy format wyświetlania ich na ekranie z dokładnością dwóch miejsc po kropce.

Przykładowe rozwiązanie — listing 1.1

```
#include <iostream.h> // Zadanie 1.1
#include <iomanip.h>
#include <conio.h>

main()
{
    float a, b, pole;

    cout << "Program oblicza pole prostokata." << endl;
    cout << "Podaj bok a." << endl;
    cin >> a;
    cout << "Podaj bok b." << endl;
    cin >> b;
    pole = a*b;
```

[2] Więcej informacji na temat obiektowych operacji wejścia-wyjścia, flag i manipulatorów znajdzie czytelnik na stronach WWW poświęconych językowi programowania C++ pod adresem *http://www.cplusplus.com/*.

```
    cout << fixed; // flaga
    cout << setprecision(2); // ustalenie precyzji
    cout << "Pole prostokata o boku a = " << a << " i boku b = " << b;
    cout << " wynosi " << pole << "." << endl;

    getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Linijka kodu

```
float a, b, pole;
```

umożliwia zadeklarowanie zmiennych `a`, `b` i `pole` (wszystkie zmienne w programie są typu rzeczywistego `float`). Instrukcja

```
cout << "Program oblicza pole prostokata." << endl;
```

wyświetla na ekranie komputera komunikat *Program oblicza pole prostokata*. Instrukcja `cin >> a;` czeka na wprowadzenie z klawiatury komputera liczby, która następnie zostanie przypisana zmiennej `a`. Pole prostokąta zostaje obliczone w wyrażeniu

```
pole = a*b;
```

Za wyświetlenie wartości zmiennych `a` i `b` oraz `pole` wraz z odpowiednim opisem są odpowiedzialne następujące linijki kodu:

```
    cout << "Pole prostokata o boku a = " << a << " i boku b = " << b;
    cout << " wynosi " << pole << "." << endl;
```

Flaga `fixed` używa ustalonej kropki dziesiętnej dla liczb zmiennoprzecinkowych. Zapis

```
cout << setprecision(2);
```

oznacza, że liczby te będą wyświetlane na ekranie z dokładnością dwóch miejsc po kropce. Natomiast funkcja

```
getch();
```

(ang. *get character* — wczytaj znak) czeka na wczytanie dowolnego znaku z klawiatury (naciśnięcie dowolnego klawisza). Prototyp tej funkcji znajduje się w pliku nagłówkowym *conio.h*. Instrukcja

```
endl;
```

(ang. *end of line* — koniec linii) przenosi kursor na początek następnej linii.

Komentarze w programie oznaczamy dwoma ukośnikami

```
// to jest komentarz do programu
```

Są one ignorowane w procesie kompilacji.

Rezultat działania programu można zobaczyć na rysunku 1.1.

```
Program oblicza pole prostokata.
Podaj bok a.
1
Podaj bok b.
2
Pole prostokata o boku a = 1.00 i boku b = 2.00 wynosi 2.00.
```

Rysunek 1.1. *Efekt działania programu Zadanie 1.1*

ZADANIE

1.2

Napisz program, który wyświetla na ekranie komputera wartość predefiniowanej stałej $\pi = 3{,}14\ldots$ Należy przyjąć format prezentowania tej stałej, oznaczanej w języku C++ jako `M_PI`, z dokładnością pięciu miejsc po kropce.

> **Wskazówka**
>
> Stała `M_PI` znajduje się w pliku nagłówkowym *math.h*, który poleceniem `#include <math.h>` należy dołączyć do programu.

Przykładowe rozwiązanie — listing 1.2

```
#include <iostream.h> // Zadanie 1.2
#include <iomanip.h>
#include <math.h>
#include <conio.h>

main()
{
 cout << "Program wyswietla wartosc predefiniowanej stalej pi" << endl;
 cout << "z dokladnoscia pieciu miejsc po kropce." << endl;
 cout << "pi = " << fixed << setprecision(5) << M_PI << endl;

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 1.2.

```
Program wyswietla wartosc predefiniowanej stalej pi
z dokladnoscia pieciu miejsc po kropce.
pi = 3.14159
```

Rysunek 1.2. *Efekt działania programu Zadanie 1.2*

ZADANIE

1.3 Napisz program, który wyświetla na ekranie komputera pierwiastek kwadratowy z wartości predefiniowanej stałej $\pi = 3{,}14...$ Należy przyjąć format wyświetlania tego pierwiastka z dokładnością dwóch miejsc po kropce.

> **Wskazówka**
>
> Pierwiastek kwadratowy ze stałej π obliczamy, korzystając z funkcji `sqrt()`. Funkcja ta znajduje się w pliku nagłówkowym *math.h*.

Przykładowe rozwiązanie — listing 1.3

```
#include <iostream.h> // Zadanie 1.3
#include <iomanip.h>
#include <math.h>
#include <conio.h>

main()
{
 cout << "Program wyswietla pierwiastek kwadratowy z pi";
 cout << " z dokladnoscia dwoch miejsc po kropce." << endl;
 cout << "Sqrt(pi) = " << fixed << setprecision(2) << sqrt(M_PI) << endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 1.3.

```
Program wyswietla pierwiastek kwadratowy
z pi z dokladnoscia dwoch miejsc po kropce.
Sqrt(pi) = 1.77
```

Rysunek 1.3. *Efekt działania programu Zadanie 1.3*

ZADANIE

1.4 Napisz program, który oblicza objętość kuli o promieniu `r`. Wartość promienia wprowadzamy z klawiatury. W programie należy przyjąć, że `r` jest typu `float` (rzeczywistego). Dla zmiennych `r` oraz `objetosc` należy przyjąć format wyświetlania ich na ekranie z dokładnością dwóch miejsc po kropce.

Przykładowe rozwiązanie — listing 1.4

```
#include <iostream.h> // Zadanie 1.4
#include <iomanip.h>
#include <math.h>
#include <conio.h>

main()
{
  float r, objetosc;

  cout << "Program oblicza objetosc kuli o promieniu r." << endl;
  cout << "Podaj promien r." << endl;
  cin >> r;
  objetosc = 4*M_PI*r*r*r/3;
  cout << fixed;
  cout << setprecision(2);
  cout << "Objetosc kuli o promieniu r = " << r << " wynosi ";
  cout << objetosc << "." << endl;

  getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Objętość kuli o promieniu `r` oblicza linijka kodu

```
objetosc = 4*M_PI*r*r*r/3;
```

gdzie potęgowanie zamieniono na mnożenie.

Rezultat działania programu można zobaczyć na rysunku 1.4.

```
Program oblicza objetosc kuli o promieniu r.
Podaj promien r.
1
Objetosc kuli o promieniu r = 1.00 wynosi 4.19.
```

Rysunek 1.4. *Efekt działania programu Zadanie 1.4*

ZADANIE

1.5 Napisz program, który oblicza wynik dzielenia całkowitego bez reszty dla dwóch liczb całkowitych a = 37 i b = 11.

> **Wskazówka**
>
> W języku C++ w przypadku zastosowania operatora dzielenia / dla liczb całkowitych reszta wyniku jest pomijana[3].

Przykładowe rozwiązanie — listing 1.5

```
#include <iostream.h> // Zadanie 1.5
#include <conio.h>

main()
{
 int a = 37;
 int b = 11;

 cout << "Program oblicza wynik dzielenia calkowitego" << endl;
 cout << "dla dwoch liczb calkowitych." << endl;
 cout << "Dla liczb a = " << a << " i b = " << b << endl;
 cout << a << "/" << b << " = " << a/b << "." << endl;

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 1.5.

```
Program oblicza wynik dzielenia calkowitego
dla dwoch liczb calkowitych.
Dla liczb a = 37 i b = 11
37/11 = 3.
```

Rysunek 1.5. *Efekt działania programu Zadanie 1.5*

[3] W języku Turbo Pascal należy zastosować operator dzielenia całkowitego bez reszty `div`.

ZADANIE

1.6

Napisz program, który oblicza resztę z dzielenia całkowitego dla dwóch liczb całkowitych a = 37 i b = 11.

Wskazówka

Należy zastosować operator reszty z dzielenia całkowitego modulo, który oznaczamy w języku C++ symbolem %. Operator ten umożliwia uzyskanie tylko reszty z dzielenia, natomiast całkowita wartość liczbowa jest odrzucana.

Przykładowe rozwiązanie — listing 1.6

```
#include <iostream.h> // Zadanie 1.6
#include <conio.h>

main()
{
 int a = 37;
 int b = 11;

 cout << "Program oblicza reszte z dzielenia calkowitego";
 cout << " dwoch liczb calkowitych." << endl;
 cout << "Dla liczb a = " << a << " i b = " << b << endl;
 cout << a << "%" << b << " = " << a%b << "." << endl;

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 1.6.

```
Program oblicza reszte z dzielenia calkowitego dwoch liczb
calkowitych.
Dla liczb a = 37 i b = 11
37%11 = 4.
```

Rysunek 1.6. *Efekt działania programu Zadanie 1.6*

ZADANIE

1.7 Napisz program, który oblicza sumę, różnicę, iloczyn i iloraz dla dwóch liczb `x` i `y` wprowadzanych z klawiatury. W programie przyjmujemy, że liczby `x` i `y` są typu `float` (rzeczywistego). Dla zmiennych `x`, `y`, `suma`, `roznica`, `iloczyn` i `iloraz` należy przyjąć format wyświetlania ich na ekranie z dokładnością dwóch miejsc po kropce.

Przykładowe rozwiązanie — listing 1.7

```
#include <iostream.h> // Zadanie 1.7
#include <iomanip.h>
#include <conio.h>

main()
{
 float x, y, suma, roznica, iloczyn, iloraz;

 cout << "Program oblicza sume, roznice, iloczyn i iloraz" << endl;
 cout << "dla dwoch liczb x i y wprowadzanych z klawiatury." << endl;
 cout << endl;
 cout << "Podaj x." << endl;
 cin >> x;
 cout << "Podaj y." << endl;
 cin >> y;

 suma = x+y;
 roznica = x-y;
 iloczyn = x*y;
 iloraz = x/y;

 cout << fixed;
 cout << setprecision (2);
 cout << "Dla x = " << x << " i y = " << y << endl;
 cout << endl; // wydruk pustej linii
 cout << "suma = " << suma << "," << endl;
 cout << "roznica = " << roznica << "," << endl;
 cout << "iloczyn = " << iloczyn << "," << endl;
 cout << "iloraz = " << iloraz << ".";

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 1.7.

```
Program oblicza sume, roznice, iloczyn i iloraz
dla dwoch liczb x i y wprowadzanych z klawiatury.

Podaj x.
1
Podaj y.
2
Dla x = 1.00 i y = 2.00

suma = 3.00,
roznica = -1.00,
iloczyn = 2.00,
iloraz = 0.50.
```

Rysunek 1.7. *Efekt działania programu Zadanie 1.7*

2

Podejmujemy decyzje w programie

W tym rozdziale przedstawimy typowe zadania wraz z przykładowymi rozwiązaniami z wykorzystaniem instrukcji warunkowych.

W języku C++ istnieją dwie instrukcje warunkowe:

- instrukcja warunkowa `if ... else`,
- instrukcja wyboru `switch ... case`.

Instrukcja warunkowa `if ... else` służy do sprawdzania poprawności wyrażenia warunkowego i w zależności od tego, czy dany warunek jest prawdziwy, czy nie, pozwala na wykonanie różnych bloków programu.

Jej ogólna postać jest następująca:

```
if (warunek)
{
   // instrukcje do wykonania, kiedy warunek jest prawdziwy
}
else
{
   // instrukcje do wykonania, kiedy warunek jest fałszywy
}
```

Blok `else` jest opcjonalny i instrukcja warunkowa w wersji skróconej ma postać

```
if (warunek)
{
   // instrukcje do wykonania, kiedy warunek jest prawdziwy
}
```

Jeżeli *warunek* nie jest prawdziwy, to instrukcja warunkowa `if` nie zostanie wykonana.

Instrukcja wyboru `switch ... case` pozwala w wygodny i przejrzysty sposób sprawdzić ciąg warunków i wykonywać kod w zależności od tego, czy są one prawdziwe, czy fałszywe. Jej ogólna postać jest następująca:

```
switch (wyrażenie)
{
   case wartość_1 : instrukcje_1;
   break;
   case wartość_2 : instrukcje_2;
   break;

   ......................
   case wartość_n : instrukcje_n;
   break;
   default : instrukcje;
}
```

Instrukcja `break` przerywa wykonywanie całego bloku `case`. **UWAGA:** jej brak może doprowadzić do nieoczekiwanych wyników i błędów w programie.

ZADANIE

2.1 Napisz program, który dla trzech boków trójkąta `a`, `b` i `c` wprowadzonych z klawiatury sprawdza, czy tworzą one trójkąt prostokątny (zakładamy, że `a > 0`, `b > 0`, `c > 0`).

Przykładowe rozwiązanie — listing 2.1

```
#include <iostream.h> // Zadanie 2.1
#include <conio.h>

main()
{
 int a, b, c;

 cout << "Program sprawdza, czy boki a, b oraz c tworza trojkat
 ↪prostokatny." << endl;
```

```
 cout << "Podaj bok a." << endl;
 cin >> a;
 cout << "Podaj bok b." << endl;
 cin >> b;
 cout << "Podaj bok c." << endl;
 cin >> c;

 if ((a*a+b*b) == (c*c))
 {
  cout << "Boki a = " << a << ", b = " << b << " i c = " << c;
  cout << " tworza trojkat prostokatny." << endl;
 }
 else
 {
  cout << "Boki a = " << a << ", b = " << b << " i c = " << c;
  cout << " nie tworza trojkata prostokatnego." << endl;
 }

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Sprawdzenie twierdzenia Pitagorasa dla wczytanych boków `a`, `b` i `c` zostało zawarte w następujących linijkach kodu:

```
if ((a*a+b*b) == (c*c))
 {
  cout << "Boki a = " << a << ", b = " << b << " i c = " << c;
  cout << " tworza trojkat prostokatny." << endl;
 }
 else
 {
  cout << "Boki a = " << a << ", b = " << b << " i c = " << c;
  cout << " nie tworza trojkata prostokatnego." << endl;
 }
```

Łatwo sprawdzić, że boki `a = 3`, `b = 4`, `c = 5` tworzą trójkąt prostokątny (liczby te spełniają twierdzenie Pitagorasa), i na ekranie pojawi się komunikat *Boki… tworza trojkat prostokatny*, natomiast boki `a = 1`, `b = 2`, `c = 3` nie tworzą trójkąta prostokątnego (liczby te nie spełniają twierdzenia Pitagorasa), więc na ekranie zostanie wyświetlony komunikat *Boki… nie tworza trojkata prostokatnego*.

Rezultat działania programu dla `a = 1`, `b = 2`, `c = 3` można zobaczyć na rysunku 2.1.

Program sprawdza, czy boki a, b oraz c tworza trojkat prostokatny.
Podaj bok a.
1
Podaj bok b.
2
Podaj bok c.
3
Boki a = 1, b = 2 i c = 3 nie tworza trojkata prostokatnego.

Rysunek 2.1. *Efekt działania programu Zadanie 2.1*

ZADANIE

2.2 Napisz program, który z wykorzystaniem instrukcji warunkowej `if` oblicza pierwiastki równania kwadratowego $ax^2+bx+c = 0$, gdzie zmienne `a`, `b`, `c` to liczby rzeczywiste wprowadzane z klawiatury. Dla zmiennych `a`, `b`, `c`, `x1` oraz `x2` należy przyjąć format wyświetlania ich z dokładnością dwóch miejsc po kropce.

