ALEX ALLAIN

JUŻ DZIŚ NAUCZ SIĘ PROGRAMOWAĆ!

PRZEWODNIK DLA POCZĄTKUJĄCYCH

C++

Helion

Tytuł oryginału: Jumping into C++

Tłumaczenie: Ireneusz Jakóbik

ISBN: 978-83-246-8920-0

Jumping into C++. Copyright © 2012 by F. Alexander Allain

All rights reserved. This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the author except for the use of brief quotations in a book review.

Polish edition copyright © 2014 by Helion S.A.
All rights reserved.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/cppppo.zip

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

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

■ ■ ■ ROZDZIAŁ 3

Interakcja z użytkownikiem. Zapisywanie informacji w zmiennych

Wiesz już, jak napisać prosty program wyświetlający informacje wprowadzone przez Ciebie — to jest przez programistę — oraz w jaki sposób opisywać programy za pomocą komentarzy. To całkiem nieźle, ale co zrobić, gdybyś chciał wejść w interakcję z użytkownikiem?

Aby współdziałać z użytkownikiem, powinieneś mieć możliwość przyjmowania **danych wejściowych**, czyli informacji pochodzących spoza programu. W tym celu musisz dysponować jakimś miejscem na przechowanie tych danych. W programach dane wejściowe, a także inne informacje, są zapisywane w **zmiennych**. Istnieje kilka różnych typów zmiennych, które przechowują różne rodzaje informacji (na przykład liczby albo litery). Kiedy mówisz kompilatorowi, że deklarujesz zmienną, powinieneś podać mu jej **typ**, a także nazwę.

Najczęściej używane podstawowe typy zmiennych, z których będziesz korzystać, to `char`, `int` i `double`. Zmienna typu `char` przechowuje pojedynczy znak, zmienne typu `int` przechowują liczby całkowite (czyli liczby bez miejsc dziesiętnych), natomiast zmienne typu `double` przechowują liczby z miejscami dziesiętnymi. Każdy z tych typów jest zarazem słowem kluczowym używanym podczas deklarowania zmiennej.

Deklarowanie zmiennych w C++

Zanim użyjesz zmiennej, powinieneś poinformować o tym kompilator, deklarując ją (kompilator jest bardzo czuły na punkcie wcześniejszego informowania go o wszystkim). Aby zadeklarować zmienną, należy skorzystać ze składni `typ <zmienna>;` (zwróć uwagę na średnik!).

Oto kilka przykładów deklaracji zmiennych:

```
int liczba_calkowita;
char litera;
double liczba_dziesietna;
```

W jednym wierszu można zadeklarować wiele zmiennych tego samego typu. Wszystkie zmienne powinny być rozdzielone przecinkami.

```
int a, b, c, d;
```

Zalecam jednak deklarowanie każdej ze zmiennych w osobnym wierszu, co ułatwia czytanie kodu.

Kup książkę Poleć książkę

Korzystanie ze zmiennych

Wiesz już, jak powiedzieć kompilatorowi o zmiennych, ale w jaki sposób z nich korzystać?

To proste — należy skorzystać z instrukcji `cin` w celu pobrania danych wejściowych, operatora wstawiania skierowanego w prawą stronę (>>) oraz ze zmiennej, do której chcemy „wstawić" wartość wpisaną przez użytkownika.

Oto przykładowy program ilustrujący użycie zmiennej:

Przykładowy kod 5.: ***czytaj_liczbe.cpp***

```
#include <iostream>

using namespace std;

int main ()
{
      int tojestliczba;
      cout << "Proszę wprowadź liczbę: ";
      cin >> tojestliczba;
      cout << "Wprowadziłeś: " << tojestliczba << "\n";
}
```

Rozłóżmy nasz kod na składniki i przeanalizujmy go wiersz po wierszu. Pierwszą część programu już znasz, a zatem skupmy się na ciele funkcji `main`.

```
int tojestliczba;
```

W wierszu tym zadeklarowano zmienną typu całkowitego `tojestliczba`. Następna nowa linia to:

```
cin >> tojestliczba;
```

Funkcja `cin >> tojestliczba` powoduje zapisanie wprowadzonej wartości w zmiennej `tojest` ↪`liczba`. Zanim liczba zostanie odczytana przez program, użytkownik musi nacisnąć *Enter*.

Co zrobić, gdy program błyskawicznie kończy działanie?

