DO NOWEJ PODSTAWY PROGRAMOWEJ

Część 1

PODRĘCZNIK dla szkół ponadpodstawowych

Informatyka Europejczyka

Zakres rozszerzony

Danuta Korman
Grażyna Szabłowicz-Zawadzka

Helion EDUKACJA

Redaktor prowadzący: Joanna Zaręba

Ilustracja na okładce została wykorzystana za zgodą Shutterstock.

Helion SA
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?ieppr1
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-4188-3

Printed in Poland.

Spis treści

Wstęp

W realizacji programu **informatyki w zakresie rozszerzonym** korzystamy zarówno z podręczników do zakresu podstawowego, jak i rozszerzonego. Nauczyciel omawia materiał z zakresu podstawowego, wplatając w wymagających tego miejscach zagadnienia zawarte w podręcznikach do poziomu rozszerzonego. Rozkład materiału dokładnie wskazuje kolejność omawianych zagadnień. Ponieważ siatka godzin do informatyki rozszerzonej może być różna w różnych szkołach, nie można go podzielić na poszczególne klasy. Nauczyciel, znając przydział godzin lekcyjnych w danym roku, zaplanuje ich rozkład, jednak w sumie musi to być 9 godzin w cyklu kształcenia. Rozkład materiału jest przygotowany dokładnie na tyle.

Na poziomie podstawowym realizowane są 3 godziny lekcyjne w cyklu kształcenia, czyli łącznie 90 godzin. Tutaj realizacja godzin jest dokładnie określona: po 30 godzin w klasach I, II i III.

Na poziomie rozszerzonym dodatkowo dochodzi 6 godzin lekcyjnych w cyklu kształcenia, czyli łącznie 180 godzin. W sumie na tym poziomie jest 9 godzin lekcyjnych w cyklu kształcenia (3+6), czyli 270 godzin. Materiał omawiany w zakresie rozszerzonym obejmuje zarówno zakres podstawowy, jak i rozszerzony.

Podręczniki zawierają **wiele przykładów i ćwiczeń**, na podstawie których uczeń może przystąpić do **rozwiązywania zadań**. Każdy rozdział zaczyna się od podania informacji, co uczeń powinien potrafić po zakończeniu nauki w szkole podstawowej i czego się nauczy na lekcjach na poziomie rozszerzonym. Na końcu każdego rozdziału są **zadania podsumowujące** omówiony materiał.

Na podstawie części pierwszej podręcznika do poziomu rozszerzonego uczeń:

- Znacznie rozszerzy znajomość **tekstowych języków programowania dopuszczonych do egzaminu maturalnego** z informatyki rozszerzonej. Będzie to kontynuacja nauki języka Python. Ponadto zostanie wprowadzony język C++, również dopuszczony do egzaminu maturalnego.
- Zajmie się **własnościami oraz analizą efektywności algorytmów**. Ponadto pozna **źródła błędów występujących w obliczeniach komputerowych**.
- Pozna i będzie stosować **zaawansowane algorytmy iteracyjne i rekurencyjne** z użyciem typów złożonych, realizowane na liczbach i na tekstach. Będzie wykorzystywać nabyte umiejętności w zadaniach z **kryptografii i kryptoanalizy**, analizując i konstruując **algorytmy szyfrujące i deszyfrujące**.
- Pozna sposoby **reprezentowania w komputerze obrazów, dźwięków i animacji**. Nauczy się **edycji dźwięku** i tworzenia dwuwymiarowych oraz trójwymiarowych **wizualizacji i animacji**. Będzie stosować właściwe formaty plików graficznych.
- Pozna **metody kompresji danych**.
- Będzie stosować **zaawansowane funkcje arkusza kalkulacyjnego** w zależności od rodzaju danych. Nauczy się definiowania **makropoleceń**. Pozna możliwości **wbudowanego języka programowania VBA**.

- Zaprojektuje i stworzy **relacyjną bazę złożoną z wielu tabel**. Będzie formułować kwerendy, tworzyć i modyfikować formularze oraz raporty. Pozna i zastosuje **język SQL** do wyszukiwania informacji w bazie danych. Będzie uwzględniać kwestie **integralności danych**, **bezpieczeństwa** i **ochrony danych** w bazie.

Danuta Korman
Grażyna Szabłowicz-Zawadzka

Rozdział 1.

ROZUMIENIE I ANALIZOWANIE PROBLEMÓW. ALGORYTMY I ICH ZASTOSOWANIE

Czego się nauczysz na poziomie rozszerzonym?

Rozszerzysz znajomość **tekstowych języków programowania dopuszczonych do egzaminu maturalnego** z informatyki rozszerzonej. Będzie to kontynuacja nauki języka Python. Ponadto poznasz język C++, również dopuszczony do egzaminu maturalnego, w tym:

- strukturę programu zapisanego w języku C++,
- strumieniowe operacje wejścia-wyjścia i manipulatory strumieniowe,
- wyrażenia arytmetyczne, relacje i operatory logiczne,
- dostęp do bibliotek i funkcje matematyczne w języku C++,
- podstawowe konstrukcje algorytmiczne, na przykład instrukcje warunkowe, instrukcję wyboru, instrukcje iteracyjne, instrukcje sterujące,
- strukturalizację programów, czyli zastosowanie funkcji w języku C++,
- proste typy danych,
- strukturalne typy danych, na przykład tablice, łańcuchy znaków.

Poznasz i będziesz wykorzystywać reprezentację algorytmów w postaci **pseudokodu**.

Zajmiesz się **własnościami i złożonością obliczeniową poznawanych algorytmów**. Będziesz **analizować efektywność algorytmów**. Poznasz pojęcia **poprawności**, **skończoności** i **optymalności algorytmów**.

Poznasz tematykę **błędów pojawiających się w obliczeniach komputerowych**, w tym notację naukową, błędy względne i bezwzględne.

Rozszerzysz znajomość **pozycyjnych i niepozycyjnych systemów liczbowych**, w tym konwersje liczb rzeczywistych, konwersje między systemami niedziesiętnymi, operacje arytmetyczne wykonywane w różnych systemach liczbowych.

Poznasz **reprezentacje liczb naturalnych i całkowitych w komputerze**, w tym stałopozycyjną i zmiennopozycyjną reprezentację liczb oraz kod U2.

Poznasz i będziesz stosować **zaawansowane algorytmy iteracyjne i rekurencyjne**, z wykorzystaniem typów złożonych, realizowane na liczbach, ciągach liczbowych i tekstach.

Będziesz wykorzystywać nabyte umiejętności, realizując zadania z **kryptografii** i **kryptoanalizy**, analizując i konstruując **algorytmy szyfrujące i deszyfrujące**.

Temat 1. Sposoby reprezentowania algorytmów

Istnieje wiele sposobów przedstawiania algorytmów. Wybór odpowiedniej metody powinien zależeć od rozwiązywanego problemu. Do wykonywanego zadania należy dobrać taki sposób reprezentowania algorytmu, który najdokładniej i najczytelniej pokaże jego przebieg. Najważniejsze **metody przedstawiania algorytmów wykorzystywane w tym podręczniku** to:

- lista kroków,
- schemat blokowy,
- pseudokod,
- programy w tekstowych językach programowania Python i C++.

Do prezentowania algorytmów można stosować też inne metody, na przykład opis słowny, drzewo algorytmu, drzewo wyrażenia, inne języki programowania. Większość metod zastosowanych w podręczniku poznaliście już w szkole podstawowej. Jedyną nową techniką jest pseudokod, który opisano w podrozdziale 1.1. W tym temacie przypominamy również inne metody, które pojawiają się w podręczniku.