Przykładowe rozwiązanie — listing 2.2

```
#include <iostream.h> // Zadanie 2.2
#include <iomanip.h>
#include <math.h>
#include <conio.h>

main()
{
 float a, b, c, delta, x1, x2;

 cout << "Program oblicza pierwiastki rownania kwadratowego";
 cout << " dla dowolnych wspolczynnikow a, b, c." << endl;
 cout << "Podaj a." << endl;
 cin >> a;

 if (a == 0)
 {
  cout << "Niedozwolona wartosc wspolczynnika a. Nacisnij dowolny klawisz.";
 }
 else
 {
  cout << "Podaj b." << endl;
  cin >> b;
  cout << "Podaj c." << endl;
  cin >> c;
  cout << fixed;
  cout << setprecision(2);
```

```
 cout << "Dla wprowadzonych liczb:" << endl;
 cout << "a = " << a << "," << endl;
 cout << "b = " << b << "," << endl;
 cout << "c = " << c << "," << endl;

 delta = b*b-4*a*c;

if (delta < 0)
{
 cout << "brak pierwiastkow rzeczywistych." << endl;
}
else
 {
 if (delta == 0)
  {
   x1 = -b/(2*a);
   cout << "trojmian ma jeden pierwiastek podwojny x1 = " << x1 << "."
   ↪<< endl;
  }
 else
  {
   x1 = (-b-sqrt(delta))/(2*a);
   x2 = (-b+sqrt(delta))/(2*a);
   cout << "trojmian ma dwa pierwiastki:" << endl;
   cout << "x1 = " << x1 << "," << endl;
   cout << "x2 = " << x2 << "." << endl;
  }
 }
}

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

W pierwszej części programu sprawdzamy, czy wartość współczynnika `a` jest równa zero. Ilustrują to następujące linijki kodu:

```
if (a == 0)
{
   cout << "Niedozwolona wartosc wspolczynnika a. Nacisnij dowolny
   ↪klawisz.";
}
else
{
  .....
}
```

Jeśli `a = 0`, to zostanie wyświetlony komunikat *Niedozwolona wartosc wspolczynnika a…* i program zostanie zakończony. Dla `a` różnego od zera program będzie oczekiwał na wprowadzenie wartości `b` i `c`. Po ich wpisaniu zostanie obliczona `delta` według wzoru

```
delta = b*b-4*a*c;
```

Jeśli `delta < 0`, to zostanie wyświetlony komunikat *brak pierwiastkow rzeczywistych*.

W przypadku gdy `delta = 0`, równanie kwadratowe ma jeden pierwiastek podwójny, który obliczymy ze wzoru

```
x1 = -b/(2*a);
```

Jeśli `delta > 0` równanie ma dwa pierwiastki, które zostaną obliczone przy użyciu wzorów

```
x1 = (-b-sqrt(delta))/(2*a);
x2 = (-b+sqrt(delta))/(2*a);
```

Przykładowo dla `a = 1`, `b = 5` i `c = 4` wartości pierwiastków równania wynoszą odpowiednio `x1 = -4` i `x2 = -1`.

Dla `a = 1`, `b = 4` i `c = 4` trójmian ma z kolei jeden pierwiastek podwójny `x1 = -2`.

Dla `a = 1`, `b = 2` oraz `c = 3` trójmian nie ma pierwiastków rzeczywistych.

Rezultat działania programu dla `a = 1`, `b = 5`, `c = 4` można zobaczyć na rysunku 2.2.

Program oblicza pierwiastki rownania kwadratowego dla dowolnych wspolczynnikow a, b, c.
Podaj a.
1
Podaj b.
5
Podaj c.
4
Dla wprowadzonych liczb:
a = 1.00,
b = 5.00,
c = 4.00,
trojmian ma dwa pierwiastki:
x1 = -4.00,
x2 = -1.00.

Rysunek 2.2. *Efekt działania programu Zadanie 2.2*

ZADANIE

2.3 Napisz program, który z wykorzystaniem instrukcji wyboru `switch` oblicza pierwiastki równania kwadratowego $ax^2+bx+c = 0$, gdzie zmienne `a`, `b`, `c` to liczby rzeczywiste wprowadzane z klawiatury. Dla zmiennych `a`, `b`, `c`, `x1` oraz `x2` należy przyjąć format wyświetlania ich na ekranie z dokładnością dwóch miejsc po kropce.

> **Wskazówka**
>
> Należy wprowadzić do programu zmienną pomocniczą `liczba_pierwiastkow`.

Przykładowe rozwiązanie — listing 2.3

```
#include <iostream.h> // Zadanie 2.3
#include <iomanip.h>
#include <math.h>
#include <conio.h>

main()
{
 float a, b, c, delta, x1, x2;
 char liczba_pierwiastkow;

 cout << "Program oblicza pierwiastki rownania kwadratowego";
 cout << " dla dowolnych wspolczynnikow a, b, c." << endl;
 cout << "Podaj a." << endl;
 cin >> a;

 if (a == 0)
  {
   cout << "Niedozwolona wartosc wspolczynnika a. Nacisnij dowolny
   ↪klawisz.";
  }
 else
  {
   cout << "Podaj b." << endl;
   cin >> b;
   cout << "Podaj c." << endl;
   cin >> c;

   cout << fixed;
   cout << setprecision(2);
   cout << "Dla wprowadzonych liczb:" << endl;
   cout << "a = " << a << "," << endl;
   cout << "b = " << b << "," << endl;
   cout << "c = " << c << "," << endl;

   delta = b*b-4*a*c;
```

```
    if (delta < 0) liczba_pierwiastkow = 0;
    if (delta == 0) liczba_pierwiastkow = 1;
    if (delta > 0) liczba_pierwiastkow = 2;

    switch (liczba_pierwiastkow)
    {
     case 0 : cout << "brak pierwiastkow rzeczywistych." << endl;
     break;
     case 1 : { x1 = -b/(2*a);
      cout << "trojmian ma jeden pierwiastek podwojny";
      cout << " x1 = " << x1 << "." << endl;
     }
     break;
     case 2 : { x1 = (-b-sqrt(delta))/(2*a);
      x2 = (-b+sqrt(delta))/(2*a);
      cout << "trojmian ma dwa pierwiastki:" << endl;
      cout << "x1 = " << x1 << "," << endl;
      cout << "x2 = " << x2 << "." << endl;
     }
     break;
    }
   }

  getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu dla `a = 1`, `b = 4`, `c = 4` można zobaczyć na rysunku 2.3.

Program oblicza pierwiastki rownania kwadratowego dla dowolnych wspolczynnikow a, b, c.
Podaj a.
1
Podaj b.
4
Podaj c.
4
Dla wprowadzonych liczb:
a = 1.00,
b = 4.00,
c = 4.00,
trojmian ma jeden pierwiastek podwojny x1 = -2.00.

Rysunek 2.3. *Efekt działania programu Zadanie 2.3*

ZADANIE

2.4 Napisz program, który oblicza wartość x z równania ax+b = c. Wartości a, b i c należą do zbioru liczb rzeczywistych i są wprowadzane z klawiatury. Dodatkowo należy zabezpieczyć program na wypadek sytuacji, kiedy wprowadzona wartość a jest równa zero. Dla zmiennych a, b, c oraz x należy przyjąć format wyświetlania ich z dokładnością dwóch miejsc po kropce.

Przykładowe rozwiązanie — listing 2.4

```
#include <iostream.h> // Zadanie 2.4
#include <iomanip.h>
#include <conio.h>

main()
{
 float a, b, c, x;

 cout << "Program oblicza wartosc x z rownania liniowego ax+b = c." << endl;
 cout << "Podaj a." << endl;
 cin >> a;

 if (a == 0)
 {
  cout << "Niedozwolona wartosc wspolczynnika. Nacisnij dowolny klawisz.";
 }
 else
 {
 cout << "Podaj b." << endl;
 cin >> b;
 cout << "Podaj c." << endl;
 cin >> c;

 x = (c-b)/a;

 cout << fixed;
 cout << setprecision(2);
 cout << "Dla a = " << a << ", b = " << b << ", c = " << c;
 cout << " wartosc x = " << x << "." << endl;
 }

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 2.4.

```
Program oblicza wartosc x z rownania liniowego ax+b = c.
Podaj a.
1
Podaj b.
2
Podaj c.
3
Dla a = 1.00, b = 2.00, c = 3.00 wartosc x = 1.00.
```

Rysunek 2.4. *Efekt działania programu Zadanie 2.4*

ZADANIE

2.5

Napisz program, w którym użytkownik zgaduje liczbę losową z przedziału od 0 do 9 generowaną przez komputer.

> **Wskazówka**
>
> W programie należy zastosować instrukcję (makro) `random()`.

Przykładowe rozwiązanie — listing 2.5

```
#include <iostream.h> // Zadanie 2.5
#include <math.h>
#include <conio.h>

main()
{
 int losuj_liczbe, zgadnij_liczbe;

 cout << "Program losuje liczbe od 0 do 9. Zgadnij ja." << endl;
 randomize();
 losuj_liczbe = random(10);
 cin >> zgadnij_liczbe;

 if (zgadnij_liczbe == losuj_liczbe)
 {
  cout << "Gratulacje! Zgadles liczbe!" << endl;
 }
 else
 {
```

```
  cout << "Bardzo mi przykro, ale wylosowana liczba to: " <<
  ↪losuj_liczbe << "." << endl;
 }

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Za losowanie liczby przez komputer odpowiadają w programie następujące linijki kodu:

```
randomize();
losuj_liczbe = random(10);
```

Instrukcja (makro) `random(10)` zwraca liczbę pseudolosową z zakresu od 0 do 9 (tj. 10–1).

Instrukcja (makro) `randomize()` inicjalizuje generator liczb pseudolosowych (tzn. ustala jego wartość początkową).

Rezultat działania programu można zobaczyć na rysunku 2.5.

```
Program losuje liczbe od 0 do 9. Zgadnij ja.
1
Bardzo mi przykro, ale wylosowana liczba to: 4.
```

Rysunek 2.5. *Efekt działania programu Zadanie 2.5*

3

Iteracje

W tym rozdziale przedstawimy typowe zadania wraz z przykładowymi rozwiązaniami z wykorzystaniem iteracji, czyli popularnych pętli. O ile młodzi programiści nie mają problemu z programami, w których wykorzystano instrukcję `for`*, to zamiana jej na* `do ... while` *oraz* `while` *powoduje pewne trudności. Proste przykłady z użyciem instrukcji* `for` *zostały więc rozwiązane zarówno z instrukcją* `do ... while`*, jak i* `while`*.*

Iteracja (łac. *iteratio* — powtarzanie) to czynność powtarzania (najczęściej wielokrotnego) tej samej albo wielu różnych instrukcji w pętli.

W języku C++ istnieją trzy instrukcje iteracyjne:

- `for` (dla),
- `do ... while` (powtarzaj),
- `while` (dopóki).

Pętlę `for` stosujemy w sytuacji, kiedy dokładnie wiemy, ile razy ma ona zostać wykonana. Istnieje wiele wariantów tej pętli, ale zawsze możemy wyróżnić trzy główne części.

1. **Inicjalizacja** to zwykle instrukcja przypisania stosowana do ustawienia początkowej wartości zmiennej sterującej pętlą.
2. **Warunek** jest wyrażeniem relacyjnym określającym moment zakończenia wykonywania pętli.
3. **Inkrementacja** (zwiększanie) lub **dekrementacja** (zmniejszanie) definiuje sposób modyfikacji zmiennej sterującej pętlą po zakończeniu każdego przebiegu (powtórzenia).

Te trzy główne składowe oddzielone są od siebie średnikami.

Pętla `for` wykonywana jest tak długo, dopóki wartość warunku wynosi `true`. Gdy warunek osiągnie wartość `false`, działanie programu jest kontynuowane od pierwszej instrukcji znajdującej się za pętlą.

W przeciwieństwie do Turbo Pascala w języku C++ zmienna sterująca pętlą `for` nie musi być typu całkowitego, znakowego lub logicznego. Może być ona np. typu `float`.

Pętla ta może być wykonywana tyle razy, ile wartości znajduje się w przedziale

```
inicjalizacja; warunek; zwiększanie
```

lub

```
inicjalizacja; warunek; zmniejszanie
```

Ogólna postać tej instrukcji jest następująca:

```
for (inicjalizacja; warunek; zwiększanie)
{
   // instrukcje
}
```

lub

```
for (inicjalizacja; warunek; zmniejszanie)
{
   // instrukcje
}
```

W języku C++ jest możliwa zmiana przyrostu zmiennej sterującej pętlą.

Kolejną instrukcją iteracyjną jest `do ... while`. Jej ogólna postać jest następująca:

```
do
{
   // instrukcje
}
while (warunek);
```

Cechą charakterystyczną instrukcji iteracyjnej `do ... while` jest to, że bez względu na wartość zmiennej *warunek* pętla musi zostać wykonana co najmniej jeden raz. Program po napotkaniu instrukcji `do ... while` wchodzi do pętli i wykonuje instrukcje znajdujące się w nawiasach klamrowych {}, a następnie sprawdza, czy warunek jest spełniony.

Jeśli tak, wraca na początek pętli, natomiast jeśli warunek osiągnie wartość `false` (nieprawda), pętla się zakończy.

Ostatnią instrukcją iteracyjną jest `while`. Jej ogólna postać jest następująca:

```
while (warunek)
{
  // instrukcje
}
```

Cechą charakterystyczną tej instrukcji jest sprawdzenie warunku jeszcze przed jej wykonaniem. W szczególnym przypadku pętla może nie zostać wcale wykonana. Instrukcja `while` powoduje wykonywanie instrukcji tak długo, dopóki warunek jest prawdziwy.

ZADANIE

3.1 Napisz program, który za pomocą instrukcji `for` dla danych wartości `x` zmieniających się od 0 do 10 oblicza wartość funkcji `y = 3x`.

Przykładowe rozwiązanie — listing 3.1

```
#include <iostream.h> // Zadanie 3.1
#include <conio.h>

main()
{
int x, y;

cout << "Program oblicza wartosc funkcji y = 3x dla x zmieniajacego sie
➥od 0 do 10." << endl;

for (x = 0; x <= 10; x++)
{
 y = 3*x;
 cout << "x = " << x << '\t' << "y = " << y << endl;
}

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

W pętli

```
for (x = 0; x <= 10; x++)
```

kolejne wartości `x`, zmieniające się automatycznie od 0 (inicjalizacja) do 10 (warunek) z krokiem 1 (zwiększanie), będą podstawiane do wzoru

```
y = 3*x;
```

a następnie zostaną wyświetlone na ekranie. Znak `'\t'` oznacza przejście do następnej pozycji w tabulacji linii.

Rezultat działania programu można zobaczyć na rysunku 3.1.

```
Program oblicza wartosc funkcji y = 3x dla x zmieniajacego
sie od 0 do 10.
x = 0   y = 0
x = 1   y = 3
x = 2   y = 6
x = 3   y = 9
x = 4   y = 12
x = 5   y = 15
x = 6   y = 18
x = 7   y = 21
x = 8   y = 24
x = 9   y = 27
x = 10  y = 30
```

Rysunek 3.1. *Efekt działania programu Zadanie 3.1*

ZADANIE

3.2 Napisz program, który za pomocą instrukcji `do ... while` dla danych wartości `x` zmieniających się od 0 do 10 oblicza wartość funkcji `y = 3x`.

Przykładowe rozwiązanie — listing 3.2

```
#include <iostream.h> // Zadanie 3.2
#include <conio.h>

main()
{
int x = 0, y = 0;

cout << "Program oblicza wartosc funkcji y = 3x dla x zmieniajacego sie
↪od 0 do 10." << endl;
```

```
do
{
 y = 3*x;
 cout << "x = " << x << '\t' << "y = " << y << endl;
 x++;
}
while (x <= 10);

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Pętla `do ... while`

```
do
{
 y = 3*x;
 cout << "x = " << x << '\t' << "y = " << y << endl;
 x++;
}
while (x <= 10);
```

nie posiada wbudowanego mechanizmu zmiany sterującej nią zmiennej, dlatego musimy do niej ten mechanizm dobudować. Rolę zmiennej sterującej pełni tutaj `x`. Zmienną tą powinniśmy przed pętlą wyzerować, stąd zapis

```
x = 0;
```

Następnie `x` należy zwiększać o krok, który w naszym przypadku wynosi 1. Ilustruje to następująca linijka kodu:

```
x++;
```

Pętla będzie powtarzana tak długo, aż zostanie spełniona zależność `x <= 10`. Zwróćmy uwagę, że warunek sprawdzający zakończenie działania pętli, tzn. `while (x <= 10)`, znajduje się na jej końcu.

ZADANIE

3.3 Napisz program, który za pomocą instrukcji `while` dla danych wartości `x` zmieniających się od 0 do 10 oblicza wartość funkcji `y = 3x`.

Przykładowe rozwiązanie — listing 3.3

```
#include <iostream.h> // Zadanie 3.3
#include <conio.h>

main()
{
```

```
int x = 0, y = 0;

cout << "Program oblicza wartosc funkcji y = 3x dla x zmieniajacego sie
↪od 0 do 10." << endl;

while (x <= 10)
{
 y = 3*x;
 cout << "x = " << x << '\t' << "y = " << y << endl;
 x++;
}

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Pętla `while`, podobnie jak `do ... while`, nie posiada wbudowanego mechanizmu zmiany sterującej nią zmiennej, musimy więc do niej ten mechanizm dobudować. Rolę zmiennej sterującej pełni tutaj zmienna `x`.