Jeżeli Twój poprzedni program błyskawicznie kończył działanie i musiałeś użyć instrukcji `cin.get ()`, aby temu przeciwdziałać, może się zdarzyć, że okno z obecnym programem także od razu zostanie zamknięte, nawet jeśli dołączysz wywołanie funkcji `cin.get ()`. Możesz ominąć ten problem, dodając przed instrukcją `cin.get ()` następujący wiersz:

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

Funkcja ta odczytuje i pozbywa się znaku — w tym przypadku jest to *Enter* naciśnięty przez użytkownika. Tak! Kiedy użytkownik wprowadza do komputera dane, program odczytuje także ten znak. Nie jest on nam potrzebny i dlatego możemy się go pozbyć. Zazwyczaj powyższy wiersz będzie wymagany tylko wtedy, gdy używasz instrukcji `cin.get ()` w celu wstrzymania działania programu w oczekiwaniu na naciśnięcie klawisza. Bez niego `cin.get ()` wczyta znak nowego wiersza (*Enter*), a działanie Twojego programu zostanie od razu zakończone.

Pamiętaj, że zmienna `tojestliczba` jest zadeklarowana jako liczba całkowita. Jeśli użytkownik wprowadzi liczbę z miejscami dziesiętnymi, zostanie ona **obcięta** (miejsca po przecinku będą

Kup książkę Poleć książkę

pomijane; na przykład 3,1415 zostanie odczytane jako 3). Po uruchomieniu programu spróbuj wpisać ciąg liter albo liczbę dziesiętną. Reakcja programu będzie zależeć od wprowadzonych danych, ale w żadnym przypadku nie będzie pożądana. Na razie nie zajmiemy się obsługą błędów, która jest potrzebna, aby radzić sobie w takich sytuacjach.

```
cout << "Wprowadziłeś: " << tojestliczba << "\n";
```

W wierszu tym wypisywana jest liczba podana przez użytkownika. Zwróć uwagę, że przy wyświetlanej zmiennej nie użyto znaków cudzysłowu. Gdybyśmy wokół zmiennej `tojestliczba` zastosowali cudzysłowy, na ekranie pokazałby się napis `Wprowadziłeś: tojestliczba`. Brak znaków cudzysłowu informuje kompilator, że ma do czynienia ze zmienną i że przed wyświetleniem jej na ekranie powinien sprawdzić jej wartość w celu zastąpienia nazwy zmiennej jej treścią.

Przy okazji, nie dziw się dwóm oddzielnym operatorom wstawiania umieszczonym w jednym wierszu. Taki zapis jest jak najbardziej poprawny, a informacje wyjściowe zostaną wyświetlone w odpowiednich miejscach. W rzeczywistości literały łańcuchowe (czyli napisy zawarte między cudzysłowami) muszą być oddzielane od zmiennych za pomocą operatorów wstawiania (<<). Próba wyświetlenia zmiennej razem z literałem łańcuchowym przy użyciu jednego tylko operatora << wywoła błąd o następujące treści:

```
ZŁY KOD
cout << "Wprowadziłeś: " tojestliczba;
```

Tak jak w przypadku wszystkich innych funkcji, wiersz kończy się średnikiem. Jeśli o nim zapomnisz, podczas próby skompilowania programu kompilator wyświetli komunikat informujący o błędzie.

Zmiana wartości zmiennych oraz ich porównywanie

Odczytywanie i wyświetlanie zmiennych bardzo szybko robi się nudne. Zajmijmy się możliwością modyfikowania zmiennych oraz wpływania na zachowanie programu w zależności od ich wartości. Dzięki temu będziemy mogli w różny sposób reagować na różne dane wprowadzane przez użytkownika.

Zmiennej można przypisać wartość za pomocą **operatora przypisania**, którym jest znak równości =.

```
int x;
x = 5;
```

Powyższy zapis nadaje zmiennej x wartość 5. Mógłbyś pomyśleć, że znak równości **porównuje** wartości z jego lewej i prawej strony, ale tak nie jest. W C++ do sprawdzania równości jest używany oddzielny operator, składający się z dwóch znaków równości ==. Ze znaku tego będziesz często korzystać w instrukcjach warunkowych albo pętlach. W kilku kolejnych rozdziałach będziemy używać operatora porównania podczas poznawania sposobów na wybieranie różnych ścieżek działania programu w zależności od danych wprowadzonych przez użytkownika.

```
a == 5 // NIE przypisuje wartości 5 do zmiennej a, tylko sprawdza, czy zmienna ta jest równa 5.
```

Na zmiennych możesz przeprowadzać operacje arytmetyczne.