W podręczniku algorytmy będą zapisywane w dwóch językach programowania dopuszczonych do egzaminu maturalnego z informatyki:

- **Python**, którego podstawy już poznaliście,
- **C++**, którego opis jest w temacie 2.

1.1. Pseudokod

Pseudokod to taki sposób zapisu algorytmu, który zachowuje strukturę charakterystyczną dla kodu zapisanego w języku programowania. Rezygnujemy tutaj ze ścisłych reguł składniowych na rzecz prostoty i czytelności. Pseudokod nie zawiera niektórych szczegółów implementacyjnych, takich jak deklaracja zmiennych i alokacja pamięci. Kroki algorytmu opisywane są za pomocą formuł matematycznych lub zdań w języku naturalnym.

Nie istnieją obecnie standardy zapisu pseudokodu. Przeważnie używa się składni, która opiera się na istniejących językach programowania.

Przykład 1.1.

Przeanalizujmy zapisany w postaci pseudokodu prosty algorytm, który po wczytaniu liczby sprawdza, czy jest ona dodatnia. Jeśli warunek jest spełniony, wypisuje wczytaną liczbę, w przeciwnym razie wyświetla informację o błędzie.

Specyfikacja:

Dane: liczba rzeczywista: *liczba*.

Wynik: jeśli liczba jest dodatnia, wypisanie zmiennej *liczba*, w przeciwnym razie wypisanie komunikatu o błędzie.

Pseudokod:

```
wprowadź liczbę
jeżeli liczba jest dodatnia, to
  wypisz liczbę
w przeciwnym razie
  wypisz informację o błędzie
```

Ten sam algorytm możemy zapisać wszystkimi poznanymi metodami. Na rysunku 1.1 przedstawiono schemat blokowy tego algorytmu, a poniżej jego zapis w postaci listy kroków i programów w językach programowania Python i C++. Zauważ, że tylko reprezentacja w postaci programu umożliwia jego uruchomienie. Ponadto zapis algorytmu w pseudokodzie bardzo przypomina kod programu.

Lista kroków:

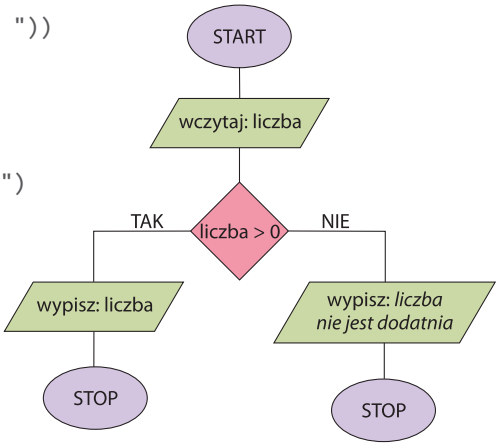
Krok 1. Jeżeli *liczba* jest dodatnia, przejdź do kroku 2., w przeciwnym razie przejdź do kroku 3.

Krok 2. Wypisz zmienną *liczba* i zakończ algorytm.

Krok 3. Wypisz komunikat „liczba nie jest dodatnia" i zakończ algorytm.

Program w języku Python:

```
liczba = float(input("podaj liczbę: "))
if liczba > 0:
    print(liczba)
else:
    print("liczba nie jest dodatnia")
```



Rysunek 1.1. Schemat blokowy algorytmu do przykładu 1.1

Program w języku C++:

```
#include <iostream>
using namespace std;
int main()
{
  double liczba;
  cout << "podaj liczbę: ";
  cin >> liczba;
  if (liczba > 0) cout << liczba << endl;
  else cout << "liczba nie jest dodatnia" << endl;
}
```

1.2. Rozwiązywanie równania kwadratowego

Zajmijmy się analizą i rozwiązaniem równania kwadratowego podawanego przeważnie w postaci $ax^2 + bx + c = 0$. Współczynniki rzeczywiste a, b i c są tutaj danymi wejściowymi. Należy pamiętać, aby współczynnik a był różny od zera. W przeciwnym razie, czyli dla a równego zero, otrzymalibyśmy równanie liniowe.

Znane metody realizujące to zadanie to **algorytm „delty”** oraz **wzory Viète’a**. Jednak aby skonstruować algorytm poprawny dla obliczeń wykonywanych przez komputer, należy wykorzystać obie podane metody. Taki sposób wyznaczania pierwiastków równania kwadratowego nazywamy **stabilnym algorytmem rozwiązującym równanie kwadratowe**.

Analizowany algorytm zapiszemy wszystkimi poznanymi sposobami, które dopuszczone są do egzaminu maturalnego.

Bez względu na zastosowaną metodę rozwiązywania równania kwadratowego specyfikacja tego zadania jest taka sama i ma następującą postać:

Specyfikacja:

Dane: liczba rzeczywista: $a \neq 0$;

dowolne liczby rzeczywiste: b, c.

Wynik: pierwiastki rzeczywiste równania kwadratowego lub komunikat informujący o braku rozwiązania.

1.2.1. Niestabilny algorytm rozwiązujący równanie kwadratowe — algorytm „delty”

Najczęściej stosowaną metodą rozwiązywania równania kwadratowego jest algorytm „delty”. Pierwiastki trójmianu wyznaczane są tutaj z wykorzystaniem „delty”, którą obliczamy następująco:

$$\Delta = b^2 - 4ac.$$

Następnie analizujemy wartość uzyskanej „delty” i w zależności od niej wyznaczamy pierwiastki. Jeśli „delta” jest większa od zera, to otrzymujemy dwa różne pierwiastki x_1 i x_2, które obliczamy za pomocą podanych poniżej wzorów:

$$x_1 = \frac{-b - \sqrt{\Delta}}{2a},$$

$$x_2 = \frac{-b + \sqrt{\Delta}}{2a}.$$

Jeśli „delta” jest równa zero, to otrzymujemy dwa identyczne pierwiastki x_1 i x_2, które wyznaczamy za pomocą poniższego wzoru:

$$x_1 = x_2 = \frac{-b}{2a}.$$

Mówimy wówczas, że równanie ma jeden pierwiastek.

W przypadku gdy „delta" jest mniejsza od zera, równanie nie ma rozwiązania w zbiorze liczb rzeczywistych.

Algorytm „delty" jest najczęściej stosowanym sposobem wyznaczania pierwiastków równania kwadratowego. Metoda ta jednak nie sprawdza się w sytuacji, gdy do obliczeń wykorzystywany jest komputer. Dla niektórych wartości współczynników a, b, c generuje ona niedokładne wartości pierwiastków. Wynika to z błędów zaokrągleń wyników pośrednich, które mogą mieć wpływ na niedokładność wyników końcowych. Stąd algorytm „delty" określamy jako **niestabilny algorytm rozwiązujący równanie kwadratowe**. Poniżej przedstawiono ten algorytm w postaci schematu blokowego (rysunek 1.2), listy kroków, pseudokodu oraz programów w językach Python i C++.

Lista kroków:

Krok 1. Jeśli $a = 0$, to wypisz komunikat „to nie jest równanie kwadratowe" i zakończ algorytm.

Krok 2. (W tym przypadku $a \neq 0$). Przypisz $\Delta = b^2 - 4ac$.