```
while (x <= 10)
{
 y = 3*x;
 cout << "x = " << x << '\t' << "y = " << y << endl;
 x++;
}
```

Zmienną `x` powinniśmy przed pętlą wyzerować, zatem

```
x = 0;
```

Następnie należy ją zwiększać o krok, który w naszym przypadku wynosi 1. Ilustruje to następująca linijka kodu:

```
x++;
```

Pętla będzie powtarzana tak długo, aż stanie się prawdziwa zależność `x <= 10`. Zwróćmy uwagę, że warunek sprawdzający zakończenie działania pętli, tzn. `while (x <= 10)`, znajduje się na jej początku.

ZADANIE

3.4 Napisz program, który za pomocą instrukcji `for` wyświetla liczby całkowite od 1 do 20.

Przykładowe rozwiązanie — listing 3.4

```
#include <iostream.h> // Zadanie 3.4
#include <conio.h>
```

```
main()
{
int i;

cout << "Program wyswietla liczby calkowite od 1 do 20." << endl;

for (i = 1; i <= 20; i++)
{
if (i < 20)
{
 cout << i << ", ";
}
else
{
 cout << i << ".";
}
}

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 3.2.

```
Program wyswietla liczby calkowite od 1 do 20.
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20.
```

Rysunek 3.2. *Efekt działania programu Zadanie 3.4*

ZADANIE

3.5 Napisz program, który za pomocą instrukcji `do ... while` wyświetla liczby całkowite od 1 do 20.

Przykładowe rozwiązanie — listing 3.5

```
#include <iostream.h> // Zadanie 3.5
#include <conio.h>

main()
{
int i = 1; // ustalenie wartosci poczatkowej

cout << "Program wyswietla liczby calkowite od 1 do 20." << endl;

do
{
 if (i < 20)
```

```
 {
  cout << i << ", ";
 }
 else
 {
  cout << i << ".";
 }
 i++;
}
while (i <= 20);

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

ZADANIE

3.6 Napisz program, który za pomocą instrukcji `while` wyświetla liczby całkowite od 1 do 20.

Przykładowe rozwiązanie — listing 3.6

```
#include <iostream.h> // Zadanie 3.6
#include <conio.h>

main()
{
int i = 1; // ustalenie wartosci poczatkowej

cout << "Program wyswietla liczby calkowite od 1 do 20." << endl;

while (i <= 20)
{
if (i < 20)
{
 cout << i << ", ";
}
else
{
 cout << i << ".";
}
i++;
}

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

ZADANIE

3.7 Napisz program, który za pomocą instrukcji `for` sumuje liczby całkowite od 1 do 100.

Przykładowe rozwiązanie — listing 3.7

```
#include <iostream.h> // Zadanie 3.7
#include <conio.h>

main()
{
int i, suma = 0;

cout << "Program sumuje liczby calkowite od 1 do 100." << endl;

for (i = 1; i <= 100; i++)
{
 suma = suma+i;
}
cout << "Suma liczb calkowitych od 1 do 100 wynosi " << suma << "." <<
↪endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Za sumowanie liczb całkowitych od 1 do 100 odpowiedzialne są następujące linijki kodu:

```
for (i = 1; i <= 100; i++)
{
 suma = suma+i;
}
```

Oczywiście przed pętlą zmienna `suma` musi zostać wyzerowana, stąd zapis

```
suma = 0;
```

Rezultat działania programu można zobaczyć na rysunku 3.3.

```
Program sumuje liczby calkowite od 1 do 100.
Suma liczb calkowitych od 1 do 100 wynosi 5050.
```

Rysunek 3.3. *Efekt działania programu Zadanie 3.7*

ZADANIE

3.8 Napisz program, który za pomocą instrukcji `do ... while` sumuje liczby całkowite od 1 do 100.

Przykładowe rozwiązanie — listing 3.8

```
#include <iostream.h> // Zadanie 3.8
#include <conio.h>

main()
{
int i = 1, suma = 0; // ustalenie wartosci poczatkowych

cout << "Program sumuje liczby calkowite od 1 do 100." << endl;

do
{
 suma = suma+i;
 i++;
}
while (i <= 100);

cout << "Suma liczb calkowitych od 1 do 100 wynosi " << suma << "." <<
↪endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

ZADANIE

3.9 Napisz program, który za pomocą instrukcji `while` sumuje liczby całkowite od 1 do 100.

Przykładowe rozwiązanie — listing 3.9

```
#include <iostream.h> // Zadanie 3.9
#include <conio.h>

main()
{
int i = 1, suma = 0; // ustalenie wartosci poczatkowych

cout << "Program sumuje liczby calkowite od 1 do 100." << endl;

while (i <= 100)
{
 suma = suma+i;
```

```
 i++;
}

cout << "Suma liczb calkowitych od 1 do 100 wynosi " << suma << "." << endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

ZADANIE

3.10 Napisz program, który za pomocą instrukcji `for` sumuje liczby parzyste w przedziale od 1 do 100.

> **Wskazówka**
>
> Należy skorzystać z właściwości operatora modulo %.

Przykładowe rozwiązanie — listing 3.10

```
#include <iostream.h> // Zadanie 3.10
#include <conio.h>

main()
{
int i, suma = 0;

cout << "Program sumuje liczby parzyste w przedziale od 1 do 100." <<
➥endl;

for (i = 1; i <= 100; i++)
{
if (i%2 == 0)
 suma = suma+i;
}
cout << "Suma liczb parzystych z przedzialu od 1 do 100 wynosi " << suma
➥<< "." << endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Za sumowanie liczb parzystych z przedziału od 1 do 100 odpowiedzialne są następujące linijki kodu:

```
for (i = 1; i <= 100; i++)
{
if (i%2 == 0)
 suma = suma+i;
}
```

Do wyodrębnienia liczb parzystych wykorzystaliśmy właściwości operatora modulo oznaczonego symbolem %. Użyliśmy w tym celu zapisu `i%2 == 0` — jeśli reszta z dzielenia całkowitego zmiennej `i%2` wynosi zero, to mamy do czynienia z liczbą parzystą, którą dodajemy do zmiennej `suma`.

Rezultat działania programu można zobaczyć na rysunku 3.4.

```
Program sumuje liczby parzyste w przedziale od 1 do 100.
Suma liczb parzystych z przedzialu od 1 do 100 wynosi 2550.
```

Rysunek 3.4. *Efekt działania programu Zadanie 3.10*

ZADANIE

3.11

Napisz program, który za pomocą instrukcji `do ... while` sumuje liczby parzyste w przedziale od 1 do 100.

Wskazówka

Należy skorzystać z właściwości operatora modulo %.

Przykładowe rozwiązanie — listing 3.11

```
#include <iostream.h> // Zadanie 3.11
#include <conio.h>

main()
{
int i, suma = 0;

cout << "Program sumuje liczby parzyste w przedziale od 1 do 100." <<
➥endl;

do
{
if (i%2 == 0)
 suma = suma+i;
 i++;
}
while (i <= 100);

cout << "Suma liczb parzystych z przedzialu od 1 do 100 wynosi " << suma
➥<< "." << endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

ZADANIE

3.12 Napisz program, który za pomocą instrukcji `while` sumuje liczby parzyste w przedziale od 1 do 100.

> **Wskazówka**
>
> Należy skorzystać z właściwości operatora modulo %.

Przykładowe rozwiązanie — listing 3.12

```
#include <iostream.h> // Zadanie 3.12
#include <conio.h>

main()
{
int i, suma = 0;

cout << "Program sumuje liczby parzyste w przedziale od 1 do 100."
↪<< endl;

while (i <= 100)
{
if (i%2 == 0)
 suma = suma+i;
 i++;
}

cout << "Suma liczb parzystych z przedzialu od 1 do 100 wynosi "
↪<< suma << "." << endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

ZADANIE

3.13 Napisz program, który za pomocą instrukcji `for` sumuje liczby nieparzyste z przedzialu od 1 do 100.

> **Wskazówka**
>
> Należy skorzystać z właściwości operatora modulo % i oznaczonego symbolem *!* operatora negacji.

Przykładowe rozwiązanie — listing 3.13

```
#include <iostream.h> // Zadanie 3.13
#include <conio.h>
```

```
main()
{
int i, suma = 0;

cout << "Program sumuje liczby nieparzyste w przedziale od 1 do 100."
↪<< endl;

for (i = 1; i <= 100; i++)
{
if (!(i%2 == 0))
 suma = suma+i;
}
cout << "Suma liczb nieparzystych z przedzialu od 1 do 100 wynosi " <<
↪suma << "." << endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Za sumowanie liczb nieparzystych od 1 do 100 odpowiedzialne są następujące linijki kodu:

```
for (i = 1; i <= 100; i++)
{
if (!(i%2 == 0))
 suma = suma+i;
}
```

Do wyodrębnienia liczb nieparzystych wykorzystaliśmy właściwości operatora modulo oznaczonego symbolem % oraz właściwości oznaczonego znakiem *!* operatora negacji. Drugi z nich przekształca warunek prawdziwy w fałszywy, a fałszywy w prawdziwy. Zapis `(!(i%2 == 0))` oznacza, że reszta z dzielenia całkowitego zmiennej `i%2` jest różna od zera, a więc mamy do czynienia z liczbą nieparzystą, którą dodajemy do zmiennej `suma`.

Rezultat działania programu można zobaczyć na rysunku 3.5.

```
Program sumuje liczby nieparzyste w przedziale od 1 do 100.
Suma liczb nieparzystych z przedzialu od 1 do 100 wynosi 2500.
```

Rysunek 3.5. *Efekt działania programu Zadanie 3.13*

ZADANIE

3.14 Napisz program, który za pomocą instrukcji `do ... while` sumuje liczby nieparzyste w przedziale od 1 do 100.

> **Wskazówka**
>
> Należy skorzystać z właściwości operatora modulo % i operatora negacji *!*.

Przykładowe rozwiązanie — listing 3.14

```
#include <iostream.h> // Zadanie 3.14
#include <conio.h>

main()
{
int i, suma = 0;

cout << "Program sumuje liczby nieparzyste w przedziale od 1 do 100."
↪<< endl;

do
{
if (!(i%2 == 0))
 suma = suma+i;
 i++;
}
while (i <= 100);

cout << "Suma liczb nieparzystych z przedzialu od 1 do 100 wynosi " <<
↪suma << "." << endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

ZADANIE

3.15 Napisz program, który za pomocą instrukcji `while` sumuje liczby nieparzyste w przedziale od 1 do 100.

> **Wskazówka**
>
> Należy skorzystać z właściwości operatora modulo % i operatora negacji *!*.

Przykładowe rozwiązanie — listing 3.15

```
#include <iostream.h> // Zadanie 3.15
#include <conio.h>
```

```
main()
{
int i, suma = 0;

cout << "Program sumuje liczby nieparzyste w przedziale od 1 do 100." <<
↪endl;

while (i <= 100)
{
if (!(i%2 == 0))
 suma = suma+i;
 i++;
}

cout << "Suma liczb nieparzystych z przedzialu od 1 do 100 wynosi " <<
↪suma << "." << endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

ZADANIE

3.16 Napisz program, który za pomocą instrukcji `for` znajduje największą i najmniejszą liczbę ze zbioru `n` liczb losowych z przedziału od 0 do 99 oraz oblicza ich średnią (w zadaniu `n = 5`).

Przykładowe rozwiązanie — listing 3.16

```
#include <iostream.h> // Zadanie 3.16
#include <math.h>
#include <iomanip.h>
#include <conio.h>

main()
{
const
ilosc_liczb = 5;

int i;
float liczba, suma, min, max;

cout << "Program losuje " << ilosc_liczb << " liczb z przedzialu od 0 do
↪99," << endl;
cout << "a nastepnie znajduje najmniejsza i najwieksza oraz" << endl;
cout << "oblicza srednia ze wszystkich wylosowanych liczb." << endl;

suma = 0;
randomize();
```

```
min = random(100);
cout << endl;
cout << "Wylosowano liczby: " << min << ", ";
max = min;
suma = suma+max;

for (i = 1; i <= ilosc_liczb-1; i++)
{
liczba = random(100);

if (i <= ilosc_liczb-2)
{
 cout << liczba << ", ";
}
else
{
 cout << liczba << ".";
}

if (max < liczba) max = liczba;
if (liczba < min) min = liczba;

suma = suma+liczba;
}

cout << endl;
cout << "Najwieksza liczba to " << max << "." << endl;
cout << "Najmniejsza liczba to " << min << "." << endl;
cout << "Srednia wynosi " << fixed << setprecision(2) <<
suma/ilosc_liczb << "." << endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

W pierwszej kolejności w programie losujemy liczbę i przypisujemy jej wartość `min`.

```
min = random(100);
```

W kolejnym kroku wartości `max` nadajemy wartość `min`.

```
max = min;
```

Następnie w pętli pomniejszonej o 1 (`for (i = 1; i <= ilosc_liczb-1; i++)`) sprawdzamy, czy następna wylosowana liczba jest większa od poprzedniej. Jeśli tak, to staje się ona największą liczbą (`max`); w przeciwnym wypadku przypisujemy jest wartość `min`. Ilustrują to poniższe linijki:

```
if (max < liczba) max = liczba;
if (liczba < min) min = liczba;
```

Sumę wszystkich wylosowanych liczb wyliczają następujące linijki kodu: `suma = suma+max` (przed pętlą) i `suma = suma+liczba` (w pętli). Średnia ze wszystkich liczb jest natomiast obliczana i wyświetlana na ekranie przez taki fragment kodu:

```
cout << "Srednia wynosi " << fixed << setprecision(2) <<
↪suma/ilosc_liczb << "." << endl;
```

Rezultat działania programu dla pięciu wylosowanych liczb można zobaczyć na rysunku 3.6.

Program losuje 5 liczb z przedzialu od 0 do 99,
a nastepnie znajduje najmniejsza i najwieksza oraz
oblicza srednia ze wszystkich wylosowanych liczb.

Wylosowano liczby: 93, 76, 61, 12, 23.
Najwieksza liczba to 93.
Najmniejsza liczba to 12.
Srednia wynosi 53.00.

Rysunek 3.6. *Efekt działania programu Zadanie 3.16*

ZADANIE

3.17

Napisz program, który za pomocą instrukcji `do ... while` znajduje największą i najmniejszą liczbę ze zbioru `n` liczb losowych z przedziału od 0 do 99 oraz oblicza ich średnią (w zadaniu `n = 5`).

Przykładowe rozwiązanie — listing 3.17

```
#include <iostream.h> // Zadanie 3.17
#include <math.h>
#include <iomanip.h>
#include <conio.h>

main()
{
const
ilosc_liczb = 5;

int i = 1;
float liczba, suma, min, max;
```

```
cout << "Program losuje " << ilosc_liczb << " liczb z przedzialu od 0 do
↪99," << endl;
cout << "a nastepnie znajduje najmniejsza i najwieksza oraz" << endl;
cout << "oblicza srednia ze wszystkich wylosowanych liczb." << endl;

suma = 0;
randomize();
min = random(100);
cout << endl;
cout << "Wylosowano liczby: " << min << ", ";
max = min;
suma = suma+max;

do
{
liczba = random(100);

if (i <= ilosc_liczb-2)
{
 cout << liczba << ", ";
}
else
{
 cout << liczba << ".";
}

if (max < liczba) max = liczba;
if (liczba < min) min = liczba;

suma = suma+liczba;
i++;
}
while (i <= ilosc_liczb-1);

cout << endl;
cout << "Najwieksza liczba to " << max << "." << endl;
cout << "Najmniejsza liczba to " << min << "." << endl;
cout << "Srednia wynosi " << fixed << setprecision(2) <<
suma/ilosc_liczb << "." << endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

ZADANIE

3.18 Napisz program, który za pomocą instrukcji `while` znajduje największą i najmniejszą liczbę ze zbioru `n` liczb losowych z przedziału od 0 do 99 oraz oblicza ich średnią (w zadaniu `n = 5`).

Przykładowe rozwiązanie — listing 3.18

```
#include <iostream.h> // Zadanie 3.18
#include <math.h>
#include <iomanip.h>
#include <conio.h>

main()
{
const
ilosc_liczb = 5;

int i = 1;
float liczba, suma, min, max;

cout << "Program losuje " << ilosc_liczb << " liczb z przedzialu od 0 do
↪99," << endl;
cout << "a nastepnie znajduje najmniejsza i najwieksza oraz" << endl;
cout << "oblicza srednia ze wszystkich wylosowanych liczb." << endl;

suma = 0;
randomize();
min = random(100);
cout << endl;
cout << "Wylosowano liczby: " << min << ", ";
max = min;
suma = suma+max;

while (i <= ilosc_liczb-1)
{
liczba = random(100);

if (i <= ilosc_liczb-2)
{
 cout << liczba << ", ";
}
else
{
 cout << liczba << ".";
}

if (max < liczba) max = liczba;
if (liczba < min) min = liczba;

suma = suma+liczba;
i++;
}
```

```
cout << endl;
cout << "Maksymalna liczba to " << max << "." << endl;
cout << "Minimalna liczba to " << min << "." << endl;
cout << "Srednia wynosi " << fixed << setprecision(2) << suma/
↪ilosc_liczb << "." << endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

