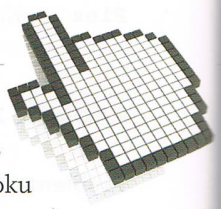
3 Projektowanie rozwiązania prostych problemów w języku C++



Język programowania **C** powstał w latach 70. ubiegłego stulecia. Szybko stał się popularnym (obok języka **Pascal**) językiem **programowania strukturalnego**, głównie do programowania systemów operacyjnych i aplikacji. W 1983 roku pojawiła się kolejna[4] wersja o nazwie **C++**, rozszerzona o mechanizmy **programowania obiektowego**[5].

Program napisany w języku C++ jest zbudowany z funkcji. Każda z nich może mieć parametry i określone typy wartości. W standardowej wersji ten język nie ma wielu potrzebnych instrukcji (np. instrukcji wejścia / wyjścia). Można je jednak w prosty sposób dołączyć za pomocą tzw. **bibliotek standardowych**.

Zadania zamieszczone w tym punkcie można również rozwiązać w języku Pascal.

Wprowadzenie do języka Pascal, w tym: ogólne uwagi o konstrukcji języka, struktura programu, środowisko programowania, wybrane typy danych i wybrane instrukcje, podstawy programowania strukturalnego, rozwiązanie równania kwadratowego, sortowanie bąbelkowe.

Zanim program napisany w języku C++ zostanie wykonany, musi być wcześniej przetłumaczony na **język wynikowy** – zrozumiały przez komputer (procesor). Tłumaczenie nazywamy **translacją**. Program jest w całości sprawdzany pod względem składni (poprawności językowej), a użytkownik otrzymuje ewentualną listę błędów.

Nawet najdrobniejszy błąd kompilacji uniemożliwia wykonanie programu. Przy tej okazji nasuwa się bardzo ważna refleksja. Programowanie uczy logicznego myślenia, dokładności i staranności.

Ze względu na ograniczoną objętość podręcznika, na kilku prostych przykładach omówimy wybrane typy danych i podstawowe instrukcje stosowane w programowaniu strukturalnym.

[4] **http://www.apple.com/pl/ipad/features/ [dostęp 5.08.2011].**

[5] W **programowaniu obiektowym** program stanowi zbiór obiektów komunikujących się ze sobą w celu wykonywania zadań. Każdy **obiekt** zawiera **dane** i **metody**, czyli funkcje służące do wykonywania określonych zadań na tych danych. Tymi zagadnieniami w podręczniku nie będziemy się zajmować. Ograniczymy się do poznania metod programowania strukturalnego.

3.1. Ogólne uwagi o konstrukcji języka i budowie programu

Program napisany w języku C++ składa się z ciągu funkcji. Jedna z nich jest wyróżniona i nosi nazwę `main`. Od niej zawsze zaczyna się wykonywanie programu.

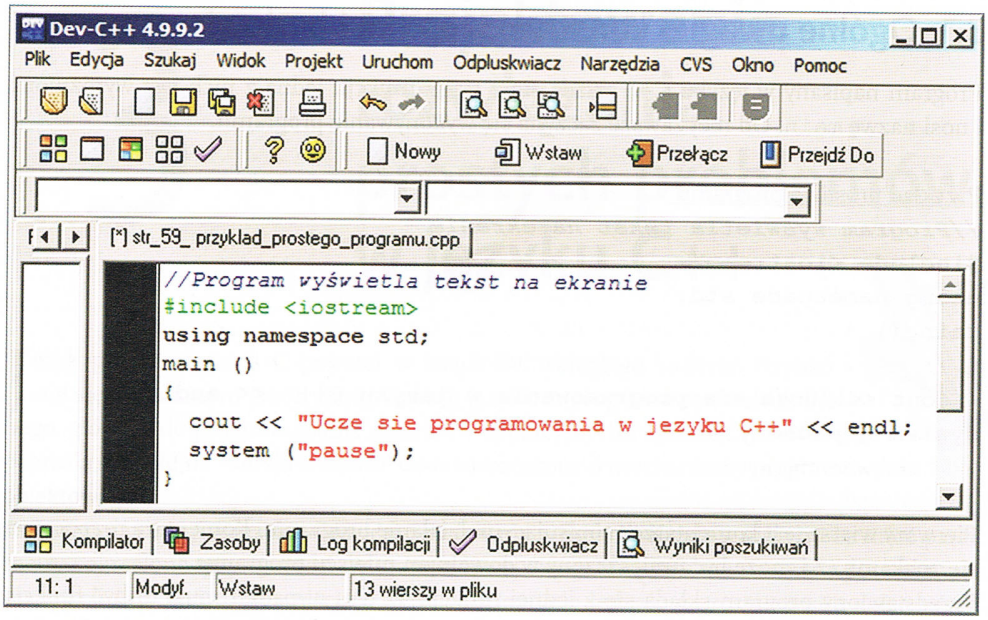
Przykład prostego programu

```
//Program wyświetla tekst na ekranie
#include <iostream>
using namespace std;
main ()
{
  cout << "Ucze sie programowania w jezyku C++" << endl;
system ("pause");
}
```