Kup książkę Poleć książkę

*	Mnoży dwie wartości
-	Odejmuje jedną wartość od drugiej
+	Dodaje dwie wartości
/	Dzieli jedną wartość przez drugą

Oto kilka przykładów:

```
a = 4 * 6; // Daje w wyniku 24 (zwróć uwagę na użycie średnika oraz komentarza)
a = a + 5; // Daje w wyniku poprzednią wartość zmiennej a powiększoną o 5
```

Skrócone zapisy na dodawanie i odejmowanie jedynki

W C++ bardzo często zachodzi potrzeba zwiększenia zmiennej o jeden:

```
int x = 0;
x = x + 1;
```

Z podobnym wzorcem będziesz mieć do czynienia w dalszej części tej książki, kiedy zaczniemy poznawać takie koncepcje jak pętle. Jest on do tego stopnia powszechny, że nawet istnieje operator, którego jedynym celem jest zwiększanie zmiennej o jeden: ++.

Pokazany wcześniej kod można zapisać następująco:

```
int x = 0;
x++;
```

Po wykonaniu powyższych instrukcji zmienna x przyjmie wartość 1. Operator ten jest znany pod nazwą operatora **inkrementacji**, a dodanie 1 do zmiennej nazywane jest **inkrementacją** tej zmiennej.

Operator -- działa podobnie, tylko że odejmuje od zmiennej 1. Jest on znany pod nazwą operatora **dekrementacji**, natomiast odjęcie 1 od zmiennej nazywane jest **dekrementacją** tej zmiennej.

Wiedząc to, możesz domyślić się, skąd pochodzi nazwa C++. C++ bazuje na języku programowania C. C++ znaczy dosłownie „C plus jeden”. C++ to bardziej C z pewnymi dodatkami niż zupełnie nowy język. Myślę, że gdyby twórcy C++ wiedzieli, o ile potężniejszy będzie ten język od C, nadaliby mu nazwę C do kwadratu.

Istnieje podobny, skrócony operator umożliwiający dodanie dowolnej wartości do zmiennej:

```
x += 5; // Dodaje 5 do x
```

Są także operatory odejmowania, mnożenia i dzielenia:

```
x -= 5; // Odejmuje 5 od x
x *= 5; // Mnoży x przez 5
x /= 5; // Dzieli x przez 5
```

Operatory ++ i -- możesz umieszczać nie tylko po zmiennej, ale także przed zmienną:

```
--x;
++y;
```

Różnica między zapisami „przed” i „po” kryje się w wartości, jaka jest zwracana w danym wyrażeniu. Jeśli napiszesz:

Kup książkę Poleć książkę

```
int x = 0;
cout << x++;
```

Zostanie wyświetlone 0. Dzieje się tak dlatego, że pomimo zmodyfikowania zmiennej x została zwrócona jej wartość początkowa. Ponieważ operator ++ występuje po zmiennej, zwiększenie jej wartości ma miejsce już po jej odczytaniu.

Jeżeli umieścisz operator przed zmienną, uzyskasz nową wartość:

```
int x = 0;
cout << ++x;
```

Zostanie wyświetlone 1, ponieważ najpierw nastąpi zwiększenie zmiennej x o 1, a dopiero później pobranie jej wartości. Znając wszystkie te operacje, możesz napisać w C++ niewielki kalkulator.

Przykładowy kod 6.: ***kalkulator.cpp***

```
#include <iostream>

using namespace std;

int main()
{
      int pierwszy_argument;
      int drugi_argument;
      cout << "Podaj pierwszy argument: ";
      cin >> pierwszy_argument;
      cout << "Podaj drugi argument: ";
      cin >> drugi_argument;
      cout << pierwszy_argument << " * " << drugi_argument << " = " <<
      ↪pierwszy_argument * drugi_argument << endl;
      cout << pierwszy_argument << " + " << drugi_argument << " = " <<
      ↪pierwszy_argument + drugi_argument << endl;
      cout << pierwszy_argument << " / " << drugi_argument << " = " <<
      ↪pierwszy_argument / drugi_argument << endl;
      cout << pierwszy_argument << " - " << drugi_argument << " = " <<
      ↪pierwszy_argument - drugi_argument << endl;
}
```

Poprawne i niepoprawne użycie zmiennych

Najczęściej popełniane błędy podczas deklarowania zmiennych w C++

Deklarowanie zmiennych daje Ci w programach wiele nowych możliwości, ale niepoprawne zadeklarowanie zmiennej może wywołać szereg problemów. Jeśli na przykład spróbujesz użyć zmiennej, której nie zadeklarowałeś, kompilacja nie powiedzie się i wystąpi błąd dotyczący **niezadeklarowanej zmiennej**. Jeżeli skorzystasz ze zmiennej, która nie była zadeklarowana, kompilator zazwyczaj wygeneruje następujący błąd (w tym przypadku zmienną tą jest x):

```
error: 'x' was not declared in this scope
```

Dokładna treść komunikatu zależy od kompilatora. Powyższy przykład dotyczy kompilatora MinGW oraz środowiska Code::Blocks.