ZADANIE

3.19 Napisz program wyświetlający tabliczkę mnożenia dla liczb od 1 do 100 z wykorzystaniem podwójnej pętli `for`.

Przykładowe rozwiązanie — listing 3.19

```
#include <iostream.h> // Zadanie 3.19
#include <iomanip.h>
#include <conio.h>

main()
{
 const n = 10;

 int wiersze, kolumny;

 cout << "Program wyswietla tabliczke mnozenia dla liczb od 1 do 100."
 ↪<< endl;
 cout << endl;

 for (wiersze = 1; wiersze <= n; wiersze++)
 {
  for (kolumny = 1; kolumny <= n; kolumny++)
  {
   cout << wiersze*kolumny << '\t';
  }
  cout << endl;
 }

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 3.7.

Program wyswietla tabliczke mnozenia dla liczb od 1 do 100.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Rysunek 3.7. *Efekt działania programu Zadanie 3.19*

ZADANIE

3.20

Napisz program wyświetlający tabliczkę mnożenia dla liczb od 1 do 100 z wykorzystaniem podwójnej pętli `do ... while`.

Przykładowe rozwiązanie — listing 3.20

```
#include <iostream.h> // Zadanie 3.20
#include <iomanip.h>
#include <conio.h>

main()
{
 const n = 10;
 int wiersze, kolumny;
```

```
 cout << "Program wyswietla tabliczke mnozenia dla liczb od 1 do 100."
 ↪<< endl;
 cout << endl;

 wiersze = 1;
 do
 {
  kolumny = 1;
  do
  {
   cout << wiersze*kolumny << '\t';
   kolumny++;
  }
  while (kolumny <= n);
  wiersze++;
  cout << endl;
 }
 while (wiersze <= n);

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

ZADANIE

3.21 Napisz program wyświetlający tabliczkę mnożenia dla liczb od 1 do 100 z wykorzystaniem podwójnej pętli `while`.

Przykładowe rozwiązanie — listing 3.21

```
#include <iostream.h> // Zadanie 3.21
#include <iomanip.h>
#include <conio.h>

main()
{
 const n = 10;
 int wiersze, kolumny;

 cout << "Program wyswietla tabliczke mnozenia dla liczb od 1 do 100."
 ↪<< endl;
 cout << endl;

 wiersze = 1;
 while (wiersze <= n)
  {
   kolumny = 1;
   while (kolumny <= n)
    {
```

```
    cout << wiersze*kolumny << '\t';
    kolumny++;
    }
   wiersze++;
   cout << endl;
  }

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

ZADANIE

3.22 Napisz program, który wyświetla duże litery alfabetu od A do Z i od Z do A, z wykorzystaniem pętli `for`.

Przykładowe rozwiązanie — listing 3.22

```
#include <iostream.h> // Zadanie 3.22
#include <iomanip.h>
#include <conio.h>

main()
{
 char znak;

 cout << "Program wyswietla duze litery alfabetu od A do Z i od Z do A."
 ↪<< endl;
 cout << endl;

 for (znak = 'A'; znak <= 'Z'; znak++)
 {
  if (znak < 'Z')
  {
   cout << znak << ", ";
  }
  else
  {
   cout << znak << ".";
  }
 }

 cout << endl;

 for (znak = 'Z'; znak >= 'A'; znak--)
 {
  if (znak > 'A')
  {
   cout << znak << ", ";
```

```
  }
  else
  {
   cout << znak << ".";
  }
 }

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 3.8.

Program wyswietla duze litery alfabetu od A do Z i od Z do A.

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z.

Z, Y, X, W, V, U, T, S, R, Q, P, O, N, M, L, K, J, I, H, G, F, E, D, C, B, A.

Rysunek 3.8. *Efekt działania programu Zadanie 3.22*

ZADANIE

3.23 Napisz program, który wyświetla duże litery alfabetu od A do Z i od Z do A, z wykorzystaniem pętli `do ... while`.

Przykładowe rozwiązanie — listing 3.23

```
#include <iostream.h> // Zadanie 3.23
#include <iomanip.h>
#include <conio.h>

main()
{
 char znak;

 cout << "Program wyswietla duze litery alfabetu od A do Z i od Z do A."
 ↪<< endl;
 cout << endl;

 znak = 'A';

 do
 {
  if (znak < 'Z')
  {
```

```
  cout << znak << ", ";
 }
 else
 {
  cout << znak << ".";
 }
 znak++;
}
while (znak <= 'Z');

cout << endl;

znak = 'Z';

do
{
 if (znak > 'A')
 {
  cout << znak << ", ";
 }
 else
 {
  cout << znak << ".";
 }
 znak--;
}
while (znak >= 'A');


getch(); // czeka na nacisniecie dowolnego klawisza
}
```

ZADANIE

3.24 Napisz program, który wyświetla duże litery alfabetu od A do Z i od Z do A, z wykorzystaniem pętli `while`.

Przykładowe rozwiązanie — listing 3.24

```
#include <iostream.h> // Zadanie 3.24
#include <iomanip.h>
#include <conio.h>

main()
{
 char znak;

 cout << "Program wyswietla duze litery alfabetu od A do Z i od Z do A."
 ↪<< endl;
```

```
 cout << endl;

 znak = 'A';

 while (znak <= 'Z')
 {
  if (znak < 'Z')
  {
   cout << znak << ", ";
  }
  else
  {
   cout << znak << ".";
  }
  znak++;
 }

 cout << endl;

 znak = 'Z';

 while (znak >= 'A')
 {
  if (znak > 'A')
  {
   cout << znak << ", ";
  }
  else
  {
   cout << znak << ".";
  }
  znak--;
 }

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

4

Tablice

W tym rozdziale przedstawimy typowe zadania wraz z rozwiązaniami z wykorzystaniem tablic jedno- i dwuwymiarowych.

Tablice jednowymiarowe

Tablica jest strukturą danych, która umożliwia przechowywanie w sposób zorganizowany wielu zmiennych tego samego typu (całkowitego, rzeczywistego itd.). Aby utworzyć taką strukturę musimy dokonać deklaracji tablicy. W deklaracji tej określamy typ wartości, jaki ma przechowywać tablica, a także liczbę jej elementów. Tablice mogą być jednowymiarowe, dwuwymiarowe itd.

Ogólna postać deklaracji tablicy jednowymiarowej i związanej z nią zmiennej jest w języku C++ następująca:

```
typ_tablicy nazwa_tablicy[rozmiar_tablicy];
```

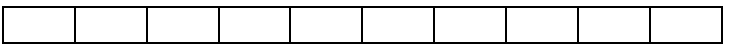
Przykład poniżej ilustruje deklarację tablicy jednowymiarowej typu całkowitego o nazwie `dane` zawierającej 10 elementów.

```
int dane[10];
```

Dostęp do elementów tablicy jest realizowany za pośrednictwem indeksu, który wskazuje dany element. Dla deklaracji w języku C++ o postaci

```
int dane[10];
```

pierwszy element tablicy dane ma indeks 0, drugi dostępny jest przez indeks 1 itd. Ostatni element ma indeks równy wymiarowi tablicy pomniejszonemu o 1, czyli 9, co przedstawiono w reprezentacji graficznej poniżej.



0 1 2 3 4 5 6 7 8 9

ZADANIE

4.1 Napisz program, który w 10-elementowej tablicy jednowymiarowej o nazwie dane umieszcza liczby od 0 do 9 (zobacz poniżej jej reprezentację graficzną).

Indeks tablicy	Wartość tablicy
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

Przykładowe rozwiązanie — listing 4.1

```
#include <iostream.h> // Zadanie 4.1
#include <conio.h>

main()
{
   const n = 10;
   int i, dane[n];
   for (i = 0; i < 10; i++)
   {
      dane[i] = i;
      cout << "dane[" << i << "] = " << dane[i] << endl;
```

```
    }
    getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 4.1.

```
dane[0] = 0
dane[1] = 1
dane[2] = 2
dane[3] = 3
dane[4] = 4
dane[5] = 5
dane[6] = 6
dane[7] = 7
dane[8] = 8
dane[9] = 9
```

Rysunek 4.1. *Efekt działania programu Zadanie 4.1*

ZADANIE

4.2 Napisz program, który w 10-elementowej tablicy jednowymiarowej o nazwie `dane` umieszcza liczby od 9 do 0 (zobacz poniżej jej reprezentację graficzną).

Indeks tablicy	Wartość tablicy
0	9
1	8
2	7
3	6
4	5
5	4
6	3
7	2
8	1
9	0

Wskazówka

Zadanie to rozwiążemy poprawnie, zamieniając w zadaniu 4.1 linijkę kodu

```
dane[i] = i;
```

na następującą:

```
dane[i] = n-1-i;
```

Przykładowe rozwiązanie — listing 4.2

```
#include <iostream.h> // Zadanie 4.2
#include <conio.h>

main()
{
   const n = 10;
   int i, dane[n];

   for (i = 0; i < 10; i++)
   {
      dane[i] = n-i-1;
      cout << "dane[" << i << "] = " << dane[i] << endl;
   }

   getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 4.2.

```
dane[0] = 9
dane[1] = 8
dane[2] = 7
dane[3] = 6
dane[4] = 5
dane[5] = 4
dane[6] = 3
dane[7] = 2
dane[8] = 1
dane[9] = 0
```

Rysunek 4.2. *Efekt działania programu Zadanie 4.2*

Tablice dwuwymiarowe

Tablice dwuwymiarowe deklarujemy w języku C++ podobnie jak jednowymiarowe. Ogólna postać deklaracji takiej tablicy i związanej z nią zmiennej jest następująca:

```
typ_tablicy nazwa_tablicy[rozmiar_tablicy][rozmiar_tablicy];
```

Dostęp do elementów tablicy jest realizowany za pośrednictwem dwóch indeksów, które wskazują dany element. Przykład poniżej ilustruje deklarację tablicy dwuwymiarowej 10×10 typu całkowitego.

```
int dane[10][10];
```

Tablicę dwuwymiarową, jako produkt o wymiarach 10×10, możemy sobie wyobrazić następująco:

1	**2**	0	0	0	0	0	0	0	0
3	**1**	0	0	0	0	0	0	0	0
0	0	**1**	0	0	0	0	0	0	0
0	0	0	**1**	0	0	0	0	0	0
0	0	0	0	**1**	0	0	0	0	0
0	0	0	0	0	**1**	0	0	0	0
0	0	0	0	0	0	**1**	0	0	0
0	0	0	0	0	0	0	**1**	0	0
0	0	0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0	0	**7**

Wartości liczbowe możemy wpisywać do tablicy wierszami lub kolumnami. Pierwszy element tablicy `dane[0][0]` jest równy `1`, element `dane[0][1]` to `2`, element `dane[1][0]` wynosi `3` itd. Ostatni element tablicy ma indeks równy jej wymiarowi minus 1, czyli 9 — w naszym przypadku `dane[9][9] = 7`.

ZADANIE

4.3

Napisz program, który w zadeklarowanej tablicy dwuwymiarowej 10×10 o nazwie `macierz` umieszcza na przekątnej liczbę 1, a poza przekątną 0. Dodatkowo program powinien obliczać sumę elementów wyróżnionych w tablicy, tj. tych znajdujących się na jej przekątnej.

1	0	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	0	**1**	0	0	0	0	0	0	0
0	0	0	**1**	0	0	0	0	0	0
0	0	0	0	**1**	0	0	0	0	0
0	0	0	0	0	**1**	0	0	0	0
0	0	0	0	0	0	**1**	0	0	0
0	0	0	0	0	0	0	**1**	0	0
0	0	0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0	0	**1**

Przykładowe rozwiązanie — listing 4.3

```
#include <iostream.h> // Zadanie 4.3
#include <conio.h>

main()
{
 const int n = 10;
 int i, j, suma, macierz[n][n];

cout << "Wpisywanie do tablicy liczby 1 na przekatnej, a 0 poza nia." <<
↪endl;
cout << endl;

for (i = 0; i < n; i++) // wpisywanie do tablicy
{
for (j = 0; j < n; j++)
{
 if (i == j)
  macierz[i][j] = 1;
 else
  macierz[i][j] = 0;
}
}
```

```
// wyswietlenie zawartosci tablicy
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
 cout << macierz[i][j] << " ";
}
 cout << endl;
}

// obliczanie sumy liczb znajdujacych sie na przekatnej
suma = 0;
cout << endl;

for (i = 0; i < n; i++)
{
 suma = suma+macierz[i][i];
}

cout << "Suma elementow na przekatnej wynosi " << suma << "." << endl;

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Do wpisywania danych do tablicy o nazwie `macierz` użyliśmy dwóch pętli `for`. Linijki kodu z instrukcją warunkową `if`:

```
if (i == j)
  macierz[i][j] = 1;
 else
  macierz[i][j] = 0;
```

odpowiadają za wpisywanie liczby 1 na przekątnej i liczby 0 poza nią.

Całość kodu umieszczającego w tablicy liczby 1 na przekątnej i liczby 0 poza przekątną znajduje się poniżej.

```
for (i = 0; i < n; i++) // wpisywanie do tablicy
{
for (j = 0; j < n; j++)
{
 if (i == j)
  macierz[i][j] = 1;
 else
  macierz[i][j] = 0;
}
}
```

Za wyświetlenie zawartości tablicy na ekranie komputera odpowiadają następujące linijki kodu:

```
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
 cout << macierz[i][j] << " ";
}
 cout << endl;
}
```

Obliczanie sumy elementów znajdujących się na przekątnej należy do następujących linijek:

```
for (i = 0; i < n; i++)
{
 suma = suma+macierz[i][i];
}
```

Oczywiście zmienną `suma` trzeba wcześniej wyzerować:

```
suma = 0;
```

Rezultat działania programu można zobaczyć na rysunku 4.3.

```
Wpisywanie do tablicy liczby 1 na przekatnej, a 0 poza nia.

1 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1

Suma elementow na przekatnej wynosi 10.
```

Rysunek 4.3. *Efekt działania programu Zadanie 4.3*

ZADANIE

4.4 Napisz program, który w zadeklarowanej tablicy dwuwymiarowej 10×10 o nazwie `macierz` umieszcza na przekątnej liczby od 0 do 9, a poza przekątną 0. Dodatkowo program powinien obliczać sumę elementów wyróżnionych w tablicy, tj. znajdujących się na jej przekątnej.

0	0	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	0	**2**	0	0	0	0	0	0	0
0	0	0	**3**	0	0	0	0	0	0
0	0	0	0	**4**	0	0	0	0	0
0	0	0	0	0	**5**	0	0	0	0
0	0	0	0	0	0	**6**	0	0	0
0	0	0	0	0	0	0	**7**	0	0
0	0	0	0	0	0	0	0	**8**	0
0	0	0	0	0	0	0	0	0	**9**

Wskazówka

Zadanie to rozwiążemy poprawnie, zamieniając w zadaniu 4.3 linijkę kodu

```
macierz[i][j] = 1;
```

na następującą:

```
macierz[i][j] = i;
```

Przykładowe rozwiązanie — listing 4.4

```
#include <iostream.h> // Zadanie 4.4
#include <conio.h>

main()
{
 const int n = 10;
 int i, j, suma, macierz[n][n];

cout << "Wpisywanie do tablicy liczb od 0 do 9 na przekatnej,
↪a 0 poza nia." << endl;
cout << endl;
```

```
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
 if (i == j)
  macierz[i][j] = i;
 else
  macierz[i][j] = 0;
}
}

// wyswietlenie zawartosci tablicy
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
 cout << macierz[i][j] << " ";
}
 cout << endl;
}

// obliczanie sumy liczb znajdujacych sie na przekatnej
suma = 0;
cout << endl;

for (i = 0; i < n; i++)
{
 suma = suma+macierz[i][i];
}
cout << "Suma liczb na przekatnej wynosi " << suma << "." << endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 4.4.

ZADANIE

4.5 Napisz program, który w zadeklarowanej tablicy dwuwymiarowej 10×10 o nazwie `macierz` (jej interpretacja graficzna poniżej) umieszcza liczby 1 i 0. Program powinien dodatkowo obliczać sumę wyróżnionych elementów.

Wpisywanie do tablicy liczb od 0 do 9 na przekatnej, a 0 poza nia.

0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 2 0 0 0 0 0 0 0
0 0 0 3 0 0 0 0 0 0
0 0 0 0 4 0 0 0 0 0
0 0 0 0 0 5 0 0 0 0
0 0 0 0 0 0 6 0 0 0
0 0 0 0 0 0 0 7 0 0
0 0 0 0 0 0 0 0 8 0
0 0 0 0 0 0 0 0 0 9

Suma liczb na przekatnej wynosi 45.

Rysunek 4.4. *Efekt działania programu Zadanie 4.4*

0	0	0	0	0	0	0	0	0	**1**
0	0	0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	**1**	0	0
0	0	0	0	0	0	**1**	0	0	0
0	0	0	0	0	**1**	0	0	0	0
0	0	0	0	**1**	0	0	0	0	0
0	0	0	**1**	0	0	0	0	0	0
0	0	**1**	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0

Wskazówka

Zadanie to rozwiążemy poprawnie, zamieniając w zadaniu 4.3 linijki kodu

```
if (i == j)
   macierz[i][j] = 1;
else
   macierz[i][j] = 0;
```

na następujące:

```
if (n == i+j+1)
   macierz[i][j] = 1;
else
   macierz[i][j] = 0;
```

Sumę wyróżnionych elementów obliczymy, zastępując linijki

```
for (i = 0; i < n; i++)
{
   suma = suma+macierz[i][i];
}
```

poniższymi.

