owoce PROGRAMOWANIA

Helion

>>> JĘZYK C++

WYDANIE IX

TONY GADDIS

Tytuł oryginału: Starting Out with C++ from Control Structures through Objects (9th Edition)

Tłumaczenie: Patryk Wierzchoń (rozdz. 1 – 6, dodatki), Andrzej Watrak (rozdz. 7 – 21)

ISBN: 978-83-283-4680-2

Authorized translation from the English language edition, entitled: STARTING OUT WITH C++ FROM CONTROL STRUCTURES THROUGH OBJECTS, Ninth Edition; ISBN 0134498372; by Tony Gaddis; published by Pearson Education, Inc.

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/jezcow
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 3

Wyrażenia i interaktywność

TEMATYKA

3.1. Obiekt cin

WYJAŚNIENIE: **Obiekt `cin` możemy wykorzystać do odczytu informacji wpisywanych na klawiaturze.**

Jak dotąd pisałeś programy, które miały dane wbudowane w kod. Użytkownik nie miał możliwości wprowadzenia własnych danych, a początkowe wartości zmiennych zostały zainicjowane przez twórcę kodu. Tego typu programy ograniczają się tylko do jednego zestawu początkowych danych. Każda zmiana danych oznacza modyfikację kodu i ponowną kompilację.

W praktyce większość programów wymaga wprowadzenia do nich danych, które zostaną przypisane do zmiennych. Oznacza to, że nie trzeba modyfikować programu za każdym razem, gdy użytkownik chce zmienić zestaw danych. Na przykład program, który oblicza wynagrodzenia w małej firmie, będzie wymagał od użytkownika wprowadzenia imienia i nazwiska pracownika, liczby przepracowanych godzin i wynagrodzenia za

godzinę pracy. Po wydrukowaniu informacji o wysokości wypłaty pracownika program może zacząć od początku i poprosić o wpisanie danych dla kolejnego pracownika.

Tak jak `cout` jest standardowym obiektem wyjścia języka C++, tak `cin` jest standardowym obiektem wejścia. Odczytuje on wejście z konsoli (lub z klawiatury) w sposób pokazany w programie na listingu 3.1.

Zamiast obliczać pole tylko jednego prostokąta, program może obliczyć pole dowolnego prostokąta. Wartości zapisywane w zmiennych `length` i `width` wprowadzane są przez użytkownika w trakcie pracy programu. Spójrz na wiersze 13. – 14.:

```
cout << "Jaka jest długość prostokąta? ";
cin >> length;
```

Listing 3.1

```
1  // Program prosi użytkownika o wpisanie długości i szerokości
2  // prostokąta. Oblicza jego pole i wyświetla
3  // obliczoną wartość na ekranie.
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9     int length, width, area;
10
11   cout << "Ten program oblicza pole";
12   cout << "prostokąta.\n";
13   cout << "Jaka jest długość prostokąta? ";
14   cin >> length;
15   cout << "Jaka jest szerokość prostokąta? ";
16   cin >> width;
17    area = length * width;
18    cout << "Pole prostokąta wynosi " << area << ".\n";
19    return 0;
20 }
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Ten program oblicza pole prostokąta.
Jaka jest długość prostokąta? 10 [Enter]
Jaka jest szerokość prostokąta? 20 [Enter]
Pole prostokąta wynosi 200.
```

W wierszu 13. wyrażenie `cout` zostało wykorzystane do wyświetlenia pytania `"Jaka jest długość prostokąta?"`. To pytanie nazywamy **zachętą**. Wskazuje ono użytkownikowi, jakie dane powinny zostać wprowadzone. Program zawsze powinien wyświetlić zachętę, zanim użyje obiektu `cin`. W ten sposób użytkownik będzie wiedział, że musi wpisać wartość za pomocą klawiatury.

W wierszu 14. wykorzystano `cin` do odczytania wartości z klawiatury. Symbol `>>` nazywamy **operatorem ekstrakcji strumienia**. Pobiera on znaki z obiektu strumienia, który znajduje się po lewej, i zapisuje je w zmiennej, po prawej stronie operatora. W tym wierszu wartość zostaje pobrana z obiektu `cin` (a ten pobiera ją z klawiatury) i zapisana w zmiennej `length`.

Pobieranie wejścia od użytkownika to zazwyczaj proces dwuetapowy:

1. Wyświetlenie zachęty za pomocą `cout`.
2. Pobranie wejścia z klawiatury za pomocą `cin`.

Zachęta powinna być pytaniem do użytkownika albo wskazaniem, jaką wartość powinien wpisać. Na przykład kod z listingu 3.1 wyświetla następującą zachętę:

```
Jaka jest długość prostokąta?
```

Użytkownik, widząc takie pytanie, wie, że musi wprowadzić długość prostokąta. Po wyświetleniu zachęty program wykorzystuje obiekt `cin` do pobrania wartości z klawiatury i zapisania jej w zmiennej `length`.

Zauważ, że operatory << i >> wskazują na kierunek, w którym przepływają dane. W wyrażeniu korzystającym z `cout` operator << zawsze wskazuje na `cout`. Wskazuje więc, że dane przepływają ze zmiennej do obiektu. W wyrażeniu korzystającym z `cin` operator >> zawsze wskazuje zmienną, która otrzymuje wartość, zatem przepływ danych odbywa się od obiektu `cin` do zmiennej. Zostało to przedstawione na rysunku 3.1.

```
cout << "Jaka jest długość prostokąta? ";
cin >> length;
```

Potraktuj operatory << i >> jako strzałki, które wskazują na kierunek przepływu danych

```
cout ←— "Jaka jest długość prostokąta? ";
cin —→ length;
```

Rysunek 3.1. Operatory << i >>

Obiekt `cin` powoduje wstrzymanie programu do czasu wpisania danych i naciśnięcia klawisza *Enter*. Program nie przejdzie do wykonywania kolejnych wierszy, dopóki `cin` nie otrzyma danych.

Obiekt `cin` automatycznie konwertuje dane odczytane z klawiatury na typ odpowiadający zmiennej wskazanej w wyrażeniu. Jeżeli użytkownik wpisze `10`, `cin` odczyta to jako znaki `'1'` i `'0'`. Jest to na tyle sprytny obiekt, że sam zamieni wpisane znaki na typ `int`, zanim zapisze je w zmiennej. Jest on również na tyle sprytny, że nie zapisze wartości `10.7` w zmiennej typu `int`. Jeśli użytkownik wprowadzi liczbę zmiennoprzecinkową w wyrażeniu wpisującym wejście użytkownika w zmiennej `int`, `cin` pominie część ułamkową.

UWAGA. W każdym programie, który korzysta z `cin`, trzeba załączyć plik nagłówkowy `<iostream>`.

Wprowadzanie wielu wartości

Obiekt `cin` możemy wykorzystać do zebrania kilku wartości jednocześnie. Przyjrzyjmy się listingowi 3.2, na którym znajduje się nieco zmodyfikowany program z listingu 3.1.

Listing 3.2

```
1 // Program prosi użytkownika o wpisanie długości i szerokości
2 // prostokąta. Oblicza jego pole i wyświetla
3 // obliczoną wartość na ekranie.
4 #include <iostream>
5 using namespace std;
6
7 int main()
8 {
9    int length, width, area;
10
11    cout << "Ten program oblicza pole ";
12    cout << "prostokąta.\n";
13    cout << "Wprowadź długość i szerokość prostokąta, ";
14    cout << "rozdzielając je spacją.\n";
15    cin >> length >> width;
16    area = length * width;
17    cout << "Pole prostokąta wynosi " << area << "." << endl;
18    return 0;
19 }
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Ten program oblicza pole prostokąta.
Wprowadź długość i szerokość prostokąta, rozdzielając je spacją.
10 20 [Enter]
Pole prostokąta wynosi 200.
```

W wierszu 15. program oczekuje na wpisanie przez użytkownika dwóch wartości. Pierwsza z nich zostanie zapisana w zmiennej `length`, a druga w `width`.

```
cin >> length >> width;
```

W przykładowym wyjściu użytkownik wprowadził wartości 10 i 20, więc program wpisał 10 w zmiennej `length` i 20 w `width`.

Zwróć uwagę, że wprowadzone wartości są rozdzielone spacjami. W ten sposób `cin` może rozróżnić początek i koniec każdej z liczb. Liczba spacji nie ma znaczenia. Użytkownik mógłby wpisać przykładowo:

```
10              20
```

UWAGA. Klawisz *Enter* należy nacisnąć po wprowadzeniu ostatniej wartości.

Obiekt `cin` jest w stanie odczytać również kilka wartości różnych typów, jak na listingu 3.3.

Listing 3.3

```
1 // Program przedstawia, jak cin może odczytać wiele wartości
2 // danych różnego typu.
3 #include <iostream>
```

```
 4 using namespace std;
 5
 6 int main()
 7 {
 8    int whole;
 9    double fractional;
10    char letter;
11
12    cout << "Wpisz liczbę całkowitą, liczbę z ułamkiem i literę: ";
13    cin >> whole >> fractional >> letter;
14    cout << "Liczba całkowita: " << whole << endl;
15    cout << "Liczba z ułamkiem: " << fractional << endl;
16    cout << "Litera: " << letter << endl;
17    return 0;
18 }
```

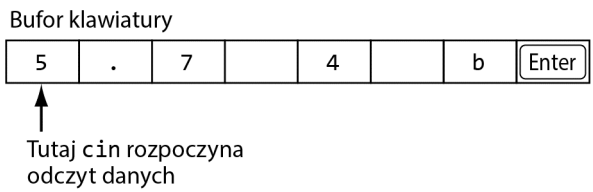
Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Wpisz liczbę całkowitą, liczbę z ułamkiem i literę: 4 5.7 b [Enter]
Liczba całkowita: 4
Liczba z ułamkiem: 5.7
Litera: b
```

Na powyższym przykładzie możemy zobaczyć, że wartości zostają przypisane do odpowiednich zmiennych. Co by się jednak stało, gdyby użytkownik odpowiedział w następujący sposób?

```
Wpisz liczbę całkowitą, liczbę z ułamkiem i literę: 5.7 4 b [Enter]
```

Wartości wpisane przez użytkownika trafiłyby najpierw do lokalizacji w pamięci nazywanej **buforem klawiatury**. Zatem wartości `5.7`, `4` i `b` przechowywane są w buforze klawiatury tak, jak zilustrowano to na rysunku 3.2.



Rysunek 3.2. Bufor klawiatury

Gdy użytkownik naciśnie *Enter*, `cin` zapisze wartość `5` w zmiennej `whole`. Nie odczyta kropki dziesiętnej, ponieważ zmienna jest typu `int`. Następnie odczyta wartość `.7` i zapisze ją w zmiennej `fractional`. Spacja zostaje pominięta, natomiast liczba `4`, jako kolejna odczytana wartość, zostaje wpisana jako znak do zmiennej `letter`. Ponieważ wyrażenie `cin` odczytuje tylko trzy wartości, wartość `b` pozostanie w buforze nieodczytana. Dlatego niezwykle ważne jest wprowadzenie wartości w odpowiedniej kolejności.

Punkt kontrolny

3.1. Jaki plik nagłówkowy należy załączyć w programach korzystających z `cin`?

3.2. Prawda czy fałsz? Po wprowadzeniu danych za pomocą `cin` użytkownik musi nacisnąć *Enter*.

3.3. Przyjmij, że zmienna `wartosc` jest typu `int`. Co zostanie zapisane w zmiennej, jeżeli użytkownik wpisze `3.14` w poniższym wyrażeniu?

```
cin >> wartosc;
```

A) `3.14`
B) `3`
C) `0`
D) Nic. Pojawi się komunikat błędu

3.4. W programie znajdują się następujące definicje zmiennych:

```
long mile;
int stopy;
float cale;
```

Napisz wyrażenie `cin`, które wpisuje wartości do każdej z tych zmiennych.

3.5. Poniższy program uruchomi się, jednak użytkownik będzie miał problem ze zrozumieniem, co ma zrobić. Jak można go poprawić?

```
// Ten program mnoży dwie liczby i wyświetla wynik.
#include <iostream>
using namespace std;
int main()
{
    double first, second, product;
    cin >> first >> second;
    product = first * second;
cout << product;
    return 0;
}
```

3.6. Uzupełnij poniższy szkielet programu tak, aby prosił on o wprowadzenie wagi użytkownika w kilogramach i wyświetlał wartość przeliczoną na funty.

```
#include <iostream>
using namespace std;
int main()
{
    double pounds, kilograms;

    // Napisz kod, który prosi użytkownika o wpisanie swojej wagi i zapisuje wejście
    // w zmiennej kilograms.
    // Poniższy wiersz odpowiada za konwersję jednostek.

    pounds = kilograms * 2.2;

    // Napisz wyrażenie, które wyświetli wagę użytkownika w funtach.

    return 0;
}
```

3.2. Wyrażenia matematyczne

WYJAŚNIENIE: C++ pozwala na tworzenie złożonych wyrażeń matematycznych poprzez użycie wielu operatorów i grupowanie symboli.

W rozdziale 2. poznałeś podstawowe operatory matematyczne stosowane do budowania wyrażeń. **Wyrażenie** to instrukcja programu, która zwraca wartość. Wyrażenie zazwyczaj składa się z operatora i argumentów. Spójrz na poniższą instrukcję:

```
suma = 21 + 3;
```

Ponieważ `21 + 3` zwraca wartość, możemy nazwać je wyrażeniem. Jego wartość, czyli `24`, zostanie zapisana w zmiennej `suma`. Wyrażenia nie muszą być działaniami matematycznymi. W poniższej instrukcji liczba `3` jest wyrażeniem:

```
liczba = 3;
```

Poniżej umieszczono przykłady instrukcji programistycznych, w których do zmiennej `wynik` przypisywane są wartości wyrażeń.

```
wynik = x;
wynik = 4;
wynik = 15 / 3;
wynik = 22 * liczba;
wynik = sizeof(int);
wynik = a + b + c;
```

W każdym z tych wyrażeń po prawej stronie operatora `=` pojawia się liczba, zmienna lub wyrażenie matematyczne. Zwracana przez nie wartość zapisywana jest w zmiennej. Wszystkie powyższe przykłady pokazują wpisywanie do zmiennej wartości wyrażenia.

Na listingu 3.4 przedstawiono możliwości wykorzystania wyrażeń matematycznych z obiektem `cout`.

Listing 3.4

```
// Program prosi użytkownika o wpisanie
// licznika i mianownika ułamka,
// a następnie wyświetla wartość dziesiętną.

#include <iostream>
using namespace std;

int main()
{
   double numerator, denominator;

   cout << "Ten program pokazuje wartość dziesiętną ";
   cout << "ułamka.\n";
   cout << "Wpisz licznik: ";
   cin >> numerator;
   cout << "Wpisz mianownik: ";
   cin >> denominator;
   cout << "Wartość dziesiętna wynosi ";
   cout << (numerator / denominator) << "." << endl;
```

```
20    return 0;
21 }
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Ten program pokazuje wartość dziesiętną ułamka.
Wpisz licznik: 3 [Enter]
Wpisz mianownik: 16 [Enter]
Wartość dziesiętna wynosi 0.1875.
```

Obiekt `cout` wyświetli każdą poprawną wartość wyrażenia w C++. Program z listingu 3.4 wyświetla wartość wyrażenia `numerator / denominator`.

UWAGA. W przykładzie z listingu 3.4 użytkownik wpisuje `3` i `16`. Ponieważ wartości te przypisywane są do zmiennych `double`, będą przechowywane jako `3.0` i `16.0`.

UWAGA. Jeśli wysyła się wyrażenie zawierające operator do obiektu `cout`, dobrym pomysłem jest umieszczenie go w nawiasie. W przeciwnym razie kilka bardziej zaawansowanych operatorów może zwrócić nieoczekiwane wyniki.

Kolejność działań

Możemy tworzyć wyrażenia matematyczne zawierające kilka operatorów. W poniższym przykładzie w zmiennej `odpowiedz` zapisywana jest suma `17`, `x`, `12` i `y`.

```
odpowiedz = 17 + x + 12 + y;
```

Niektóre wyrażenia nie są jednak tak oczywiste. Rozważ poniższe wyrażenie:

```
wynik = 12 + 6 / 3;
```

Jaka wartość zostanie zapisana w zmiennej `wynik`? Liczba `6` jest argumentem i operatora dodawania, i dzielenia. Wynikiem może być zarówno `6`, jak i `14`, w zależności od kolejności umiejscowienia operatorów. Tak naprawdę wynikiem tego działania będzie `14`, ponieważ dzielenie ma *pierwszeństwo* przed dodawaniem.

Wyrażenia matematyczne są sprawdzane od lewej do prawej. Jeżeli argument jest współdzielony przez dwa operatory, pierwszy operację wykona operator o wyższym pierwszeństwie. Mnożenie i dzielenie mają pierwszeństwo przed dodawaniem i odejmowaniem, zatem powyższy przykład wykona się w następującej kolejności:

1. 6 zostaje podzielone przez 3, co zwraca wynik 2.
2. 12 zostaje dodane do 2, co zwraca wynik 14.

Możemy ten przykład zilustrować za pomocą diagramu:

```
wynik = 12 + 6 / 3
            \  /
wynik = 12 +  2
wynik = 14
```

Kolejność działań została przedstawiona w tabeli 3.1. Operatory na szczycie tabeli mają pierwszeństwo przed operatorami niżej.

Tabela 3.1. Priorytet operatorów arytmetycznych (od najwyższego do najniższego)

(negacja jednoargumentowa) -
* / %
+ -

Mnożenie, dzielenie oraz modulo mają taki sam priorytet, podobnie jak operatory dodawania i odejmowania. W tabeli 3.2 przedstawiono kilka wyrażeń z wynikami.

Tabela 3.2. Proste wyrażenia i ich wartości

Wyrażenie	Wartość
`5 + 2 * 4`	13
`10 / 2 - 3`	2
`8 + 12 * 2 - 4`	28
`4 + 17 % 2 - 1`	4
`6 - 3 * 2 + 7 - 1`	6

Łączność

Łączność operatora występuje albo od lewej do prawej, albo od prawej do lewej. Jeżeli dwa operatory w wyrażeniu mają taki sam priorytet, będą działały na podstawie swojej łączności. Łączność operatorów arytmetycznych została przedstawiona w tabeli 3.3. W celu lepszego zrozumienia spójrz na poniższy przykład:

```
5 - 3 + 2
```

Operatory - i + mają takie samo pierwszeństwo oraz łączność od lewej do prawej. Powyższe wyrażenie możemy zapisać jako:

```
((5 - 3) + 2)
```

Kolejny przykład:

```
12 / 6 * 4
```

Ponieważ operatory / i * mają taki sam priorytet i łączność od lewej do prawej, powyższe wyrażenie możemy zapisać jako:

```
((12 / 6) * 4)
```

Tabela 3.3. Łączność operatorów arytmetycznych

Operator	Łączność
(negacja jednoargumentowa) -	Od prawej do lewej
* / %	Od lewej do prawej
+ -	Od lewej do prawej

Grupowanie za pomocą nawiasów

Części wyrażeń matematycznych mogą zostać pogrupowane za pomocą nawiasów, aby wymusić wykonanie jednych operacji przed drugimi. W poniższym wyrażeniu suma `a + b` zostaje podzielona przez `4`.