Kup książkę Poleć książkę

Chociaż może istnieć wiele zmiennych tego samego typu, nie można mieć wielu zmiennych o tej samej nazwie. Nie ma na przykład możliwości utworzenia jednej zmiennej o typie zmiennoprzecinkowym oraz drugiej zmiennej o typie całkowitym, z których obie będą mieć nazwę `moja_wartosc`. Komunikat błędu dotyczący zadeklarowania dwóch zmiennych o tej samej nazwie może wyglądać następująco:

```
error: conflicting declaration 'double moja_wartosc'
error: 'moja_wartosc' has a previous declaration as 'int moja_wartosc'
error: declaration of 'double moja_wartosc'
error: conflicts with previous declaration 'int moja_wartosc'
```

Trzeci najczęściej występujący błąd polega na pominięciu średnika na końcu wiersza:

```
ZŁY KOD
int x
```

W zależności od instrukcji znajdującej się po deklaracji zmiennej brak średnika może wywołać ze strony kompilatora wiele różnych komunikatów o błędach. Zwykle komunikat będzie się odnosić do wiersza występującego bezpośrednio po deklaracji.

Niektóre błędy nie pojawią się podczas kompilacji, ale w czasie działania programu. Kiedy zadeklarujesz zmienną, pozostaje ona **niezainicjalizowana**. Zanim użyjesz zmiennej, musisz ją **zainicjalizować**. Aby przeprowadzić inicjalizację zmiennej, powinieneś nadać jej wartość. Jeśli tego nie zrobisz, program będzie się zachowywać w sposób nieprzewidywalny. Często spotykany problem wygląda mniej więcej tak:

```
int x;
int y;
y = 5;
x = x + y;
```

W powyższym przykładzie tylko zmiennej `y` nadano wartość 5 przed jej użyciem. Początkowa wartość `x` pozostaje nieznana. Podczas działania programu zostanie ona wybrana losowo, tak więc ostateczna wartość zmiennej `x` może być zupełnie dowolna! Nie należy zakładać, że zmienne są inicjalizowane jakąś wygodną wartością, taką jak na przykład 0.

Jedna z technik, z której możesz korzystać, polega na inicjalizowaniu zmiennych podczas ich deklarowania:

```
int x = 0;
```

Takie rozwiązanie w zupełności wystarczy, aby po utworzeniu zmiennej jej wartość była znana. Przyjęcie takiego nawyku z pewnością pozwoli Ci uniknąć w przyszłości popełnienia wielu paskudnych błędów, a czym wśród przyjaciół jest tych kilka dodatkowych stuknięć w klawisze?

Rozróżnianie wielkości liter

Mamy teraz dobrą okazję do porozmawiania o kolejnym ważnym zagadnieniu, przez które łatwo możesz się pogubić. Jest nim **rozróżnienie wielkości liter**. W C++ ma znaczenie, czy użyjesz liter wielkich, czy małych. Nazwy `Kot` i `kot` mają dla kompilatora różne znaczenie. We wszystkich słowach kluczowych, nazwach funkcji oraz zmiennych w C++ rozróżniana jest wielkość liter.

Różnica w wielkości liter (na przykład `X` i `x`) między deklaracją zmiennej i miejscami jej użycia jest jednym z powodów, dla których może wystąpić błąd niezadeklarowanej zmiennej, gdy jesteś przekonany, że zmienną jednak zadeklarowałeś.

Kup książkę Poleć książkę

Nazwy zmiennych

Wybieranie zrozumiałych i opisowych nazw zmiennych także jest bardzo ważne. Oto przykład złego doboru nazw zmiennych:

```
wart1 = wart2 * wart3;
```

Co to znaczy? Tego nie wie nikt. Nazwy użyte w powyższej instrukcji są praktycznie bezużyteczne. Kiedy tworzysz program, myślisz, że kod, który piszesz, jest dość oczywisty — jesteś o tym przekonany w dniu, w którym programujesz. Następnego dnia ten sam kod będzie dla Ciebie niezrozumiały. Dzięki nadawaniu opisowych nazw zmiennym będziesz mniej zdezorientowany, gdy następnym razem będziesz czytać swój kod. Oto przykład:

```
powierzchnia = szerokosc * wysokosc;
```

Taki zapis jest o wiele czytelniejszy niż pierwsze równanie, i to tylko dzięki zmianie nazw.