```
for (i = 0; i < n; i++)
{
 suma = suma+macierz[i][n-i-1];
}
```

Przykładowe rozwiązanie — listing 4.5

```
#include <iostream.h> // Zadanie 4.5
#include <conio.h>

main()
{
 const int n = 10;
 int i, j, suma, macierz[n][n];

cout << "Wpisywanie do tablicy liczb 1 i 0 oraz ich wyswietlenie." <<
↪endl;
cout << endl;
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
 if (n == i+j+1)
  macierz[i][j] = 1;
 else
  macierz[i][j] = 0;
}
}

// wyswietlenie zawartosci tablicy
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
 cout << macierz[i][j] << " ";
```

```
}
 cout << endl;
}

// obliczanie sumy liczb wyroznionych w zadaniu
suma = 0;
cout << endl;

for (i = 0; i < n; i++)
{
 suma = suma+macierz[i][n-i-1];
}

cout << "Suma wyroznionych w zadaniu elementow wynosi " << suma << "."
↪<< endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 4.5.

```
Wpisywanie do tablicy liczb 1 i 0 oraz ich wyswietlenie.

0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0

Suma wyroznionych w zadaniu elementow wynosi 10.
```

Rysunek 4.5. *Efekt działania programu Zadanie 4.5*

ZADANIE

4.6 Napisz program, który w zadeklarowanej tablicy dwuwymiarowej 10×10 o nazwie `macierz` (jej interpretacja graficzna poniżej) umieszcza liczby od 0 do 9. Program powinien dodatkowo obliczać sumę wyróżnionych elementów.

0	0	0	0	0	0	0	0	0	**0**
0	0	0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	**2**	0	0
0	0	0	0	0	0	**3**	0	0	0
0	0	0	0	0	**4**	0	0	0	0
0	0	0	0	**5**	0	0	0	0	0
0	0	0	**6**	0	0	0	0	0	0
0	0	**7**	0	0	0	0	0	0	0
0	**8**	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0

Wskazówka

Zadanie to rozwiążemy poprawnie, zamieniając w zadaniu 4.5 linijki kodu

```
if (n == i+j+1)
   macierz[i][j] = 1;
else
   macierz[i][j] = 0;
```

na następujące:

```
if (n == i+j+1)
   macierz[i][j] = i;
else
   macierz[i][j] = 0;
```

Przykładowe rozwiązanie — listing 4.6

```
#include <iostream.h> // Zadanie 4.6
#include <conio.h>

main()
{
 const int n = 10;
 int i, j, suma, macierz[n][n];

cout << "Wpisywanie do tablicy liczb od 0 do 9 i ich wyswietlenie." <<
↪endl;
cout << endl;
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
 if (n == i+j+1)
  macierz[i][j] = i;
 else
```

```
  macierz[i][j] = 0;
}
}

// wyswietlenie zawartosci tablicy
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
 cout << macierz[i][j] << " ";
}
 cout << endl;
}

// obliczanie sumy liczb wyroznionych w zadaniu
suma = 0;
cout << endl;

for (i = 0; i < n; i++)
{
 suma = suma+macierz[i][n-i-1];
}

cout << "Suma wyroznionych w tablicy elementow wynosi " << suma << "."
↪<< endl;

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 4.6.

```
Wpisywanie do tablicy liczb od 0 do 9 i ich wyswietlenie.

0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 3 0 0 0
0 0 0 0 0 4 0 0 0 0
0 0 0 0 5 0 0 0 0 0
0 0 0 6 0 0 0 0 0 0
0 0 7 0 0 0 0 0 0 0
0 8 0 0 0 0 0 0 0 0
9 0 0 0 0 0 0 0 0 0

Suma wyroznionych w tablicy elementow wynosi 45.
```

Rysunek 4.6. *Efekt działania programu Zadanie 4.6*

ZADANIE

4.7

Napisz program, który w zadeklarowanej tablicy dwuwymiarowej 10×10 umieszcza w pierwszej kolumnie liczby od 0 do 9, w drugiej kwadraty tych liczb, natomiast w pozostałych kolumnach 0 (interpretacja graficzna tablicy poniżej). Dodatkowo program powinien obliczać sumę elementów znajdujących się w pierwszej kolumnie oraz sumę liczb z kolumny drugiej.

0	**0**	0	0	0	0	0	0	0	0
1	**1**	0	0	0	0	0	0	0	0
2	**4**	0	0	0	0	0	0	0	0
3	**9**	0	0	0	0	0	0	0	0
4	**16**	0	0	0	0	0	0	0	0
5	**25**	0	0	0	0	0	0	0	0
6	**36**	0	0	0	0	0	0	0	0
7	**49**	0	0	0	0	0	0	0	0
8	**64**	0	0	0	0	0	0	0	0
9	**81**	0	0	0	0	0	0	0	0

Przykładowe rozwiązanie — listing 4.7

```
#include <iostream.h> // Zadanie 4.7
#include <conio.h>
#include <iomanip.h>

main()
{
const int n = 10;
int i, j, suma, tablica[n][n];

// wpisywanie liczb do tablicy
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
 if (j == 0) tablica[i][j] = i;
 if (j == 1) tablica[i][j] = i*i;
 if (j > 1) tablica[i][j] = 0;
}
}
```

```
// wyswietlenie zawartosci tablicy
cout << "Zawartosc tablicy:" << endl;
cout << endl;

for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
 cout << setw(2) << tablica[i][j] << " ";
}
cout << endl;
}

suma = 0;
for (i = 0; i < n; i++)
{
 suma = suma+tablica[i][0];
}

cout << endl;
cout << "Suma liczb znajdujacych sie w pierwszej kolumnie wynosi " <<
↪suma << "." << endl;

suma = 0;
for (i = 0; i < n; i++)
{
 suma = suma+tablica[i][1];
}

cout << endl;
cout << "Suma liczb znajdujacych sie w drugiej kolumnie wynosi " << suma
↪<< "." << endl;

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Następujące linijki kodu:

```
for (i = 0; i < n; i++)
{
   for (j = 0; j < n; j++)
   {
      if (j == 0) tablica[i][j] = i;
      if (j == 1) tablica[i][j] = i*i;
      if (j > 1) tablica[i][j] = 0;
   }
}
```

są odpowiedzialne za wpisywanie do tablicy liczb. Do pierwszej kolumny są wpisywane liczby od 0 do 9:

```
if (j == 0) tablica[i][j] = i;
```

do drugiej ich kwadraty:

```
if (j == 1) tablica[i][j] = i*i;
```

a do pozostałych kolumn 0:

```
if (j > 1) tablica[i][j] = 0;
```

Sumowaniem liczb znajdujących się w pierwszej kolumnie zajmują się następujące linijki:

```
for (i = 0; i < n; i++)
{
   suma = suma+tablica[i][0];
}
```

Za sumowanie liczb z drugiej kolumny odpowiada następujący fragment kodu:

```
for (i = 0; i < n; i++)
{
   suma = suma+tablica[i][1];
}
```

W programie wykorzystano manipulator `setw(int n)`, który określa szerokość pola — *n* znaków.

```
cout << setw(2) << tablica[i][j] << " ";
```

Aby mógł on funkcjonować, należy dodać na początku plik nagłówkowy:

```
#include <iomanip.h>
```

Rezultat działania programu można zobaczyć na rysunku 4.7.

Zawartosc tablicy:

```
0  0  0  0  0  0  0  0  0  0
1  1  0  0  0  0  0  0  0  0
2  4  0  0  0  0  0  0  0  0
3  9  0  0  0  0  0  0  0  0
4 16  0  0  0  0  0  0  0  0
5 25  0  0  0  0  0  0  0  0
6 36  0  0  0  0  0  0  0  0
7 49  0  0  0  0  0  0  0  0
8 64  0  0  0  0  0  0  0  0
9 81  0  0  0  0  0  0  0  0
```

Suma liczb znajdujacych sie w pierwszej kolumnie wynosi 45.

Suma liczb znajdujacych sie w drugiej kolumnie wynosi 285.

Rysunek 4.7. *Efekt działania programu Zadanie 4.7*

ZADANIE 4.8

Dane są dwie tablice dwuwymiarowe 10×10 o nazwach a i b. Tablica a zawiera elementy przedstawione poniżej.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Tablica `b` zawiera same zera. Napisz program, który przepisuje zawartość tablicy `a` do tablicy `b`, zamieniając kolumny na wiersze (interpretacja graficzna tablicy wynikowej poniżej).

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
2	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
3	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
4	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
6	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
7	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
8	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
9	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**

Przykładowe rozwiązanie — listing 4.8

```
#include <iostream.h> // Zadanie 4.8
#include <conio.h>

main()
{
const int n = 10;
int i, j, a[n][n], b[n][n];

// wpisywanie liczb do tablicy a
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
 a[i][j] = j;
}
}

// przepisywanie liczb z tablicy a do tablicy b
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
```

```
 b[i][j] = a[j][i]; // zamiana kolumn na wiersze
}
}

// wyswietlenie zawartosci tablicy a
cout << "Zawartosc tablicy a:" << endl;
cout << endl;

for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
 cout << a[i][j] << " ";
}
 cout << endl;
}

cout << endl; // wyswietlenie pustej linii

// wyswietlanie zawartosci tablicy b
cout << "Zawartosc tablicy b:" << endl;
cout << endl;

for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
 cout << b[i][j] << " ";
}
 cout << endl;
}

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Następująca linijka kodu:

```
b[i][j] = a[j][i];
```

jest odpowiedzialna za zamianę kolumn na wiersze.

Rezultat działania programu można zobaczyć na rysunku 4.8.

```
Zawartosc tablicy a:

0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9

Zawartosc tablicy b:

0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9
```

Rysunek 4.8. *Efekt działania programu Zadanie 4.8*

5

Podprogramy

W tym rozdziale przedstawimy typowe zadania wraz z przykładowymi rozwiązaniami z użyciem podprogramów. Umiejętność wykorzystywania ich w pisanych programach zwykle owocuje szybkim przyswojeniem zasad programowania obiektowego.

W języku C++ podprogramy stanowią pewną logiczną całość i nazywamy je funkcjami. Każda funkcja w programie powinna realizować jakieś określone zadanie, np. obliczać średnią arytmetyczną z określonych liczb, czytać elementy tablicy lub obliczać nadgodziny zatrudnionego pracownika itd. Gdy program musi wykonać określone zadanie, wywołuje odpowiednią funkcję, podając jej informacje potrzebne do zrealizowania tego zadania. Komunikacja pomiędzy funkcjami a pozostałą częścią programu odbywa się w ściśle określony sposób.

W C++ ogólna postać funkcji jest następująca:

```
typ_wartości nazwa_funkcji(lista_parametrów)

   {
      deklaracje_zmiennych;

      ciąg instrukcji;
   }
```

Każda funkcja stanowi w tym języku spójny, tworzący pewną całość blok instrukcji. Jej zawartość należy wyłącznie do niej samej i jest ona niedostępna dla instrukcji znajdujących się we wszystkich innych

funkcjach. Jedyną możliwością skorzystania z niej jest jej wywołanie. Instrukcje, które tworzą główną część funkcji, są ukryte przed pozostałą częścią programu i jeśli nie korzystają ze zmiennych (lub danych) globalnych, to nie mają wpływu na działanie innych jego fragmentów; same również nie pozostają pod ich wpływem. Oznacza to, że instrukcje i dane zdefiniowane w jednej funkcji nie mogą współdziałać z instrukcjami i danymi określonymi w innej, ponieważ funkcje mają różny zakres. Wszystkie zmienne zdefiniowane wewnątrz nich nazywamy zmiennymi lokalnymi. Istnieją one tylko w obrębie funkcji i po wyjściu z nich ulegają zniszczeniu. Dlatego w funkcjach lokalnych nie można przechowywać wartości pomiędzy ich wywołaniami. W języku C++ zakres wszystkich funkcji znajduje się na tym samym poziomie. Oznacza to, że nie jest możliwe definiowanie jednej z nich wewnątrz innej.

ZADANIE

5.1 Napisz program obliczający pole prostokąta. Powinien on zawierać jeden podprogram: bezparametrową funkcję `pole_prostokata()`, w której zawarty będzie algorytm obliczania pola. Wartości boków `a` i `b` wprowadzamy z klawiatury. W programie należy założyć, że zmienne `a`, `b` oraz `pole` są typu `float` (rzeczywistego) i są one zmiennymi globalnymi. Należy dla nich przyjąć format wyświetlania ich z dwoma miejscami po kropce.

Przykładowe rozwiązanie — listing 5.1

```
#include <iostream.h> // Zadanie 5.1
#include <iomanip.h>
#include <conio.h>

float a, b, pole;

void pole_prostokata() //deklaracja i definicja funkcji pole_prostokata()
{
cout << "Program oblicza pole prostokata." << endl;
cout << "Podaj bok a." << endl;
cin >> a;
cout << "Podaj bok b." << endl;
cin >> b;
pole = a*b;
cout << fixed << setprecision(2);
cout << "Pole prostokata o boku a = " << a << " i boku b = " << b;
cout << " wynosi " << pole << "." << endl;
}

main()
```

```
{
pole_prostokata(); // wywolanie funkcji pole_prostokata()

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Zdefiniowane tu zmienne

```
float a, b, pole;
```

są zmiennymi globalnymi, czyli są one widoczne w całym programie i podprogramie.

Funkcja `pole_prostokata()` zawiera algorytm obliczający pole prostokąta:

```
void pole_prostokata() //deklaracja i definicja funkcji pole_prostokata()
{
cout << "Program oblicza pole prostokata." << endl;
cout << "Podaj bok a." << endl;
cin >> a;
cout << "Podaj bok b." << endl;
cin >> b;
pole = a*b;
cout << fixed << setprecision(2);
cout << "Pole prostokata o boku a = " << a << " i boku b = " << b;
cout << " wynosi " << pole << "." << endl;
}
```

Funkcja ta wywołana zostaje w programie głównym.

```
main()
{
pole_prostokata(); // wywolanie funkcji pole_prostokata()

..........
}
```

Jej wywołanie realizuje algorytm obliczający pole prostokąta.

Rezultat działania programu można zobaczyć na rysunku 5.1.

```
Program oblicza pole prostokata.
Podaj bok a.
2.01
Podaj bok b.
1.02
Pole prostokata o boku a = 2.01 i boku b = 1.02 wynosi 2.05.
```

Rysunek 5.1. *Efekt działania programu Zadanie 5.1*

ZADANIE

5.2 Napisz program obliczający pole prostokąta. Powinien on zawierać jeden podprogram: bezparametrową funkcję `pole_prostokata`, w której zawarty będzie algorytm obliczania pola. Wartości boków `a` i `b` wprowadzamy z klawiatury. W programie należy założyć, że zmienne `a`, `b` oraz `pole` są typu `float` (rzeczywistego) i są zmiennymi lokalnymi. Należy dla nich przyjąć format wyświetlania ich z dwoma miejscami po kropce.