```
wynik = (a + b) / 4;
```

Bez nawiasów najpierw `b` zostałoby podzielone przez `4`, a wynik dodany do `a`. Więcej wyrażeń i wartości znajduje się w tabeli 3.4.

Tabela 3.4. Więcej prostych wyrażeń i ich wartości

Wyrażenie	Wartość
`(5 + 2) * 4`	28
`10 / (5 - 3)`	5
`8 + 12 * (6 – 2)`	56
`(4 + 17) % 2 - 1`	0
`(6 - 3) * (2 + 7) / 3`	9

Zamiana wyrażeń algebraicznych na instrukcje programu

W algebrze nie zawsze używamy operatora mnożenia. Jednak C++ wymaga operatorów dla każdej operacji matematycznej. Tabela 3.5 zawiera wyrażenia algebraiczne wykonujące mnożenie oraz ich odpowiedniki w wyrażeniach C++.

Tabela 3.5. Algebraiczne wyrażenia mnożenia oraz ich zapis w C++

Wyrażenie algebraiczne	Działanie	Wyrażenie w C++
6B	*6* razy *B*	`6 * B`
(3)(12)	*3* razy *12*	`3 * 12`
4xy	*4* razy *x* razy *y*	`4 * x * y`

Zamieniając wyrażenia algebraiczne na wyrażenia C++, będziesz musiał czasami wstawiać nawiasy, których nie ma w oryginalnym wyrażeniu. Przeanalizujmy poniższy przykład:

$$x = \frac{a+b}{c}$$

Zamieniając go na instrukcję C++, będziesz musiał umieścić wyrażenie $a + b$ w nawiasie:

```
x = (a + b) / c;
```

Więcej wyrażeń algebraicznych i ich równoważników w języku C++ znajdziesz w tabeli 3.6.

Tabela 3.6. Wyrażenia algebraiczne i ich odpowiedniki w C++

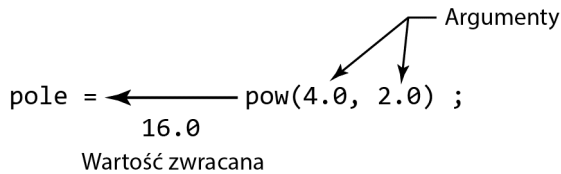
Wyrażenie algebraiczne	Wyrażenie w C++
$y = 3\frac{x}{2}$	`y = x / 2 * 3`
$z = 3bc + 4$	`z = 3 * b * c + 4`
$a = \frac{3x+2}{4a-1}$	`a = (3 * x + 2) / (4 * a - 1)`

Proszę, żadnych wykładników!

W przeciwieństwie do wielu języków programowania C++ nie ma operatora wykładnika. Potęgowanie wymaga zastosowania **funkcji bibliotecznej**. Biblioteka C++ to nie miejsce, w którym wypożyczasz książki, ale zbiór wyspecjalizowanych funkcji. Pomyśl o funkcji bibliotecznej jako o procedurze wykonującej określone operacje. Jedna z takich funkcji nazywa się `pow()` i służy do potęgowania liczb. Oto przykład jej zastosowania:

```
pole = pow(4.0, 2.0);
```

Powyższa instrukcja zawiera *wywołanie* funkcji `pow()`. Liczby w nawiasie to argumenty, czyli dane wysyłane do funkcji. Funkcja `pow()` zawsze podnosi pierwszy argument do potęgi wyrażonej przez drugi argument. W tym przypadku `4` jest podniesione do potęgi `2`. Funkcja ta *zwraca* wynik, który można wykorzystać w instrukcji, w której została użyta. W tym przypadku zwróci ona `16`, a wynik zostanie wpisany do zmiennej `pole`. Działanie omówionej instrukcji zilustrowano na rysunku 3.3.



Rysunek 3.3. Funkcja pow()

Wyrażenie `pole = pow(4.0, 2.0)` jest równoważne z poniższym wyrażeniem algebraicznym:

$$pole = 4^2$$

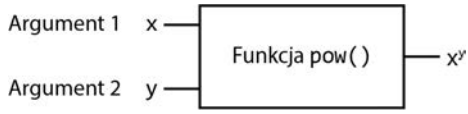
Oto kolejny przykład wyrażenia wykorzystującego funkcję `pow()`. Przypisuje wartość działania 3 razy 6^3 do zmiennej `x`:

```
x = 3 * pow(6.0, 3.0);
```

Natomiast poniższe wyrażenie wyświetla wartość `5` podniesioną do potęgi `4`:

```
cout << pow(5.0, 4.0);
```

Pomocne może być rozpatrywanie funkcji `pow()` jako „czarnej skrzynki", do której wkłada się dwie liczby, a następnie wyjmuje się z niej wynik, którym jest wartość pierwszej liczby podniesionej do potęgi określonej przez drugą liczbę. Zostało to przedstawione na rysunku 3.4.



Rysunek 3.4. Funkcja pow() jako „czarna skrzynka"

Korzystając z funkcji `pow()`, powinieneś przestrzegać kilku wytycznych. Po pierwsze, należy dołączyć plik nagłówkowy `<cmath>`. Po drugie, argumenty przekazywane do funkcji powinny być typu `double`. Po trzecie, zmienna przechowująca wynik zwrócony przez funkcję również powinna być typu `double`. Przykładowo, w poniższym wyrażeniu zmienna `pole` powinna być typu `double`:

```
pole = pow(4.0, 2.0);
```

Program z listingu 3.5 rozwiązuje prosty problem algebraiczny. Prosi użytkownika o wprowadzenie promienia koła, a następnie oblicza jego pole. Wzór jest taki:

$$P = \pi r^2$$

co zapisujemy w programie jako:

```
area = PI * pow(radius, 2.0);
```

Listing 3.5

```
 1 // Program oblicza pole koła.
 2 // Wzór na pole koła to pi razy
 3 // promień do kwadratu. Pi wynosi 3,14159.
 4 #include <iostream>
 5 #include <cmath>    // Potrzebne do zastosowania funkcji pow()
 6 using namespace std;
 7
 8 int main()
 9 {
10    const double PI = 3.14159;
11    double area, radius;
12
13    cout << "Ten program oblicza pole koła.\n";
14    cout << "Ile wynosi promień koła? ";
15    cin >> radius;
16    area = PI * pow(radius, 2.0);
17    cout << "Pole wynosi " << area << "." << endl;
18    return 0;
19 }
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Ten program oblicza pole koła.
Ile wynosi promień koła? 10 [Enter]
Pole wynosi 314.159.
```

UWAGA. Program z listingu 3.5 wykorzystuje funkcję `pow()` tylko w ramach demonstracji. Nie ma jednak powodu stosowania jej w tak prostych obliczeniach. Wyrażenie algebraiczne obliczające pole możemy zapisać jako:

```
area = PI * radius * radius;
```

Funkcja `pow()` jest jednak bardzo przydatna w wyrażeniach stosujących większe wykładniki.

W centrum uwagi
Obliczanie średniej

Obliczanie wartości średniej grupy liczb to bardzo proste działanie: dzielimy sumę wartości przez ich liczbę. Chociaż to proste, łatwo pomylić się podczas pisania programu obliczającego średnią. Załóżmy na przykład, że zmienne `a`, `b` i `c` są typu `double`. Każda z nich przechowuje jakąś wartość. Chcielibyśmy obliczyć średnią z tych wartości. Z powodu niedbalstwa moglibyśmy zapisać wyrażenie w następujący sposób:

```
average = a + b + c / 3.0;
```

Zauważyłeś już błąd? Jako pierwsze wykona się dzielenie. Wartość `c` zostanie podzielona przez `3.0`. Następnie wynik zostanie dodany do `a` i `b`. Nie jest to prawidłowy sposób obliczania średniej. W celu poprawnego obliczenia średniej należy umieścić sumę w nawiasie:

```
average = (a + b + c) / 3.0;
```

Prześledźmy proces pisania programu obliczającego średnią. Załóżmy, że zdałeś trzy testy na lekcjach informatyki i chciałbyś napisać program, który obliczy średnią ocen z tych testów. Oto algorytm w pseudokodzie:

```
Pobierz pierwszą ocenę.
Pobierz drugą ocenę.
Pobierz trzecią ocenę.
Oblicz średnią, sumując oceny i dzieląc je przez trzy.
Wyświetl średnią.
```

W pierwszych trzech krokach każemy użytkownikowi wpisać trzy oceny z testów. Załóżmy, że zmienne do ich przechowywania nazwiemy `test1`, `test2` i `test3`. Następnie, w czwartym kroku, obliczamy średnią ocen za pomocą poniższego wyrażenia. Wynik zapisujemy w zmiennej typu `double` o nazwie `average`.

```
average = (a + b + c) / 3.0;
```

Ostatni krok to wyświetlenie średniej. Całość programu znajduje się na listingu 3.6.

Listing 3.6

```
// Program oblicza średnią
// trzech ocen z testów.
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
   double test1, test2, test3;     // Przechowywanie ocen
    double average;                // Przechowywanie średniej

    // Pobranie trzech ocen
    cout << "Wpisz pierwszą ocenę: ";
    cin >> test1;
    cout << "Wpisz drugą ocenę: ";
    cin >> test2;
    cout << "Wpisz trzecią ocenę: ";
```

```
18     cin >> test3;
19
20     // Obliczenie średniej ocen
21     average = (test1 + test2 + test3) / 3.0;
22
23     // Wyświetlenie średniej
24     cout << "Średnia ocen: " << average << endl;
25     return 0;
26 }
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Wpisz pierwszą ocenę: 4 [Enter]
Wpisz drugą ocenę: 5 [Enter]
Wpisz trzecią ocenę: 3 [Enter]
Średnia ocen: 4
```

Punkt kontrolny

3.7. Uzupełnij poniższą tabelę, określając wartości każdego z wyrażeń.

Wyrażenie	Wartość
`6 + 3 * 5`	
`12 / 2 - 4`	
`9 + 14 * 2 - 6`	
`5 + 19 % 3 - 1`	
`(6 + 2) * 3`	
`14 / (11 - 4)`	
`9 + 12 * (8 - 3)`	
`(6 + 17) % 2 - 1`	
`(9 - 3) * (6 + 9) / 3`	

3.8. Napisz wyrażenia w C++ odpowiadające poniższym wyrażeniom algebraicznym:

$y = 6x$

$a = 2b + 4c$

$y = x^2$

$g = \frac{x+2}{z^2}$

$y = \frac{x^2}{z^2}$

3.9. Przeanalizuj poniższy program i uzupełnij tabelę.

```
#include <iostream>
#include <cmath>
using namespace std;
```

```
int main()
{
    double value1, value2, value3;
    cout << "Wpisz liczbę: ";
    cin >> value1;
    value2 = 2 * pow(value1, 2.0);
    value3 = 3 + value2 / 2 - 1;
    cout << value3 << endl;
return 0;
}
```

Jeśli użytkownik wpisze…	program wyświetli (wartość zmiennej value3)…
2	
5	
4.3	
6	

3.10. Uzupełnij szkielet programu, aby wyświetlał objętość zbiornika paliwa w kształcie walca. Oto wzór na objętość:

$V = \pi r^2 h$

gdzie:

π wynosi 3,14159,
r to promień zbiornika,
h to wysokość zbiornika.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    double volume, radius, height;
    cout << "Ten program oblicza objętość\n";
    cout << "zbiornika paliwa w kształcie walca.\n";
    cout << "Jaka jest wysokość zbiornika? ";
    cin >> height;
    cout << "Ile wynosi promień zbiornika? ";
    cin >> radius;
    // Dokończ program.
}
```

3.3. Gdy pomylisz jabłka z pomarańczami: konwersja typów

WYJAŚNIENIE: **Jeżeli argumenty operatora są różnych typów, C++ automatycznie zmieni je na ten sam typ. Może to wpłynąć na wynik wyrażeń matematycznych.**

Jakiego typu będzie wynik, jeżeli wartość typu `int` pomnożymy przez wartość typu `float`? Co się stanie, jeżeli wartość typu `double` pomnożymy przez `unsigned int`? Czy jest jakikolwiek sposób, aby przewidzieć, co się stanie w powyższych sytuacjach? Odpowiedź brzmi: tak. C++ posiada zestaw reguł, którymi kieruje się przy wykonywaniu operacji matematycznych na zmiennych o różnych typach.

Wszystkie typy danych mają swoje rangi, tak jak żołnierze w armii. Typ danych, który może pomieścić większą ilość danych, stoi wyżej w hierarchii. Typ `float` ma wyższą rangę niż `int`. Tabela 3.7 zawiera typy danych uszeregowane od najwyższej rangi do najniższej.

Tabela 3.7. Typy danych według rang

`long double`
`double`
`float`
`unsigned long long int`
`long long int`
`unsigned long int`
`long int`
`unsigned int`
`int`

Jedynym wyjątkiem od rankingu z tabeli 3.7 jest sytuacja, gdy `int` i `long` są tego samego rozmiaru. W tym przypadku `unsigned int` ma wyższą rangę niż `long`, ponieważ może przechowywać wyższą wartość.

Gdy C++ korzysta z operatorów, dąży do sprowadzenia argumentów do tego samego typu. Ten typ automatycznej konwersji znany jest jako **koercja typów**. Konwersję wartości na typ stojący wyżej w rankingu nazywamy **promocją**. Konwersję na typ danych będący niżej w rankingu nazywamy **degradacją**. Przyjrzyjmy się teraz zasadom ewaluacji wyrażeń matematycznych.

Zasada 1. Typy `char`, `short`, `unsigned short` są automatycznie promowane do `int`.

Zwróć uwagę, że w tabeli 3.7 nie ma typów `char`, `short`, `unsigned short`. Jest tak, ponieważ promowane są do `int` automatycznie za każdym razem, gdy używa się ich w wyrażeniach matematycznych. Jedynym wyjątkiem jest sytuacja, gdy `unsigned short` przechowuje wartość większą niż `int`. Może się to zdarzyć w systemach, w których typ `short` jest tego samego rozmiaru co `int`. W takim przypadku `unsigned short` jest promowany do `unsigned int`.

Zasada 2. Gdy operator działa na argumentach dwóch różnych typów danych, typ o niższej randze jest promowany do typu o wyższej randze.

Załóżmy, że w poniższym wyrażeniu zmienna `lata` jest typu `int`, a zmienna `stopaOprocentowania` jest typu `float`:

```
lata * stopaOprocentowania
```

Przed wykonaniem mnożenia zmienna `lata` zostanie wypromowana do `float`.

Zasada 3. Jeżeli wartość wyrażenia przypisywana jest do zmiennej, jej typ zostanie zmieniony na typ tej zmiennej.

Przyjmijmy, że w poniższym wyrażeniu zmienna `pole` jest typu `long`, a zmienne `dlugosc` i `szerokosc` są typu `int`.

```
pole = dlugosc * szerokosc;
```

Ponieważ `dlugosc` i `szerokosc` są typu `int`, nie zostaną zamienione na inny typ danych. Natomiast wynik mnożenia zostanie zamieniony na `long`, aby można było zapisać go w zmiennej.

Uważaj na sytuacje, w których wynik będący liczbą z ułamkiem miałby zostać zapisany w zmiennej typu `int`. Oto przykład:

```
int x, y = 4;
float z = 2.7;
x = y * z;
```

W wyrażeniu `y * z`, zmienna `y` zostanie zamieniona na typ `float`. Wynikiem mnożenia będzie `10.8`. Jednak ponieważ zmienna `x` jest typu `int`, zostanie w niej zapisana wartość `10`.

Dzielenie liczb całkowitych

Wynikiem dzielenia liczby całkowitej przez liczbę całkowitą będzie zawsze liczba całkowita. Reszta zostanie odrzucona. Na przykład w poniższym kodzie do zmiennej `czesci` zostanie wpisana wartość `2.0`.

```
double czesci;
czesci = 15 / 6;
```

Chociaż `15 / 6` to `2.5`, ułamek zostanie odrzucony, ponieważ dzielimy liczbę całkowitą przez liczbę całkowitą. Deklaracja zmiennej jako `double` nie ma w tym przypadku żadnego znaczenia, ponieważ ułamek jest odrzucany jeszcze *przed* przypisaniem do niej wartości. Żeby działanie zwróciło liczbę zmiennoprzecinkową, co najmniej jeden argument musi być liczbą zmiennoprzecinkową. Poprzedni przykład moglibyśmy napisać w taki sposób:

```
double czesci;
czesci = 15.0 / 6;
```

W tym przykładzie wartość literału `15.0` jest traktowana jako liczba zmiennoprzecinkowa. Dlatego do zmiennej zostanie przypisana liczba `2.5`.