Przechowywanie łańcuchów tekstowych

Zapewne zauważyłeś, że wszystkie wspomniane do tej pory typy zmiennych pozwalają na przechowywanie prostych wartości, na przykład pojedynczej liczby całkowitej albo znaku. Za ich pomocą można całkiem sporo zdziałać, ale C++ oferuje także inne typy danych[1].

Jednym z najprzydatniejszych typów danych jest typ łańcuchowy **string**. Umożliwia on przechowywanie wielu znaków. Widziałeś już typ łańcuchowy w działaniu, gdy na ekranie był wyświetlany napis:

```
cout << "Hej, jestem tutaj! Och, i oczywiście Hello World!\n";
```

Klasa `string` w C++ umożliwia przechowywanie, modyfikowanie oraz wykonywanie innych działań na łańcuchach tekstowych.

Zadeklarowanie zmiennej łańcuchowej jest proste:

Przykładowy kod 7.: *string.cpp*

```
#include <string>

using namespace std;

int main ()
{
       string my_string;
}
```

Zwróć uwagę, że w odróżnieniu od innych typów wbudowanych, w celu użycia typu `string` powinieneś dołączyć plik nagłówkowy `<string>`. Jest tak dlatego, że typ `string` nie jest wbudowany bezpośrednio w kompilator, tak jak to jest w przypadku typów całkowitych. Łańcuchy tekstowe są udostępniane poprzez standardową bibliotekę C++, która jest ogromnym źródłem kodu przeznaczonego do wielokrotnego użycia.

Tak samo jak w przypadku pozostałych typów podstawowych dostępnych w C++, łańcuchy tekstowe wpisywane przez użytkownika można wczytywać za pomocą instrukcji `cin`.

[1] C++ umożliwia programiście tworzenie własnych typów danych. Powrócimy do tego tematu w dalszej części książki, podczas omawiania struktur.

Kup książkę Poleć książkę

Przykładowy kod 8.: ***string_imie.cpp***

```
#include <iostream>
#include <string>

using namespace std;

int main ()
{
	string imie_uzytkownika;
	cout << "Proszę podać swoje imię: ";
	cin >> imie_uzytkownika;
	cout << "Cześć " << imie_uzytkownika << "\n";
}
```

Program ten tworzy zmienną łańcuchową, prosi użytkownika o podanie swojego imienia, po czym wyświetla je na ekranie.

Tak jak w przypadku innych zmiennych, zmienne łańcuchowe można inicjalizować wartością początkową:

```
string imie_uzytkownika = '<nieznane>'
```

Jeśli chcesz zestawić ze sobą dwa łańcuchy tekstowe (operacja ta jest nazywana **dołączaniem**[2] jednego łańcucha do drugiego), możesz skorzystać ze znaku +:

Przykładowy kod 9.: ***string_dolacz.cpp***

```
#include <iostream>
#include <string>

using namespace std;

int main ()
{
	string imie_uzytkownika;
	string nazwisko_uzytkownika;

	cout << "Proszę podać swoje imię: ";
	cin >> imie_uzytkownika;
	cout << "Proszę podać swoje nazwisko: ";
	cin >> nazwisko_uzytkownika;
	string pelne_imie_uzytkownika = imie_uzytkownika + " " + nazwisko_uzytkownika;

	cout << "Nazywasz się: " << pelne_imie_uzytkownika << "\n";
}
```

Program bierze trzy różne łańcuchy tekstowe: imię użytkownika, pojedynczy odstęp oraz nazwisko użytkownika, i łączy je w jeden łańcuch.

Kiedy wczytujesz łańcuchy tekstowe, czasami chciałbyś pobrać jednorazowo cały wiersz. Istnieje specjalna funkcja, `getline`, której można w tym celu użyć; pozbywa się ona nawet końcowego znaku nowego wiersza.

[2] Czasami spotkasz się z terminem **konkatenacja**, oznaczającym łączenie łańcuchów tekstowych. Słowo „konkatenacja" wywodzi się z łacińskiego określenia łączenia za pomocą łańcucha; *catena* w łacinie to łańcuch.

Kup książkę Poleć książkę

Aby skorzystać z funkcji `getline`, należy przekazać jej źródło danych wejściowych (w tym przypadku `cin`), łańcuch, do którego nastąpi wczytanie, oraz znak, na którym należy zakończyć wczytywanie łańcucha. Na przykład poniższy kod pobierze imię użytkownika:

```
getline( cin, user_first_name, '\n' );
```

Funkcja `getline` jest przydatna również wtedy, gdy chcesz pobrać od użytkownika dane wejściowe tylko do pewnego znaku, na przykład przecinka (chociaż użytkownik nadal musi nacisnąć *Enter*, zanim program przyjmie dane).

```
getline ( cin, moj_lancuch, '.' );
```

Jeśli użytkownik wpisze teraz:

```
Hello, World
```

do zmiennej `moj_lancuch` wejdzie wartość `Hello`. Dalsza część tekstu, w tym przypadku `World`, będzie pozostawać w buforze wejściowym do chwili, w której program odczyta bufor za pomocą następnej instrukcji pobierającej z niego dane.