Pierwszy wiersz jest komentarzem opisującym działanie programu. Komentarz rozpoczyna się znakami `//` i może być umieszczony w dowolnym miejscu programu.
Przedstawiony program składa się z jednej funkcji. W tym wypadku musi to być funkcja o nazwie **`main`**. Bezpośrednio po nazwie funkcji umieszcza się listę jej parametrów ujętą w nawiasy. Parametry reprezentują wartości, które mogą być przekazane do funkcji. Jeżeli funkcja nie ma danych, zaznacza się to pustą listą **`()`**.
Instrukcje wchodzące w skład funkcji są ograniczone nawiasami klamrowymi **`{}`**. Każda pojedyncza instrukcja jest zakończona znakiem średnika (`;`).
W przedstawionym przykładzie chcemy, aby program wyświetlał określony tekst. Ponieważ język C++ nie ma instrukcji wejścia / wyjścia, trzeba się posłużyć strumieniem wyjściowym **`cout`** ze **standardowej biblioteki wejścia / wyjścia** dostarczanej wraz z kompilatorem języka. Aby można było jej użyć, trzeba kompilatorowi dostarczyć informacji umieszczonych w **pliku nagłówkowym**. Dla biblioteki funkcji wejścia / wyjścia jest to plik o nazwie **iostream**. Do programu wstawia się go za pomocą dyrektywy **`#include <iostream>`** oraz instrukcji **`using namespace std`** definiującej tzw. przestrzeń nazw należącą do standardu C++.
Tekst, który ma być wyświetlony, należy umieścić w cudzysłowie i podać jako parametr strumienia wyjściowego **`cout`** wraz z podwójnym znakiem mniejszości <<. Umieszczenie instrukcji **`endl`**, (ang. *end of line* – koniec wiersza) w tej sekwencji na końcu wyświetlanego tekstu (także poprzedzonej znakiem <<) przeniesie kursor na początek następnego wiersza.
Aby po **kompilacji** oraz **uruchomieniu** programu zatrzymać wyświetlony tekst na ekranie, należy skorzystać z instrukcji **`system ("pause")`**, która wyświetli komunikat „Aby kontynuować naciśnij dowolny klawisz...". Dopiero po naciśnięciu dowolnego klawisza nastąpi powrót do okna środowiska programistycznego **Dev-C++** (rys. 5).

3.2. Środowisko programowania Dev-C++

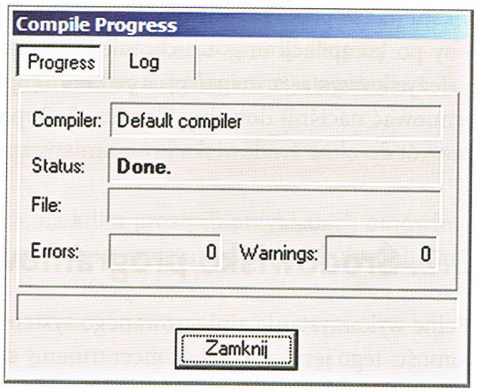
Pełne wykorzystanie zintegrowanego systemu programowania **Dev-C++** wymaga dobrej znajomości tego języka. Tutaj skoncentrujemy się na wyjaśnieniu, w jaki sposób zainicjować pracę w tym środowisku (rys. 5).