Przykładowe rozwiązanie — listing 5.2

```
#include <iostream.h> // Zadanie 5.2
#include <iomanip.h>
#include <conio.h>

void pole_prostokata() // deklaracja i definicja funkcji pole_prostokata()
{
float a, b, pole;

cout << "Program oblicza pole prostokata." << endl;
cout << "Podaj bok a." << endl;
cin >> a;
cout << "Podaj bok b." << endl;
cin >> b;
pole = a*b;
cout << fixed << setprecision(2);
cout << "Pole prostokata o boku a = " << a << " i boku b = " << b;
cout << " wynosi " << pole << "." << endl;
}

main()
{
pole_prostokata(); //wywolanie funkcji pole_prostokata()

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

W tym zadaniu zmienne

```
float a, b, pole;
```

zostały zdefiniowane jako zmienne lokalne, więc są one widoczne tylko w podprogramie. Reszta zadania została rozwiązana jak wyżej.

Rezultat działania programu można zobaczyć na rysunku 5.1.

ZADANIE

5.3 Napisz program obliczający pole prostokąta. Powinien on zawierać jeden podprogram: funkcję `pole_prostokata()`, do której parametry przekazywane są przez wartość. Wartości boków `a` i `b` wprowadzamy z klawiatury. W programie należy założyć, że zmienne `a`, `b` oraz `pole` są typu `float` (rzeczywistego) i są zmiennymi globalnymi. Należy dla nich przyjąć format wyświetlania ich z dwoma miejscami po kropce.

Przykładowe rozwiązanie — listing 5.3

```
#include <iostream.h> // Zadanie 5.3
#include <iomanip.h>
#include <conio.h>

float a, b, pole;

void pole_prostokata(float x, float y) // deklaracja i definicja funkcji
{
 pole = x*y;
 cout << fixed << setprecision(2);
 cout << "Pole prostokata o boku a = " << x << " i boku b = " << y;
 cout << " wynosi " << pole << "." << endl;
}

main()
{
cout << "Program oblicza pole prostokata." << endl;
cout << "Podaj bok a." << endl;
cin >> a;
cout << "Podaj bok b." << endl;
cin >> b;

pole_prostokata(a, b); // wywolanie funkcji

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Funkcja `void pole_prostokata(float x, float y)` zawiera dwa parametry formalne, `x` i `y` typu `float`, których używamy tylko do jej opisu. Cała jej postać jest następująca:

```
void pole_prostokata(float x, float y) // deklaracja i definicja funkcji
{
 pole = x*y;
 cout << fixed << setprecision(2);
```

```
 cout << "Pole prostokata o boku a = " << x << " i boku b = " << y;
 cout << " wynosi " << pole << "." << endl;
}
```

Wywołując tę funkcję w programie głównym, wywołujemy ją z parametrami aktualnymi, których wartości wprowadzamy z klawiatury:

```
{
   .......
   pole_prostokata(a, b); // wywolanie funkcji
   .......
}
```

Rezultat działania programu można zobaczyć na rysunku 5.1.

Łatwo zauważyć, że większość zadań z programowania z wykorzystaniem podprogramów można rozwiązać według następującego schematu[1]:

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
.....................

void czytaj_dane() // deklaracja i definicja funkcji czytaj_dane()
{
.........
}

void przetworz_dane() // deklaracja i definicja funkcji przetworz_dane()
{
.........
}

void wyswietl_wynik() // deklaracja i definicja funkcji wyswietl_wynik()
{
.........
}

main()
{
   czytaj_dane(); // wywolanie funkcji czytaj_dane()
   przetworz_dane(); // wywolanie funkcji przetworz_dane()
   wyswietl_wynik(); // wywolanie funkcji wyswietl_wynik()

   getch(); // czeka na nacisniecie dowolnego klawisza
}
```

[1] Schemat ten jest bardzo przydatny w programowaniu strukturalnym, kiedy operujemy pojęciem podprogramu, oraz w programowaniu obiektowym, gdzie posługujemy się pojęciem obiektu.

Funkcja czytaj_dane() zajmuje się tylko czytaniem danych. Za ich przetworzenie odpowiedzialna jest funkcja przetworz_dane(). Ostatnia z funkcji, wyswietl_wynik(), wyświetla przetworzone dane (wyniki) np. na ekranie monitora. Funkcje mogą być z parametrem lub bezparametrowe w zależności od upodobań programisty. Powyższy schemat zilustrujemy poznanym wcześniej przykładem programu, który oblicza pole prostokąta.

ZADANIE

5.4 Napisz program, który oblicza pole prostokąta. Powinien on zawierać trzy bezparametrowe funkcje: czytaj_dane(), przetworz_dane() oraz wyswietl_wynik(). Wartości boków a i b wprowadzamy z klawiatury. W programie należy założyć, że zmienne a, b oraz pole są typu float (rzeczywistego) i są zmiennymi globalnymi. Należy dla nich przyjąć format wyświetlania ich z dwoma miejscami po kropce. Funkcja czytaj_dane() czyta wartości boków a i b, przetworz_dane() oblicza pole prostokąta, a wyswietl_wynik() prezentuje wyniki na ekranie komputera.

Przykładowe rozwiązanie — listing 5.4

```
#include <iostream.h> // Zadanie 5.4
#include <iomanip.h>
#include <conio.h>

float a, b, pole;

void czytaj_dane() // deklaracja i definicja funkcji czytaj_dane()
{
cout << "Program oblicza pole prostokata." << endl;
cout << "Podaj bok a." << endl;
cin >> a;
cout << "Podaj bok b." << endl;
cin >> b;
}

void przetworz_dane() // deklaracja i definicja funkcji przetworz_dane()
{
pole = a*b;
}

void wyswietl_wynik() // deklaracja i definicja funkcji wyswietl_wynik()
{
```

```
cout << fixed << setprecision(2);
cout << "Pole prostokata o boku a = " << a << " i boku b = " << b;
cout << " wynosi " << pole << "." << endl;
}

main()
{
czytaj_dane(); // wywolanie funkcji czytaj_dane()
przetworz_dane(); // wywolanie funkcji przetworz_dane()
wyswietl_wynik(); // wywolanie funkcji wyswietl_wynik()

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

W naszym programie funkcja `czytaj_dane()`:

```
void czytaj_dane() // deklaracja i definicja funkcji czytaj_dane()
{
   cout << "Program oblicza pole prostokata." << endl;
   cout << "Podaj bok a." << endl;
   cin >> a;
   cout << "Podaj bok b." << endl;
   cin >> b;
}
```

wczytuje z klawiatury wartość boku `a` i wartość boku `b`.

Funkcja `przetworz_dane()`:

```
void przetworz_dane() // deklaracja i definicja funkcji przetworz_dane()
{
   pole = a*b;
}
```

przetwarza te dane i oblicza pole prostokąta według wzoru `pole = a*b`.

Natomiast funkcja `wyswietl_wynik()`:

```
void wyswietl_wynik() // deklaracja i definicja funkcji wyswietl_wynik()
{
cout << fixed << setprecision(2);
cout << "Pole prostokata o boku a = " << a << " i boku b = " << b;
cout << " wynosi " << pole << "." << endl;
}
```

prezentuje wartości boków `a` i `b` oraz wartość zmiennej `pole` w określonym formacie. W programie głównym zostają wywołane wszystkie trzy funkcje:

```
main()
{
   czytaj_dane(); // wywolanie funkcji czytaj_dane()
   przetworz_dane(); // wywolanie funkcji przetworz_dane()
   wyswietl_wynik(); // wywolanie funkcji wyswietl_wynik()

.........
}
```

Rezultat działania programu można zobaczyć na rysunku 5.1.

Prawda, że wszystko jest jasne i proste? Następne zadania w tym rozdziale spróbujemy rozwiązać według powyższego schematu.

ZADANIE

5.5 Napisz program, który z wykorzystaniem instrukcji wyboru `switch ... case` oblicza pierwiastki równania kwadratowego $ax^2+bx+c = 0$, gdzie zmienne `a`, `b` oraz `c` to liczby rzeczywiste wprowadzane z klawiatury. Dla zmiennych `a`, `b`, `c`, `x1` oraz `x2` należy przyjąć format wyświetlania ich z dwoma miejscami po kropce. Program powinien zawierać trzy bezparametrowe funkcje: `czytaj_dane()`, `przetworz_dane()` i `wyswietl_wynik()`.

> **Wskazówka**
>
> Funkcja `czytaj_dane()` jest odpowiedzialna za wczytanie danych do programu oraz obsłużenie sytuacji, kiedy `a = 0`. Funkcja `przetworz_dane()` zajmuje się wykonaniem niezbędnych obliczeń, natomiast funkcja `wyswietl_wynik()` jest natomiast odpowiedzialna za pokazanie rezultatów na ekranie monitora.

Przykładowe rozwiązanie — listing 5.5

```
#include <iostream.h> // Zadanie 5.5
#include <iomanip.h>
#include <conio.h>

float a, b, c, delta, x1, x2;
char liczba_pierwiastkow;

void czytaj_dane()
{
cout << "Program oblicza pierwiastki rownania kwadratowego" << endl;
cout << "dla dowolnych wspolczynnikow a, b, c." << endl;
cout << "Podaj a." << endl;
cin >> a;
```

```
if (a == 0)
{
cout << "Niedozwolona wartosc wspolczynnika. Nacisnij dowolny klawisz."
↪<< endl;
getch(); // czeka na nacisniecie dowolnego klawisza
exit(1); // wyjscie z programu
}
else
{
cout << "Podaj b." << endl;
cin >> b;
cout << "Podaj c." << endl;
cin >> c;
}
}

przetworz_dane()
{
delta = b*b-4*a*c;

if (delta < 0) liczba_pierwiastkow = 0;
if (delta == 0) liczba_pierwiastkow = 1;
if (delta > 0) liczba_pierwiastkow = 2;

switch(liczba_pierwiastkow)
{
case 1 : x1 = -b/(2*a);
             break;
case 2 : { x1 = (-b-sqrt(delta))/(2*a);
              x2 = (-b+sqrt(delta))/(2*a);
             }
             break;
}
}

void wyswietl_wynik()
{
cout << "Dla wprowadzonych liczb:" << endl;
cout << "a = " << a << "," << endl;
cout << "b = " << b << "," << endl;
cout << "c = " << c << "," << endl;

switch(liczba_pierwiastkow)
{
case 0 : cout << "brak pierwiastkow rzeczywistych." << endl;
             break;
case 1 : cout << "trojmian ma jeden pierwiastek podwojny x1 = " << x1
↪<< "." << endl;
             break;
```

```
case 2 : { cout << "trojmian ma dwa pierwiastki:" << endl;
           cout << "x1 = " << x1 << "," << endl;
           cout << "x2 = " << x2 << "." << endl;
         }
         break;
}
}

main()
{
cout << fixed << setprecision(2);
czytaj_dane();
przetworz_dane();
wyswietl_wynik();

getch(); //czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu dla `a = 1`, `b = 2` i `c = 3` można zobaczyć na rysunku 5.2.

```
Program oblicza pierwiastki rownania kwadratowego
dla dowolnych wspolczynnikow a, b, c.
Podaj a.
1
Podaj b.
2
Podaj c.
3
Dla wprowadzonych liczb:
a = 1.00,
b = 2.00,
c = 3.00,
brak pierwiastkow rzeczywistych.
```

Rysunek 5.2. *Efekt działania programu Zadanie 5.5*

ZADANIE

5.6 Napisz program, który w tablicy 10×10 umieszcza losowo na przekątnej cyfry od 0 do 9, a poza nią zera. Dodatkowo oblicza on sumę liczb znajdujących się na przekątnej. Program powinien zawierać trzy bezparametrowe funkcje: `czytaj_dane()`, `przetworz_dane()` i `wyswietl_wynik()`. Funkcja `czytaj_dane()` umieszcza dane w tablicy, `przetworz_dane()` oblicza sumę liczb znajdujących się na przekątnej, natomiast funkcja `wyswietl_wynik()` prezentuje zawartość tablicy na ekranie monitora.

Przykładowe rozwiązanie — listing 5.6

```
#include <iostream.h> // Zadanie 5.6
#include <iomanip.h>
#include <conio.h>

const int rozmiar = 10;

int tablica[rozmiar][rozmiar];

czytaj_dane()
{
int i, j;

randomize();

for (i = 0; i < rozmiar; i++)
{
for (j = 0; j < rozmiar; j++)
{
if (i == j )
tablica[i][j] = random(10);
else
tablica[i][j] = 0;
}
}
}

przetworz_dane()
{
int i, suma=0;

for (i = 0; i < rozmiar; i++)
{
suma = suma+tablica[i][i];
}
cout << "Suma elementow na przekatnej wynosi " << suma << "." << endl;
}
```

```
wyswietl_wynik()
{
int i, j;

cout << "Zawartosc tablicy:" << endl;
cout << endl;

for (i = 0; i < rozmiar; i++)
{
for (j = 0; j < rozmiar; j++)
{
cout << tablica[i][j] << " ";
}
cout << endl;
}
}

main()
{
czytaj_dane();
przetworz_dane();
wyswietl_wynik();

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 5.3.

```
Suma elementow na przekatnej wynosi 40.
Zawartosc tablicy:

3 0 0 0 0 0 0 0 0 0
0 3 0 0 0 0 0 0 0 0
0 0 2 0 0 0 0 0 0 0
0 0 0 9 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 9 0 0 0 0
0 0 0 0 0 0 4 0 0 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 5 0
0 0 0 0 0 0 0 0 0 2
```

Rysunek 5.3. *Efekt działania programu Zadanie 5.6*

ZADANIE

5.7 Napisz program, który sortuje *n* liczb wczytanych z klawiatury (w zadaniu jest ich sześć). Powinien on zawierać trzy bezparametrowe funkcje: `czytaj_dane()`, `przetworz_dane()` i `wyswietl_wynik()`. Funkcja `czytaj_dane()` czyta dane wprowadzone z klawiatury i umieszcza je w tablicy o nazwie `liczby`. Funkcja `przetworz_dane()` sortuje je według wybranego algorytmu (w programie należy zastosować algorytm sortowania bąbelkowego). Funkcja `wyswietl_wynik()` prezentuje natomiast zawartość posortowanej tablicy `liczby` na ekranie monitora.

Przykładowe rozwiązanie — listing 5.7

```
#include <iostream.h> // Zadanie 5.7
#include <iomanip.h>
#include <conio.h>

const int n = 6; // ilosc liczb
int liczby[n];
int x, i, j;

czytaj_dane()
{
cout << "Podaj " << n << " liczb calkowitych." << endl;

for (i = 0; i <= n-1; i++)
{
cin >> liczby[i];
}
}

przetworz_dane() // algorytm sortowania babelkowego
{
for (i = 1; i <= n-1; i++)
{
for (j = n-1; j >= i; j--)
{

if (liczby[j-1] > liczby[j])
{
x = liczby[j-1];
liczby[j-1] = liczby[j];
liczby[j] = x;
}
}
}
}
```

```
wyswietl_wynik()
{
cout << "Liczby uporzadkowane: " << endl;

for (i = 0; i <= n-1; i++)
{
cout << liczby[i] << " ";
}
cout << endl;
}

main()
{

cout << "Program sortuje " << n << " liczb calkowitych." << endl;

czytaj_dane();
przetworz_dane();
wyswietl_wynik();

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 5.4.

```
Program sortuje 6 liczb calkowitych.
Podaj 6 liczb calkowitych.
5
12
67
456
3
-10
Liczby uporzadkowane:
-10 3 5 12 67 456
```

Rysunek 5.4. *Efekt działania programu Zadanie 5.7*

ZADANIE

5.8 Napisz program, który losuje `n` liczb całkowitych (w zadaniu `n = 10`) z przedziału od 0 do 99, a następnie znajduje najmniejszą i największą z nich. Powinien on zawierać trzy bezparametrowe funkcje: `czytaj_dane()`, `przetworz_dane()` i `wyswietl_wynik()`. Funkcja `czytaj_dane()` losuje *n* całkowitych liczb od 0 do 99. `przetworz_dane()` porównuje je ze sobą i ustala, która z nich jest najmniejsza oraz największa. Funkcja `wyswietl_wynik()` prezentuje natomiast największą i najmniejszą liczbę z wylosowanego zbioru.

Przykładowe rozwiązanie — listing 5.8

```
#include <iostream.h> // Zadanie 5.8
#include <iomanip.h>
#include <conio.h>

const int n = 10; // ilosc liczb
int liczba, mini, maxi;

czytaj_dane()
{
cout << "Program losuje " << n << " liczb calkowitych z przedzialu od 0
↪do 99," << endl;
cout << "a nastepnie znajduje najwieksza i najmniejsza liczbe." << endl;
↪randomize();
mini = random(100);
maxi = mini;
cout << "Wylosowane liczby: ";
cout << maxi << "  ";
}

przetworz_dane()
{
int i;

for (i = 1; i <= n-1; i++)
{
liczba = random(100);
cout << liczba << "  ";
if (maxi < liczba) maxi = liczba;
if (liczba < mini) mini = liczba;
}
}

wyswietl_wynik()
{
```

```
cout << endl;
cout << "Najwieksza liczba to " << maxi << "." << endl;
cout << "Najmniejsza liczba to " << mini << "." << endl;
}

main()
{
czytaj_dane();
przetworz_dane();
wyswietl_wynik();

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu dla wylosowanych liczb można zobaczyć na rysunku 5.5.

```
Program losuje 10 liczb calkowitych z przedzialu od 0 do 99,
a nastepnie znajduje najwieksza i najmniejsza liczbe.
Wylosowane liczby: 40  21  62  7  74  13  2  75  12  23
Najwieksza liczba to 75.
Najmniejsza liczba to 2.
```

Rysunek 5.5. *Efekt działania programu Zadanie 5.8*

ZADANIE

5.9 Napisz program, który dla x zmieniającego się od 0 do 5 z krokiem 0,5 oblicza wartość funkcji $y = x^2+1$.

Przykładowe rozwiązanie — listing 5.9

```
#include <iostream.h> // Zadanie 5.9
#include <iomanip.h>
#include <conio.h>

const n = 10;
int i;
float x = 0, y, krok = 0.5;

float oblicz(float a)
{
float oblicz;

oblicz = a*a+1;
```

```
return oblicz;
}

main()
{
cout << " x" << "\t" << " y" << endl;
cout << "==============" << endl;
for (i = 0; i <= n; i++)
{
y = oblicz(x);
cout << fixed;
cout << setprecision(2);
cout << x << "\t" << y << endl;
x=x+krok;
}

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 5.6.