3.4. Przepełnienie i zaniżenie

WYJAŚNIENIE: Jeżeli wartość przypisana do zmiennej jest poniżej lub powyżej jej pojemności, mamy do czynienia z przepełnieniem lub zaniżeniem.

Jeżeli do zmiennej próbujemy przypisać wartość wykraczającą poza jej zakres, możemy napotkać problemy. Spójrz na to przykładowe wyrażenie, w którym zmienne `a`, `b` i `c` są typu `short int`:

```
a = b * c;
```

Jeżeli wartości `b` i `c` są dostatecznie duże, wynik mnożenia może okazać się zbyt duży, aby mógł być wpisany do `a`. Aby uniknąć błędów, zmienna `a` powinna zostać zdeklarowana jako `int` albo `longint`.

Gdy do zmiennej przypiszemy liczbę przekraczającą zakres jej typu, spowodujemy **przepełnienie**. Analogicznie wpisanie liczby zbyt małej, leżącej poza zakresem zmiennej, spowoduje **zaniżenie** (wyjście listingu 3.7 przedstawia wyniki z systemu, w którym typ `short` zajmuje 2 bajty).

Listing 3.7

```
// Program pokazuje przepełnienie i zaniżenie w zmiennej typu short.
#include <iostream>
using namespace std;

int main()
{
    // testVar zostaje zainicjowana maksymalną wartością dla typu short.
short testVar = 32767;

    // Wyświetlenie testVar
    cout << testVar << endl;

    // Dodanie 1 do testVar, aby spowodować przepełnienie
    testVar = testVar + 1;
    cout << testVar << endl;

    // Odjęcie 1 od testVar, aby spowodować zaniżenie
    testVar = testVar - 1;
    cout << testVar << endl;
    return 0;
}
```

Wyjście programu

```
32767
-32768
32767
```

Zazwyczaj przepełnienie zmiennej typu całkowitego powoduje przejście do najniższej wartości danego typu. Na listingu 3.7 `testVar` przechodzi z 32 767 do –32 768 po dodaniu 1. Po odjęciu 1 od `testVar` w kolejnym kroku nastąpiło zaniżenie i jej wartość

wróciła z powrotem do 32 767. Nie zostanie zwrócony żaden komunikat, więc zachowaj ostrożność, pracując z liczbami bliskimi maksymalnych i minimalnych zakresów zmiennych. Przepełnienia i zaniżenia powodują, że wartości zwracane przez program będą niepoprawne.

Wyniki przepełnień i zaniżeń liczb zmiennoprzecinkowych zależą od konfiguracji kompilatora. W zależności od systemu program może zachować się w jeden z następujących sposobów:

- Zwróci zły wynik i będzie działał dalej.
- Wyświetli komunikat błędu i przestanie działać w momencie przepełnienia lub zaniżenia wartości zmiennej typu zmiennoprzecinkowego.
- Wyświetli komunikat błędu i przestanie działać w przypadku przepełnienia, jednak w przypadku zaniżenia wartości do zmiennej zostanie przypisane 0.
- Da Ci możliwość wyboru różnych zachowań w przypadku przepełnienia lub zaniżenia.

Zachowanie swojego systemu sprawdzisz, kompilując kod z listingu 3.8.

Listing 3.8

```
// Ten program możemy wykorzystać, aby sprawdzić zachowanie systemu
// w przypadku przepełnienia lub zaniżenia zmiennej typu zmiennoprzecinkowego.
#include <iostream>
using namespace std;

int main()
{
   float test;

   test = 2.0e38 * 1000;     // Test przepełnienia
   cout << test << endl;
   test = 2.0e-38 / 2.0e38; // Test zaniżenia
   cout << test << endl;
  return 0;
}
```

3.5. Rzutowanie typów

WYJAŚNIENIE: Rzutowanie typów pozwala na ręczną konwersję danych.

Rzutowanie typów umożliwia ręczne promowanie lub degradowanie wartości. Wyrażenie rzutujące wygląda tak:

```
static_cast<TypDanych>(Wartość)
```

Wartość to zmienna lub literał, który chcesz rzutować, a *TypDanych* to typ, na który zmieniona ma zostać wartość. Spójrz na poniższy przykład wykorzystujący rzutowanie typów:

```
double number = 3.7;
int val;
val = static_cast<int>(number);
```

Kod definiuje dwie zmienne: `number` typu `double` i `val` typu `int`. Wyrażenie rzutujące w trzecim wierszu zwraca kopię wartości zapisanej w `number`, zmienioną na `int`. Konwersja `double` na `int` polega na przycięciu części ułamkowej. Dlatego ta instrukcja przypisze wartość 3 do zmiennej `val`, zaś wartość zmiennej `number` nie zostanie zmodyfikowana.

Rzutowanie typów przydaje się w sytuacjach, w których C++ nie wykona poprawnej konwersji automatycznie. Na listingu 3.9 pokazano sytuację, w której rzutowanie typu zostało użyte, aby zapobiec dzieleniu liczb całkowitych. Wyrażenie korzystające z rzutowania typów to:

```
perMonth = static_cast<double>(books) / months;
```

Listing 3.9

```
1 // Ten program stosuje rzutowanie typów, aby zapobiec dzieleniu liczb całkowitych.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7    int books;         // Liczba książek do przeczytania
8    int months;        // Liczba miesięcy spędzonych na czytaniu
9    double perMonth;   // Średnia liczba książek na miesiąc
10
11   cout << "Ile książek planujesz przeczytać? ";
12   cin >> books;
13   cout << "Ile miesięcy zajmie Ci ich przeczytanie? ";
14   cin >> months;
15   perMonth = static_cast<double>(books) / months;
16   cout << "To jest " << perMonth << " książki na miesiąc.\n";
17   return 0;
18 }
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Ile książek planujesz przeczytać? 30 [Enter]
Ile miesięcy zajmie Ci ich przeczytanie? 7 [Enter]
To jest 4.28571 książki na miesiąc.
```

Zmienna `books` jest typu `int`, lecz jej wartość jest rzutowana na `double` przed podzieleniem. Bez rzutowania typów w wierszu 15. miałoby miejsce dzielenie liczb całkowitych, co dałoby zły wynik.

Listing 3.10 przedstawia inne zastosowanie rzutowania typów.

Listing 3.10

```
1 // Ten program wykorzystuje wyrażenie rzutowania typu, aby wyświetlić
2 // znak na podstawie liczby.
3 #include <iostream>
4 using namespace std;
5
```

```
 6 int main()
 7 {
 8    int number = 65;
 9
10    // Wyświetlenie wartości zmiennej number
11    cout << number << endl;
12
13    // Wyświetlenie wartości zmiennej number zamienionej
14    // na typ char
15    cout << static_cast<char>(number) << endl;
16    return 0;
17 }
```

Wyjście programu

```
65
A
```

OSTRZEŻENIE! Jeżeli w kodzie z listingu 3.9 zastosowalibyśmy następującą instrukcję, miałoby miejsce również dzielenie liczb całkowitych:

```
perMonth = static_cast<double>(books / months);
```

Wynikiem dzielenia zmiennej `books` przez `months` jest `4`. Po konwersji `4` na `double` będzie to `4.0`. Aby zapobiec dzieleniu liczb całkowitych, jedna z wartości powinna zostać zamieniona na zmiennoprzecinkową jeszcze przed dzieleniem. Dzięki temu drugi argument zostanie automatycznie zamieniony przez C++ na `double`.

Przyjrzyjmy się temu programowi. W wierszu 8. zainicjowano zmienną `number` o wartości `65`. W wierszu 11. wartość zmiennej zostaje wysłana do obiektu `cout`, powodując wyświetlenie `65` na ekranie. W wierszu 15. zastosowano rzutowanie typów, aby zamienić wartość zmiennej `number` na typ `char`. Przypomnij sobie wiadomości z rozdziału 2.: znaki przechowywane są w pamięci jako numery kodu ASCII. Ponieważ liczba 65 oznacza w kodzie ASCII literę A, na ekranie zostanie wyświetlone `A`.

UWAGA. C++ dostarcza kilka wyrażeń rzutowania typów. `static_cast` jest najpowszechniej używanym wyrażeniem rzutowania typów, więc w tej książce użyjemy go w pierwszej kolejności.

Punkt kontrolny

3.11. Załóżmy, że mamy następujące definicje zmiennych:

```
int a = 5, b = 12;
double x = 3.4, z = 9.1;
```

Jakie będą wartości poniższych wyrażeń?

A) `b / a`
B) `x * a`
C) `static_cast<double>(b / a)`
D) `static_cast<double>(b) / a`
E) `b / static_cast<double>(a)`

F) `static_cast<double>(b) / static_cast<double>(a)`
G) `b / static_cast<int>(x)`
H) `static_cast<int>(x) * static_cast<int>(z)`
I) `static_cast<int>(x * z)`
J) `static_cast<double>(static_cast<int>(x) * static_cast<int>(z))`

3.12. Uzupełnij poniższy szkielet programu tak, aby program poprosił użytkownika o wprowadzenie znaku. Zapisz znak w zmiennej `letter`. Wykorzystaj rzutowanie typów, aby wyświetlić na ekranie kod ASCII wprowadzonego znaku.

```
#include <iostream>
using namespace std;
int main()
{
   char letter;
   // Dokończ program zgodnie ze specyfikacją.
   return 0;
}
```

3.13. Co wyświetli poniższy program?

```
#include <iostream>
using namespace std;
int main()
{
    int integer1, integer2;
    double result;
    integer1 = 19;
    integer2 = 2;
    result = integer1 / integer2;
    cout << result << endl;
    result = static_cast<double>(integer1) / integer2;
    cout << result << endl;
    result = static_cast<double>(integer1 / integer2);
    cout << result << endl;
    return 0;
}
```

3.6. Wielokrotne przypisania i przypisania łączone

WYJAŚNIENIE: Przypisanie wielokrotne to przypisanie jednej wartości do wielu zmiennych za pomocą jednej instrukcji.

C++ pozwala na przypisanie wartości do wielu zmiennych jednocześnie. Jeżeli program ma kilka zmiennych, na przykład `a`, `b`, `c` i `d`, a każda z tych zmiennych powinna mieć przypisaną wartość 12, możemy utworzyć instrukcję taką jak poniżej:

```
a = b = c = d = 12;
```

Wartość 12 zostanie przypisana do każdej ze zmiennych w powyższym poleceniu[1].

[1] Operator przypisania działa od prawej do lewej. Wartość 12 zostaje najpierw przypisana do zmiennej `d`, potem do `c`, `b` i `a`.

Łączone operatory przypisania

Bardzo często programy mają przypisania w następującej formie:

```
liczba = liczba + 1;
```

Wyrażenie po prawej stronie przypisania zwraca wartość zmiennej `liczba` powiększoną o 1. Wynik zostaje zapisany w zmiennej `liczba`, co zastępuje poprzednią wartość. Rezultatem wyrażenia będzie po prostu powiększenie wartości zmiennej o 1. W podobny sposób poniższe wyrażenie zmniejsza wartość zmiennej o 5:

```
liczba = liczba - 5;
```

Jeśli nigdy wcześniej nie widziałeś instrukcji tego typu, możesz być początkowo nieco zdezorientowany, ponieważ ta sama nazwa zmiennej pojawia się po obu stronach operatora przypisania. Tabela 3.8 pokazuje inne przykłady instrukcji pisanych w ten sposób.

Tabela 3.8. (Przyjmij, że x = 6)

Wyrażenie	Co robi	Wartość x po wykonaniu wyrażenia
`x = x + 4`	Dodaje 4 do x.	10
`x = x – 3`	Odejmuje 3 od x.	3
`x = x * 10`	Mnoży x razy 10.	60
`x = x / 2`	Dzieli x przez 2.	3
`x = x % 4`	Przypisuje do x resztę z dzielenia `x / 4`.	2

Tego typu działania są bardzo częste w programowaniu. Dlatego, w celu ułatwienia pracy, C++ oferuje specjalny zestaw operatorów przeznaczonych do wykonywania tych poleceń. W tabeli 3.9 znajdują się **łączone operatory przypisania**, znane również jako **operatory złożone** lub **arytmetyczne operatory przypisania**.

Tabela 3.9. Łączone operatory przypisania

Operator	Przykładowe zastosowanie	Równoważne z
`+=`	`x += 5`	`x = x + 5`
`-=`	`y -= 2`	`y = y - 2`
`*=`	`z *= 10`	`z = z * 10`
`/=`	`a /= b`	`a = a / b`
`%=`	`c %= 3`	`c = c % 3`

Dzięki operatorom złożonym programista nie musi wpisywać nazw zmiennych dwa razy. Wskazują one również jasno, co się dzieje w wyrażeniu. Zastosowanie operatorów złożonych przedstawiono na listingu 3.11.

Listing 3.11

```
// Ten program obserwuje wyposażenie trzech sklepów z gadżetami,
// które otwarto w tym samym czasie. Każdy z nich zaczął
// z taką samą liczbą gadżetów w wyposażeniu. Poprzez odejmowanie
// sprzedanych przez każdy sklep gadżetów
// możemy obliczyć aktualny stan wyposażenia.
#include <iostream>
using namespace std;

int main()
{
   int begInv,     // Wyposażenie początkowe dla wszystkich sklepów
       sold,       // Liczba sprzedanych gadżetów
       store1,     // Wyposażenie pierwszego sklepu
       store2,     // Wyposażenie drugiego sklepu
       store3;     // Wyposażenie trzeciego sklepu

   // Pobranie początkowego wyposażenia wszystkich sklepów
   cout << "Tydzień temu otwarto trzy sklepy z gadżetami\n";
   cout << "w tym samym czasie i z tym samym początkowym\n";
   cout << "wyposażeniem. Jakie było wyposażenie początkowe? ";
   cin >> begInv;

   // Ustawienie wyposażenia każdego ze sklepów
   store1 = store2 = store3 = begInv;

   // Pobranie liczby gadżetów sprzedanych w pierwszym sklepie
   cout << "Ile gadżetów sprzedano w pierwszym sklepie? ";
   cin >> sold;
   store1 -= sold; // Zaktualizowanie wyposażenia pierwszego sklepu

   // Pobranie liczby gadżetów sprzedanych w drugim sklepie
   cout << "Ile gadżetów sprzedano w drugim sklepie? ";
   cin >> sold;
   store2 -= sold; // Zaktualizowanie wyposażenia drugiego sklepu

   // Pobranie liczby gadżetów sprzedanych w trzecim sklepie
   cout << "Ile gadżetów sprzedano w trzecim sklepie? ";
   cin >> sold;
   store3 -= sold; // Zaktualizowanie wyposażenia trzeciego sklepu

   // Wyświetlenie bieżącego stanu wyposażenia każdego ze sklepów
   cout << "\nBieżący stan wyposażenia każdego ze sklepów:\n";
   cout << "Sklep 1: " << store1 << endl;
   cout << "Sklep 2: " << store2 << endl;
   cout << "Sklep 3: " << store3 << endl;
   return 0;
}
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Tydzień temu otwarto trzy sklepy z gadżetami
w tym samym czasie i z tym samym początkowym
wyposażeniem. Jakie było wyposażenie początkowe? 100 [Enter]
Ile gadżetów sprzedano w pierwszym sklepie? 25 [Enter]
Ile gadżetów sprzedano w drugim sklepie? 15 [Enter]
Ile gadżetów sprzedano w trzecim sklepie? 45 [Enter]
```

```
Bieżący stan wyposażenia każdego ze sklepów:
Sklep 1: 75
Sklep 2: 85
Sklep 3: 55
```

Bardziej złożone wyrażenia możemy uprościć za pomocą łączonych operatorów przypisania. Spójrz na poniższy przykład:

```
wynik *= a + 5;
```

W tym wyrażeniu zmienna `wynik` zostaje pomnożona przez wartość sumy `a + 5`. Tworząc wyrażenia tego typu, musisz jednak pamiętać, że priorytet operatorów łączonych jest niższy niż zwykłych operatorów matematycznych. Powyższe wyrażenie jest równoważne z:

```
wynik = wynik * (a + 5);
```

które różni się od:

```
wynik = wynik * a + 5;
```

W tabeli 3.10 znajdziesz kilka podobnych przykładów i ich odpowiedniki zapisane z użyciem zwykłej instrukcji przypisania.

Tabela 3.10. Przykłady zastosowania łączonych operatorów przypisania

Przykładowe użycie	Równoważne z
`x += b + 5;`	`x = x + (b + 5);`
`y -= a * 2;`	`y = y - (a * 2);`
`z *= 10 - c;`	`z = z * (10 - c);`
`a /= b + c`	`a = a / (b + c)`
`c %= d - 3`	`c = c % (d – 3)`

Punkt kontrolny

3.14. Napisz wyrażenie wielokrotnego przypisania, które przypisuje wartość `0` do zmiennych `total`, `subtotal`, `tax` i `shipping`.

3.15. Napisz wyrażenia wykonujące poniższe działania, używając operatorów łączonych:
A) dodanie 6 do `x`,
B) odjęcie 4 od zmiennej `amount`,
C) pomnożenie `y` przez 4,
D) podzielenie zmiennej `total` przez 27,
E) zapisanie reszty z dzielenia `x` przez 7 w zmiennej `x`,
F) dodanie `y * 5` do `x`,
G) odjęcie zmiennej `discount` pomnożonej razy 4 od zmiennej `total`,
H) pomnożenie wartości zmiennej `increase` przez `salesRep` razy 5,
I) podzielenie zmiennej `profit` przez `shares` minus 1000.