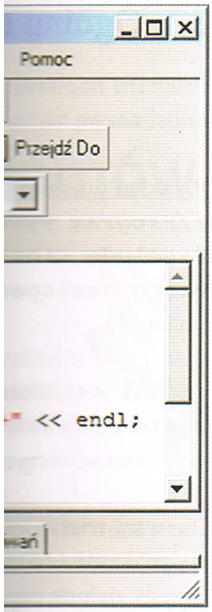
Rysunek 5. Okno środowiska programistycznego **Dev-C++**

Przykładowa sekwencja czynności, która umożliwi wykonanie podanego wyżej programu:
1. Poleceniem **Plik** | **Nowy** | **Plik źródłowy** otwieramy okno edycji, w którym piszemy program.
2. Poleceniem **Plik** | **Zapisz** zapisujemy program na dysku, nadając mu unikatową nazwę z rozszerzeniem **cpp** (tu: **1_program_prosty.cpp**).
3. Poleceniem **Uruchom** | **Kompiluj** lub kombinacją klawiszy **Ctrl+F9** przeprowadzamy kompilację programu. Jeśli pojawią się błędy składniowe, poprawiamy je. Jeśli nie ma błędów, pojawi się komunikat prezentowany na rys. 6.
4. Poleceniem **Uruchom** | **Uruchom** lub kombinacją klawiszy **Ctrl+F10** uruchamiamy program.

Uruchomienie programu, który nie był wcześniej tłumaczony z języka wysokiego poziomu na język wynikowy, powoduje jego automatyczną kompilację. Służy do tego klawisz **F9**. Zwróć uwagę, że w folderze, w którym jest zapisany twój program z rozszerzeniem **cpp**, pojawił się dodatkowo plik o tej samej nazwie, lecz z rozszerzeniem **exe**. Jest to plik **wykonywalny**, czyli taki, który może być uruchomiony bezpośrednio z poziomu systemu operacyjnego (bez potrzeby uruchamiania aplikacji **Dev-C++**).



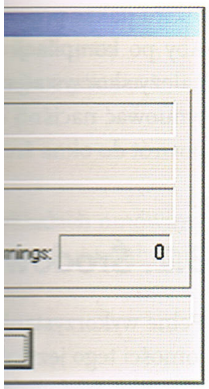
Rysunek 6. Komunikat informujący o poprawnej kompilacji



wyżej programu:
zemy program.
unikatową nazwę

prowadzamy kom-
śli nie ma błędów,

hamiamy program.



cy o poprawnej kompilacji

3.3. Deklarowanie stałych i zmiennych, stosowanie operatorów w języku C++

Do oznaczania nazw stałych, zmiennych, funkcji i innych zdefiniowanych obiektów używamy **identyfikatorów**. Identyfikatorem jest ciąg znaków zaczynających się od litery, np. **bok_kwadratu**.
Stałą może być liczba, pojedynczy znak lub tekst. Z kolei **zmienna** jest nazwą, pod którą mogą być podstawiane różne wartości. Zarówno ze stałą, jak i ze zmienną jest związany jej typ. Wyróżniamy następujące podstawowe trzy typy:

`char` – 1-bajtowy typ znakowy – pojedyncze znaki dostępne w kodzie ASCII;
`int` – 4-bajtowy typ całkowity – liczby z zakresu −2147 483 648 ... 2147 483 647;
`double` – 8-bajtowy typ zmiennoprzecinkowy (rzeczywisty) – liczby z zakresu $-1{,}7 \times 10^{308}$ ... $1{,}7 \times 10^{308}$.

Deklaracja stałej polega na podaniu jej typu i nazwy, której przypisano określoną wartość, stosując znak równości =, np.

```
float pi = 3.14, bok = 0.24E-2;
char litera = 't';
```

Deklaracja zmiennej polega na podaniu jej typu i nazwy lub kilku nazw oddzielonych przecinkami, np.:

```
int liczba_a, liczba_b;
double bok;
char znak_1, znak_2;
```

Liczbę wyrażamy w sposób zbliżony do konwencjonalnego zapisu matematycznego. Jedyne różnice polegają na tym, że zamiast przecinka dziesiętnego używa się **kropki dziesiętnej**, natomiast zamiast podstawy potęgowania 10 używa się litery **E** lub **e**.

Stałą znakową umieszczamy zawsze w pojedynczych apostrofach (' '), natomiast **stałą tekstową** w cudzysłowie (" ").