```
x         y
==============
0.00    1.00
0.50    1.25
1.00    2.00
1.50    3.25
2.00    5.00
2.50    7.25
3.00   10.00
3.50   13.25
4.00   17.00
4.50   21.25
5.00   26.00
```

Rysunek 5.6. *Efekt działania programu Zadanie 5.9*

6

Programowanie obiektowe

W tym rozdziale przedstawimy typowe zadania wraz z przykładowymi rozwiązaniami z wykorzystaniem zasad programowania obiektowego.

Język C++ pozwala na programowanie obiektowe (ang. *object-oriented programming*). Jest to taki paradygmat programowania, w którym programy definiuje się za pomocą obiektów — elementów łączących *stan* (są to dane nazywane **polami**) i *zachowanie* (są to **metody** — w C++ są nimi funkcje służące do wykonywania na tych danych określonych zadań, których deklaracja występuje w ramach obiektu). W języku C++ podstawowym pojęciem programowania obiektowego jest klasa (ang. *class*). Klasa definiuje projekt i strukturę obiektu. Jej schemat ma następującą postać:

```
class nazwa_klasy
{
   // pola
   // metody
}
```

Pola służą do przechowywania danych i zmiennych określonych typów — zarówno prostych, jak i obiektowych. Metody służą natomiast do wykonywania operacji na tych danych. W klasach możemy wyróżnić między innymi następujące elementy: stałe, zmienne składowe, zmienne statyczne, metody, konstruktory i destruktory.

Konstruktor jest specjalną metodą stosowaną przy tworzeniu obiektu danej klasy. Jest on używany do inicjalizacji zmiennych składowych obiektu oraz do przydzielania pamięci potrzebnej do jego działania. **Destruktor** to natomiast specjalna metoda wywoływana tuż przed zwolnieniem obiektu. Jest on przeciwieństwem konstruktora i jest używany do zwolnienia pamięci zajmowanej przez obiekt.

Jeśli w klasie nie zadeklarowano jawnie konstruktora i destruktora, to zostaną one utworzone automatycznie przez kompilator i użyte w chwili tworzenia i niszczenia obiektu.

Język C++ posiada następujące trzy poziomy dostępu (nadawane przez modyfikatory lub specyfikatory) do swoich składników:

- **prywatny** (ang. *private*) — słowo kluczowe `private` definiuje pola i metody klasy dostępne tylko dla jej składników, co oznacza, że metody w innych klasach nie mają dostępu do tak zadeklarowanych elementów;
- **publiczny** (ang. *public*) — słowo kluczowe `public` definiuje pola i metody dostępne z dowolnego miejsca programu, co oznacza, że metody w innych klasach mają dostęp do składników zadeklarowanych przy użyciu tego modyfikatora;
- **chroniony** (ang. *protected*) — słowo kluczowe `protected` stosowane jest tylko w przypadku dziedziczenia.

Zadania z programowania obiektowego przedstawione w dalszej części tej książki będziemy rozwiązywali według poniższego schematu.

```
#include <iostream.h>
.....

class nazwa_klasy
{
   public:
   deklaracja zmiennych;

   czytaj_dane(); // prototyp metody czytaj_dane()
   przetworz_dane(); // prototyp metody przetworz_dane()
   wyswietl_wynik(); // prototyp metody wyswietl_wynik()
};

nazwa_klasy::czytaj_dane() // definicja metody czytaj_dane()
{
.............
}
```

```
nazwa_klasy::przetworz_dane() // definicja metody przetworz_dane()
{
..............
}

nazwa_klasy::wyswietl_wynik() // definicja metody wyswietl_wynik()
{
..............
}

main()
{
   nazwa_klasy zmienna; // utworzenie obiektu zmienna

   zmienna.czytaj_dane(); // wywolanie metody czytaj_dane()
   zmienna.przetworz_dane(); // wywolanie metody przetworz_dane()
   zmienna.wyswietl_wynik(); // wywolanie metody wyswietl_wynik()
   ......................
}
```

:: to operator zasięgu identyfikujący klasę, do której należy metoda.

Klasa o nazwie `nazwa_klasy` zawiera oprócz deklaracji zmiennych trzy metody. `czytaj_dane()` zajmuje się tylko odczytem danych. Za ich przetworzenie odpowiedzialna jest metoda `przetworz_dane()`. Ostatnia z metod, `wyswietl_wynik()`, prezentuje przetworzone dane (wyniki) np. na ekranie monitora. Metody mogą być z parametrem lub bezparametrowe w zależności od upodobań programisty. Powyższy schemat zilustrujemy przykładem znanego nam programu obliczającego pole prostokąta.

ZADANIE

6.1 Napisz zgodnie z zasadami programowania obiektowego program, który oblicza pole prostokąta. Klasa powinna zawierać trzy metody:

- `czytaj_dane()` — umożliwia wprowadzenie do programu wartości boków `a` i `b` z klawiatury. W programie należy przyjąć, że boki oraz zmienna `pole` są typu `float` (rzeczywistego).
- `przetworz_dane()` — oblicza pole prostokąta według wzoru `pole = a*b`.
- `wyswietl_wynik()` — wyświetla wartości boków `a` i `b` oraz zmiennej `pole` w określonym formacie. Dla tych trzech zmiennych należy przyjąć format wyświetlania ich na ekranie z dwoma miejscami po kropce.

Przykładowe rozwiązanie — listing 6.1

```
#include <iostream.h> // Zadanie 6.1
#include <iomanip.h>
#include <conio.h>

class pole_prostokata // definicja klasy pole_prostokata
{
public:
float a, b, pole;

czytaj_dane(); // prototyp funkcji czytaj_dane
przetworz_dane(); // prototyp funkcji przetworz_dane
wyswietl_wynik(); // prototyp funkcji wyswietl_wynik
};

pole_prostokata::czytaj_dane() // definicja metody czytaj_dane()
{
cout << "Program oblicza pole prostokata." << endl;
cout << "Podaj bok a." << endl;
cin >> a;
cout << "Podaj bok b." << endl;
cin >> b;
}

pole_prostokata::przetworz_dane() // definicja metody przetworz_dane()
{
pole = a*b;
}

pole_prostokata::wyswietl_wynik() // definicja metody wyswietl_wynik()
{
cout << "Pole prostokata o boku a = ";
cout << fixed;
cout << setprecision(2);
cout << a << " i boku b = " << b;
cout << " wynosi " << pole << "." << endl;
}

main()
{

pole_prostokata pole; // utworzenie obiektu pole

pole.czytaj_dane(); // wywolanie metody czytaj_dane()
pole.przetworz_dane(); // wywolanie metody przetworz_dane()
pole.wyswietl_wynik(); // wywolanie metody wyswietl_wynik()

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Definicja klasy `pole_prostokata` jest następująca:

```
class pole_prostokata // definicja klasy pole_prostokata
{
public:
float a, b, pole;

czytaj_dane(); // prototyp funkcji czytaj_dane
przetworz_dane(); // prototyp funkcji przetworz_dane
wyswietl_wynik(); // prototyp funkcji wyswietl_wynik
};
```

Klasa jest publiczna i zawiera, oprócz trzech rzeczywistych zmiennych, trzy metody: `czytaj_dane()`, `przetworz_dane()` oraz `wyswietl_wynik()`.

W naszym programie metoda `czytaj_dane()`:

```
pole_prostokata::czytaj_dane() // definicja metody czytaj_dane()
{
cout << "Program oblicza pole prostokata." << endl;
cout << "Podaj bok a." << endl;
cin >> a;
cout << "Podaj bok b." << endl;
cin >> b;
}
```

wczytuje z klawiatury wartości boków `a` i `b`.

Metoda `przetworz_dane()`:

```
pole_prostokata::przetworz_dane() // definicja metody przetworz_dane()
{
   pole = a*b;
}
```

oblicza pole prostokąta, korzystając ze wzoru `pole = a*b`.

Natomiast metoda `wyswietl_wynik()`:

```
pole_prostokata::wyswietl_wynik() // definicja metody wyswietl_wynik()
{
cout << "Pole prostokata o boku a = ";
cout << fixed;
cout << setprecision(2);
cout << a << " i boku b = " << b;
cout << " wynosi " << pole << "." << endl;
}
```

prezentuje wartości boków `a` i `b` oraz wartość zmiennej `pole` w określonym formacie.

Linijka kodu

```
pole_prostokata pole; // utworzenie obiektu pole
```

pozwala utworzyć obiekt o nazwie `pole`.

W programie głównym zostają wywołane wszystkie trzy metody:

```
pole.czytaj_dane(); // wywolanie metody czytaj_dane()
pole.przetworz_dane(); // wywolanie metody przetworz_dane()
pole.wyswietl_wynik(); // wywolanie metody wyswietl_wynik()
```

Ponieważ wszystkie składowe klasy są publiczne, program ma do nich dostęp przy użyciu operatora wyboru składowej oznaczonego kropką (.). Prawda, że wszystko jest jasne i proste? Następne zadania w tej książce spróbujemy rozwiązać według powyższego schematu.

Rezultat działania programu można zobaczyć na rysunku 6.1.

```
Program oblicza pole prostokata.
Podaj bok a.
3
Podaj bok b.
4
Pole prostokata o boku a = 3.00 i boku b = 4.00 wynosi 12.00.
```

Rysunek 6.1. *Efekt działania programu Zadanie 6.1*

ZADANIE

6.2

Napisz zgodnie z zasadami programowania obiektowego program, który oblicza pierwiastki równania kwadratowego $ax^2+bx+c = 0$ z wykorzystaniem instrukcji wyboru `switch ... case`. Klasa powinna zawierać trzy metody:

- `czytaj_dane()` — jest odpowiedzialna za wczytanie danych do programu oraz obsłużenie sytuacji, kiedy `a = 0`. Zmienne `a`, `b` oraz `c` to liczby rzeczywiste wprowadzane z klawiatury.
- `przetworz_dane()` — odpowiada za wykonanie niezbędnych obliczeń.
- `wyswietl_wynik()` — jest odpowiedzialna za pokazanie wyników na ekranie monitora. Dla zmiennych `a`, `b`, `c`, `x1` oraz `x2` należy przyjąć format wyświetlania ich z dwoma miejscami po kropce.

Przykładowe rozwiązanie — listing 6.2

```
#include <iostream.h> // Zadanie 6.2
#include <iomanip.h>
#include <conio.h>

class trojmian // definicja klasy trojmnian
{
public:
float a, b, c, delta, x1, x2;
char liczba_pierwiastkow;
czytaj_dane(); // prototyp funkcji czytaj_dane()
przetworz_dane(); // prototyp funkcji przetworz_dane()
wyswietl_wynik(); // prototyp funkcji wyswietl_wynik()
};

trojmian::czytaj_dane()
{
cout << "Program oblicza pierwiastki rownania kwadratowego" << endl;
cout << "dla dowolnych wspolczynnikow a, b, c." << endl;
cout << "Podaj a." << endl;
cin >> a;

if (a == 0)
{
cout << "Niedozwolona wartosc wspolczynnika. Nacisnij dowolny klawisz."
↪<< endl;
getch(); // czeka na nacisniecie dowolnego klawisza
exit(1); // wyjscie z programu
}
else
{
cout << "Podaj b." << endl;
cin >> b;
cout << "Podaj c." << endl;
cin >> c;
}
}

trojmian::przetworz_dane()
{
delta = b*b-4*a*c;

if (delta < 0) liczba_pierwiastkow = 0;
if (delta == 0) liczba_pierwiastkow = 1;
if (delta > 0) liczba_pierwiastkow = 2;

switch(liczba_pierwiastkow)
{
case 1 : x1 = -b/(2*a);
```

```
break;
case 2 : { x1 = (-b-sqrt(delta))/(2*a);
           x2 = (-b+sqrt(delta))/(2*a);
         }
break;
}
}

trojmian::wyswietl_wynik()
{
cout << "Dla wprowadzonych liczb:" << endl;
cout << "a = " << a << "," << endl;
cout << "b = " << b << "," << endl;
cout << "c = " << c << "," << endl;

switch(liczba_pierwiastkow)
{
case 0 : cout << "brak pierwiastkow rzeczywistych." << endl;
break;
case 1 : cout << "trojmian ma jeden pierwiastek podwojny x1 = " << x1
↪<< "." << endl;
break;
case 2 : { cout << "trojmian ma dwa pierwiastki:" << endl;
           cout << "x1 = " << x1 << "," << endl;
           cout << "x2 = " << x2 << "." << endl;
         }
break;
}
}

main()
{

trojmian trojmian1; // utworzenie obiektu trojmian1

cout << fixed << setprecision(2);

trojmian1.czytaj_dane();
trojmian1.przetworz_dane();
trojmian1.wyswietl_wynik();

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 6.2.

```
Program oblicza pierwiastki rownania kwadratowego
dla dowolnych wspolczynnikow a, b, c.
Podaj a.
1.02
Podaj b.
5
Podaj c.
4
Dla wprowadzonych liczb:
a = 1.02,
b = 5.00,
c = 4.00,
trojmian ma dwa pierwiastki:
x1 = -3.90,
x2 = -1.01.
```

Rysunek 6.2. *Efekt działania programu Zadanie 6.2*

ZADANIE

6.3 Napisz zgodnie z zasadami programowania obiektowego program, który w tablicy 10×10 umieszcza losowo na przekątnej liczby od 0 do 9, a poza nią zera. Dodatkowo oblicza on sumę liczb znajdujących się na przekątnej. Klasa powinna zawierać trzy metody:

- `czytaj_dane()` — umieszcza dane w tablicy;
- `przetworz_dane()` — oblicza i wyświetla sumę liczb znajdujących się na przekątnej;
- `wyswietl_wynik()` — pokazuje zawartość tablicy na ekranie monitora.

Przykładowe rozwiązanie — listing 6.3

```
#include <iostream.h> // Zadanie 6.3
#include <iomanip.h>
#include <conio.h>

const int rozmiar = 10;
int macierz[rozmiar][rozmiar];

class matrix // definicja klasy matrix
```

```
{
public:
czytaj_dane(int macierz[rozmiar][rozmiar], int rozmiar);
  // prototyp funkcji czytaj_dane()
przetworz_dane(int macierz[rozmiar][rozmiar], int rozmiar);
  // prototyp funkcji przetworz_dane()
wyswietl_wynik(int macierz[rozmiar][rozmiar], int rozmiar);
  // prototyp funkcji wyswietl_wynik()
};

matrix::czytaj_dane(int tablica[rozmiar][rozmiar], int rozmiar)
{
int i, j;
randomize();
for (i = 0; i < rozmiar; i++)
{
for (j = 0; j < rozmiar; j++)
{
if (i == j)
tablica[i][j] = random(10);
else
tablica[i][j] = 0;
}
}
}

matrix::przetworz_dane(int tablica[rozmiar][rozmiar], int rozmiar)
{
int i, suma = 0;
for (i = 0; i < rozmiar; i++)
{
suma = suma+tablica[i][i];
}
cout << "Suma elementow na przekatnej wynosi " << suma << "." << endl;
}

matrix::wyswietl_wynik(int tablica[rozmiar][rozmiar], int rozmiar)
{
int i, j;
cout << endl;
cout << "Zawartosc tablicy:" << endl;
cout << endl;
for (i = 0; i < rozmiar; i++)
{
for (j = 0; j < rozmiar; j++)
{
cout << tablica[i][j] << " ";
}
cout << endl;
}
}
```

```
main()
{
int tablica[rozmiar][rozmiar];
matrix matrix1; // utworzenie obiektu matrix1

matrix1.czytaj_dane(tablica, rozmiar);
matrix1.przetworz_dane(tablica, rozmiar);
matrix1.wyswietl_wynik(tablica, rozmiar);

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 6.3.

```
Suma elementow na przekatnej wynosi 45.

Zawartosc tablicy:

3 0 0 0 0 0 0 0 0 0
0 9 0 0 0 0 0 0 0 0
0 0 5 0 0 0 0 0 0 0
0 0 0 3 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 6 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 9 0 0
0 0 0 0 0 0 0 0 2 0
0 0 0 0 0 0 0 0 0 8
```

Rysunek 6.3. *Efekt działania programu Zadanie 6.3*

ZADANIE

6.4 Napisz zgodnie z zasadami programowania obiektowego program, który sortuje `n` liczb (w zadaniu jest ich sześć). Klasa powinna zawierać trzy metody:

- `czytaj_dane()` — odczytuje dane i umieszcza je w tablicy o nazwie `liczby`;
- `przetworz_dane()` — sortuje dane, korzystając z wybranego algorytmu (w programie zastosowano algorytm sortowania bąbelkowego);
- `wyswietl_wynik()` — prezentuje zawartość posortowanej tablicy `liczby` na ekranie monitora.