3.16. Co wyświetli poniższy program?

```
#include <iostream>
using namespace std;
int main()
{
    int unus, duo, tres;
    unus = duo = tres = 5;
    unus += 4;
    duo *= 2;
    tres -= 4;
    unus /= 3;
    duo += tres;
    cout << unus << endl;
    cout << duo << endl;
cout << tres << endl;
return 0;
}
```

3.7. Formatowanie wyjścia

WYJAŚNIENIE: Obiekt `cout` umożliwia formatowanie wyświetlanych danych. Wpływa to na sposób, w jaki dane pojawiają się na ekranie.

Te same dane mogą być wydrukowane lub wyświetlone na kilka różnych sposobów. Wszystkie poniższe liczby mają tę samą wartość, chociaż zapisano je inaczej:

```
720
720.0
720.00000000
7.2e+2
+720.0
```

Sposób, w jaki wyświetlana jest wartość, nazywamy **formatowaniem**. Obiekt `cout` ma standardowy sposób formatowania każdego typu danych. Czasami jednak potrzebujemy nieco większej kontroli nad sposobem prezentacji danych. Przeanalizuj listing 3.12, który wyświetla trzy wiersze składające się z liczb rozdzielonych spacjami.

Listing 3.12

```
1 // Program wyświetla trzy wiersze liczb.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7    int num1 = 2897, num2 = 5,    num3 = 837,
8        num4 = 34,   num5 = 7,    num6 = 1623,
9        num7 = 390,  num8 = 3456, num9 = 12;
10
11   // Wyświetlenie pierwszego wiersza
12   cout << num1 << "  " << num2 << "  " << num3 << endl;
13
14   // Wyświetlenie drugiego wiersza
```

```
15    cout << num4 << "  " << num5 << "  " << num6 << endl;
16
17    // Wyświetlenie trzeciego wiersza
18    cout << num7 << "  " << num8 << "  " << num9 << endl;
19    return 0;
20 }
```

Wyjście programu

```
2897  5  837
34  7  1623
390  3456  12
```

Niestety liczby nie układają się w kolumnach. Dzieje się tak, ponieważ niektóre liczby, `5` czy `7`, zajmują jedną pozycję na ekranie, a inne dwie lub więcej. Obiekt `cout` oddziela liczby tylko jedną spacją.

Rozwiązaniem może być ustawienie minimalnej liczby spacji przy każdej liczbie. Wyświetlając informacje za pomocą `cout`, możemy skorzystać z manipulatora strumienia o nazwie `setw()`, aby utworzyć pola wyświetlania o zadanej szerokości. Oto przykład takiego polecenia:

```
wartosc = 23;
cout << setw(5) << wartosc;
```

Liczba w nawiasach określa *szerokość pola* przeznaczonego do wyświetlenia następnej wartości. Szerokość pola to minimalna liczba znaków lub spacji, która zostanie użyta do wyświetlenia wartości. W powyższym przykładzie liczba `23` będzie wyświetlona w polu o szerokości pięciu znaków. Ponieważ `23` to tylko dwa znaki, pozostałe znaki (spacje) zostaną wyświetlone przed liczbą. Aby lepiej zilustrować działanie tego manipulatora strumienia, zmodyfikujmy nieco poprzedni przykład:

```
wartosc = 23;
cout <<'('<< setw(5) << wartosc <<')';
```

Wynik będzie taki:

```
(   23)
```

Zwróć uwagę, że liczba zajmuje tylko dwie ostatnie pozycje zadeklarowanego pola. Ponieważ nie wykorzystuje ona całego pola, `cout` dodaje przed nią trzy spacje „wypełnienia", czyli **wyrównuje do prawej strony**.

Program na listingu 3.13 pokazuje jak liczby z listingu 3.12 mogą być wyświetlone w równych kolumnach za pomocą `setw()`.

Listing 3.13

```
1 // Ten program wyświetla trzy wiersze liczb.
2 #include <iostream>
3 #include <iomanip>       // Wymagane dla setw()
4 using namespace std;
5
6 int main()
7 {
8    int num1 = 2897, num2 = 5,    num3 = 837,
9        num4 = 34,   num5 = 7,    num6 = 1623,
```

```
10        num7 = 390,  num8 = 3456, num9 = 12;
11
12    // Wyświetlenie pierwszego wiersza
13    cout << setw(6) << num1 << setw(6)
14         << num2 << setw(6) << num3 << endl;
15
16    // Wyświetlenie drugiego wiersza
17    cout << setw(6) << num4 << setw(6)
18          << num5 << setw(6) << num6 << endl;
19
20    // Wyświetlenie trzeciego wiersza
21    cout << setw(6) << num7 << setw(6)
22          << num8 << setw(6) << num9 << endl;
23    return 0;
24 }
```

Wyjście programu

```
  2897     5   837
    34     7  1623
   390  3456    12
```

Dzięki wyświetlaniu każdej liczby w polu o szerokości 6 możemy uzyskać równe kolumny.

UWAGA. Program z listingu 3.13 zawiera nowy plik nagłówkowy `<iomanip>`. Należy go dołączyć do każdego programu korzystającego z `setw()`.

Zauważ, że manipulator `setw()` jest używany przed każdym wyświetleniem wartości, ponieważ ustala on szerokość pola wyłącznie dla wartości następującej po nim. Po wyświetleniu wartości `cout` wraca do swojego domyślnego sposobu wyświetlania.

Co, jeśli liczba jest zbyt duża i nie mieści się w ustalonym polu, tak jak w poniższym przykładzie?

```
value = 18397;
cout << setw(2) << value;
```

W tym przypadku `cout` wyświetli całą liczbę. Manipulator `setw()` określa tylko minimalną szerokość. Przy dłuższych wartościach `cout` zignoruje szerokość ustawioną przez `setw()`.

Szerokość pola możesz określać dla każdego typu danych. Na listingu 3.14 zademonstrowano wykorzystanie `setw()` z wartościami typu `int`, `double` i obiektem `string`.

Listing 3.14

```
1  // Ten program demonstruje użycie manipulatora setw()
2  // z wartościami różnych typów danych.
3  #include <iostream>
4  #include <iomanip>
5  #include <string>
6  using namespace std;
7
8  int main()
9  {
10    int intValue = 3928;
```

```
11    double doubleValue = 91.5;
12    string stringValue = "Jan Kowalski";
13
14    cout << "(" << setw(5) << intValue << ")" << endl;
15    cout << "(" << setw(8) << doubleValue << ")" << endl;
16    cout << "(" << setw(16) << stringValue << ")" << endl;
17    return 0;
18 }
```

Wyjście programu

```
( 3928)
(    91.5)
(    Jan Kowalski)
```

Przykład z listingu 3.14 służy do zilustrowania następujących zagadnień:

- Kropka dziesiętna jest uwzględniona w szerokości pola liczby zmiennoprzecinkowej.
- Szerokość pola obiektu `string` uwzględnia wszystkie znaki, w tym spacje.
- Domyślnie wyświetlane wartości wyrównywane są do prawej. Spacje wstawiane w celu wyrównania długości dodawane są wyłącznie z lewej strony.

Manipulator setprecision()

Liczby zmiennoprzecinkowe możemy zaokrąglić do określonej liczby **cyfr znaczących**, innymi słowy, do określonej **precyzji**. Jest to liczba miejsc po przecinku i przed nim. Za pomocą manipulatora `setprecision()` możemy wpływać na sposób wyświetlania liczb zmiennoprzecinkowych. Na listingu 3.15 przedstawiono wyświetlanie wyniku dzielenia z różną precyzją.

Listing 3.15

```
1  // Ten program pokazuje, jak użyć setprecision()
2  // do zaokrąglenia liczby zmiennoprzecinkowej.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9     double quotient, number1 = 132.364, number2 = 26.91;
10
11    quotient = number1 / number2;
12    cout << quotient << endl;
13    cout << setprecision(5) << quotient << endl;
14    cout << setprecision(4) << quotient << endl;
15    cout << setprecision(3) << quotient << endl;
16    cout << setprecision(2) << quotient << endl;
17    cout << setprecision(1) << quotient << endl;
18    return 0;
19 }
```

Wyjście programu

```
4.91877
4.9188
4.919
4.92
4.9
5
```

Pierwszy raz wartość jest wyświetlana w wierszu 12. bez manipulatora precyzji (domyślnie system operacyjny wyświetla liczby zmiennoprzecinkowe z sześcioma cyframi znaczącymi). Kolejne instrukcje `cout` wyświetlają tę samą wartość, ale zaokrągloną, kolejno, do pięciu, czterech, trzech, dwóch i jednej cyfry znaczącej.

Jeżeli liczba jest wyrażona z precyzją mniejszą niż określona w manipulatorze `setpre` ↪`cision()`, sposób jej wyświetlania nie zostanie zmieniony. W poniższym przykładzie zmienna `dolary` ma jedynie 4 cyfry precyzji. Dlatego `cout` wyświetli `24.51` w obu wierszach:

```
double dolary = 24.51;
cout <<dolary<< endl; // Wyświetlenie 24.51
cout << setprecision(5) <<dolary<< endl; // Wyświetlenie 24.51
```

W tabeli 3.11 przedstawiono, jak `setprecision()` wpływa na wyświetlanie różnych wartości.

Tabela 3.11. Manipulator strumienia setprecision()

Liczba	Manipulator	Wyświetlona wartość
28.92786	setprecision(3)	28.9
21	setprecision(5)	21
109.5	setprecision(4)	109.5
34.28596	setprecision(2)	34

W przeciwieństwie do szerokości pola ustawienie precyzji działa, dopóki nie zostanie zmienione na inne. Podobnie jak przy innych manipulatorach formatowania, również w tym przypadku należy dodać plik nagłówkowy `<iomanip>`.

Na listingu 3.16 przedstawiono połączenie manipulatorów `setw()` i `setprecision()` wykorzystane do pełnej kontroli sposobu wyświetlania liczb zmiennoprzecinkowych.

Listing 3.16

```
1 // Program pyta o wartość sprzedaży z trzech dni.
2 // Suma sprzedaży jest obliczana i wyświetlana w tabeli.
3 #include <iostream>
4 #include <iomanip>
5 using namespace std;
6
7 int main()
8 {
9    double day1, day2, day3, total;
10
```

```
11    // Pobranie wartości sprzedaży z każdego dnia
12    cout << "Sprzedaż w dniu 1: ";
13    cin >> day1;
14    cout << "Sprzedaż w dniu 2: ";
15    cin >> day2;
16    cout << "Sprzedaż w dniu 3: ";
17    cin >> day3;
18
19    // Obliczenie sumy sprzedaży
20    total = day1 + day2 + day3;
21
22    // Wyświetlenie wartości sprzedaży
23    cout << "\nWartość sprzedaży\n";
24    cout << "-------------\n";
25    cout << setprecision(5);
26    cout << "Dzień 1: " << setw(8) << day1 << endl;
27    cout << "Dzień 2: " << setw(8) << day2 << endl;
28    cout << "Dzień 3: " << setw(8) << day3 << endl;
29    cout << "Razem: " << setw(10) << total << endl;
30    return 0;
31 }
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Sprzedaż w dniu 1: 321.57 [Enter]
Sprzedaż w dniu 2: 269.62 [Enter]
Sprzedaż w dniu 3: 307.77 [Enter]

Wartość sprzedaży
-------------
Dzień 1:   321.57
Dzień 2:   269.62
Dzień 3:   307.77
Razem:     898.96
```

Manipulator fixed

Czasami manipulator `setprecision()` może zaskoczyć Cię w niepożądany sposób. Jeżeli ustawimy bardzo małą precyzję, może się zdarzyć, że liczby zostaną wyświetlone w notacji naukowej. Poniżej znajdziesz wyjście dla programu z listingu 3.16 ilustrujące taką sytuację.

Listing 3.16a

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Sprzedaż w dniu 1: 145678.99 [Enter]
Sprzedaż w dniu 2: 205614.85 [Enter]
Sprzedaż w dniu 3: 198645.22 [Enter]

Wartość sprzedaży
-------------
Dzień 1: 1.4568e+05
Dzień 2: 2.0561e+05
Dzień 3: 1.9865e+05
Razem: 5.4994e+05
```

Kolejnym manipulatorem jest `fixed`, który zmusza `cout` do wyświetlania liczb w notacji stałoprzecinkowej. Zastosowanie tego manipulatora przedstawiono na listingu 3.17.

Listing 3.17

```
 1 // Program pyta o wartość sprzedaży z trzech dni.
 2 // Suma sprzedaży jest obliczana i wyświetlana w tabeli.
 3 #include <iostream>
 4 #include <iomanip>
 5 using namespace std;
 6
 7 int main()
 8 {
 9    double day1, day2, day3, total;
10
11    // Pobranie wartości sprzedaży z każdego dnia
12    cout << "Sprzedaż w dniu 1: ";
13    cin >> day1;
14    cout << "Sprzedaż w dniu 2: ";
15    cin >> day2;
16    cout << "Sprzedaż w dniu 3: ";
17    cin >> day3;
18
19    // Obliczenie sumy sprzedaży
20    total = day1 + day2 + day3;
21
22    // Wyświetlenie wartości sprzedaży
23    cout << "\nWartość sprzedaży\n";
24    cout << "-------------\n";
25    cout << setprecision(2) << fixed;
26    cout << "Dzień 1: " << setw(8) << day1 << endl;
27    cout << "Dzień 2: " << setw(8) << day2 << endl;
28    cout << "Dzień 3: " << setw(8) << day3 << endl;
29    cout << "Razem: " << setw(10) << total << endl;
30    return 0;
31 }
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Sprzedaż w dniu 1: 1321.87 [Enter]
Sprzedaż w dniu 2: 1869.26 [Enter]
Sprzedaż w dniu 3: 1403.77 [Enter]

Wartość sprzedaży
-------------
Dzień 1:  1321.87
Dzień 2:  1869.26
Dzień 3:  1403.77
Razem:    4594.90
```

Instrukcja w wierszu 25. zawiera manipulator `fixed`:

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

Gdy zostanie on zastosowany w programie, każda kolejna liczba zmiennoprzecinkowa zostanie zapisana w notacji stałoprzecinkowej z liczbą miejsc po przecinku określoną przez `setprecision()`.

Użycie manipulatorów `fixed` i `setprecision()` razem spowoduje, że wartość określona przez manipulator `setprecision()` będzie liczbą wyświetlanych miejsc po przecinku. Spójrz na poniższy przykład:

```
double x = 123.4567;
cout << setprecision(2) << fixed << x << endl;
```

Ponieważ zastosowano manipulator `fixed`, `setprecision()` spowoduje, że liczba zostanie wyświetlona z dwoma miejscami po przecinku. W związku z tym obiekt `cout` wyświetli `123.46`.

Manipulator showpoint

Liczby zmiennoprzecinkowe nie są domyślnie wyświetlane z zerami końcowymi i jeśli nie mają części ułamkowej, zostają wyświetlone bez kropki dziesiętnej. Przeanalizujmy poniższy kod:

```
double x = 123.4, y = 456.0;
cout << setprecision(6) << x << endl;
cout << y << endl;
```

Powyższe wyrażenia `cout` wyświetlą następujące wyjście:

```
123.4
456
```

Chociaż dla obu liczb określono precyzję, żadna z nich nie została wyświetlona z zerami końcowymi. Jeżeli chcemy wyświetlać liczby z określoną liczbą miejsc po przecinku i uzupełniać brakujące pozycje zerami, musimy wykorzystać manipulator `showpoint`, jak w poniższym kodzie:

```
double x = 123.4, y = 456.0;
cout << setprecision(6) << showpoint << x << endl;
cout << y << endl;
```

Powyższe polecenia zwrócą następujące wyjście:

```
123.400
456.000
```

UWAGA. W większości kompilatorów zera końcowe wyświetlane są, jeżeli użyto `fixed` i `setprecision()`.

Manipulatory left i right

Zazwyczaj wyjście wyrównane jest do prawej. Spójrz na poniższy przykład:

```
double x = 146.789, y = 24.2, z = 1.783;
cout << setw(10) << x << endl;
cout << setw(10) << y << endl;
cout << setw(10) << z << endl;
```

Każda zmienna jest wyświetlana w polu o szerokości 10 znaków. Wyjściem `cout` będzie w tym przypadku:

```
146.789
   24.2
  1.783
```

Zauważ, że wszystkie wartości są wyrównane do prawej. Za pomocą manipulatora `left` możesz wyrównać je do lewej, jak w kodzie poniżej:

```
double x = 146.789, y = 24.2, z = 1.783;
cout << left << setw(10) << x << endl;
cout << setw(10) << y << endl;
cout << setw(10) << z << endl;
```

Wyjście wygląda tak:

```
146.789
24.2
1.783
```

W tej sytuacji wartości są wyrównane do lewej strony zdefiniowanych pól wyświetlania. Manipulator `left` pozostaje ważny do czasu użycia manipulatora `right`, który powoduje, że dalsze wyjście jest wyrównywane do prawej przy wyświetlaniu.

Omówione manipulatory podsumowano w tabeli 3.12.

Tabela 3.12. Manipulatory strumienia

Manipulator strumienia	Opis
`setw(`*n*`)`	Ustala pole o szerokości *n* znaków.
`fixed`	Wyświetla liczby zmiennoprzecinkowe w notacji stałoprzecinkowej.
`showpoint`	Powoduje wyświetlenie zer końcowych, nawet jeśli liczba nie posiada ułamka.
`setprecision(`*n*`)`	Ustala precyzję liczb zmiennoprzecinkowych.
`left`	Wyrównuje następujące po nim wyjścia do lewej.
`right`	Wyrównuje następujące po nim wyjścia do prawej.

Punkt kontrolny

3.17. Napisz instrukcje `cout` z zastosowaniem manipulatorów strumienia, które wykonają następujące zadania:
- A) wyświetlenie liczby 34,789 w polu o szerokości dziewięciu znaków z dokładnością do dwóch miejsc po przecinku,
- B) wyświetlenie liczby 7,0 w polu o szerokości pięciu znaków z dokładnością do trzech miejsc po przecinku,
- C) wyświetlenie liczby 5,789e+12 w notacji stałoprzecinkowej,
- D) wyświetlenie liczby 67 w polu o szerokości siedmiu znaków z wyrównaniem do lewej.

3.18. Poniższy program nie skompiluje się, ponieważ jego wiersze zostały wymieszane.