ĆWICZENIE 4

Oblicz obwód okręgu o promieniu 5 cm.

```
//Obliczenie obwodu okręgu o promieniu 5 cm
#include <iostream>
using namespace std;
main()
{
  int r = 5;
  double pi = 3.14, obwod;
  obwod = 2 * pi * r;
  cout<< "Obwod okregu o promieniu    5 cm = " << obwod<< " cm" <<endl;
  system ("pause");
}
```

Jedną z cech wyróżniających język C++ jest liczba dostępnych **operatorów**. Operator służy do obliczania wartości. Podstawowym operatorem jest **operator przypisania**, czyli znak równości =. Może on wystąpić wielokrotnie w jednej instrukcji, np.

```
int r = 5, a, b;
a = b = r;
cout << a  << endl << b;
```

Instrukcja `a = b = r` zostanie wykonana następująco: *przypisz do b wartość r, następnie do a wartość b*. W efekcie na ekranie zostanie wyświetlone w dwóch wierszach:

5
5

Zapamiętaj różnicę między operatorami przypisania: w języku Pascal jest nim :=, a w C++ jest nim =.

Operatory arytmetyczne

`+` – dodawanie;
`-` – odejmowanie;
`*` – mnożenie;
`/` – dzielenie;
`%` – modulo, czyli reszta z dzielenia, np. `cout 11 % 4`; wyświetli na ekranie wartość 3.

Język C++ pozwala łączyć operator przypisania z operatorami arytmetycznymi, np.
`a += 2;` zwiększa *a* o wartość 2 (zapis alternatywny to: `a = a + 2;`);
`x *= 10;` pomnóż *x* przez 10 (zapis alternatywny to: `x = x * 10;`).

Operatorami dostępnymi tylko dla zmiennych całkowitych są operatory automatycznego zwiększania lub zmniejszania o jeden, np.
`licznik++;` zwiększa wartość licznika o 1 (zapis alternatywny to: `licznik = licznik +1;`);
`licznik--;` zmniejsza wartość licznika o 1 (zapis alternatywny to: `licznik = licznik - 1;`).

Operatory relacji

`==` czy wartości są równe;
`!=` czy wartości są różne;
`>` czy pierwsza wartość jest większa od drugiej;
`<` czy pierwsza wartość jest mniejsza od drugiej;
`>=` czy pierwsza wartość jest większa lub równa drugiej;
`<=` czy pierwsza wartość jest mniejsza lub równa drugiej.

Zapamiętaj różnicę między operatorami relacji *wartości są różne*. W Pascalu jest to <>, natomiast w C++ jest to !=.

Różnice także występują w operatorze *wartości są równe*. W Pascalu jest to =, natomiast w C++ jest to ==. Pozostałe operatory są takie same.

3.4. Wybra

Skoncentrujem
nym języku pro

Wczytanie dan

```
cin >> z
```

Strumień wejśc
pisana zmienne
dotyczy wprowa
nia drugiej dane

Wyprowadzan

```
cout <<
```

Strumień wyjśc
ekranie komuni
Jeśli wartość zm

Nawiasy klam

Instrukcje umie
czą funkcję.

Instrukcja war

prosta:
z alternatywą:

Zarówno akcja_
sach klamrowyc

Przykłady

```
if (a > b) c
  else cout

if (a != 0)
  {
    x = -b/a
    cout <<
  };
```

Instrukcja itera

```
do
  {
    pojedync
  }
while (warun
```

ów. Operator służy
nia, czyli znak rów-

wartość r, następnie
szach:

est nim :=,

wartość 3.

znymi, np.
r);

omatycznego zwięk-

= licznik +1;);
licznik - 1;).

t to =,

3.4. Wybrane instrukcje języka C++

Skoncentrujemy się na tych samych instrukcjach, które opisaliśmy w umownym strukturalnym języku programowania.

Wczytanie danych (w języku umownym: `Podaj`)

```
cin >> zmienna;
```

Strumień wejściowy umożliwiający wprowadzenie danej z klawiatury, która zostanie przypisana zmiennej. Poprawny jest także zapis: `cin >> zmienna_1 >> zmienna_2`, który dotyczy wprowadzenia pierwszej danej, naciśnięcia klawisza **Enter**, a następnie wprowadzenia drugiej danej.