Przykładowe rozwiązanie — listing 6.4

```
#include <iostream.h> // Zadanie 6.4
#include <iomanip.h>
#include <conio.h>

const int n = 6;
int liczby[n];

class sortowanie // definicja klasy sortowanie
{
public:
czytaj_dane(); // prototyp funkcji czytaj_dane()
przetworz_dane(); // prototyp funkcji przetworz_dane()
wyswietl_wynik(); // prototyp funkcji wyswietl_wynik()
};

sortowanie::czytaj_dane()
{
int i;

liczby[0] = 574;
liczby[1] = 303;
liczby[2] = -134;
liczby[3] = 125;
liczby[4] = 80;
liczby[5] = 236;

cout << "Liczby nieposortowane: " << endl;
for (i = 0; i < n; i++)
{
cout << liczby[i] << "  ";
}
cout << endl;
}

sortowanie::przetworz_dane() // algorytm sortowania babelkowego
{
int i, j, x;

for (i = 1; i <= n-1; i++)
{
for (j = n-1; j >= i; --j)
{
if (liczby[j-1] > liczby[j])
{
x = liczby[j-1];
liczby[j-1] = liczby[j];
liczby[j] = x;
}
```

```
} // j
} // i
}

sortowanie::wyswietl_wynik()
{
int i;

cout << endl;
cout << "Liczby posortowane: " << endl;
for (i = 0; i < n; i++)
{
cout << liczby[i] << "  ";
}
cout << endl;
}

main()
{
sortowanie babelki;  // powstaje obiekt babelki

babelki.czytaj_dane();
babelki.przetworz_dane();
babelki.wyswietl_wynik();

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 6.4.

```
Liczby nieposortowane:
574  303  -134  125  80  236

Liczby posortowane:
-134  80  125  236  303  574
```

Rysunek 6.4. *Efekt działania programu Zadanie 6.4*

7

Pliki tekstowe

W tym rozdziale przedstawimy typowe zadania wraz z przykładowymi rozwiązaniami z wykorzystaniem plików tekstowych.

Pliki tekstowe zawierają informację niezakodowaną, bezpośrednio czytelną. Są one plikami o dostępie sekwencyjnym.

W języku C++ plik otwiera się przez połączenie go ze strumieniem. Istnieją trzy typy strumieni:

- wejściowe,
- wyjściowe,
- wejściowo-wyjściowe.

Warunkiem wykonywania na plikach operacji wejścia-wyjścia jest dołączenie do programu pliku nagłówkowego *fstream.h*. Aby otworzyć strumień wejściowy, należy zadeklarować go jako obiekt klasy `ifstream` (*in file stream*). Strumienie wyjściowe są obiektami klasy `ofstream` (*out file stream*). Obiekty klasy `fstream` to z kolei strumienie, na których będą wykonywane zarówno operacje wejścia, jak i wyjścia.

W języku C++ operacje wejścia-wyjścia dotyczące plików są analogiczne do standardowych operacji tego typu. Chcąc zapisać dane do pliku, trzeba utworzyć obiekt strumienia klasy `ofstream` reprezentujący

dany plik i skorzystać z operatora wstawiania <<. Podobnie, chcąc odczytać dane z pliku, należy utworzyć obiekt strumienia klasy `ifstream`, a następnie czytać dane za pomocą operatora pobierania >>.

ZADANIE

7.1 Napisz zgodnie z zasadami programowania obiektowego program, który wczytuje z klawiatury imię i nazwisko, zapisuje te dane do pliku tekstowego *dane.txt*, a następnie odczytuje je z niego i wyświetla na ekranie komputera. Klasa powinna zawierać trzy metody:

- `czytaj_dane()` — wczytuje z klawiatury imię i nazwisko;
- `zapisz_dane_do_pliku()` — zapisuje imię i nazwisko do pliku tekstowego *dane.txt*;
- `czytaj_dane_z_pliku()` — odczytuje dane z pliku *dane.txt* i wyświetla je na ekranie komputera.

Przykładowe rozwiązanie — listing 7.1

```
#include <iostream.h> // Zadanie 7.1
#include <fstream.h>
#include <conio.h>

class plik // definicja klasy plik
{
 public:
 char dane[20], dane1[20];

 czytaj_dane(); // prototyp funkcji czytaj_dane()
 zapisz_dane_do_pliku(); // prototyp funkcji zapisz_dane_do_pliku()
 czytaj_dane_z_pliku(); // prototyp funkcji czytaj_dane_z_pliku();
};

plik::czytaj_dane() // definicja funkcji czytaj_dane()
{
 cout << "Podaj imie i nazwisko." << endl;
 cin.getline(dane, sizeof(dane));
}

plik::zapisz_dane_do_pliku() // definicja funkcji zapisz_dane_do_pliku()
{
 cout << "Zapisujemy dane do pliku." << endl;
 ofstream plik_zapis("dane.txt"); // tworzymy i otwieramy plik
 plik_zapis << dane << endl; // zapisujemy dane do pliku
 plik_zapis.close();
}
```

```
plik::czytaj_dane_z_pliku() // definicja funkcji czytaj_dane_z_pliku();
{
 cout << "Odczytujemy dane1 z pliku." << endl;
 ifstream plik_odczyt("dane.txt"); // otwieramy plik, ktory juz istnieje

 while (!plik_odczyt.eof())
 {
  plik_odczyt >> dane1; // odczytujemy dane1 z pliku i wyswietlamy je
  cout << dane1 << " ";
 }

 plik_odczyt.close(); // zamykamy plik
}

main()
{
 plik plik1; // powstaje obiekt plik1

 plik1.czytaj_dane();
 plik1.zapisz_dane_do_pliku();
 plik1.czytaj_dane_z_pliku();

 getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Zwróćmy uwagę na to, że w programie zadeklarowano dwie zmienne łańcuchowe:

```
char dane[20], dane1[20];
```

Zmienna o nazwie `dane` przechowuje dane przed zapisaniem ich do pliku tekstowego, natomiast `dane1` przechowuje dane odczytane z pliku.

Metoda `czytaj_dane()`:

```
plik::czytaj_dane() // definicja funkcji czytaj_dane()
{
 cout << "Podaj imie i nazwisko." << endl;
 cin.getline(dane, sizeof(dane));
}
```

wczytuje z klawiatury dane, czyli imię i nazwisko. Funkcja `cin.getline()` umożliwia wczytanie całej linii tekstu, natomiast zastosowanie operatora `sizeof()` gwarantuje poprawność rozmiaru.

Metoda `zapisz_dane_do_pliku()`:

```
plik::zapisz_dane_do_pliku() // definicja funkcji zapisz_dane_do_pliku()
{
 cout << "Zapisujemy dane do pliku." << endl;
 ofstream plik_zapis("dane.txt"); // tworzymy i otwieramy plik
 plik_zapis << dane << endl; // zapisujemy dane do pliku
 plik_zapis.close();
}
```

zapisuje dane do pliku *dane.txt*.

Ostatnia z metod, `czytaj_dane_z_pliku()`:

```
plik::czytaj_dane_z_pliku() // definicja funkcji czytaj_dane_z_pliku();
{
 cout << "Odczytujemy dane1 z pliku." << endl;
 ifstream plik_odczyt("dane.txt"); // otwieramy plik, ktory juz istnieje

 while (!plik_odczyt.eof())
 {
  plik_odczyt >> dane1; // odczytujemy dane1 z pliku i wyswietlamy je
  cout << dane1 << " ";
 }

 plik_odczyt.close(); // zamykamy plik
}
```

odczytuje `dane1` z pliku i wyświetla je na ekranie monitora. Wszystkie trzy metody zostają wywołane w programie głównym.

Rezultat działania programu można zobaczyć na rysunku 7.1.

```
Podaj imie i nazwisko.
Jan Kowalski
Zapisujemy dane do pliku.
Odczytujemy dane1 z pliku.
Jan Kowalski
```

Rysunek 7.1. *Efekt działania programu Zadanie 7.1*

ZADANIE

7.2 Napisz zgodnie z zasadami programowania obiektowego program, który tablicę 10×10 o postaci

1	0	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	0	**1**	0	0	0	0	0	0	0
0	0	0	**1**	0	0	0	0	0	0
0	0	0	0	**1**	0	0	0	0	0
0	0	0	0	0	**1**	0	0	0	0
0	0	0	0	0	0	**1**	0	0	0
0	0	0	0	0	0	0	**1**	0	0
0	0	0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0	0	**1**

zapisuje do pliku tekstowego *dane.txt*, a następnie zapisane dane odczytuje i wyświetla na ekranie komputera. Klasa powinna zawierać trzy metody:

- `czytaj_dane()` — tworzy tablicę 10×10;
- `zapisz_dane_do_pliku()` — zapisuje tablicę 10×10 do pliku tekstowego *dane.txt*;
- `czytaj_dane_z_pliku()` — odczytuje tablicę 10×10 z pliku *dane.txt* i wyświetla ją na ekranie komputera.

Przykładowe rozwiązanie — listing 7.2

```
#include <iostream.h> // Zadanie 7.2
#include <fstream.h>
#include <conio.h>

const int rozmiar = 10;

class matrix // definicja klasy matrix
{
 public:
 czytaj_dane(int tablica[rozmiar][rozmiar], int rozmiar);
 zapisz_dane_do_pliku(int tablica[rozmiar][rozmiar], int rozmiar);
 czytaj_dane_z_pliku(int tablica1[rozmiar][rozmiar], int rozmiar);
};

matrix::czytaj_dane(int tablica[rozmiar][rozmiar], int rozmiar)
{
 int i, j;

 for (i = 0; i < rozmiar; i++) // tworzymy tablice 10x10
 {
  for (j = 0; j < rozmiar; j++)
```

```
  {
   if (i == j)
    tablica[i][j] = 1;
   else
    tablica[i][j] = 0;
  } // j
 } // i
}

matrix::zapisz_dane_do_pliku(int tablica[rozmiar][rozmiar], int rozmiar)
{
 int i, j;

 cout << "Zapisujemy tablice 10x10 do pliku." << endl;

 ofstream plik_zapis("dane.txt"); // tworzymy i otwieramy plik do zapisu

 for (i = 0; i < rozmiar; i++) // zapisujemy tablice 10x10
 {
  for (j = 0; j < rozmiar; j++)
  {
   cout << tablica[i][j] << " ";
   plik_zapis << tablica[i][j];
  } // j
  cout << endl;
 } // i
 plik_zapis.close(); // zamykamy plik
}

matrix::czytaj_dane_z_pliku(int tablica1[rozmiar][rozmiar], int rozmiar)
{
 int i, j;

 cout << endl;
 cout << "Odczytujemy tablice 10x10 z pliku." << endl;

 ifstream plik_odczyt("dane.txt"); // otwieramy istniejacy plik

 for (i = 0; i < rozmiar; i++)
 {
  for (j = 0; j < rozmiar; j++)
  {
   plik_odczyt >> tablica1[i][j]; // odcztujemy tablice z pliku
   cout << tablica1[i][j] << " ";
  } // j
  cout << endl;
 } // i
 plik_odczyt.close(); //zamykamy plik
}

main()
{
 int tab[rozmiar][rozmiar];
```

```
 matrix matrix1; // powstaje obiekt matrix1
 matrix1.czytaj_dane(tab, rozmiar);
 matrix1.zapisz_dane_do_pliku(tab, rozmiar);
 matrix1.czytaj_dane_z_pliku(tab, rozmiar);
getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Zwróćmy uwagę, że w programie zadeklarowano dwie tablice:

```
int tablica[rozmiar][rozmiar]
```

oraz

```
int tablica1[rozmiar][rozmiar]
```

Tablica o nazwie `tablica[][]` przechowuje dane przed zapisaniem ich do pliku, natomiast `tablica1[][]` przechowuje dane odczytane z niego.

Rezultat działania programu można zobaczyć na rysunku 7.2.

```
Zapisujemy tablice 10x10 do pliku.
1 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1

Odczytujemy tablice 10x10 z pliku.
1 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1
```

Rysunek 7.2. *Efekt działania programu Zadanie 7.2*

ZADANIE

7.3

Napisz zgodnie z zasadami programowania obiektowego program, który tablicę a o wymiarach 10×10 i postaci

0	0	0	0	0	0	0	0	0	0
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	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

przekształca w tablicę b

0	**1**	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0

i zapisuje wynik do pliku tekstowego *dane.txt*, a następnie odczytuje z niego tablicę i wyświetla ją na ekranie. Klasa powinna zawierać cztery metody:

- `czytaj_dane()` — tworzy tablicę `a` o wymiarach 10×10;
- `przetworz_dane()` — przepisuje elementy tablicy `a` o wymiarach 10×10 do tablicy `b` o tej samej wielkości;
- `zapisz_dane_do_pliku()` — zapisuje tablicę `b` o wymiarach 10×10 do pliku;
- `czytaj_dane_z_pliku()` — odczytuje tablicę `c` o wymiarach 10×10 z pliku i wyświetla ją na ekranie.

Przykładowe rozwiązanie — listing 7.3

```
#include <iostream.h> // Zadanie 7.3
#include <fstream.h>
#include <conio.h>

const int n = 10;

class matrix // definicja klasy matrix
{
 public:
 int a[n][n], b[n][n], c[n][n];
 czytaj_dane(); // prototyp funkcji czytaj_dane()
 przetworz_dane(); // prototyp funkcji przetworz_dane()
 zapisz_dane_do_pliku(); // prototyp funkcji zapisz_dane_do_pliku()
 czytaj_dane_z_pliku(); // prototyp funkcji czytaj_dane_z_pliku();
};

matrix::czytaj_dane()
{
 int i, j;

 cout << "Tworzymy tablice a." << endl;

 for (i = 0; i < n; i++)
 {
  for (j = 0; j < n; j++)
  {
   if (i == 1)
    a[i][j] = 1;
   else
    a[i][j] = 0;
    cout << a[i][j] << " ";
  } // j
   cout << endl;
```

```
 }  // i
 // koniec wpisywania
}

matrix::przetworz_dane()
{
 int i, j;

 cout << endl;
 cout << "Przepisujemy elementy z tablicy a do tablicy b." << endl;
 for (i = 0; i < n; i++)
 {
  for (j = 0; j < n; j++)
  {
   b[i][j] = a[j][i]; // przepisujemy zawartosc tablicy a do tablicy b
  } // j
 } // i
}

matrix::zapisz_dane_do_pliku()
{
 int i, j;

 cout << "Zapisujemy tablice b do pliku dane.txt." << endl;

 ofstream plik_zapis("dane.txt"); // tworzymy i otwieramy plik do zapisu

 for (i = 0; i < n; i++)
 {
  for (j = 0; j < n; j++)
  {
   cout << b[i][j] << " ";
   plik_zapis << b[i][j] << endl; // zapisujemy tablice do pliku
  } // j
   cout << endl;
 }  // i
  // koniec zapisywania do pliku
 plik_zapis.close(); // zamykamy plik
}

matrix::czytaj_dane_z_pliku()
{
 int i, j;

 cout << endl;
 cout << "Odczytujemy tablice c z pliku dane.txt i wyswietlamy ja na
 ↪ekranie." << endl;
```

```
 ifstream plik_odczyt("dane.txt"); // otwieramy istniejacy plik

 for (i = 0; i < n; i++)
 {
  for (j = 0; j < n; j++)
  {
   plik_odczyt >> c[i][j]; // odczytujemy tablice z pliku
   cout << c[i][j] << " ";
  } // j
   cout << endl;
 } // i
  // koniec odczytywania z pliku

 plik_odczyt.close(); // zamykamy plik
}

main()
{
 matrix matrix1; // powstaje obiekt matrix1

 matrix1.czytaj_dane();
 matrix1.przetworz_dane();
 matrix1.zapisz_dane_do_pliku();
 matrix1.czytaj_dane_z_pliku();

getch(); // czeka na nacisniecie dowolnego klawisza
}
```

Rezultat działania programu można zobaczyć na rysunku 7.3 (rysunek przedstawiony na następnej stronie).

```
Tworzymy tablice a.
0 0 0 0 0 0 0 0 0 0
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 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

Przepisujemy elementy z tablicy a do tablicy b.
Zapisujemy tablice b do pliku dane.txt.
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0

Odczytujemy tablice c z pliku dane.txt i wyswietlamy ja na ekranie.
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
```

Rysunek 7.3. *Efekt działania programu Zadanie 7.3*