```
#include <iomanip>
}
cout << person << endl;
```

```
string person = "Wolfgang Smith";
int main()
cout << person << endl;
{
#include <iostream>
return 0;
cout << left;
using namespace std;
cout << setw(20);
cout << right;
```

Prawidłowe ułożenie wierszy powinno spowodować, że program wyświetli poniższe wyjście:

```
 Wolfgang Smith
Wolfgang Smith
```

Ułóż wiersze kodu w odpowiedniej kolejności. Przetestuj program na swoim komputerze.

3.19. Poniższy szkielet programu prosi o podanie kąta w stopniach i zamienia go na radiany. Twoim zadaniem jest napisanie kodu formatującego wyświetlany wynik.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    const double PI = 3.14159;
    double degrees, radians;
    cout << "Wpisz kąt w stopniach, a ja zamienię go\n";
    cout << "dla Ciebie na radiany: ";
    cin >> degrees;
    radians = degrees * PI / 180;
    // Wyświetl wartość w radianach wyrównaną do lewej,
    // w notacji stałoprzecinkowej i z precyzją 4, w polu
    // szerokim na pięć znaków. Upewnij się, że kropka
    // dziesiętna jest zawsze wyświetlana.
    return 0;
}
```

3.8. Operacje na znakach i obiekcie string

WYJAŚNIENIE: Istnieją specjalne funkcje do pracy na znakach i obiektach `string`.

Chociaż możemy wykorzystywać `cin` oraz operator `>>` do wprowadzania tekstów, wiąże się z tym kilka problemów, o których trzeba wiedzieć. Obiekt `cin` podczas odczytywania wejścia przechodzi dalej, ignorując białe znaki (spacje, tabulatory, znaki nowego wiersza). Odczytuje, gdy natrafi na pierwszy znak, który nie jest białym znakiem, i kontynuuje odczytywanie do napotkania kolejnego białego znaku. Przykład z listingu 3.18 pomoże Ci lepiej zrozumieć problem.

Listing 3.18

```
// Ten program ilustruje problem, który może wystąpić,
// jeśli cin zostanie użyty do odczytania tekstu do obiektu string.
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string name;
    string city;

    cout << "Wprowadź, proszę, swoje imię i nazwisko: ";
    cin >> name;
    cout << "Wprowadź miasto, w którym mieszkasz: ";
    cin >> city;

    cout << "Witaj, " << name << "." << endl;
    cout << "Mieszkasz w " << city << "." << endl;
    return 0;
}
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Wprowadź, proszę, swoje imię i nazwisko: Adam Nowak [Enter]
Wprowadź miasto, w którym mieszkasz: Witaj, Adam.
Mieszkasz w Nowak.
```

Zauważ, że użytkownik nie miał nawet szansy, żeby wprowadzić nazwę miasta. Gdy w pierwszej instrukcji wejściowej `cin` dotarł do spacji między `Adam` a `Nowak`, przestał odczytywać dalej. W drugiej instrukcji `cin` wykorzystał pozostałe znaki w buforze i zapisał `Nowak` jako nazwę miasta.

Aby ominąć ten problem, należy skorzystać z funkcji języka C++ o nazwie `getline()`. Odczytuje ona cały wiersz, łącznie z wszelkimi spacjami na początku czy oddzielającymi wyrazami, które zapisuje w obiekcie `string`. Funkcja `getline()` wygląda jak poniżej. `cin` to strumień wejściowy, z którego odczytujemy dane, a `wierszWejscia` to nazwa obiektu klasy `string`, który otrzymuje odczytane dane:

```
getline(cin, wierszWejscia);
```

Listing 3.19 ilustruje zastosowanie funkcji `getline()`.

Listing 3.19

```
// Ten program demonstruje użycie funkcji getline()
// do zapisania danych tekstowych do obiektu klasy string.
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string name;
    string city;

    cout << "Wprowadź, proszę, swoje imię i nazwisko: ";
```

```
13     getline(cin, name);
14     cout << "Wprowadź miasto, w którym mieszkasz: ";
15     getline(cin, city);
16
17     cout << "Witaj, " << name << "." << endl;
18     cout << "Mieszkasz w " << city << "." << endl;
19     return 0;
20 }
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Wprowadź, proszę, swoje imię i nazwisko: Adam Nowak [Enter]
Wprowadź miasto, w którym mieszkasz: Szczytno [Enter]
Witaj, Adam Nowak.
Mieszkasz w Szczytno.
```

Wpisywanie pojedynczych znaków

Czasami będziesz chciał, aby wejście było pojedynczym znakiem. Na przykład niektóre programy wyświetlają menu, z którego użytkownik może wybrać jedną pozycję. Często opcje do wyboru oznaczone są literami A, B, C itd. Użytkownik wybiera jedną pozycję z menu poprzez wpisanie znaku. Najprostszym sposobem na odczytanie jednego znaku jest zastosowanie `cin` oraz operatora `>>`, jak na listingu 3.20.

Listing 3.20

```
1 // Program odczytuje pojedynczy znak wprowadzony z klawiatury i zapisuje go w zmiennej typu char.
2 #include<iostream>
3 using namespace std;
4
5 int main()
6 {
7     char ch;
8
9     cout << "Wpisz znak i naciśnij Enter: ";
10    cin >> ch;
11    cout << "Wpisałeś " << ch << "." << endl;
12    return 0;
13 }
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Wpisz znak i naciśnij Enter: A [Enter]
Wpisałeś A.
```

Zastosowanie cin.get()

Podobnie jak w przypadku wejścia tekstowego, istnieją sytuacje, w których wykorzystanie `cin >>` do odczytania pojedynczego znaku nie działa tak, jak byś tego chciał. Przykładowo, ponieważ `cin` pomija białe znaki, nie można nic nie wpisać lub wprowadzić tylko znak naciśnięciem klawisza *Enter*. Program nie przejdzie do instrukcji występujących po `cin >>`, dopóki użytkownik nie wprowadzi znaku innego niż za pomocą klawisza *Spacja* czy *Enter*. (Jeśli taki znak zostanie wprowadzony, i tak należy

nacisnąć *Enter*, aby program przeszedł dalej). Dlatego programy podpowiadające użytkownikowi `"Aby kontynuować, naciśnij Enter"` nie mogą w tym celu korzystać z operatora `>>`.

W takich sytuacjach niezwykle przydatna okazuje się wbudowana funkcja o nazwie `get()` obiektu `cin`. Ponieważ funkcja `get()` jest częścią obiektu `cin`, nazywamy ją **metodą**. Metoda `get()` odczytuje pojedynczy znak, w tym białe znaki. Jeżeli program musi przechować odczytany znak, metodę `get()` można wykorzystać na jeden z dwóch poniższych sposobów. W obydwu przypadkach przyjmij, że `ch` to nazwa zmiennej, w której zapisujemy znak:

```
cin.get(ch);
ch = cin.get();
```

Jeżeli program musi skorzystać z metody `cin.get()`, aby wstrzymać wyświetlanie do czasu naciśnięcia klawisza *Enter*, i nie musi zapisywać nigdzie znaku, można ją zapisać jako:

```
cin.get();
```

Na listingu 3.21 znajdziesz wszystkie trzy sposoby zastosowania metody `cin.get()`.

Listing 3.21

```
// Ten program demonstruje trzy sposoby
// wykorzystania metody cin.get() do wstrzymania programu.
#include<iostream>
using namespace std;

int main()
{
    char ch;

    cout << "Program zatrzymał się. Naciśnij Enter, aby kontynuować.";
    cin.get(ch);
    cout << "Program zatrzymał się po raz drugi. Naciśnij Enter jeszcze raz.";
    ch = cin.get();
    cout << "Program zatrzymał się po raz trzeci. Naciśnij Enter jeszcze raz.";
    cin.get();
    cout << "Dziękuję!";
    return 0;
}
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Program zatrzymał się. Naciśnij Enter, aby kontynuować. [Enter]
Program zatrzymał się po raz drugi. Naciśnij Enter jeszcze raz. [Enter]
Program zatrzymał się po raz trzeci. Naciśnij Enter jeszcze raz. [Enter]
Dziękuję!
```

Mieszanie cin >> z cin.get()

Mieszanie `cin >>` z `cin.get()` powoduje pojawienie się denerwującego i trudnego do zidentyfikowania problemu. Celem lepszego zrozumienia spójrz na listing 3.22.

Listing 3.22

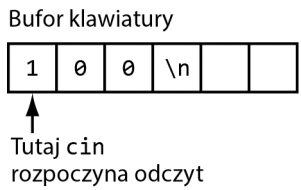
```
 1 // Ten program demonstruje problem, który pojawia się
 2 // przy mieszaniu cin >> z cin.get().
 3 #include <iostream>
 4 using namespace std;
 5
 6 int main()
 7 {
 8    char ch;                          // Zdefiniowanie zmiennej typu char
 9    int number;                       // Zdefiniowanie zmiennej typu int
10
11    cout << "Wpisz liczbę: ";
12    cin >> number;                    // Odczytanie liczby
13    cout << "Wpisz znak: ";
14    ch = cin.get();                   // Odczytanie znaku
15    cout << "Dziękuję!\n";
16    return 0;
17 }
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

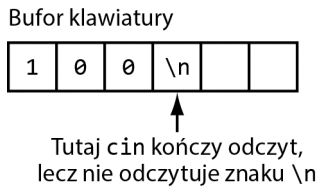
```
Wpisz liczbę: 100 [Enter]
Wpisz znak: Dziękuję!
```

Podczas działania programu wiersz 12. „prosi" o wpisanie liczby, jednak wygląda na to, że instrukcja z wiersza 14. zostaje pominięta. Dzieje się tak, ponieważ `cin >>` oraz `cin.get()` używają nieco innych technik do odczytu danych.

W przykładowym uruchomieniu programu, gdy wykonywana jest instrukcja z wiersza 12., użytkownik wpisuje 100 i naciska klawisz *Enter*. Naciśnięcie go powoduje wpisanie znaku nowego wiersza (`'\n'`) do bufora klawiatury, tak jak pokazano to na rysunku 3.5. Wyrażenie `cin >>` w wierszu 12. rozpoczyna odczytywanie wprowadzonych danych i przestaje po dotarciu do znaku nowego wiersza. Zilustrowano to na rysunku 3.6. Znak nowego wiersza nie zostaje odczytany, lecz pozostaje w buforze klawiatury.



Rysunek 3.5. Zawartość bufora klawiatury



Rysunek 3.6. Obiekt cin przestaje odczytywać, gdy natknie się na znak nowego wiersza

Gdy w wierszu 14. zacznie działać metoda `cin.get()`, rozpoczyna ona odczytywanie od miejsca w buforze klawiatury, w którym skończył czytać poprzedni operator wejścia. Oznacza to, że `cin.get()` odczyta znak nowego wiersza i nie da użytkownikowi możliwości wprowadzenia kolejnych znaków. Możesz temu zapobiec, stosując metodę `cin.ignore()` omówioną w kolejnej sekcji.

Zastosowanie cin.ignore()

Aby rozwiązać problem omówiony pod koniec poprzedniej sekcji, powinniśmy skorzystać z metody `ignore()` obiektu `cin`. Sprawia ona, że `cin` pominie jeden znak z bufora lub więcej znaków. Ogólna postać tej metody jest taka:

```
cin.ignore(n, c);
```

Argumenty w nawiasie są opcjonalne. Argument `n` to liczba całkowita, a argument `c` to znak. Pierwszy z nich powoduje, że `cin` ominie określoną liczbę znaków w buforze. Drugi natomiast sprawi, że obiekt `cin` będzie ignorował wszystkie znaki do momentu napotkania znaku określonego tym argumentem. Na przykład poniższa instrukcja spowoduje, że `cin` pominie 20 znaków lub będzie pomijał wszystkie znaki do napotkania znaku nowego wiersza, w zależności od tego, który z warunków zostanie spełniony jako pierwszy:

```
cin.ignore(20, '\n');
```

Jeżeli metoda zostanie wywołana bez argumentów, tak jak poniżej, pominie ona tylko pierwszy napotkany znak:

```
cin.ignore();
```

Program z listingu 3.23 jest zmodyfikowanym kodem z listingu 3.22 i demonstruje wykorzystanie metody `cin.ignore()`. Zwróć uwagę, że metoda ta została wywołana w wierszu 13., tuż po `cin >>`.

Listing 3.23

```
 1 // Ten program z powodzeniem wykorzystuje
 2 // zarówno cin >>, jak i cin.get() do odczytu klawiatury.
 3 #include<iostream>
 4 using namespace std;
 5
 6 int main()
 7 {
 8     char ch;
 9     int number;
10
11     cout << "Wpisz liczbę: ";
12     cin >> number;
13     cin.ignore();     // Pominięcie znaku nowego wiersza
14     cout << "Wpisz znak: ";
15     ch = cin.get();
16     cout << "Dziękuję!\n";
17     return 0;
18 }
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Wpisz liczbę: 100 [Enter]
Wpisz znak: Z [Enter]
Dziękuję!
```

Metody i operatory klasy string

Obiekty `string` w języku C++ mają szereg metod. Przykładowo, jeżeli chcesz sprawdzić długość ciągu znaków przechowywanego w obiekcie `string`, możesz wykorzystać metodę `length()`. Oto przykład jej zastosowania:

```
string stan =  "Teksas";
int rozmiar = stan.length();
```

Pierwsze polecenie tworzy obiekt klasy `string` i inicjuje go ciągiem znaków `"Teksas"`. Drugie wyrażenie definiuje zmienną `int` o nazwie `rozmiar` i inicjuje ją długością obiektu `stan`. Po wykonaniu kodu w zmiennej `rozmiar` znajdzie się wartość `5`.

Niektóre operatory mogą być również wykorzystane z obiektami klasy `string`. Jednym z nich jest operator `+`. Poznałeś już zastosowanie tego operatora przy operacjach dodawania dwóch wartości liczbowych. Ponieważ ciągów znaków nie można dodawać, operator `+` służy do ich **konkatenacji**, czyli łączenia. Załóżmy, że w programie mamy następujące definicje i inicjacje:

```
string powitanie1 = "Witaj, ";
string powitanie2;
string nazwa1 = "świecie";
string nazwa2 = "człowieku";
```

Poniższe wyrażenia ilustrują działanie konkatenacji:

```
powitanie2 = powitanie1 + nazwa1; // powitanie2 przechowuje teraz tekst "Witaj, świecie".
powitanie1 = powitanie1 + nazwa2; // powitanie1 przechowuje teraz tekst "Witaj, człowieku".
```

Zwróć uwagę, że na końcu tekstu z `powitanie1` znajduje się spacja. Jeśli by jej nie było, w obiekcie `powitanie2` zostałby zapisany ciąg `"Witaj,świecie"`.

Ostatnie wyrażenie z przykładu możemy zapisać za pomocą operatora łączonego`+=`, jak na poniższym przykładzie:

```
powitanie1 += nazwa2;
```

W rozdziale 10. poznasz więcej przydatnych metod i operatorów klasy `string`.

3.9. Więcej matematycznych funkcji bibliotecznych

WYJAŚNIENIE: Biblioteka uruchomieniowa C++ posiada szereg funkcji pozwalających wykonywać złożone operacje matematyczne.

W poprzednich podrozdziałach dowiedziałeś się, jak wykorzystać funkcję `pow()` do potęgowania liczb. Biblioteka języka C++ posiada wiele innych funkcji wykonujących działania matematyczne. Są one niezwykle przydatne w programach naukowych oraz w programach do specjalnych zastosowań. Kilka z tych funkcji znajdziesz w tabeli 3.13. Każda z nich wymaga załączenia pliku nagłówkowego `<cmath>`.

Tabela 3.13. Funkcje biblioteki <cmath>

Funkcja	Przykład	Opis
`abs()`	`y = abs(x);`	Zwraca wartość bezwzględną argumentu. Argument i zwracana wartość są typu `int`.
`cos()`	`y = cos(x);`	Zwraca cosinus argumentu. Argument powinien być kątem wyrażonym w radianach. Argument i wynik są typu `double`.
`exp()`	`y = exp(x);`	Oblicza funkcję wykładniczą argumentu `x`. Argument i wynik są typu `double`.
`fmod()`	`y = fmod(x, z);`	Zwraca resztę z dzielenia argumentów jako wartość typu `double`. Funkcja ta działa tak samo jak operator modulo, jednak argumenty i wynik są typu `double`.
`log()`	`y = log(x);`	Zwraca logarytm naturalny argumentu. Argument i wynik są typu `double`.
`log10()`	`y = log10(x);`	Zwraca logarytm dziesiętny argumentu. Argument i wynik są typu `double`.
`round()`	`y = round(x)`	Argument `x` może być typu `float`, `double` lub `longdouble`. Funkcja zwraca wartość `x` zaokrągloną do najbliższej pełnej wartości. Na przykład `2.8` zostanie zaokrąglone do `3.0`, a `2.1` do `2.0`. Wynik jest tego samego typu co argument.
`sin()`	`y = sin(x);`	Zwraca sinus argumentu. Argument powinien być kątem wyrażonym w radianach. Argument i wynik są typu `double`.
`sqrt()`	`y = sqrt(x);`	Zwraca pierwiastek kwadratowy argumentu. Argument i wynik są typu `double`.
`tan()`	`y = tan(x);`	Zwraca tangens argumentu. Argument powinien być kątem wyrażonym w radianach. Argument i wynik są typu `double`.

Każda z tych funkcji jest prosta w użyciu, tak jak funkcja `pow()`. Poniżej zaprezentowano fragment programu, który wykorzystuje funkcję `sqrt()` zwracającą pierwiastek kwadratowy liczby:

```
cout << "Wpisz liczbę: ";
cin >> num;
s = sqrt(num);
cout << "Pierwiastkiem kwadratowym liczby " << num << " jest " << s << "." <<
endl;
```

Oto wyjście wygenerowane przez powyższy fragment, z wartością `25` wpisaną przez użytkownika:

```
Wpisz liczbę: 25
Pierwiastkiem kwadratowym liczby 25 jest 5.
```

Listing 3.24 pokazuje zastosowanie funkcji `sqrt()` do obliczenia przeciwprostokątnej trójkąta prostokątnego. Program wykorzystuje równanie wyprowadzone z twierdzenia Pitagorasa:

$$c = \sqrt{a^2 + b^2}$$

We wzorze c to długość przeciwprostokątnej, zaś a i b to długości przyprostokątnych.

Listing 3.24

```
// Ten program prosi o wprowadzenie długości dwóch
// przyprostokątnych trójkąta. Następnie oblicza i wyświetla
// długość przeciwprostokątnej.
#include <iostream>
#include <iomanip>    // Na potrzeby funkcji setprecision()
#include <cmath>      // Na potrzeby funkcji sqrt() i pow()
using namespace std;

int main()
{
   double a, b, c;

   cout << "Wprowadź długość boku a: ";
   cin >> a;
   cout << "Wprowadź długość boku b: ";
   cin >> b;
   c = sqrt(pow(a, 2.0) + pow(b, 2.0));
   cout << "Długość przeciwprostokątnej wynosi ";
   cout << setprecision(2) << c << "." << endl;
   return 0;
}
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Wprowadź długość boku a: 5.0 [Enter]
Wprowadź długość boku b: 12.0 [Enter]
Długość przeciwprostokątnej wynosi 13.
```

Poniższe wyrażenie z listingu 3.24 oblicza pierwiastek z sumy kwadratów przyprostokątnych:

```
c = sqrt(pow(a, 2.0) + pow(b, 2.0));
```

Zwróć uwagę, że argumentem funkcji jest całe poniższe wyrażenie matematyczne:

```
pow(a, 2.0) + pow(b, 2.0)
```

W tym wyrażeniu funkcja `pow()` została wywołana dwa razy, do obliczenia kwadratu każdej z przyprostokątnych. Następnie wyniki funkcji zostały dodane, a ich suma została wysłana do funkcji `sqrt()`.

Liczby losowe

Liczby losowe przydają się w wielu zadaniach programistycznych. Poniżej wymieniono kilka przykładów:

- Liczby losowe często używane są w grach. Przykładem są gry komputerowe, w których gracz może rzucić kostką. Liczby losowe reprezentują wylosowaną wartość.
- Program, który pokazuje karty wyciągnięte z przetasowanej talii. Liczby losowe reprezentują wartości wyciągniętych kart.
- Liczby losowe są niezwykle przydatne w programach symulacyjnych. W symulacjach komputer decyduje, jak zachowa się osoba, zwierzę, owad lub inna istota. Możemy w tym celu stworzyć wzory, w których liczby losowe wpłyną na zachowania i zdarzenia w programie.
- Liczby losowe przydają się w programach statystycznych, które muszą wyciągnąć losowe dane do analizy.
- Liczby losowe są powszechnie stosowane w systemach bezpieczeństwa do szyfrowania ważnych danych.

W bibliotece C++ istnieje funkcja `rand()`, którą możesz wykorzystać do generowania liczb losowych (możemy z niej skorzystać po dołączeniu pliku nagłówkowego `<cstdlib>`). Funkcja zwraca wartość typu `int`. Oto przykład użycia:

```
y = rand();
```

Po wykonaniu tej funkcji liczba losowa zostanie wpisana do zmiennej `y`. W rzeczywistości liczby generowane przez `rand()` są pseudolosowe. Funkcja wykorzystuje algorytm, który zwraca tę samą sekwencję przy każdym uruchomieniu programu na tym samym systemie operacyjnym. Załóżmy, że wykonano poniższy kod:

```
cout << rand() << endl;
cout << rand() << endl;
cout << rand() << endl;
```

Trzy liczby, które wyświetli program, mogą wydawać się losowe, jednak te same trzy wartości zostaną zwrócone przy każdym uruchomieniu programu. Aby wyniki `rand()` były bardziej losowe, należy zastosować funkcję `srand()`. Przyjmuje ona argument typu `unsigned int`, który odgrywa rolę ziarna dla algorytmu. Jeśli poda się różne wartości ziarna, algorytm `rand()` wygeneruje różne sekwencje liczb losowych.

Częstym sposobem generowania liczb losowych jest wykorzystanie funkcji `time()` z biblioteki standardowej. Zwraca ona wyrażony w sekundach czas, który upłynął od północy 1 stycznia 1970 roku. Funkcja ta wymaga pliku nagłówkowego `<ctime>`, a jako argument przyjmuje `0`. Jej użycie zaprezentowano na listingu 3.25. Program powinien wygenerować trzy różne wartości losowe po każdym uruchomieniu.

Listing 3.25

```
1 // Ten program demonstruje liczby losowe.
2 #include <iostream>
3 #include <cstdlib>    // rand() i srand()
4 #include <ctime>      // Funkcja time()
5 using namespace std;
6
7 int main()
8 {
```

```
 9    // Pobranie czasu systemowego
10    unsigned seed = time(0);
11
12    // Wprowadzenie ziarna do generatora liczb losowych
13    srand(seed);
14
15    // Wyświetlenie trzech liczb losowych
16    cout << rand() << endl;
17    cout << rand() << endl;
18    cout << rand() << endl;
19    return 0;
20 }
```

Wyjście programu

```
142568398
276266676
1291155803
```

Jeżeli chcesz ograniczyć zakres generowanych liczb, skorzystaj z poniższego wyrażenia:

```
y = (rand() % (wartoscMax - wartoscMin + 1)) + wartoscMin;
```

gdzie *wartoscMax* to najwyższa wartość przedziału, a *wartoscMin* najniższa. W poniższym przykładzie do zmiennej y zapisywana jest liczba z przedziału od 1 do 100:

```
const int WARTOSC_MIN = 1;
const int WARTOSC_MAX = 100;
y = (rand() % (WARTOSC_MAX - WARTOSC_MIN + 1)) + WARTOSC_MIN;
```

W kolejnym przykładzie liczba losowana jest z przedziału między 100 a 200:

```
const int WARTOSC_MIN = 100;
const int WARTOSC_MAX = 200;
y = (rand() % (WARTOSC_MAX - WARTOSC_MIN + 1)) + WARTOSC_MIN;
```

W sekcji „W centrum uwagi" omówiono kod wykorzystujący liczby losowe do symulowania rzutów kostką.

W centrum uwagi

Zastosowanie liczb losowych

Dr Kimura, uczący wprowadzenia do statystyki, poprosił Cię o napisanie programu, którego mógłby użyć na zajęciach, aby zasymulować rzuty kostką. Program powinien generować dwie losowe liczby w przedziale od 1 do 6 i wyświetlać je. Kod programu z trzema przykładami wyjścia znajduje się na listingu 3.26.

Listing 3.26

```
1 // Ten program symuluje rzuty kostką.
2 #include <iostream>
3 #include <cstdlib>      // rand() i srand()
4 #include <ctime>        // Funkcja time()
5 using namespace std;
6
7 int main()
8 {
```

```
 9    // Stałe
10    const int MIN_VALUE = 1;    // Minimalna wartość na kostce
11    const int MAX_VALUE = 6;    // Maksymalna wartość na kostce
12
13    // Zmienne
14    int die1;    // Przechowywanie wartości pierwszej kostki
15    int die2;    // Przechowywanie wartości drugiej kostki
16
17    // Pobranie czasu systemowego
18    unsigned seed = time(0);
19
20    // Wprowadzenie ziarna do generatora liczb losowych
21    srand(seed);
22
23    cout << "Rzut kośćmi...\n";
24    die1 = (rand() % (MAX_VALUE - MIN_VALUE + 1)) + MIN_VALUE;
25    die2 = (rand() % (MAX_VALUE - MIN_VALUE + 1)) + MIN_VALUE;
26    cout << die1 << endl;
27    cout << die2 << endl;
28    return 0;
29 }
```

Wyjście programu

```
Rzut kośćmi...
4
3
```

Wyjście programu

```
Rzut kośćmi...
1
5
```

Wyjście programu

```
Rzut kośćmi...
6
3
```

Punkt kontrolny

3.20. Opisz krótko każdą z poniższych funkcji:

```
cos()     log()     sin()
exp()     log10()   sqrt()
fmod()    pow()     tan()
```

3.21. Przyjmij, że wartości zmiennych `angle1` i `angle2` to miary kątów wyrażone w radianach. Napisz wyrażenie, które sumuje wartości sinusa `angle1` i cosinusa `angle2` i zapisuje wynik w zmiennej `x`.

3.22. Aby obliczyć pierwiastek sześcienny, należy podnieść liczbę do potęgi 1/3. Pierwiastek czwartego stopnia obliczamy, podnosząc liczbę do potęgi 1/4. Napisz wyrażenie obliczające pierwiastek piątego stopnia z `x` i zapisujący go w zmiennej `y`.

3.23. Cosecans kąta *a* wynosi:

$$\frac{1}{\sin a}$$

Napisz wyrażenie obliczające cosecans kąta zapisanego w zmiennej `a` i zapisujące wynik w zmiennej `y`.

3.10. Rzecz o debugowaniu: ręczne śledzenie programu

Ręczne śledzenie programu to proces debugowania, w którym przyjmujesz rolę komputera wykonującego program. Musisz przejść przez każdy krok programu. Analizując wyrażenia, zapisujesz wartości, które znajdą się w zmiennych po wykonaniu instrukcji. Ten proces pomaga znaleźć błędy matematyczne i logiczne w kodzie programu.

Aby ręcznie prześledzić program, musisz utworzyć tabelkę z kolumną dla każdej zmiennej. Wiersze tabeli odpowiadają wierszom programu. Przykładowo, program z listingu 3.27 umieszczono tuż obok tabeli do ręcznego śledzenia. Program korzysta z czterech zmiennych: `num1`, `num2`, `num3` i `avg`. Zwróć uwagę, że tabela ma tyle kolumn, ile jest zmiennych, i tyle wierszy, ile jest wierszy w funkcji `main()`.

Listing 3.27

```
1 // Ten program prosi o wpisanie trzech liczb,
2 // a następnie wyświetla ich średnią.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8    double num1, num2, num3, avg;
9    cout << "Wpisz pierwszą liczbę: ";
10   cin >> num1;
11   cout << "Wpisz drugą liczbę: ";
12   cin >> num2;
13   cout << "Wpisz trzecią liczbę: ";
14   cin >> num3;
15   avg = num1 + num2 + num3 / 3;
16   cout << "Średnia wynosi " << avg << "." << endl;
17   return 0;
18 }
```

num1	num2	num3	avg

Program z listingu 3.27 prosi użytkownika o wpisanie trzech liczb i wyświetla średnią. Znajduje się w nim jednak błąd. Nie wyświetla prawidłowej wartości średniej. Poniżej możesz zobaczyć wyjście przykładowego uruchomienia programu.

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Wpisz pierwszą liczbę: 10 [Enter]
Wpisz drugą liczbę: 20 [Enter]
Wpisz trzecią liczbę: 30 [Enter]
Średnia wynosi 40.
```

Prawidłowa wartość średniej z liczb 10, 20 i 30 to 20, a nie 40. Aby znaleźć błąd, prześledzimy ręcznie przebieg programu. Żeby to zrobić, musisz przejść przez każdą instrukcję, analizując wykonywane operacje i zapisując wartości zmiennych po wykonaniu. Po zakończeniu śledzenia programu z listingu 3.27 otrzymamy tabelkę taką jak poniżej. W miejscach, w których nie znamy wartości zmiennych, wpisano znaki zapytania.

Listing 3.27 (z uzupełnioną tabelą ręcznego śledzenia)

```
1 // Ten program prosi o wpisanie trzech liczb,
2 // a następnie wyświetla ich średnią.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8    double num1, num2, num3, avg;
9    cout << "Wpisz pierwszą liczbę: ";
10   cin >> num1;
11   cout << "Wpisz drugą liczbę: ";
12   cin >> num2;
13   cout << "Wpisz trzecią liczbę: ";
14   cin >> num3;
15   avg = num1 + num2 + num3 / 3;
16   cout << "Średnia wynosi " << avg << "." << endl;
17   return 0;
18 }
```

num1	num2	num3	avg
?	?	?	?
?	?	?	?
10	?	?	?
10	?	?	?
10	20	?	?
10	20	?	?
10	20	30	?
10	20	30	40
10	20	30	40

Czy widzisz już błąd? Możemy go zauważyć po przeanalizowaniu wyrażenia z wiersza 14. Dzielenie ma miejsce przed zsumowaniem wszystkich wprowadzonych liczb. Dlatego musimy nieco zmodyfikować to wyrażenie:

```
avg = (num1 + num2 + num3) / 3;
```

Ręczne śledzenie programu to prosty proces, który pozwala skupić uwagę na każdej instrukcji programu. Często pomaga w znalezieniu nieoczywistych błędów.

3.11. Rozwiązywanie problemu: analiza przypadku

Firma „Skrzynki na Miarę Sp. z o.o." produkuje drewniane skrzynki projektowane na indywidualne zamówienia. Koszt produkcji jednej skrzynki, wliczając materiały i robociznę, wynosi 0,23 zł za decymetr sześcienny. Natomiast wyprodukowana skrzynka

sprzedawana jest za 0,50 zł za decymetr sześcienny. Zostałeś poproszony o napisanie programu, który oblicza objętość (w decymetrach sześciennych), koszt wykonania, cenę dla klienta i zysk z realizacji zamówienia.

Zmienne

W tabeli 3.14 znajdują się stałe nazwane i zmienne, które muszą się znaleźć w programie.

Tabela 3.14. Stałe nazwane i zmienne

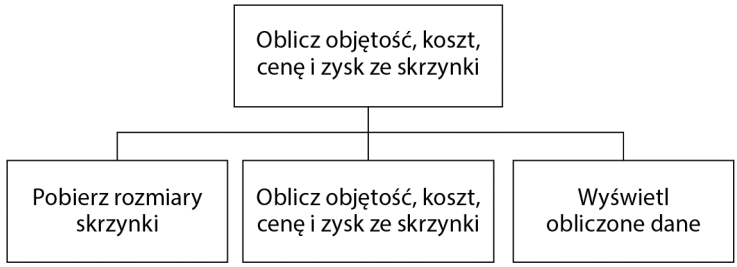
Zmienna lub stała	Opis
`COST_PER_CUBIC_DECIMETER`	Stała nazwana typu `double` o wartości `0.23`, reprezentująca koszt skrzynki za decymetr sześcienny.
`CHARGE_PER_CUBIC_DECIMETER`	Stała nazwana typu `double` o wartości `0.5`, reprezentująca cenę skrzynki za decymetr sześcienny.
`length`	Zmienna typu `double` przechowująca długość skrzynki wprowadzoną przez użytkownika.
`width`	Zmienna typu `double` przechowująca szerokość skrzynki wprowadzoną przez użytkownika.
`height`	Zmienna typu `double` przechowująca wysokość skrzynki wprowadzoną przez użytkownika.
`volume`	Zmienna typu `double` przechowująca objętość skrzynki obliczoną przez program.
`cost`	Zmienna typu `double` przechowująca koszt skrzynki obliczony przez program.
`charge`	Zmienna typu `double` przechowująca cenę skrzynki obliczoną przez program.
`profit`	Zmienna typu `double` przechowująca zysk ze sprzedaży skrzynki obliczony przez program.

Projekt programu

Program musi wykonywać poniższe ogólne kroki:

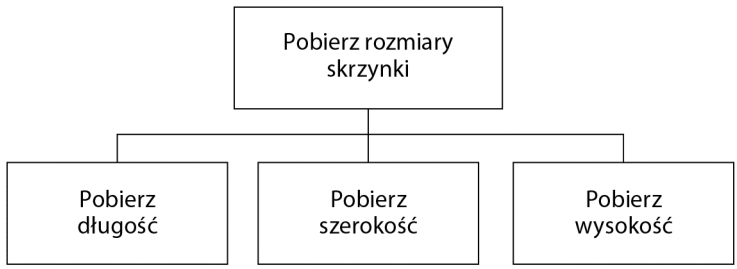
1. Poprosić użytkownika o wprowadzenie wymiarów skrzynki (długości, szerokości i wysokości).
2. Obliczyć objętość skrzynki, koszt jej produkcji, cenę i zysk.
3. Wyświetlić dane obliczone w kroku 2.

Ogólny diagram hierarchii programu przedstawiono na rysunku 3.7.