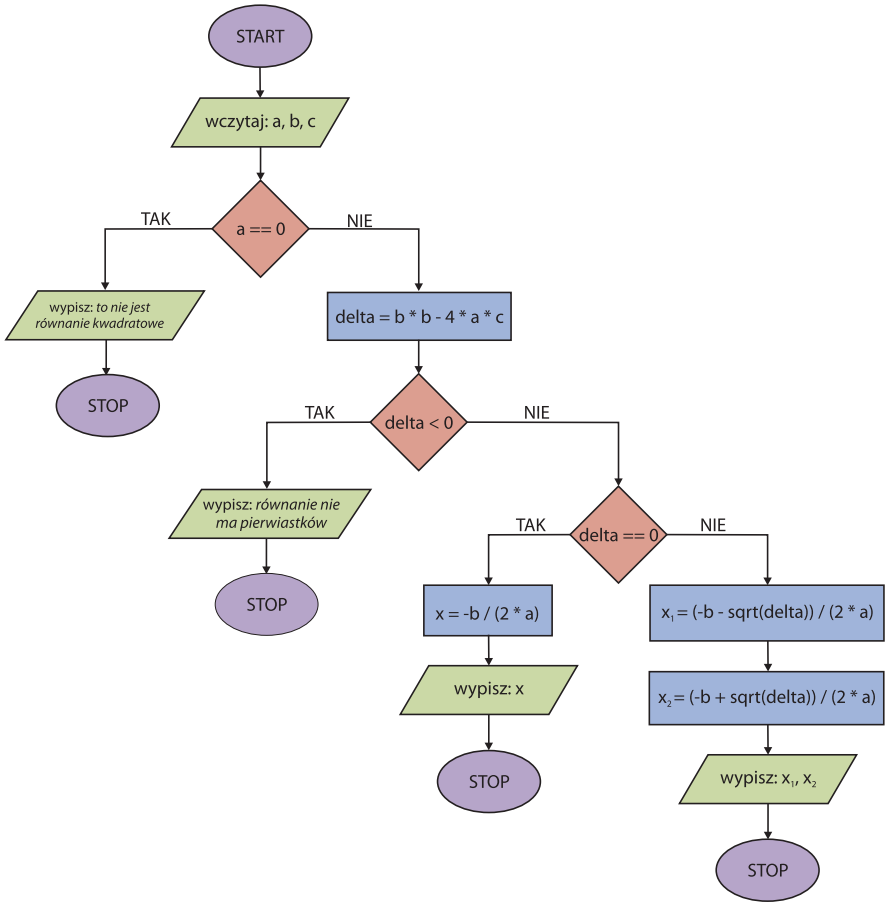
Krok 3. Jeśli $\Delta < 0$, to wypisz komunikat „równanie nie ma pierwiastków" i zakończ algorytm.

Krok 4. (W tym przypadku $\Delta \geq 0$). Jeśli $\Delta = 0$, to oblicz pierwiastek $x = \frac{-b}{2a}$, wypisz wyznaczony pierwiastek x i zakończ algorytm.

Krok 5. (W tym przypadku $\Delta > 0$). Oblicz pierwiastki $x_1 = \frac{-b-\sqrt{\Delta}}{2a}$ i $x_2 = \frac{-b+\sqrt{\Delta}}{2a}$, wypisz wartości wyznaczonych pierwiastków x_1 i x_2. Zakończ algorytm.

Pseudokod:

```
jeżeli a = 0, to
  wypisz komunikat "to nie jest równanie kwadratowe"
w przeciwnym razie
  przypisz Δ = b² - 4ac
  jeżeli Δ < 0, to
    wypisz komunikat "równanie nie ma pierwiastków"
  w przeciwnym razie
    jeżeli Δ = 0, to
      oblicz pierwiastek x = -b/2a
      wypisz x
    w przeciwnym razie
      oblicz pierwiastki x1 = (-b-√Δ)/2a  i  x2 = (-b+√Δ)/2a
      wypisz x1 i x2
```



Rysunek 1.2. Schemat blokowy algorytmu „delty” rozwiązującego równanie kwadratowe

Program w języku Python:

```
from math import *

def rownanie_kwadratowe(a, b, c):
    if a == 0:
        return "to nie jest równanie kwadratowe"
    else:
        delta = b * b – 4 * a * c
        if delta < 0:
            return "równanie nie ma pierwiastków"
        elif delta == 0:
            x1 = –b / (2 * a)
            return x1
```

```
        else:
            x1 = (-b - sqrt(delta)) / (2 * a)
            x2 = (-b + sqrt(delta)) / (2 * a)
            return x1, x2

print(rownanie_kwadratowe(2, 1, 3))
```

Wynik:

równanie nie ma pierwiastków

Program w języku C++:

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
  double a, b, c, delta, x1, x2;
  cout << "podaj współczynniki równania a, b, c:" << endl;
  cin >> a >> b >> c;
  if (a == 0) cout << "to nie jest równanie kwadratowe" << endl;
  else
  {
    delta = b * b - 4 * a * c;
    if (delta < 0) cout << "równanie nie ma pierwiastków" << endl;
    else if (delta == 0)
    {
      x1 = -b / (2 * a);
      cout << "x=" << x1 << endl;
    }
    else
    {
      x1 = (-b - sqrt(delta)) / (2 * a);
      x2 = (-b + sqrt(delta)) / (2 * a);
      cout << "x1 = " << x1 << "\tx2 = " << x2 << endl;
    }
  }
  return 0;
}
```

Ćwiczenie 1.1.

Przeanalizuj przedstawiony powyżej algorytm, który został zapisany za pomocą różnych metod. Porównaj czytelność tych technik. Przetestuj przedstawione rozwiązania dla przykładowych danych.

1.2.2. Wzory Viète'a

Obliczając pierwiastki równania kwadratowego, możemy również korzystać ze **wzorów Viète'a**. Wzory te zostały sformułowane już w XVI wieku przez francuskiego matematyka François Viète'a.

Definicja

Jeżeli liczby x_1 i x_2 są pierwiastkami równania kwadratowego $ax^2 + bx + c = 0$, to zachodzą równości:

$$x_1 + x_2 = -\frac{b}{a},$$

$$x_1 \cdot x_2 = \frac{c}{a}.$$

Należy zwrócić uwagę na to, że podane powyżej wzory Viète'a można stosować tylko wtedy, gdy równanie kwadratowe ma rozwiązania.

Wzory te wykorzystywane są na przykład do określania znaków pierwiastków trójmianu bez konieczności rozwiązywania tego równania. Zastosować je można również do skorygowania błędów zaokrągleń występujących w algorytmie „delty".

1.2.3. Stabilny algorytm rozwiązujący równanie kwadratowe

Aby uniknąć błędów zaokrągleń w algorytmie „delty", do obliczania pierwiastków równania kwadratowego należy zastosować obie podane wcześniej metody. Wybór wzorów uzależniony jest od znaku współczynnika b. Wynika to stąd, że błąd zaokrąglenia spowodowany jest odejmowaniem bliskich co do wartości liczb. Zachodzi to w liczniku wzorów na pierwiastki równania w algorytmie „delty". Jeśli $b > 0$, to odejmowanie pojawia się we wzorze $x_2 = \frac{-b + \sqrt{\Delta}}{2a}$. W sytuacji gdy $b \leq 0$, odejmowanie wykonywane jest we wzorze $x_1 = \frac{-b - \sqrt{\Delta}}{2a}$. Aby wyeliminować to działanie, w miejsce wzoru generującego błędy wstawiamy odpowiednio przekształcony wzór Viète'a $x_1 \cdot x_2 = \frac{c}{a}$, wyznaczający iloczyn pierwiastków równania.

Poprawiając algorytm „delty", wprowadzamy zmiany, gdy spełniony jest warunek $\Delta > 0$. Wówczas, jeśli $b > 0$, wykonujemy działania:

$$x_1 = \frac{-b - \sqrt{\Delta}}{2a} \text{ i } x_2 = \frac{\frac{c}{a}}{x_1},$$

natomiast jeśli $b \leq 0$, realizujemy operacje:

$$x_2 = \frac{-b + \sqrt{\Delta}}{2a} \text{ i } x_1 = \frac{\frac{c}{a}}{x_2}.$$

Realizacja omawianego algorytmu została przedstawiona w postaci listy kroków i schematu blokowego (rysunek 1.3).

Lista kroków:

Krok 1. Jeśli $a = 0$, wypisz komunikat „to nie jest równanie kwadratowe" i zakończ algorytm.

Krok 2. (W tym przypadku $a \neq 0$). Przypisz $\Delta = b^2 - 4ac$.

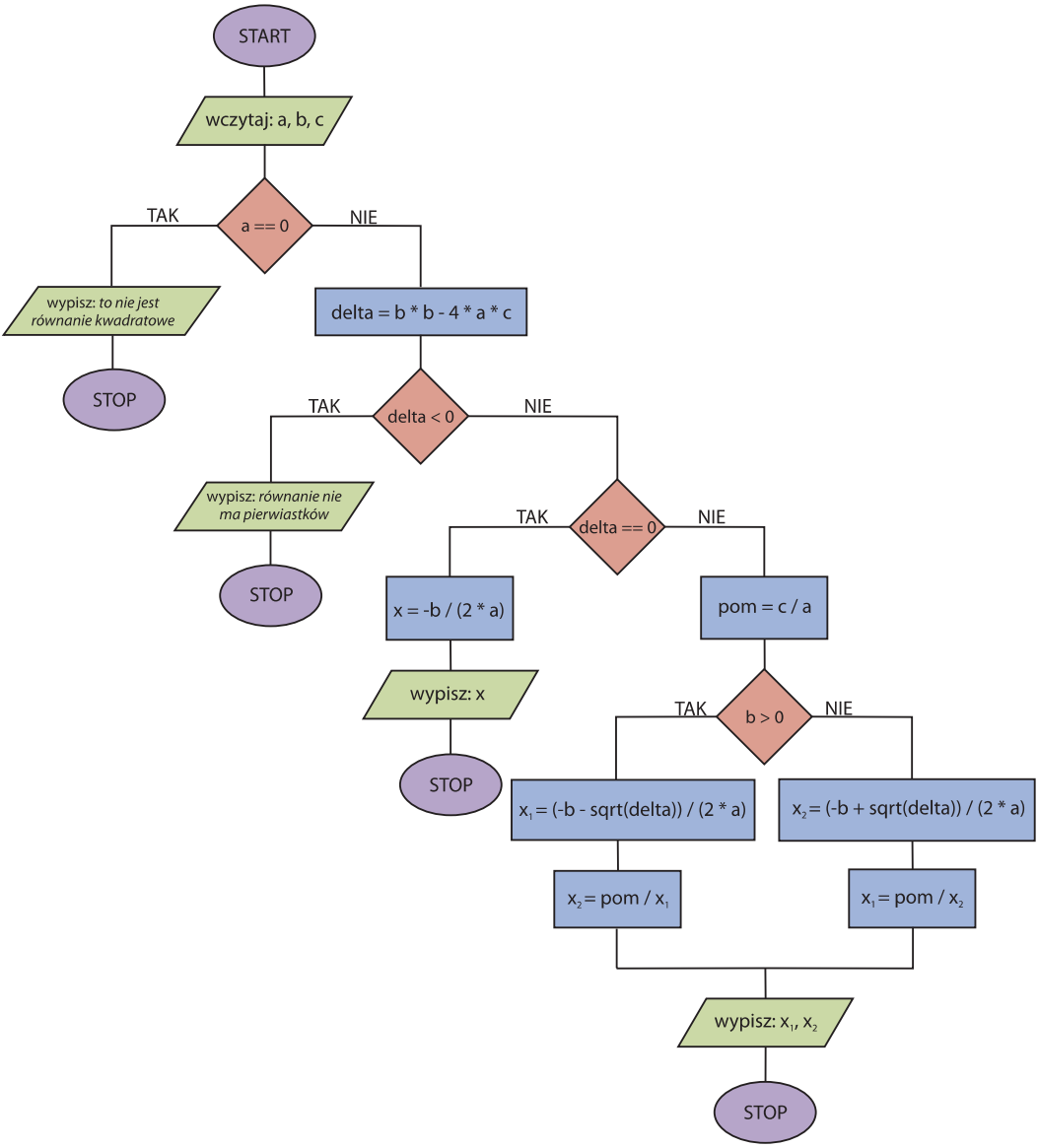
Krok 3. Jeśli $\Delta < 0$, wypisz komunikat „równanie nie ma pierwiastków" i zakończ algorytm.

Krok 4. (W tym przypadku $\Delta \geq 0$). Jeśli $\Delta = 0$, oblicz pierwiastek $x = \frac{-b}{2a}$, wypisz wyznaczony pierwiastek x i zakończ algorytm.

Krok 5. (W tym przypadku $\Delta > 0$). Przypisz $pom = \frac{c}{a}$.

Krok 6. Jeśli $b > 0$, oblicz pierwiastki $x_1 = \frac{-b - \sqrt{\Delta}}{2a}$, $x_2 = \frac{pom}{x_1}$, wypisz wartości wyznaczonych pierwiastków x_1 i x_2 oraz zakończ algorytm.

Krok 7. (W tym przypadku $b \leq 0$). Oblicz pierwiastki $x_2 = \frac{-b + \sqrt{\Delta}}{2a}$ i $x_1 = \frac{pom}{x_2}$, wypisz wartości wyznaczonych pierwiastków x_1 i x_2. Zakończ algorytm.



Rysunek 1.3. Schemat blokowy stabilnego algorytmu rozwiązującego równanie kwadratowe

Ćwiczenie 1.2.

Zapisz algorytm rozwiązujący równanie kwadratowe metodą stabilną (na podstawie listy kroków i schematu blokowego przedstawionego na rysunku 1.3) w postaci pseudokodu i programu. Przetestuj algorytm dla przykładowych danych.

Zadanie 1.1.

Przeanalizuj podany poniżej kod programu:

```
x, y, z = 0, 1, 2
if x: x += 1
elif x + y: y *= x + 3
elif y: x -= y * 3
x += y
print("x = ", x, "\ty = ", y)
```

Podaj wartość zmiennych x i y po wykonaniu podanego algorytmu. Utwórz schemat blokowy, listę kroków i pseudokod dla tego algorytmu.

Zadanie 1.2.

Podaj specyfikację zadania i skonstruuj algorytm w postaci schematu blokowego i programu wyznaczający wartość podanej funkcji:

$$f(x)=\begin{cases} 2x & dla \quad x<1 \\ -10 & dla \quad x=1 \\ (x-1)^4 & dla \quad x=3 \\ \sqrt{x-4} & dla \quad x=6 \\ 0 & dla \quad innych \quad x \end{cases}$$

Zmienną rzeczywistą x należy wprowadzić z klawiatury.

Zadanie 1.3.

Podaj specyfikację zadania i skonstruuj algorytm w postaci listy kroków i pseudokodu wyznaczający wartość podanej funkcji:

$$f(x)=\begin{cases} x^3+1 & dla \quad x<7 \\ \cos(x-1) & dla \quad x=7 \\ \sqrt{3x} & dla \quad x=9 \\ -8x & dla \quad innych \quad x \end{cases}$$

Zmienną rzeczywistą x należy wprowadzić z klawiatury.

Zadanie 1.4.

Poniżej podany jest pewien algorytm zapisany w pseudokodzie. Oblicza on wartość sumy dla danej liczby naturalnej *n* większej od 0. Użyty symbol ← często stosowany jest w pseudokodzie i oznacza operację przypisania wartości zmiennej.