Wyprowadzanie wyników i komunikatów (w języku umownym: `Pisz`)

```
cout << zmienna_lub_komunikat;
```

Strumień wyjściowy umożliwiający wyprowadzenie wartości zmiennej lub wyświetlenie na ekranie komunikatu (tekstu), np.: `cout << "wynik = " << a;`
Jeśli wartość zmiennej *a* wynosi np. **25**, wówczas na ekranie pojawi się **wynik = 25**.

Nawiasy klamrowe {...} (w języku umownym: `POCZĄTEK` ... `KONIEC`)
Instrukcje umieszczone pomiędzy nawiasami klamrowymi stanowią w języku C++ pojedynczą funkcję.

Instrukcja warunkowa `if` (w języku umownym: `JEŚLI`)

prosta:
```
if (warunek) akcja_1;
```
z alternatywą:
```
if (warunek) akcja_1;
else akcja_2;
```

Zarówno akcja_1, jak i akcja_2 mogą być instrukcjami złożonymi (umieszczonymi w nawiasach klamrowych).

Przykłady

```
if (a > b) cout << "a jest wieksze od b";
  else cout << "a nie jest wieksze od b";

if (a != 0)
  {
    x = -b/a;
    cout << x << endl;
  };
```

Instrukcja iteracyjna `do ... while` (w języku umownym: `POWTARZAJ`)

```
do
  {
    pojedyncza_akcja lub ciag_akcji;
  }
while (warunek);
```

Akcje umieszczone w nawiasach klamrowych są wykonywane dopóty, dopóki wartość warunku logicznego jest **True** (prawda). W przeciwnym razie wykonywanie tych akcji się kończy.

Przykład

```
ile = 1;
do
  {
    cout << "Ucze sie jezyka C++" << endl;
    ile = ile + 1;
  }
while (ile <= 10);
```

Instrukcja iteracyjna `while` (w języku umownym: **DOPÓKI**)

```
while (warunek) pojedyncza_akcja lub ciag_akcji;
```

Dopóki wartość warunku logicznego jest **True** (prawda), jest wykonywana pojedyncza_akcja lub ciąg_akcji umieszczony w nawiasach klamrowych. W przeciwnym razie wykonywanie tych akcji się kończy.

Przykład

```
ile = 1;
while (ile <= 10)
  {
    cout  << "Ucze sie jezyka C++" << endl;
    ile = ile + 1;
   }
```

Instrukcja pętli `for` (nie definiowaliśmy jej w języku umownym)

```
for (wartosc_poczatkowa; wartosc_koncowa; wartosc_kroku) akcja;
```

Akcja znajdująca się na końcu instrukcji **`for`** wykonywana jest tyle razy, ile wartości_kroku mieści się w zakresie wartość_początkowa ... wartość_końcowa.
Podobnie jak opisane wyżej dwie instrukcje iteracyjne, instrukcja **`for`** także wykonuje w programie powtórzenie.

Przykłady

```
s = 0;
for (i = 1; i <= 5; i++) s = s + i;{i-----> 1,2,3,4,5}

s = 0;
for (i = 5; i >= 1; i--) s = s + i; {i-----> 5,4,3,2,1}
```

W rezultacie wykonania podanych wyżej przykładów następuje przypisanie zmiennej *s* wartości równej 15.

3.5. Kodo

Tytuł tej części
cowanych wcze
z wybranych p
językowi progr
niu równania k

ĆWICZENIE

Zapisz w języku
a. Rozwiązanie
b. Obliczenie s
c. Wyznaczenie

```
//1. Rozwiąz
#include <i
using namesp
main()
{
  double a,
  cout << "R
  cout << "P
  cout << "P
  cout << en
  if (a == 0
    if (b ==
      else
    else
      {
       x =
       cout
      }
  system ("p
}


//2. Suma pię
#include <io
using namesp
main()
{
  int s, i,
  s = 0;
  i = 0;
  do
    {
```

ki wartość warunku
ię kończy.

a pojedyncza_akcja
razie wykonywanie

kroku) akcja;
ile wartości_kroku

ykonuje w programie

nie zmiennej s war-

3.5. Kodowanie prostych algorytmów w języku C++

Tytuł tej części wyraźnie sugeruje, że programowanie jest w swojej istocie wyrażaniem opracowanych wcześniej algorytmów w języku komputera. W dalszych rozważaniach skorzystamy z wybranych problemów zdefiniowanych w części poświęconej umownemu strukturalnemu językowi programowania. Ponadto pojawią się dwa kolejne ćwiczenia poświęcone rozwiązaniu równania kwadratowego i porządkowaniu elementów tablicy.