No dobrze, rozumiem już łańcuchy tekstowe, ale co z pozostałymi typami?

*Uwaga! Podrozdział ten zawiera zaawansowany materiał, **z którego nie musisz jeszcze korzystać**. Jeśli jest on dla Ciebie zbyt trudny, nie ma przeszkód, abyś go teraz pominął i powrócił do niego później.*

Być może zastanawiasz się teraz, dlaczego istnieje aż tyle różnych podstawowych typów zmiennych. Poświęćmy więc chwilę na zaznajomienie się z dwoma elementarnymi składnikami wszystkich programów komputerowych, jakimi są **bit** i **bajt**. Bit stanowi podstawową jednostkę pamięci komputera. Jest on przełącznikiem o stanach włączony i wyłączony, przechowującym w zależności od ustawienia wartość 1 lub 0. Bajt składa się z ośmiu bitów. Ponieważ w bajcie znajduje się osiem bitów, istnieje 256 różnych zestawień zer i jedynek. Jest tak, ponieważ istnieje osiem pozycji, z których każda może przyjmować dwie wartości. Omówię to dokładniej. Jeden bit może przechowywać 0 lub 1, czyli dwie wartości. Drugi bit podwaja liczbę możliwości: 00, 01, 10 i 11. Trzeci bit ponownie podwaja liczbę możliwości przez dodanie zera lub jedynki do każdej dwubitowej kombinacji. Każdy bit podwaja zatem liczbę możliwych do przedstawienia wartości. Innymi słowy, dla n bitów mamy 2^n wartości. Ponieważ bajt zawiera osiem bitów, może on przyjmować 2^8 możliwych konfiguracji. Jeśli masz dwa bajty, masz tym samym 16 bitów, co przekłada się na 2^{16} (65 536) wartości.

Nie przejmuj się, jeśli nie zrozumiałeś tego wszystkiego. Najważniejsze to zapamiętać, że im więcej masz bitów, tym większy zakres danych możesz przechowywać.

Na przykład w zmiennej typu `char` można zapisać dane o ograniczonym zakresie — tylko 256 różnych wartości; to pojedynczy bajt. Zmienna typu `integer` składa się zwykle z czterech bajtów, co oznacza, że może ona przedstawiać około cztery miliardy różnych liczb.

Dobry przykład dwóch typów zmiennych, które różnią się między sobą tylko wielkością miejsca zajmowanego w pamięci, stanowi `double` oraz jego mniejszy brat `float`. Float był tak naprawdę pierwszym typem, który mógł przechowywać liczby zmiennoprzecinkowe, a samo pojęcie liczby zmiennoprzecinkowej wiąże się z faktem, że przecinek może zmieniać swoje położenie w liczbie. Innymi słowy, przed przecinkiem dziesiętnym mogą znajdować się dwie cyfry, a po

Kup książkę Poleć książkę

przecinku cztery (12,1234), albo cztery cyfry przed przecinkiem i dwie po nim (1234,12). Nie jesteś ograniczony do konkretnej liczby cyfr przed przecinkiem dziesiętnym i po nim.

Nie martw się, jeżeli nie wszystko z tego do końca rozumiesz — liczby zmiennoprzecinkowe są tak nazywane głównie ze względów historycznych. Zapamiętaj tylko, że nazwa ta odnosi się do liczb z miejscami po przecinku (czyli miejscami dziesiętnymi). Typ `float` zajmuje tylko cztery bajty pamięci i nie może przechowywać tylu różnych wartości, ile typ `double`, który dysponuje ośmioma bajtami. Dawniej, kiedy komputery miały mniej pamięci niż teraz, gra była warta świeczki, a programiści zadawali sobie wiele trudu, aby zaoszczędzić kilka bajtów. W obecnych czasach prawie zawsze lepiej będzie korzystać z typu `double`, chociaż w przypadku ograniczonych zasobów (na przykład w systemach z niewielką pamięcią, takich jak telefony komórkowe) nadal istnieje możliwość użycia typu `float`.