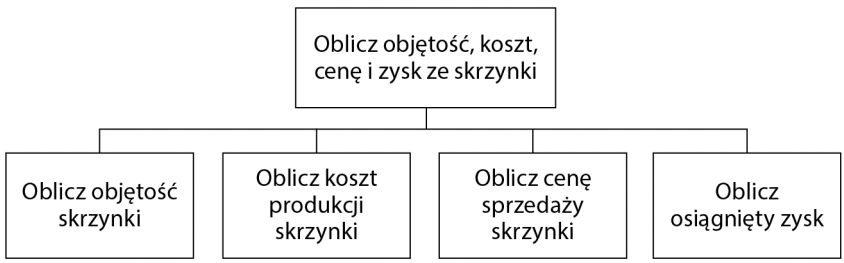
Rysunek 3.7. Diagram hierarchii

Diagram hierarchii kroku „Pobierz rozmiary skrzynki" został pokazany na rysunku 3.8.



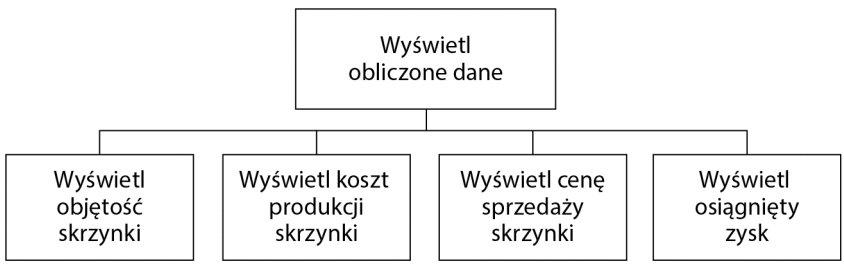
Rysunek 3.8. Diagram hierarchii kroku „Pobierz rozmiary skrzynki"

Diagram hierarchii kroku „Oblicz objętość, koszt, cenę i zysk ze skrzynki" zilustrowano na rysunku 3.9.



Rysunek 3.9. Diagram hierarchii kroku „Oblicz objętość, koszt, cenę i zysk ze skrzynki"

Diagram hierarchii kroku „Wyświetl obliczone dane" pokazano na rysunku 3.10.



Rysunek 3.10. Diagram hierarchii kroku „Wyświetl obliczone dane"

Pseudokod programu jest taki:

```
Poproś użytkownika o wprowadzenie długości skrzynki.
Poproś użytkownika o wprowadzenie szerokości skrzynki.
Poproś użytkownika o wprowadzenie wysokości skrzynki.
Oblicz objętość skrzynki.
Oblicz koszt wyprodukowania skrzynki.
Oblicz cenę za skrzynkę.
Oblicz zysk ze sprzedaży skrzynki.
Wyświetl objętość.
Wyświetl koszt produkcji.
Wyświetl cenę.
Wyświetl zysk ze sprzedaży.
```

Obliczenia

Wzory podane poniżej zostaną wykorzystane do obliczenia objętości, kosztu, ceny i zysku:

$$objętość = długość \cdot szerokość \cdot wysokość$$
$$koszt = objętość \cdot 0{,}23$$
$$cena = objętość \cdot 0{,}5$$
$$zysk = cena - koszt$$

Program

Ostatni krok to przekształcenie pseudokodu we właściwy program, który znajduje się na listingu 3.28.

Listing 3.28

```
// Program używany przez „Skrzynki na Miarę Sp. z o.o." do obliczania
// objętości, kosztów, ceny i zysku ze sprzedaży skrzynki
// dowolnego rozmiaru. Oblicza te dane na podstawie wprowadzonych przez
// użytkownika rozmiarów skrzynki.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   // Stałe dla kosztów i ceny
   const double COST_PER_CUBIC_DECIMETER = 0.23;
   const double CHARGE_PER_CUBIC_DECIMETER = 0.5;

   // Zmienne
   double length,   // Długość skrzynki
          width,    // Szerokość skrzynki
          height,   // Wysokość skrzynki
          volume,   // Objętość skrzynki
          cost,     // Koszt produkcji skrzynki
          charge,   // Cena za skrzynkę
          profit;   // Zysk osiągnięty ze sprzedaży skrzynki

```

```
24    // Ustawienie formatowania liczb na wyjściu
25    cout << setprecision(2) << fixed << showpoint;
26
27    // Poproszenie użytkownika o wpisanie długości, szerokości i wysokości skrzynki
28    cout<< "Wpisz rozmiary skrzynki(w decymetrach):\n";
29    cout << "Długość: ";
30    cin >> length;
31    cout << "Szerokość: ";
32    cin >> width;
33    cout << "Wysokość: ";
34    cin >> height;
35
36    // Obliczenie objętości skrzynki, kosztu produkcji,
37    // ceny dla klienta i zysku ze sprzedaży
38    volume = length * width * height;
39    cost = volume * COST_PER_CUBIC_DECIMETER;
40    charge = volume * CHARGE_PER_CUBIC_DECIMETER;
41    profit = charge - cost;
42
43    // Wyświetlenie obliczonych danych
44    cout << "Objętość skrzynki wynosi ";
45    cout << volume << " decymetrów sześciennych.\n";
46    cout << "Koszt produkcji: " << cost << " zł" << endl;
47    cout << "Cena: " << charge << " zł" << endl;
48    cout << "Zysk: " << profit << " zł" << endl;
49    return 0;
50 }
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Wpisz rozmiary skrzynki(w decymetrach):
Długość: 10 [Enter]
Szerokość: 8 [Enter]
Wysokość: 4 [Enter]
Objętość skrzynki wynosi 320.00 decymetrów sześciennych.
Koszt produkcji: 73.60 zł
Cena: 160.00 zł
Zysk: 86.40 zł
```

Wyjście programu (wprowadzone dane są wyróżnione pogrubioną czcionką)

```
Wpisz rozmiary skrzynki(w decymetrach):
Długość: 12.5 [Enter]
Szerokość: 10.5 [Enter]
Wysokość: 8 [Enter]
Objętość skrzynki wynosi 1050.00 decymetrów sześciennych.
Koszt produkcji: 241.50 zł
Cena: 525.00 zł
Zysk: 283.50 zł
```

Pytania i ćwiczenia kontrolne

Krótka odpowiedź

1. Załóżmy, że zdefiniowano poniższe zmienne:

```
intage;
intpay;
char section;
```

Napisz jedno wyrażenie `cin`, za pomocą którego wpiszesz wartości do każdej ze zmiennych.

2. Załóżmy, że zdefiniowano następujący obiekt `string`:

```
string description;
```

A) Napisz wyrażenie, które zapisuje do obiektu jednowyrazowy ciąg znaków.
B) Napisz wyrażenie, które zapisuje do obiektu ciąg znaków składający się z wielu wyrazów rozdzielonych białymi znakami.

3. Jakie pliki nagłówkowe musimy załączyć w poniższym programie?

```
int main()
{
double amount = 89.7;
cout<<showpoint<< fixed;
cout<<setw(8) << amount <<endl;
return 0;
}
```

4. Uzupełnij tabelę, wpisując odpowiednie wartości wyrażeń.

Wyrażenie	Wartość
`28 / 4 - 2`	
`6 + 12 * 2 - 8`	
`4 + 8 * 2`	
`6 + 17 % 3 - 2`	
`2 + 22 * (9 - 7)`	
`(8 + 7) * 2`	
`(16 + 7) % 2 - 1`	
`12 / (10 - 6)`	
`(19 - 3) * (2 + 2) / 4`	

5. Napisz w C++ wyrażenia odpowiadające poniższym wyrażeniom algebraicznym:

$a = 12x$

$z = 5x + 14y + 6k$

$y = x^4$

$$g = \frac{h+12}{4k}$$

$$c = \frac{a^3}{b^2k^4}$$

6. Załóżmy, że w programie istnieją następujące definicje zmiennych:

```
int units;
float mass;
double weight;
```

oraz poniższe wyrażenie:

```
weight = mass * units;
```

Jaki typ automatycznej konwersji będzie miał miejsce?

A) Zmienna `mass` zostanie zamieniona na `int`, `units` pozostanie `int`, a wynikiem działania będzie również wartość typu `int`.

B) Zmienna `units` zostanie zamieniona na `float`, zmienna `mass` pozostanie `float`, a wynikiem działania będzie również wartość typu `float`.

C) Zmienna `units` zostanie zamieniona na `float`, zmienna `mass` pozostanie `float`, a wynikiem działania będzie również wartość typu `double`.

7. Załóżmy, że w programie istnieją następujące definicje zmiennych:

```
int a, b = 2;
float c = 4.2;
```

oraz poniższe wyrażenie:

```
a = b * c;
```

Co zostanie zapisane w zmiennej `a`?

A) `8.4`

B) `8`

C) `0`

D) żadna z powyższych wartości

8. Załóżmy, że zmienne `qty` i `salesReps` są typu `int`. Zastosuj rzutowanie typów tak, aby w poniższym wyrażeniu uniknąć dzielenia liczb całkowitych:

```
unitsEach = qty / salesReps;
```

9. Zamień poniższą definicję zmiennej na definicję stałej nazwanej:

```
intrate;
```

10. Uzupełnij poniższą tabelę, zapisując równoważniki wyrażeń po lewej za pomocą łączonych operatorów przypisania.

Wyrażenia z operatorem przypisania	Wyrażenia z łączonym operatorem przypisania
`x = x + 5;`	
`total = total + subtotal;`	
`dist = dist / rep;`	
`ppl = ppl * period;`	
`inv = inv - shrinkage;`	
`num = num % 2;`	

11. Napisz wyrażenie wielokrotnego przypisania zastępujące poniższy kod:

```
east = 1;
west = 1;
north = 1;
south = 1;
```

12. Napisz wyrażenie wyświetlające wartość zmiennej `divSales` w polu o szerokości 8 znaków w notacji stałoprzecinkowej z dokładnością do dwóch miejsc po przecinku. Kropka dziesiętna powinna być wyświetlana zawsze.

13. Napisz wyrażenie wyświetlające wartość zmiennej `totalAge` w polu o szerokości 12 znaków w notacji stałoprzecinkowej z dokładnością do czterech miejsc po przecinku.
14. Napisz wyrażenie wyświetlające wartość zmiennej `population` w polu o szerokości 12 znaków, z wyrównaniem do lewej, z dokładnością do ośmiu miejsc po przecinku. Kropka dziesiętna powinna być wyświetlana zawsze.

Uzupełnij

15. Funkcja biblioteczna _____ zwraca cosinus kąta.
16. Funkcja biblioteczna _____ zwraca sinus kąta.
17. Funkcja biblioteczna _____ zwraca tangens kąta.
18. Funkcja biblioteczna _____ zwraca wartość funkcji wykładniczej dla liczby.
19. Funkcja biblioteczna _____ zwraca resztę z dzielenia liczb zmiennoprzecinkowych.
20. Funkcja biblioteczna _____ zwraca logarytm naturalny liczby.
21. Funkcja biblioteczna _____ zwraca logarytm dziesiętny liczby.
22. Funkcja biblioteczna _____ zwraca wartość liczby podniesionej do potęgi.
23. Funkcja biblioteczna _____ zwraca pierwiastek kwadratowy liczby.
24. Plik _____ musi zostać załączony do programu, który korzysta z funkcji matematycznych.

Warsztat projektanta algorytmów

25. Sklep detaliczny przyznaje swoim klientom maksymalną wartość kredytu. Kredyt dostępny dla każdego klienta oblicza się jako wartość maksymalną minus wykorzystana wartość kredytu. Napisz w pseudokodzie algorytm programu, w którym poprosisz użytkownika o wpisanie wartości maksymalnego kredytu dla klienta i kredytu wykorzystanego. Program powinien wyświetlić wartość dostępnego kredytu dla danego klienta.

    Po napisaniu pseudokodu napisz na jego podstawie cały program w C++.
26. Napisz w pseudokodzie algorytm programu obliczającego całkowitą wartość sprzedaży detalicznej. Program powinien poprosić o podanie wartości sprzedaży i wysokości podatku od sprzedaży. Wysokość podatku powinna być wpisana jako liczba zmiennoprzecinkowa. Na przykład jeżeli podatek wynosi 23%, użytkownik powinien wpisać `0.23`. Program powinien wyświetlić wartość podatku i całkowitą wartość sprzedaży.

    Po napisaniu pseudokodu napisz na jego podstawie cały program w C++.

27. Napisz w pseudokodzie algorytm programu, który prosi użytkownika o wpisanie punktów golfisty z trzech partii golfa, a następnie wyświetla średnią punktacji.

    Po napisaniu pseudokodu napisz na jego podstawie cały program w C++.

Znajdź błędy

W każdym z poniższych programów znajdują się błędy. Znajdź ich jak najwięcej.

28.
```
using namespace std;
int main ()
{
    double number1, number2, sum;
    Cout << "Wpisz liczbę: ";
    Cin << number1;
    Cout << "Wpisz kolejną liczbę: ";
    Cin << number2;
    number1 + number2 = sum;
    Cout "Suma dwóch liczb wynosi " << sum << "."
    return 0;
}
```

29.
```
#include <iostream>
using namespace std;
int main()
{
    int number1, number2;
    float quotient;
    cout << "Wpisz dwie liczby, a ja podzielę\n";
    cout << "pierwszą przez drugą za Ciebie.\n";
    cin >> number1, number2;
    quotient = float<static_cast>(number1) / number2;
    cout << quotient
    return 0;
}
```

30.
```
#include <iostream>;
using namespace std;
int main()
{
    const int number1, number2, product;
    cout << "Wpisz dwie liczby, a ja pomnożę\n";
    cout << "je dla Ciebie.\n";
    cin >> number1 >> number2;
    product = number1 * number2;
    cout << product
    return 0;
}
```

31.
```
#include <iostream>;
using namespace std;
int main()
{
    const int number1, number2, product;
    cout << "Wpisz dwie liczby, a ja pomnożę\n";
```

```
    cout << "je dla Ciebie.\n";
    cin >> number1 >> number2;
    product = number1 * number2;
    cout << product
    return 0;
}
```

32.

```
#include <iostream>;
using namespace std;
main
{
    double number, half;
    cout << "Wpisz liczbę, a ja podzielę ją \n"
    cout << "dla Ciebie na pół.\n"
    cin >> number1;
    half =/ 2;
    cout << fixedpoint << showpoint << half << endl;
    return 0;
}
```

33.

```
#include <iostream>;
using namespace std;
int main()
{
    char name, go;
    cout << "Wpisz swoje imię: ";
    getline >> name;
    cout << "Witaj, " << name << "." << endl;
    return 0;
}
```

Odgadnij wynik

Co wyświetlą poniższe programy (niektóre z nich powinny być śledzone ręcznie i wymagają użycia kalkulatora)?

34. (*Przyjmij, że użytkownik wpisał 38700. Użyj kalkulatora*).

```
#include <iostream>
using namespace std;

int main()
{
   double salary, monthly;
   cout << "Ile wynosi Twoje roczne wynagrodzenie? ";
   cin >> salary;
   monthly = static_cast<int>(salary) / 12;
   cout << "Twoje miesięczne wynagrodzenie wynosi" << monthly << "." <<
endl;
   return 0;
}
```

35.

```
#include <iostream>
using namespace std;
int main()
{
```

```
    long x, y, z;
    x = y = z = 4;
    x += 2;
    y -= 1;
    z *= 3;
    cout << x << " " << y << " " << z << endl;
    return 0;
}
```

36. (*Przyjmij, że użytkownik wpisał Jerzy Waszyngton*)

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
int main()
{
    string userInput;
    cout<< "Jak masz na imię? ";
    getline(cin, userInput);
    cout<< "Witaj, " <<userInput<<endl;
    return 0;
}
```

37. (*Przyjmij, że użytkownik wpisał 36720152. Użyj kalkulatora*)

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    long seconds;
    double minutes, hours, days, months, years;

    cout << "Wpisz liczbę sekund, które minęły\n";
    cout << "od jakiegoś momentu w przeszłości, a ja\n";
    cout << "powiem Ci, ile minęło minut, godzin, \n";
    cout << "dni, miesięcy i lat: ";
    cin >> seconds;
    minutes = seconds / 60;
    hours = minutes / 60;
    days = hours / 24;
    years = days / 365;
    months = years * 12;
    cout << setprecision(4) << fixed << showpoint << right;
    cout << "Minuty: " << setw(6) << minutes << endl;
    cout << "Godziny: " << setw(6) << hours << endl;
    cout << "Dni: " << setw(6) << days << endl;
    cout << "Miesiące: " << setw(6) << months << endl;
    cout << "Lata: " << setw(6) << years << endl;
    return 0;
}
```

Wyzwania programistyczne

1. **Kilometry na litry**

   Napisz program, który oblicza zużycie paliwa samochodu. Program ten powinien poprosić użytkownika o wprowadzenie maksymalnej liczby litrów paliwa, jaką

może pomieścić samochód, oraz liczby kilometrów, jaką można przejechać na pełnym baku. Następnie powinien wyświetlić liczbę kilometrów, jaką można przejechać na litrze paliwa.

2. **Miejsca stadionowe**

   Na stadionie są trzy kategorie miejsc siedzących. Na mecz softballa miejsca siedzące w klasie A kosztują 15 zł, w klasie B 12 zł, a w klasie C 9 zł. Napisz program, który zapyta, ile biletów każdej klasy na miejsca siedzące zostało sprzedanych, a następnie wyświetli dochód ze sprzedaży tych biletów. Sformatuj wyświetlanie kwoty w notacji stałoprzecinkowej z dokładnością do dwóch miejsc po przecinku i upewnij się, że kropka dziesiętna jest zawsze wyświetlana.

3. **Średnia z testów**

   Napisz program, który prosi o pięć wyników testów. Program powinien obliczać średnią wyników testów i ją wyświetlać. Wyświetlana liczba powinna być sformatowana w notacji stałoprzecinkowej z dokładnością do jednego miejsca po przecinku.