ĆWICZENIE 5

Zapisz w języku C++ rozwiązania następujących prostych problemów:
a. Rozwiązanie równania liniowego $ax + b = 0$.
b. Obliczenie sumy pięciu kolejnych liczb naturalnych.
c. Wyznaczenie największego wspólnego dzielnika NWD dwóch liczb naturalnych a i b.

```
//1. Rozwiązanie równania liniowego
#include <iostream>
using namespace std;
main()
{
  double a, b, x;
  cout <<"Rozwiazanie rownania liniowgo ax + b = 0" << endl << endl;
  cout <<"Podaj wspolczynnik a:"; cin >> a;
  cout <<"Podaj wspolczynnik b:"; cin >> b;
  cout << endl;
  if (a == 0)
    if (b == 0) cout <<"Nieskonczenie wiele rozwiazan";
      else cout <<"Rownanie sprzeczne";
    else
      {
       x = -b/a;
       cout << "wynik = " << x << endl;
      }
  system ("pause");
}


//2. Suma pięciu kolejnych liczb naturalnych rozpoczynających się liczbą 1
#include <iostream>
using namespace std;
main()
{
  int s, i, ile;
  s = 0;
  i = 0;
  do
    {
```

```
      s = s + i;
      i = i + 1;
    }
  while (i <= 5);
  cout << "suma = " << s << endl;
  system ("pause");
}

//3. Największy wspólny dzielnik
#include <iostream>
using namespace std;
main()
{
  int a, b;
  cout << "Najwiekszy wspolny dzielnik" << endl << endl;
  cout << "a = "; cin >> a;
  cout << "b = "; cin >> b;
  while (a != b)
    if (a > b) a = a - b; else b = b - a;
  cout << "nwp = " << a << endl;
  system ("pause");
}
```

ĆWICZENIE 6

Wyznacz pierwiastki równania kwadratowego $ax^2 + bx + c = 0$, a różne od 0. Skorzystaj z funkcji standardowej **Sqrt(x)** wyznaczającej pierwiastek kwadratowy z liczby x. Przedstaw rozwiązanie problemu w formie schematu blokowego.

Analiza problemu

Sposób wyznaczania pierwiastków równania kwadratowego jest powszechnie znany. W pierwszej kolejności należy obliczyć Δ, a następnie rozpatrzyć trzy możliwe przypadki:
$\Delta < 0$ – równanie nie ma rozwiązania;
$\Delta = 0$ – równanie ma jedno rozwiązanie;
$\Delta > 0$ – równanie ma dwa rozwiązania.

W programie przedstawionym poniżej użyto nieomawianych do tej pory dwóch dodatkowych bibliotek:
- **iomanip** – umożliwia zastosowanie w strumieniu wyjściowym **cout** instrukcji **setprecision (liczba)** określającej liczbę miejsc po przecinku w wyświetlanej liczbie rzeczywistej.
- **cmath** – umożliwia użycie funkcji **sqrt** obliczającej pierwiastek kwadratowy z liczby.

```
//Rozwiązanie równania kwadratowego
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
main()
{
 double a, b, c, delta;
 cout << "Rozwiazanie rownania kwadratowego ax^2 + bx + c = 0 " << endl;
 cout << "Podaj wspolczynnik a: "; cin >> a;
 cout << "Podaj wspolczynnik b: "; cin >> b;
 cout << "Podaj wspolczynnik c: "; cin >> c;
 cout << endl;
 if (a == 0) cout << "To nie jest rownanie kwadratowe!" << endl;
   else
     {
      delta = b*b-4*a*c;
      if (delta < 0) cout << "Rownanie nie ma rozwiazania" << endl;
        else
          if (delta == 0) cout << "Rownanie ma jedno rozwiazanie" << endl
          << "x0 = " << setprecision(2) << -b/(2*a) << endl;
           else
            {
             cout << "Rownanie ma dwa rozwiazania" << endl;
             cout << "x1 = " << setprecision(2) << (-b+sqrt(delta))/(2*a) << endl;
             cout << "x2 = " << setprecision(2) << (-b-sqrt(delta))/(2*a) << endl;
            }
     }
  system ("pause");
}
```