Najmniejszym typem danych jest `char`. Być może zastanawiasz się, dlaczego nadal istnieje ten typ, skoro pamięć nie stanowi już wielkiego problemu. Oto odpowiedź: typ `char` ma pewne specjalne znaczenie — operacje wejścia i wyjścia są realizowane raczej poprzez znaki niż za pośrednictwem liczb. Kiedy dla zmiennej wybierzesz typ `char`, użytkownik będzie mógł wpisać z klawiatury znak, a gdy zechcesz wyświetlić zmienną, `cout` wypisze znak w niej przechowywany zamiast liczby, która rzeczywiście tam się znajduje. „Moment — zapytasz — co to oznacza? Dlaczego liczby są znakami?”. Odpowiedź na to pytanie jest następująca: gdy komputer przechowuje to, co według nas jest znakiem (takim jak na przykład litera „a”), tak naprawdę zapamiętuje liczbę reprezentującą ten znak. Istnieje tabela zawierająca sparowane liczby oraz znaki, zwana **tabelą ASCII**, która pokazuje, które liczby odpowiadają danym znakom. Kiedy program drukuje znak, nie wyświetla liczby, tylko odszukuje w tabeli ASCII znak, który powinien zostać pokazany[3].

Mały sekret liczb zmiennoprzecinkowych

Chciałbym Ci coś wyjawić na temat liczb zmiennoprzecinkowych, takich jak `float` albo `double`. Z pewnością są one bardzo praktyczne, ponieważ mogą przyjmować szeroki zakres wartości. Największa liczba, którą może przedstawić typ `double`, to około $1{,}8 \times 10^{308}$ — jest to liczba z 308 zerami na końcu. Jednak typ ten zajmuje tylko 8 bajtów; czy to oznacza, że może on przechowywać tylko 2^{32} (18 446 744 073 709 551 616) możliwych wartości? Taka liczba ma mnóstwo zer, ale nie jest ich 308.

No właśnie! W rzeczywistości typ `double` może reprezentować około 18 trylionów liczb. To bardzo dużo — tak dużo, że aż musiałem sprawdzić, jak nazywa się liczba z 18 zerami. Nadal nie jest to jednak 308 zer. Liczby zmiennoprzecinkowe pozwalają na dokładne przedstawienie tylko niektórych wartości, które są objęte ich zakresem, przez użycie formatu podobnego do notacji naukowej.

W notacji naukowej liczby są zapisywane w postaci $\mathbf{x} \times 10^{y}$. Składnik *x* przechowuje zazwyczaj kilka pierwszych cyfr liczby, natomiast *y* — nazywany **wykładnikiem** — to potęga, do której należy podnieść liczbę. Na przykład odległość Ziemi od Słońca można zapisać jako $1{,}496 \times 10^{8}$ kilometrów (około 150 milionów kilometrów).

[3] Poważnym zaniedbaniem z mojej strony byłoby pominięcie informacji, że tabela ASCII jest raczej mała — zawiera tylko 256 pozycji. Oznacza to, że nie nadaje się ona do pracy z takimi językami jak chiński albo japoński, które dysponują więcej niż 256 znakami. Obejście tego problemu wiąże się z wprowadzeniem koncepcji Unicode, która to tematyka wykracza poza zakres niniejszej książki. Więcej informacji na ten temat znajdziesz na stronie *http://www.cprogramming.com/tutorial/unicode.html*.

Kup książkę Poleć książkę

Dzięki umieszczeniu dużych wartości w wykładniku taki sposób zapisu pozwala komputerowi na przechowywanie naprawdę dużych liczb. Część niewykładnicza nie może jednak zawierać 300 cyfr, tylko mniej więcej 15. W związku z tym, kiedy pracujesz z liczbami zmiennoprzecinkowymi, masz do dyspozycji tylko 15 cyfr **znaczących**. Jeśli masz do czynienia z względnie małymi wartościami, wówczas różnica między liczbą przechowywaną przez komputer a faktyczną liczbą będzie bardzo niewielka. Jeżeli jednak pracujesz z dużymi wartościami, to błąd bezwzględny także będzie spory, nawet jeśli błąd względny będzie mały. Na przykład przy dwóch liczbach znaczących mógłbym napisać, że Ziemia znajduje się w odległości $1{,}5 \times 10^8$ kilometrów od Słońca. Względnie jest to całkiem dobra wartość (błąd jest mniejszy niż 0,1%), ale w mierze bezwzględnej jest to ponad 70 tysięcy kilometrów — prawie dwa razy tyle, ile wynosi obwód Ziemi! Błąd taki ma miejsce oczywiście wtedy, gdy zakładamy użycie dwóch cyfr znaczących. Przyjmując 15 cyfr znaczących, możemy przybliżyć się do liczb tak małych jak na przykład milion.