4. **Średnia opadów deszczu**

   Napisz program obliczający średnią opadów deszczu z trzech miesięcy. Program powinien poprosić użytkownika o wprowadzenie nazwy każdego miesiąca, na przykład czerwca lub lipca, oraz ilości opadów (w milimetrach) w każdym z miesięcy. Program powinien wyświetlić komunikat podobny do poniższego:

   Średnia opadów deszczu w czerwcu, lipcu i sierpniu wyniosła 6,72 mm.

5. **Procent chłopców i dziewcząt**

   Napisz program, który zapyta użytkownika o liczbę chłopców i dziewcząt zapisanych do klasy. Program powinien wyświetlać wartość procentową chłopców i dziewcząt w klasie.

   Wskazówka: załóż, że w klasie jest 8 chłopców i 12 dziewczynek. W sumie w klasie jest 20 uczniów. Procent chłopców może zostać wyliczony jako `8 / 20 = 0.4` *albo 40%. Procent dziewczynek może zostać wyliczony jako* `12 / 20 = 0.6` *albo 60%.*

6. **Przeliczanie składników**

   Aby upiec ciasteczka według pewnego przepisu, potrzebujemy:
   - 1,5 szklanki cukru,
   - 1 szklanki masła,
   - 2,75 szklanki mąki.

   Podane składniki wystarczą na 48 ciasteczek. Napisz program pytający użytkownika o liczbę ciasteczek, które chciałby upiec, a następnie wyświetlający liczbę szklanek każdego ze składników potrzebną do ich upieczenia.

7. **Kasa biletowa**

   Kino przechowuje tylko pewien procent przychodu ze sprzedaży biletów. Reszta trafia do dystrybutora. Napisz program obliczający dochód brutto kina i zysk netto kasy biletowej za noc. Program powinien pytać o tytuł filmu oraz o to,

ile biletów dla dzieci i dorosłych zostało sprzedanych. (Cena biletu dla dorosłych wynosi 10 zł, a dla dziecka 6 zł). Wyświetlony powinien zostać komunikat podobny do poniższego:

Tytuł filmu:	*Wheels of Fury*
Sprzedane bilety dla dorosłych:	382
Sprzedane bilety dla dzieci:	127
Zysk kasy brutto:	4582,00 zł
Zysk kasy netto:	916,40 zł
Kwota zapłacona dystrybutorowi:	3665,60 zł

UWAGA. Załóż, że kino zatrzymuje 20% zysku brutto kasy biletowej.

8. Ile gadżetów?

Firma YukonWidgets produkuje gadżety ważące 12,5 kg każdy. Napisz program, który oblicza, ile gadżetów jest na palecie, bazując w tym celu na całkowitej wadze palety. Program powinien pytać użytkownika o wagę samej palety i wagę palety ze znajdującymi się na niej gadżetami. Następnie powinien obliczać i wyświetlać liczbę gadżetów zapakowanych na palecie.

9. Ile kalorii?

Opakowanie zawiera 30 ciastek. Znajduje się na nim informacja o kaloriach, podająca, że zawiera ono 10 „porcji", a każda porcja ma 300 kalorii. Napisz program, który prosi użytkownika o wprowadzenie liczby zjedzonych ciastek, a następnie informuje, ile kalorii zostało w sumie skonsumowanych.

10. Jak duże ubezpieczenie?

Wielu ekspertów finansowych doradza, że właściciele nieruchomości powinni ubezpieczać swoje domy i budynki na co najmniej 80% kosztów zastąpienia budowli. Napisz program, który poprosi użytkownika o wprowadzenie kosztów zastąpienia budynku, a następnie wyświetli minimalną wielkość ubezpieczenia, które powinno zostać wykupione dla tej nieruchomości.

11. Miesięczne koszty samochodu

Napisz program, który prosi użytkownika o wprowadzenie miesięcznych kosztów poszczególnych wydatków poniesionych w związku z użytkowaniem samochodu: raty kredytu, ubezpieczenia, paliwa, oleju, opon i utrzymania. Program powinien wyświetlać całkowity miesięczny koszt tych wydatków i ich całkowity koszt roczny.

12. Stopnie Celsjusza na stopnie Fahrenheita

Napisz program zamieniający temperaturę podaną w stopniach Celsjusza na temperaturę w stopniach Fahrenheita. Wzór to:

$$F = \frac{9}{5}C + 32$$

F oznacza stopnie Fahrenheita, a C stopnie Celsjusza.

13. Waluta

Napisz program, który zamieni daną wartość dolara amerykańskiego na jeny japońskie oraz na euro, zachowując współczynniki konwersji w stałych `YEN_PER_DOLLAR` i `EUROS_PER_DOLLAR`. Aby otrzymać najbardziej aktualny kurs wymiany, przeszukaj internet, używając hasła „kurs wymiany walut". Jeśli nie możesz znaleźć najnowszych danych, przyjmij, że:

1 dolar = 98,93 jena
1 dolar = 0,74 euro

Sformatuj kwotę waluty w notacji stałoprzecinkowej z dokładnością do dwóch miejsc po przecinku i upewnij się, że kropka dziesiętna jest zawsze wyświetlana.

14. Podatek od sprzedaży

Firma prowadząca sprzedaż detaliczną musi złożyć raport podatkowy o miesięcznej sprzedaży, wyszczególniając miesięczną sprzedaż i wysokość zainkasowanego podatku VAT. Napisz program, który poprosi o miesiąc, rok i całkowitą wielkość zainkasowanego przychodu w kasie (na przykład sprzedaż plus podatek od sprzedaży). Przyjmij, że wysokość podatku VAT wynosi 23%.

Jeżeli znana jest cała wielkość przychodu w kasie, a podatek wynosi 23%, wartość sprzedaży produktów możemy obliczyć ze wzoru:

$$S = \frac{T}{1{,}23}$$

S to wartość sprzedaży, a T to całkowity przychód (sprzedaż plus podatek).

Program powinien wyświetlić raport podobny do poniższego:

```
Miesiąc: październik
--------------------------------
Przychód całkowity:              12300.00 zł
Sprzedaż:                        10000.00 zł
Podatek VAT:                     2300.00 zł
```

15. Podatek od nieruchomości

Gmina pobiera podatek od nieruchomości naliczany od szacunkowej wartości nieruchomości, która wynosi 60% jej rynkowej ceny. Jeżeli wartość rynkowa hektara ziemi to 10 000 zł, wówczas jego wartość szacunkowa wyniesie 6000 zł. Podatek to kwota 75 gr od każdych 100 zł wartości szacunkowej. Podatek dla hektara o wartości szacunkowej 600 zł będzie wynosił więc 45 zł. Napisz program, który prosi o wprowadzenie wartości rynkowej nieruchomości i wyświetla jej wartość szacunkową oraz wysokość podatku.

16. Podatek od nieruchomości dla osób starszych

W jednej z gmin istnieje ulga w podatku od nieruchomości w wysokości 5000 zł dla starszych mieszkańców. Na przykład jeżeli wartość rynkowa domu starszej osoby wynosi 150 000 zł, jego wartość szacunkowa, obliczona na tej samej zasadzie jak w poprzednim zadaniu, będzie wynosiła 94 800 zł. Jednakże opodatkowaniu będzie podlegała jedynie wartość 89 800 zł. Przy zeszłorocznej wysokości podatku wynoszącej 2,67 zł za każde 100 zł wartości szacunkowej wysokość podatku wyniesie 2370,22 zł. Oprócz ulgi podatkowej seniorzy mogą rozłożyć zapłatę podatku na cztery równe płatności. W tym przypadku każda

płatność wyniosłaby 592,68 zł. Napisz program, który prosi użytkownika o wpisanie wartości rynkowej nieruchomości i aktualnej wysokości podatku za każde 100 zł wartości szacowanej. Program powinien wyświetlić wartość podatku od nieruchomości dla seniora i wysokość kwartalnej płatności.

17. Korepetytor matematyki

Napisz program, który może zostać użyty jako korepetytor matematyki dla ucznia. Program powinien wyświetlać dwie przypadkowe liczby do dodania, na przykład:

```
  247
+ 129
```

W trakcie gdy uczeń myśli nad rozwiązaniem, program powinien się zatrzymać. Gdy uczeń jest gotowy, by sprawdzić odpowiedź, powinien móc nacisnąć klawisz, a wtedy program wyświetli poprawne rozwiązanie:

```
  247
+ 129
------
  376
```

18. Zysk z odsetek

Zakładając, że mamy konto z kwotą wpłaconą raz, bez dodatkowych dopłat, jego stan po roku możemy obliczyć ze wzoru:

$$wartośa = k_p\left(1+\frac{oprocentowanie}{T}\right)^T$$

gdzie k_p to początkowy stan konta, a T to liczba kapitalizacji w ciągu roku (jeżeli kapitalizacja zachodzi co kwartał, $T = 4$).

Napisz program, który poprosi o początkowy stan konta, wysokość oprocentowania i liczbę kapitalizacji. Program powinien wyświetlić raport podobny do poniższego:

```
Oprocentowanie konta:       4.25%
Liczba kapitalizacji:         12
Saldo początkowe:     1000.00 zł
Odsetki:                43.34 zł
Końcowy stan konta:   1043.34 zł
```

19. Raty miesięczne

Miesięczne raty kredytu mogą być obliczone z poniższego wzoru:

$$rata = \frac{oprocentowanie\cdot(1+oprocentowanie)^N}{(1+oprocentowanie)^N-1}L$$

Gdzie *oprocentowanie* to miesięczna stopa oprocentowania kredytu, która jest obliczana przez podzielenie rocznego oprocentowania przez 12 (roczne oprocentowanie 12% to 1% miesięcznego oprocentowania). N to liczba spłat, a L to kwota kredytu. Napisz program, który prosi o wpisanie tych wartości, a następnie wyświetla raport podobny do poniższego:

```
Kwota kredytu:              10000.00 zł
Miesięczne oprocentowanie:          1%
Liczba rat:                        36
Rata miesięczna:             332.14 zł
Łączna wartość:            11957.15 zł
Zapłacone odsetki:          1957.15 zł
```

20. Pizza pi

Pizzeria Janka potrzebuje programu do obliczania liczby kawałków, na które może być podzielona pizza każdego rozmiaru. Program powinien wykonywać następujące kroki:

A) zapytanie użytkownika o średnicę pizzy w centymetrach;
B) obliczenie liczby kawałków, które mogą być wydzielone z pizzy tego rozmiaru;
C) wyświetlenie komunikatu informującego o liczbie kawałków.

Aby obliczyć liczbę kawałków, które mogą być wydzielone z pizzy, musisz wiedzieć, że:

- pole każdego kawałka powinno wynosić 35,877 cm^2;
- aby obliczyć liczbę kawałków, należy po prostu podzielić pole pizzy przez 35,877;
- pole pizzy oblicza się ze wzoru:

$$pole = \pi r^2$$

UWAGA. π to grecka litera *pi*. Jako jej wartość można przyjąć 3,14159. Zmienna *r* to promień pizzy. Podziel średnicę przez 2, aby otrzymać promień.

Upewnij się, że wynik pokazuje liczbę kawałków w notacji stałoprzecinkowej z dokładnością zaokrągloną do jednego miejsca po przecinku. Do zapisu *pi* wykorzystaj stałą nazwaną.

21. Ile pizz?

Zmodyfikuj program, który napisałeś w wyzwaniu 20. („Pizza pi"), tak, żeby informował o liczbie pizz potrzebnych do kupienia na przyjęcie, jeśli spodziewamy się, że każdy gość zje średnio cztery kawałki. Program powinien zapytać użytkownika o liczbę osób, które pojawią się na przyjęciu, oraz o średnicę pizz do zamówienia. Następnie powinien obliczyć i wyświetlić liczbę pizz do zakupienia.

22. Kalkulator kąta

Napisz program, który prosi użytkownika o wpisanie kąta w radianach. Potem program powinien wyświetlić sinus, cosinus i tangens kąta (aby ustalić te wartości, użyj funkcji bibliotecznych `sin()`, `cos()` i `tan()`). Wynik powinien być wyświetlony w notacji stałoprzecinkowej i zaokrąglony do czterech miejsc po przecinku.

23. Program transakcji akcji

W zeszłym miesiącu Jan dokonał zakupu akcji kapitałowych Acme Oprogramowanie S.A. Poniżej przedstawione są szczegóły dotyczące zakupu:

- Jan zakupił 1000 akcji.
- Za jedną akcję zapłacił 45,40 zł.
- Zapłacił maklerowi giełdowemu prowizję w wysokości 2% ceny, jaką zapłacił za akcje.

Dwa tygodnie później Jan sprzedał akcje. Oto szczegóły sprzedaży:

- Jan sprzedał 1000 akcji.
- Akcje sprzedał w cenie 56,90 zł za jedną.
- Zapłacił swojemu maklerowi kolejną prowizję w wysokości 2% kwoty, jaką otrzymał za akcje.

Napisz program, który wyświetli następujące informacje:

- kwotę, jaką Jan zapłacił za akcje;
- kwotę prowizji, jaką Jan zapłacił maklerowi za zakup akcji;
- kwotę, za jaką Jan sprzedał akcje;
- kwotę prowizji, jaką Jan zapłacił maklerowi za sprzedaż akcji;
- kwotę zysku Jana ze sprzedaży akcji i po opłaceniu dwóch prowizji maklera (jeśli kwota zysku wyświetlana przez program jest liczbą ujemną, oznacza to, że Jan w związku z transakcją poniósł straty).

24. Sadzenie winorośli

Właścicielka winnicy sadzi kilka nowych szpalerów winorośli i musi wiedzieć, ile winorośli posadzić w każdym szpalerze. Ustaliła, że po zmierzeniu długości przyszłego szpaleru, będzie mogła wykorzystać poniższy wzór, aby obliczyć liczbę winorośli, która zmieści się w szpalerze razem z treliażami i słupkami skrajnymi niezbędnymi do postawienia na końcu każdego szpaleru:

$$V = \frac{R - 2E}{S}$$

gdzie:

V oznacza liczbę winorośli, które zmieszczą się w szpalerze;
R to długość szpaleru, wyrażona w metrach;
E to przestrzeń w metrach zajmowana przez skrajny słupek;
S to przestrzeń między winoroślami, wyrażona w metrach.

Napisz program wykonujący obliczenia dla właścicielki winnicy. Program powinien poprosić użytkowniczkę o wprowadzenie następujących danych:

- długości rzędu w metrach,
- przestrzeni zajmowanej przez słupek skrajny w metrach,
- wielkości przestrzeni między winoroślami w metrach.

Kiedy dane wejściowe zostaną wprowadzone, program powinien obliczyć i wyświetlić liczbę winorośli, które zmieszczą się w rzędzie.

25. Gra słów

Napisz program, który rozgrywa grę z użytkownikiem. Program powinien poprosić użytkownika o wprowadzenie poniższych danych:

- imienia,
- wieku,
- nazwy miasta,

- nazwy szkoły,
- zawodu,
- gatunku zwierzęcia,
- imienia zwierzęcia.

Po wprowadzeniu potrzebnych danych program powinien wyświetlić następującą historyjkę, zawierającą wprowadzone przez użytkownika informacje w odpowiednich miejscach:

```
Była sobie raz osoba imieniem IMIĘ, która mieszkała w MIASTO. W wieku WIEK
IMIĘ poszedł do szkoły SZKOŁA. IMIĘ zakończył naukę i zaczął pracę jako
ZAWÓD. Następnie IMIĘ adoptował ZWIERZĘ o imieniu IMIĘ_ZWIERZĘCIA. Oboje żyli
razem długo i szczęśliwie.
```

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

>>> IDŹ I PROGRAMUJ W C++!

C++ powstał w 1979 roku. Od tej pory cały czas się rozwija. Mimo że wymaga od programisty pewnej dyscypliny i staranności w pracy, jest chętnie wykorzystywany, daje bowiem ogromne możliwości i cechuje się elastycznością. W języku tym zaimplementowano mechanizmy programowania obiektowego. Opanowanie C++ nie tylko pozwala programistom tworzyć aplikacje dla praktycznie wszystkich platform, komputerów, serwerów i urządzeń mobilnych, ale również ułatwia naukę innych języków programowania. Aby w pełni skorzystać z tych zalet, trzeba od początku przykładać się do nauki pisania kodu z poszanowaniem dobrych praktyk.

To kolejne wydanie lubianego podręcznika programowania w języku C++; książka przeznaczona dla osób, które dopiero rozpoczynają naukę kodowania, i tych, które mają już doświadczenie z innymi językami. Znalazło się tu przystępne wyjaśnienie podstaw działania komputera oraz wprowadzenie do samego języka, a także mnóstwo pożytecznych wskazówek dla początkujących. Bardziej zaawansowani programiści docenią szczegółowe opisy niuansów, zawiłości i źródeł możliwych problemów. Książka jest napisana prostym, zrozumiałym językiem i zawiera wiele świetnie dobranych przykładów ilustrujących nie tylko funkcje i konstrukcje języka C++, ale również przypadki i sposoby ich użycia.

Najważniejsze zagadnienia:

- przystępne wprowadzenie do C++
- funkcje i klasy, tablice i wektory
- wyrażenia lambda, wskaźniki i przeciążanie operatorów
- dziedziczenie i polimorfizm
- stosy, kolejki i rekurencja
- drzewa binarne: tworzenie i operacje na drzewach

Tony Gaddis

napisał wiele popularnych podręczników do nauki programowania. Od ponad dwudziestu lat prowadzi kursy informatyczne, przede wszystkim w Haywood Community College w Karolinie Północnej. Jest laureatem licznych nagród dla nauczycieli i trenerów. Słynie z umiejętności przystępnego wyjaśniania nawet bardzo złożonych zagadnień; dotyczy to szczególnie nauki języków programowania, w tym Javy, Visual Basic i C#.

ISBN 978-83-283-4680-2