Testowanie programu

Dla $a = 0$, $b = 4$, $c = 1$ otrzymamy: `To nie jest równanie kwadratowe!`

Dla $a = 1$, $b = 4$, $c = 1$ otrzymamy:
```
Równanie ma dwa rozwiązania
x1 = -3.73
x2 = -0.27
```

Dla $a = 1$, $b = 4$, $c = 4$ otrzymamy:
```
Równanie ma jedno rozwiązanie
x0 = -2.00
```

Dla $a = 1$, $b = 4$, $c = 5$ otrzymamy: `Równanie nie ma rozwiązania`

Wniosek

Testowanie programu nie wykazało błędów.

Dodatkowe informacje dotyczące funkcji w języku C++.

3.6. Tablice przykładem strukturalnego typu danych

Omówione dotychczas typy danych były typami prostymi. Zmienne tego typu mogły zawierać tylko jeden element (liczbę, znak). Obecnie zdefiniujemy złożoną strukturę danych, jaką jest **tablica**. W kolejnym ćwiczeniu rozpatrzymy najprostszy przypadek **tablicy jednowymiarowej**, która odpowiada matematycznemu pojęciu wektora, np. [2 3 6 5 4] oznacza tablicę jednowymiarową zawierającą pięć elementów typu całkowitego.
Deklaracja tablicy jednowymiarowej przyjmuje następującą postać:

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

Przykład

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

Podczas przypisywania tablicy konkretnych wartości pomocna jest omówiona wcześniej instrukcja pętli **FOR**.

ĆWICZENIE 7

Jednym z ciekawszych zastosowań tablic jest sortowanie jej elementów w porządku rosnącym lub malejącym. Korzystając z **sortowania przez prostą zamianę**, zwanego także **sortowaniem bąbelkowym**, uporządkuj rosnąco 10 elementów tablicy jednowymiarowej.

Analiza problemu

W metodzie sortowania przez prostą zamianę porównujemy dwa sąsiednie elementy tablicy. Gdy kolejny element jest mniejszy od poprzedniego, przestawiamy je. Po porównaniu wszystkich przylegających do siebie elementów największy z nich znajdzie się na końcu tablicy. W ten sposób wykonujemy pierwszy cykl porównań.
Ponownie wracamy do porównywania sąsiednich elementów (ale nie bierzemy już pod uwagę ostatniego). Po zakończeniu drugiego cyklu kolejny (przedostatni) element znajdzie się na właściwym miejscu.
Czynności te wykonujemy tak długo, aż stwierdzimy, że wszystkie elementy znajdują się na właściwych miejscach.

Przykład

Poniżej przedstawiono sortowanie rosnące czterech liczb całkowitych. W zapisie zastosowano następujące oznaczenia: elementy porównywane są podkreślone, a elementy, które znajdują się na właściwych miejscach – są zapisane czcionką w kolorze czerwonym.

Pierwszy cykl:

4	5	3	2	– brak przestawienia,
4	5	3	2	– przestawiamy 5 z 3,
4	3	5	2	– przestawiamy 5 z 2,
4	3	2	5	– 5 na właściwym miejscu.

Drugi cykl:

4	3	2	5	– przestawiamy 4 z 3,
3	4	2	5	– przestawiamy 4 z 2,
3	2	4	5	– 4 i 5 na właściwym miejscu.

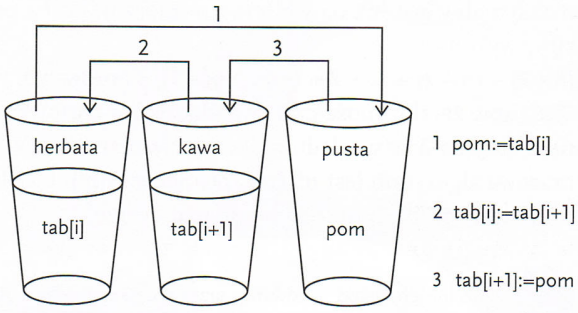
Trzeci cykl:

3	2	4	5	– przestawiamy 3 z 2,
2	3	4	5	– 2, 3, 4 i 5 na właściwym miejscu.