```
suma ← 0
liczba_1 ← 1
dla i ← 1…n wykonuj
  liczba_1 ← liczba_1 + n
  liczba_2 ← 1
  dla j ← 1…i wykonuj
    liczba_2 ← liczba_2 * j
  suma ← suma + liczba_1 + liczba_2
```

Wykonaj poniższe polecenia:

a) Podaj specyfikację algorytmu.

b) Podaj, jakie wartości przyjmą zmienne *liczba_1*, *liczba_2* i *suma* w wyniku działania powyższego algorytmu dla $n = 3$. Obliczenia wykonaj bez użycia komputera.

c) Czy potrafisz zapisać wartość sumy obliczanej przez algorytm w postaci wzoru?

d) Zapisz powyższy algorytm w postaci schematu blokowego, listy kroków i programu. Przetestuj działanie tego algorytmu dla przykładowych danych.

Temat 2. Wprowadzenie do języka C++ — podstawowe konstrukcje i analiza przykładów

Znasz już podstawy języka Python, jednak nie jest to jedyny język programowania dopuszczony do matury. W tym podręczniku na poziomie rozszerzonym algorytmy zapisywane są w dwóch językach: Python i C++. Temat ten wprowadza Cię w świat nowego języka programowania C++. Kiedy poznasz podstawy języka C++, będziesz mógł wybrać jeden z tych dwóch języków do realizacji zagadnień algorytmicznych. Możesz też poznawać obydwa języki, co na pewno przyda Ci się w przyszłości.

Temat ten omawia podstawy języka programowania C++ niezbędne do realizacji zagadnień algorytmicznych zawartych w tym podręczniku. Rozwój projektowania i programowania w kierunku obiektowym spowodował zmiany w zakresie języka C. Rozszerzono go o elementy umożliwiające programowanie zorientowane obiektowo. C++ stanowi więc nadzbiór języka C. Różnica pomiędzy tymi językami jest znacząca, jednak wszystkie poprawne programy napisane w języku C są również poprawne w C++.

W podręczniku będziemy korzystać z **międzynarodowego standardu języka C++**, stworzonego przez komitet ASC (*Accredited Standards Committee*), który działa w ramach ANSI (*American National Standards Institute*). Standard ten określany jest najczęściej mianem **ANSI/ISO**.

Standard ANSI C++ stara się zapewnić **przenośność** tego języka. Oznacza to, że kod programu powinien być poprawny dla różnego typu komputerów pracujących w wielu systemach. Ponadto standard ten powinien być czytelny dla różnych kompilatorów.

2.1. Struktura programu

Program w języku C++ zbudowany jest z dwóch części:

- części deklaracyjnej,
- części operacyjnej.

Definicja

Część deklaracyjna obejmuje wszystkie deklarowane elementy programu, które mogą zostać wykorzystane w części operacyjnej. Ta część służy do definiowania bądź deklarowania różnych elementów, jednak nie można ich w tym miejscu uruchomić. Do tego wykorzystuje się część operacyjną.

Definicja

Część operacyjna to program główny, który w tym języku jest funkcją o nazwie `main()`. Po uruchomieniu programu wykonywane są wszystkie polecenia zawarte w funkcji `main()`. W tej części programu korzystamy ze zdefiniowanych lub zadeklarowanych przez siebie elementów znajdujących się w części deklaracyjnej.

Na początkowym etapie nauki języka C++ będziesz korzystać z uproszczonej budowy programu. W następnych podrozdziałach poznasz bardziej zaawansowane możliwości konstruowania programu.

Przykład 2.1.

Przyjrzyjmy się konstrukcji pierwszego programu, który został przedstawiony w tabeli 2.1. Opisy umieszczone z prawej strony wyjaśniają znaczenie poszczególnych elementów implementacji. W języku C++ **rozróżniane są małe i wielkie litery**, kod programu należy więc zapisywać dokładnie tak, jak podano w podręczniku.

Tabela 2.1. Pierwszy program w języku C++

Kod programu	Opis
CZĘŚĆ DEKLARACYJNA	
`#include <iostream>`	Dyrektywa preprocesora — **preprocesor** to program dokonujący wstępnego przeglądu kodu źródłowego programu **Deklaracja bibliotek** (czyli plików nagłówkowych), z których korzystamy w programie Biblioteka `iostream` zawiera definicje standardowego strumienia wejścia `cin` i standardowego strumienia wyjścia `cout`
`using namespace std;`	Deklaracja korzystania ze **standardowej przestrzeni nazw** `std` (czyli biblioteki standardowej)

Kod programu	Opis
CZĘŚĆ OPERACYJNA	
`int main()`	Nagłówek funkcji głównej
`{`	Nawias otwierający (początek funkcji)
`int a, b, c;`	Deklaracja zmiennych lokalnych `a`, `b`, `c`
`a = 2;`	Przypisanie zmiennej `a` wartości 2
`cout << "podaj wartość zmiennej b:";`	Komunikat wypisany na ekranie „podaj wartość zmiennej b:”
`cin >> b;`	Wczytanie z klawiatury wartości zmiennej `b`
`c = a + b;`	Przypisanie zmiennej `c` sumy zmiennych `a` i `b`
`cout << a << " + " << b << " = " << c << endl;`	Wypisanie wyniku na ekranie
`return 0;`	Przypisanie funkcji `main()` wartości 0
`}`	Nawias zamykający (koniec funkcji)

Wskazówka

Nagłówek funkcji głównej `int main()` w pełnej postaci zapisywany jest następująco: `int main(void)`, gdzie:

- `int` oznacza, że typ wartości zwracanej z funkcji `main()` jest całkowity, funkcji tej przypisywana jest więc wartość typu `int`;
- `main` to nazwa funkcji;
- `void` oznacza „pusty”, a w tym przypadku brak parametrów, które wpisuje się w nawiasie; pusty nawias jest równoważny `void`.

W razie pominięcia deklaracji przestrzeni nazw standardowych `std` operacje wejścia i wyjścia dostępne są dopiero po podaniu nazwy klasy `std`, czyli `std::cout`, `std::cin`. Program z takimi zmianami przedstawiono poniżej:

```
#include <iostream>
int main()
{
  int a, b, c;
  a = 2;
  std::cout << "podaj wartość zmiennej b:";
  std::cin >> b;
  c = a + b;
  std::cout << a << " + " << b << " = " << c << std::endl;
  return 0;
}
```

2.2. Operacje wejścia-wyjścia

Strumieniowe operacje wejścia-wyjścia dostępne są po dołączeniu biblioteki `iostream`. Korzystać możemy z dwóch standardowych strumieni:

- `cout` — standardowy strumień wyjściowy, który zwykle skojarzony jest z monitorem; stosujemy tutaj operator kierunkowy wstawiania znaków do strumienia wyjściowego: `<<`;
- `cin` — standardowy strumień wejściowy, który zwykle skojarzony jest z klawiaturą; wykorzystujemy w tym przypadku operator kierunkowy pobierania danych ze strumienia do podanej zmiennej: `>>`.

Wypisując komunikat poleceniem `cout`, w łańcuchu oznaczanym cudzysłowem można stosować **znaki specjalne**. W tabeli 2.2 podano zestawienie podstawowych znaków specjalnych.

Tabela 2.2. Zestawienie znaków specjalnych

Znak specjalny	Opis
`\n`	*new line* (przejście do nowej linii)
`\t`	*tab* (tabulator poziomy)
`\v`	*vertical tab* (tabulator pionowy)
`\b`	*backspace* (przesunięcie kursora o jeden znak w lewo)
`\r`	*carriage return* (przesunięcie kursora na początek bieżącego wiersza)
`\'`	' (wypisanie znaku zastrzeżonego — apostrof)
`\"`	" (wypisanie znaku zastrzeżonego — cudzysłów)
`\?`	? (wypisanie znaku zastrzeżonego — znak zapytania)
`\\`	\ (wypisanie znaku zastrzeżonego — *backslash*)

Przykład 2.2.

Przyjrzyjmy się przykładom zastosowania strumieniowych operacji wejścia i wyjścia przedstawionym w tabeli 2.3. Z prawej strony umieszczono komentarze wyjaśniające wykonywane operacje.

Tabela 2.3. Przykłady zastosowania operacji wejścia i wyjścia

Przykładowe operacje wejścia i wyjścia	Efekt działania polecenia	Opis
`cout << "tekst" << " " << "tekst";`	*tekst tekst*	Nie ma potrzeby wydzielania poszczególnych słów wypisywanego tekstu, taki sam efekt uzyskamy poleceniem: `cout << "tekst tekst";`
`cout << "tekst\ttekst\ntekst";`	*tekst tekst* *tekst*	Wewnątrz wyświetlanego łańcucha można stosować znaki specjalne
`int a = 5, b = 3;` `cout << "a=" << a << "\n" << a + b;`	*a=5* *8*	W strumieniu wyjściowym można umieszczać zarówno komunikaty, jak i zmienne, które należy od siebie oddzielać operatorem `<<` Znak specjalny `\n` stosowany pojedynczo można zapisywać `"\n"` lub `'\n'`
`int a;` `cin >> a;`	Oczekiwanie na wpisanie z klawiatury wartości zmiennej `a` i potwierdzenie tego klawiszem *Enter*	Wczytanie z klawiatury wartości zmiennej `a`
`int a, b, c;` `cin >> a >> b >> c;`	Oczekiwanie na wpisanie z klawiatury wartości zmiennych `a`, `b` i `c` oddzielonych przez spacje lub *Enter* oraz potwierdzenie tego klawiszem *Enter*	Przy wczytywaniu z klawiatury wartości wielu zmiennych wygodnie jest wykonać to za pomocą jednej operacji `cin` Wczytywane zmienne mogą być różnego typu

Do formatowania strumieniowych operacji wejścia i wyjścia najczęściej wykorzystuje się **manipulatory**. Najważniejsze manipulatory przedstawione zostały w tabeli 2.4.

Tabela 2.4. Zestawienie manipulatorów strumieniowych

Manipulator	Opis
`dec`	Włącza konwersję dziesiętną
`hex`	Włącza konwersję szesnastkową
`oct`	Włącza konwersję ósemkową
`endl`	Umieszcza w strumieniu znak końca wiersza i opróżnia strumień
`ends`	Umieszcza znak `'\0'` na końcu łańcucha
`setfill(char c)`	Ustala znak wypełniania wolnych miejsc pola na `c` (dostępny po dołączeniu biblioteki `iomanip`)
`setprecision(int n)`	Określa precyzję liczb rzeczywistych równą `n` (dostępny po dołączeniu biblioteki `iomanip`)
`setw(int n)`	Wyznacza szerokość pola równą `n` znaków (dostępny po dołączeniu biblioteki `iomanip`)

Przykład 2.3.

Przyjrzyj się przykładom zastosowania manipulatorów strumieniowych w przedstawionym poniżej programie:

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  int k = 241;
  cout << "k = " << k << endl;
  cout << "konwersja systemów liczbowych:" << endl;
  cout << "szesnastkowy = " << hex << k << endl;
  cout << "dziesiętny = " << dec << k << endl;
  cout << "szesnastkowy = " << setfill('0') << setw(6) << hex << k << endl;
  cout << "ósemkowy = " << setfill('*') << setw(10) << oct << k << endl;
  return 0;
}
```

Wyniki:

k = 241
konwersja systemów liczbowych:
szesnastkowy = f1
dziesiętny = 241
szesnastkowy = 0000f1
*ósemkowy = *******361*

Powyższy program wypisuje wartość zmiennej `k` w różnych systemach liczbowych z podaną szerokością pola i znakiem wypełniania.

Przykład 2.4.

Przykład pokazuje zastosowanie manipulatorów dla zmiennych typu rzeczywistego:

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  double x = 22.111111111, y = 3.5555555555555, w;
  cout << "podaj liczbę rzeczywistą: ";
  cin >> w;
  cout << "x = " << setw(10) << x << endl;
  cout << "y = " << setfill('0') << setw(12) << y << endl;
  cout << "y = " << setprecision(8) << setfill('0') << setw(12) << y << endl;
  cout << "w = " << setfill('*') << setw(15) << w << endl;
  cout << "x + y + w = " << setprecision(9) << x + y + w << endl;
  cout << "x * y = " << setprecision(8) << x * y << endl;
  return 0;
}
```

Przykładowe wyniki:

podaj liczbę rzeczywistą: 2.5

x = 22.1111

y = 000003.55556

y = 0003.5555556

*w = **********2.5*

x + y + w = 28.1666667

*x * y = 78.617284*

Ćwiczenie 2.1.

Na podstawie kodu programu podanego w przykładzie 2.4 i wyników jego uruchomienia określ, w jaki sposób ustawiana jest precyzja dla liczb rzeczywistych za pomocą manipulatora `setprecision()`.

Podaj, z jaką precyzją wypisywane są wartości zmiennych rzeczywistych bez zastosowania manipulatora precyzji.

2.3. Zmienne, stałe, wskaźniki i referencje

2.3.1. Zmienne

Definicja

Zmienną nazywamy identyfikator, który może mieć przypisaną określoną wartość (można ją zmieniać podczas realizacji programu). Aby korzystać ze zmiennej, należy ją wcześniej zadeklarować i jednocześnie określić jej typ.

Deklaracja zmiennej ma postać:

```
typ identyfikator;
```

Nazwy zmiennych powinny być czytelne i ułatwiać piszącemu program dokonywanie w nim poprawek.

Przykład 2.5.

Przyjrzyjmy się przykładom **deklaracji zmiennych**:

```
int A;
float b, c, d;
char nazwa;
unsigned long int odleglosc;
string kolor_samochodu;
```

Podczas deklaracji zmiennych możliwe jest nadawanie im wartości początkowych przez **inicjalizację zmiennych**:

```
double liczba = 0.00125;
short int suma = 0, iloczyn = 1;
char znak = 'a';
```

2.3.2. Stałe

Definicja

Stałą nazywamy identyfikator o przypisanej określonej wartości, której podczas realizacji programu nie można zmieniać. Aby korzystać ze stałej, należy ją wcześniej zadeklarować i jednocześnie określić jej typ.

Deklaracja stałej ma postać:

```
const typ identyfikator = wartość;
```

Nazwy stałych, podobnie jak zmiennych, powinny być czytelne i ułatwiać piszącemu program dokonywanie w nim poprawek.

Przykład 2.6.

Przyjrzyjmy się przykładom deklaracji stałych:

```
const int B = 10;
const double a = 2.34, b = 3.125;
const char znak = 'W';
```

2.3.3. Wskaźniki

Definicja

Wskaźniki to adresy danych, struktur danych, obiektów klas lub funkcji. Wskaźnik zawiera informację o adresie oraz typie wskazywanego elementu w pamięci komputera.

Za pomocą operatora `*` możemy deklarować zmienne wskaźnikowe:

```
typ *identyfikator;
```

Wówczas `*identyfikator` to wartość zmiennej wskaźnikowej (czyli dynamicznej), a `identyfikator` to jej adres.

W języku C++ do dynamicznego przydzielania i zwalniania pamięci stosuje się operatory `new` i `delete`. Operator `new` służy do przydzielania pamięci i ma składnię:

```
identyfikator = new typ_obiektu;
```

Wartością wyrażenia jest tutaj wskaźnik do utworzonego obiektu.

Operator `delete` wykorzystywany jest do zwalniania pamięci i ma następującą składnię:

```
delete identyfikator;
```

Operator `delete` usuwa obiekt, który jest wskazywany przez `identyfikator`.

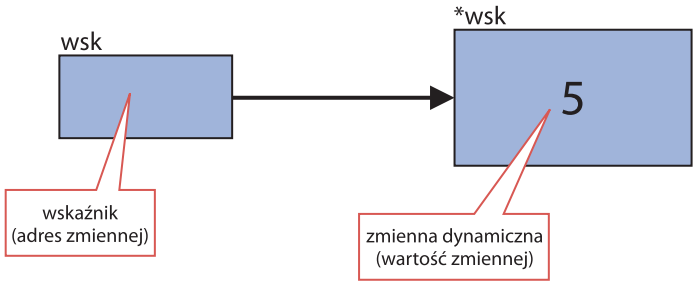
Przykład 2.7.

Poniżej przedstawiono przykład dynamicznej alokacji pamięci dla zmiennej typu `int`:

```
int *wsk;
wsk = new int;
   // operacje wykonywane na zmiennej *wsk, na przykład: *wsk=5;
delete wsk;
```

Na rysunku 2.1 pokazano efekt wykonania operacji `new`. Zmienna `wsk` zawiera adres wskazujący na zmienną dynamiczną `*wsk`, której następnie przypisywana jest wartość 5.

Do wyznaczenia wskaźnika zmiennej zadeklarowanej statycznie używany jest operator adresu `&`. Wykorzystuje się go do nadawania wartości początkowych zmiennym wskaźnikowym, a następnie dokonywania na nich zmian.



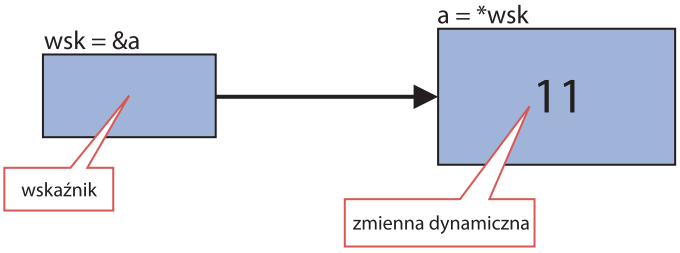
Rysunek 2.1. Powiązanie wskaźnika ze zmienną dynamiczną

Przykład 2.8.

Przeanalizujmy przykład zastosowania operatora adresu:

```
int a;
int *wsk;
wsk = &a;
*wsk = 11;
```

Na rysunku 2.2 przedstawiono efekt wykonania tych operacji.



Rysunek 2.2. Dwie nazwy dla jednej zmiennej dynamicznej

Wskaźnik `wsk` zawiera adres zmiennej dynamicznej `a`. Powoduje to, że zmienne `*wsk` i `a` są równoważne. Zmiana wartości zmiennej dynamicznej `*wsk` powoduje zmiany zmiennej `a`, i odwrotnie. Przyczyna jest prosta — obydwie zmienne mają ten sam adres, a więc przeznaczone dla nich miejsce w pamięci komputera ma dwie nazwy.

2.3.4. Referencje

Referencje, oznaczane znakiem &, zastępują zmienne, struktury danych lub obiekty klas. Operacje realizowane na referencji wykonywane są na reprezentowanej przez nią zmiennej, strukturze czy obiekcie.

Definicja

Referencja zawsze musi mieć wartość początkową, która wiąże ją z reprezentowaną zmienną, strukturą lub obiektem:

```
typ &indentyfikator = zmienna;
```

Przykład 2.9.

Przyjrzyjmy się przykładowi zastosowania referencji:

```
int a, b;
int &ref = a;
ref = 10;
b = ref;
```

Po wykonaniu powyższych instrukcji zmienna `a` reprezentowana przez referencję `ref` ma wartość 10. Zmienna `b`, będąca zmienną, która nie jest powiązana z referencją `ref`, również ma wartość 10. Operacje wykonywane na referencji `ref` i zmiennej `a` są równoważne.

2.4. Wyrażenia arytmetyczne, relacje i operatory logiczne

2.4.1. Operator przypisania (podstawienia)

Operator przypisania (tabela 2.5) realizuje nadanie wartości zmiennej. Z lewej strony tego operatora musi być określona zmienna, z prawej — wyrażenie, którego wartość przypisujemy tej zmiennej.

Tabela 2.5. Operator przypisania

Operator	Opis
`=`	Przypisanie wartości zmiennej

Przykład 2.10.

Przeanalizujmy przykłady zastosowania operatora przypisania:

```
a = 4;
b = a + 2;
```

`a = (b = 8) - 3;` — Zagnieżdżenie instrukcji przypisania, równoważne instrukcjom `b = 8; a = b - 3`.

`a = b = c = 0;` — Wielokrotna instrukcja przypisania, równoważna instrukcjom `c = 0; b = c; a = b`.

2.4.2. Operatory arytmetyczne

Operatory arytmetyczne (tabela 2.6) to operatory dwuargumentowe realizujące operacje arytmetyczne.

Operator dzielenia ma dwa zastosowania, które zależą od typu argumentów. Jeśli obydwa argumenty są całkowite, wynik dzielenia również jest liczbą całkowitą. W tym przypadku uzyskujemy część całkowitą

Tabela 2.6. Zestawienie operatorów arytmetycznych

Operator	Opis
`*`	Mnożenie
`/`	Dzielenie
`+`	Dodawanie
`-`	Odejmowanie
`%`	Reszta z dzielenia

wyniku z dzielenia tych liczb. W sytuacji gdy chociaż jeden z argumentów jest rzeczywisty, wynikiem jest również liczba rzeczywista.

Przykład 2.11.

Przeanalizuj przedstawione w tabeli 2.7 przykłady dzielenia liczb całkowitych i rzeczywistych zapisane w języku C++.

Tabela 2.7. Przykłady wyrażeń zapisanych w języku C++ wykorzystujących operator dzielenia

Wyrażenie	Wartość wyrażenia
`9.0 / 2.0`	*4.5*
`9 / 2.0`	*4.5*
`9.0 / 2`	*4.5*
`9 / 2`	*4*

Tylko w jednym przypadku uzyskujemy wynik będący liczbą całkowitą. Aby wynik był liczbą całkowitą, obydwa argumenty muszą być wartościami całkowitymi.

Jeśli argumenty wykonywanej operacji dzielenia są zmiennymi całkowitymi, a potrzebny jest wynik rzeczywisty, pojawia się problem. Aby go rozwiązać, należy wymusić, by wynik dzielenia był rzeczywisty. Stosujemy wówczas tak zwane **rzutowanie**, pokazane na poniższym przykładzie:

```
int a = 5, b = 2;
cout << a / b << "\t" << (double)a / b << endl;
```

Wyniki:

2 2.5

2.4.3. Operatory zmniejszania i zwiększania

Wyróżniamy dwa rodzaje operatorów zmniejszania i zwiększania (tabela 2.8):

- prefiksowe — `++a`, `--a`,
- postfiksowe — `a++`, `a--`.

Tabela 2.8. Zestawienie operatorów zmniejszania i zwiększania

Operator	Opis
`++`	Zwiększanie argumentu o 1 (inkrementacja)
`--`	Zmniejszanie argumentu o 1 (dekrementacja)

Jeśli operator występuje wewnątrz wyrażenia, to przy zapisie prefiksowym zwiększanie lub zmniejszanie argumentu jest wykonywane przed obliczeniem wartości wyrażenia, natomiast przy zapisie postfiksowym — po obliczeniu wartości wyrażenia. W tabeli 2.9 podano przykłady zastosowań tych operatorów.

Tabela 2.9. Przykłady zastosowań operatorów zmniejszania i zwiększania w zapisie postfiksowym i prefiksowym

Przykład zastosowania	Wartości zmiennych po wykonaniu operacji

`int a = 3, b = 4, c;`
`++a;`
`b--;`
`c = a + b;`
	a = 4
	b = 3
	c = 7

`int a = 3, b = 4, c;` `c = ++a + b;`	a = 4 b = 4 c = 8
`int a = 3, b = 4, c;` `c = a + b--;`	a = 3 b = 3 c = 7
`int a = 3, b = 4, c;` `c = ++a + b--;`	a = 4 b = 3 c = 8

2.4.4. Operatory relacyjne

W tabeli 2.10 podano operatory relacyjne, z których można korzystać w języku C++.

Tabela 2.10. Zestawienie operatorów relacyjnych

Operator	Opis
<	Mniejsze
<=	Mniejsze lub równe
==	Równe
!=	Różne (nieprawda, że równe)
>=	Większe lub równe
>	Większe

2.4.5. Operatory logiczne

Wartość wyrażenia logicznego, w którym korzystamy z operatorów logicznych (tabela 2.11), jest zawsze typu całkowitego.

Tabela 2.11. Zestawienie operatorów logicznych

Operator	Opis
!	Negacja
&&	Koniunkcja (iloczyn logiczny)
||	Alternatywa (suma logiczna)

W języku C++ dowolna niezerowa wartość interpretowana jest jako prawda (czyli `true`), natomiast wartość 0 jako fałsz (czyli `false`).

Przykład 2.12.

Przeanalizujmy następujący przykład:

```
int m = 8, n = 0, wynik;
wynik = m || n;
cout << wynik << "\t";
wynik = m && n;
cout << wynik << "\t";
wynik = !m;
cout << wynik << "\t";
wynik = !n;
cout << wynik << endl;
```

Wyniki:

1 0 0 1

Wynika to stąd, że `m` równe 8 interpretowane jest jako prawda, a `n` równe 0 jako fałsz. Jeśli wynikiem wykonanej operacji jest prawda, to wypisywana jest wartość 1, natomiast jeśli fałsz — wartość 0.

Przykład 2.13.

Następny przykład pokazuje zastosowanie operatora logicznego `&&` w konstruowaniu warunku w instrukcji warunkowej `if`:

```
int m = -3, n = -7;
if (m < 0 && n < 0) cout << "liczby są ujemne" << endl;
else cout << "co najmniej jedna liczba nie jest ujemna" << endl;
```

Wynik:

liczby są ujemne

Liczby `m` i `n` są ujemne, spełnione są więc obydwa warunki, dlatego pojawia się ten komunikat. Gdyby chociaż jedna z tych liczb była nieujemna, wyrażenie byłoby fałszywe i wypisany zostałby komunikat „co najmniej jedna liczba nie jest ujemna”.

2.4.6. Złożone operatory przypisania

Złożone operatory przypisania stosuje się do zapisywania wyrażeń `X = X • Y` w postaci `X •= Y`, gdzie `•` to operator dwuargumentowy. Do najważniejszych z nich należą:

`+=` `-=` `*=` `/=` `%=`

Przykład 2.14.

W tabeli 2.12 podano przykłady zastosowania złożonych operatorów przypisania w konstruowaniu wyrażeń.

Tabela 2.12. Przykłady zastosowania złożonych operatorów przypisania

Wyrażenie	Zapis wyrażenia bez złożonego operatora przypisania
`a *= 4 * b - 1;`	`a = a * (4 * b - 1);`
`m %= 3 - n / 2;`	`m = m % (3 - n / 2);`
`b -= 3 + d * 4;`	`b = b - (3 + d * 4);`
`k += (a *= 2) - (b += 3);`	`k = k + ((a = a * 2) - (b = b + 3));`

Zwróć uwagę na kolejność wykonywanych działań.

2.4.7. Operator warunkowy

Operator warunkowy jest jedynym w języku C++ operatorem trzyargumentowym. Jego składnia jest następująca:

$$w_1 \ ? \ w_2 \ : \ w_3$$

gdzie: w_1, w_2, w_3 — wyrażenia.

Wykonanie powyższej operacji przebiega następująco:

1. Obliczenie wartości wyrażenia w_1.
2. Jeśli wartość wyrażenia w_1 jest różna od 0 (czyli wyrażenie jest prawdziwe), rezultatem operacji jest wartość wyrażenia w_2. Wyrażenie w_3 nie jest wówczas obliczane.
3. Jeśli wartość wyrażenia w_1 jest równa 0 (czyli wyrażenie jest fałszywe), rezultatem operacji jest wartość wyrażenia w_3. Wyrażenie w_2 nie jest w tym przypadku obliczane.

Realizacja tego działania przypomina instrukcję warunkową `if`, która została omówiona w punkcie 2.8.2, „Instrukcje warunkowe".

Przykład 2.15.

Zastosujmy operator warunkowy do wyznaczenia najmniejszej wartości dla dwóch liczb całkowitych wprowadzanych z klawiatury:

```
int a, b, minimum;
cout << "podaj dwie liczby całkowite:" << endl;
cin >> a >> b;
minimum = a < b ? a : b;
cout << "minimum = " << minimum << endl;
```

Przykładowy wynik:

podaj dwie liczby całkowite:
12
7
minimum = 7

Skorowidz

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

T

U

V

W

Z

Technologia informacyjna i informatyka to dziedziny, których wykorzystanie i dostępność stale rosną, a tempo zachodzących w nich zmian jest nieporównywalne z rozwojem innych dyscyplin. Umiejętność biegłego posługiwania się komputerem i urządzeniami peryferyjnymi oraz znajomość obsługi pakietów biurowych są obowiązkowe na rynku pracy. Wkrótce każdy z nas będzie musiał się również wykazać wiedzą z zakresu programowania, tworzenia grafiki czy zagadnień związanych z prawem w sieci.

Informatyka w szkole ponadpodstawowej na poziomie rozszerzonym:

- bazuje na wiedzy, którą zdobywa się na poziomie podstawowym, i stanowi jej uzupełnienie
- przygotowuje do studiów informatycznych
- jest przedmiotem maturalnym

Dzięki pracy z książką ***Informatyka Europejczyka. Podręcznik dla szkół ponadpodstawowych. Zakres rozszerzony. Część 1*** poszerzysz znajomość języków tekstowych, takich jak Python i C++. Zgłębisz wiedzę na temat algorytmów i zaczniesz ją wykorzystywać między innymi w zadaniach z zakresu kryptografii i kryptoanalizy. Opanujesz przydatne metody edycji dźwięku i kompresji danych. Przygotujesz w programie Blender przestrzenne wizualizacje i animacje. Zastosujesz zaawansowane funkcje arkusza kalkulacyjnego dla różnego typu danych, a do zdefiniowania makropoleceń użyjesz wbudowanego do niego języka programowania VBA. Ponadto zaprojektujesz i utworzysz relacyjną bazę danych, przygotujesz formularze, kwerendy i raporty. Poznasz język zapytań SQL. Zadbasz o integralność i bezpieczeństwo zgromadzonych danych.

W skład kompletnego pakietu edukacyjnego dla szkoły ponadpodstawowej wchodzą także podręczniki:

Podręczniki z serii *Informatyka Europejczyka* ułatwią uczniom zdobywanie wiedzy i umiejętności podczas wykonywania ćwiczeń praktycznych, a nauczycielom — przekazywanie nowego materiału w interesujący i niebanalny sposób.

Wciśnij Enter i do dzieła!

http://edukacja.helion.pl

Helion EDUKACJA
zamówienia telefoniczne
0 801 339900
0 601 339900
księgarnia internetowa
http://helion.pl
Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl
Informatyka w najlepszym wydaniu
ISBN 978-83-283-4188-3
9 788328 341883