Sortowanie zakończone.

Podstawowym problemem w metodach sortujących jest określenie sposobu zamiany miejscami dwóch sąsiednich elementów. Obrazowo wyjaśnimy to na przykładzie dwóch szklanek zawierających kawę i herbatę. Jeśli chcemy zamienić zawartość tych szklanek, musimy skorzystać z trzeciej szklanki. Na poniższym rysunku pokazano sposób postępowania, w którym możemy wyróżnić trzy kroki:

1. przelewamy herbatę do pustej szklanki;
2. przelewamy kawę do szklanki po herbacie;
3. przelewamy herbatę do szklanki po kawie.




```
//Sortowanie bąbelkowe
#include <iostream>
using namespace std;
main()
{
  int i, j, pom;
  int tab [10];
  cout << "Podaj elementy tablicy do sortowania" << endl;
  for (i = 1; i <= 10; i++)
    {
      cout << "Element nr " << i << " = ";
      cin >> tab [i];
    }
  for (j = 10; j > 1; j--)
    for (i = 1; i <= j; i++)
    if (tab [i] > tab [i + 1])
      {
        pom = tab [i];
        tab [i] = tab [i + 1];
        tab [i + 1] = pom;
      }
  cout << endl << "Oto zawartosc tablicy po posortowaniu:" << endl;
  for (i = 1; i <= 10; i++) cout << tab [i] << " ";
  cout << endl;
  system ("pause");
}
```

Testowanie programu

Dla danych: 40 60 9 8 2 10 6 1 23 11 otrzymamy:

```
Oto zawartość tablicy po posortowaniu:
1  2  6  8  9  10  11 23  40  60
```

Wniosek

Testowanie programu nie wykazało błędów.

Jest wiele różnych sposobów sortowania elementów tablicy. O wyborze metody rozwiązania dowolnego problemu (w tym również sortowania) decyduje m.in. **złożoność czasowa** mierzona liczbą podstawowych operacji, które trzeba wykonać w algorytmie.
Główną operacją występującą podczas sortowania jest porównywanie elementów tablicy, a zatem liczba porównań będzie decydowała o złożoności czasowej algorytmów sortowania.
Zauważmy, że w pierwszym cyklu wykonujemy $n-1$ porównań. Za drugim razem (element maksymalny jest już na właściwym miejscu) liczba porównań wynosi $n-2$. W sumie musimy wykonać:
$(n-1) + (n-2) + \ldots + 2 + 1 = n * (n-1)/2$ porównań.
Widzimy, że złożoność czasowa algorytmu sortowania bąbelkowego wyraża się wielomianem stopnia **2** ze względu na liczbę elementów n. W takiej sytuacji mówimy, że złożoność czasowa algorytmu jest n^2, co zapisujemy $\mathbf{O}(n^2)$. Dzielenie przez stały współczynnik (w tym przypadku przez 2) nie jest brane pod uwagę.

Opis innych metod sortowania: przez proste wstawianie, proste wybieranie.

PODSUMOWANIE

- Program napisany w języku C++ zawiera kilka typowych elementów: dyrektywę **`#include`**, która nakazuje kompilatorowi włączenie do programu plików bibliotek standardowych, instrukcję **`main`** rozpoczynającą główną część programu oraz zbiór instrukcji umieszczonych w nawiasach klamrowych **`{}`**.
- Zmienne użyte w programie muszą być wcześniej zadeklarowane. Trzy podstawowe typy zmiennych to: **`int`** – typ całkowity, **`float`** – typ zmiennoprzecinkowy, **`char`** – typ znakowy.
- W programach zamieszczonych w tym podręczniku do wczytywania danych oraz wyprowadzania wyników i komunikatów stosowaliśmy strumienie **`cin >>`** i **`cout <<`**, które znajdują się w bibliotece standardowej **`iostream`**. Jednak często można się zetknąć z ich odpowiednikami **`scanf()`** i **`printf()`** znajdującymi się w bibliotece **`stdio.h`**.

PYTANIA SPRAWDZAJĄCE

1. Wyjaśnij, co oznacza kompilacja programu zapisanego w języku programowania.
2. Wymień trzy poznane instrukcje iteracyjne, wskazując na ich podobieństwa i różnice.