W większości przypadków niedokładności liczb zmiennoprzecinkowych nie będą mieć większego znaczenia, chyba że przeprowadzasz poważne obliczenia matematyczne lub naukowe.

Mały sekret liczb całkowitych

Liczby całkowite również mają swój mały sekret. Faktem jest, że liczby całkowite oraz zmiennoprzecinkowe nie mają ze sobą zbyt wiele wspólnego. Liczby całkowite, w przeciwieństwie do zmiennoprzecinkowych, przechowują dokładnie taką wartość, jaka zostanie w nich zapisana, ale szczerze nienawidzą przecinka dziesiętnego. Kiedy przeprowadzasz obliczenia na liczbach całkowitych, a wynikiem nie będzie liczba całkowita, rezultat zostanie obcięty. Część całkowita będzie dokładna, ale część dziesiętna zostanie odrzucona.

Prawdopodobnie nie zaliczyłbyś żadnego matematycznego testu, gdybyś napisał, że 5/2 = 2, ale właśnie taki wynik poda komputer! Jeśli potrzebujesz uzyskać odpowiedź z miejscami dziesiętnymi, powinieneś użyć typu niecałkowitego.

Kiedy w swoim programie zapisujesz liczby, kompilator przyjmuje, że są one całkowite, i stąd 5/2 daje w wyniku 2. Jeśli umieścisz w liczbie przecinek dziesiętny — na przykład 5,0/2,0 — kompilator zinterpretuje taką operację jako działanie na liczbach całkowitych i w rezultacie poda wynik, którego oczekujesz: 2,5.

Sprawdź się

1. Jakiego typu zmiennej powinieneś użyć, kiedy chcesz zapisać taką liczbę jak 3,1415?

A. `int`
B. `char`
C. `double`
D. `string`

2. Który z poniższych zapisów przedstawia poprawny operator służący do porównywania dwóch zmiennych?

A. `:=`
B. `=`
C. `equal`
D. `==`

Kup książkę Poleć książkę

3. W jaki sposób można uzyskać dostęp do danych typu `string`?
   A. Typ `string` jest wbudowany w język, tak więc nie trzeba nic robić.
   B. Ponieważ typ `string` jest używany podczas operacji wejścia-wyjścia, należy dołączyć plik nagłówkowy `iostream`.
   C. Należy dołączyć plik nagłówkowy `string`.
   D. C++ nie obsługuje łańcuchów tekstowych

4. Który z poniższych typów nie jest poprawnym typem zmiennej?
   A. `double`
   B. `real`
   C. `int`
   D. `char`

5. W jaki sposób można wczytać cały wiersz wprowadzony przez użytkownika?
   A. Za pomocą `cin>>`.
   B. Za pomocą `readline`.
   C. Za pomocą `getline`.
   D. Nie da się tego zrobić w prosty sposób.

6. Co zostanie wyświetlone na ekranie w wyniku wykonania następującej instrukcji w C++: `cout << 1234/2000`?
   A. 0
   B. 0,617
   C. W przybliżeniu 0,617, ale dokładny wynik nie może być zapisany w liczbie zmiennoprzecinkowej.
   D. To zależy od typów znajdujących się z obu stron równania.

7. Dlaczego w C++ jest potrzebny typ `char`, skoro istnieją już typy całkowite?
   A. Ponieważ znaki i liczby całkowite są zupełnie różnymi rodzajami danych; pierwsze z nich to litery, a drugie to liczby.
   B. Ze względu na wsteczną zgodność z C.
   C. Aby łatwiej było wczytywać oraz wyświetlać znaki zamiast liczb, ponieważ znaki są w rzeczywistości przechowywane jako liczby.
   D. Ze względu na wsparcie wielojęzykowości, aby możliwa była obsługa takich języków jak chiński albo japoński, w których występuje wiele znaków.

Odpowiedzi znajdują się na końcu książki.

Zadania praktyczne

1. Napisz program, który wyświetli Twoje imię.

2. Napisz program, który wczyta dwie liczby i zsumuje je.

3. Napisz program, który wykonuje dzielenie na dwóch liczbach podanych przez użytkownika i wyświetla dokładny wynik tego działania. Nie zapomnij przetestować swojego programu zarówno dla liczb całkowitych, jak i dziesiętnych.

Kup książkę Poleć książkę

Skorowidz

Kup książkę Poleć książkę

E

F

G

H

I

J

K

Kup książkę Poleć książkę

L

Ł

M

N

O

Kup książkę Poleć książkę

P

R

S

Kup książkę Poleć książkę

Ś

T

U

V

W

Kup książkę Poleć książkę

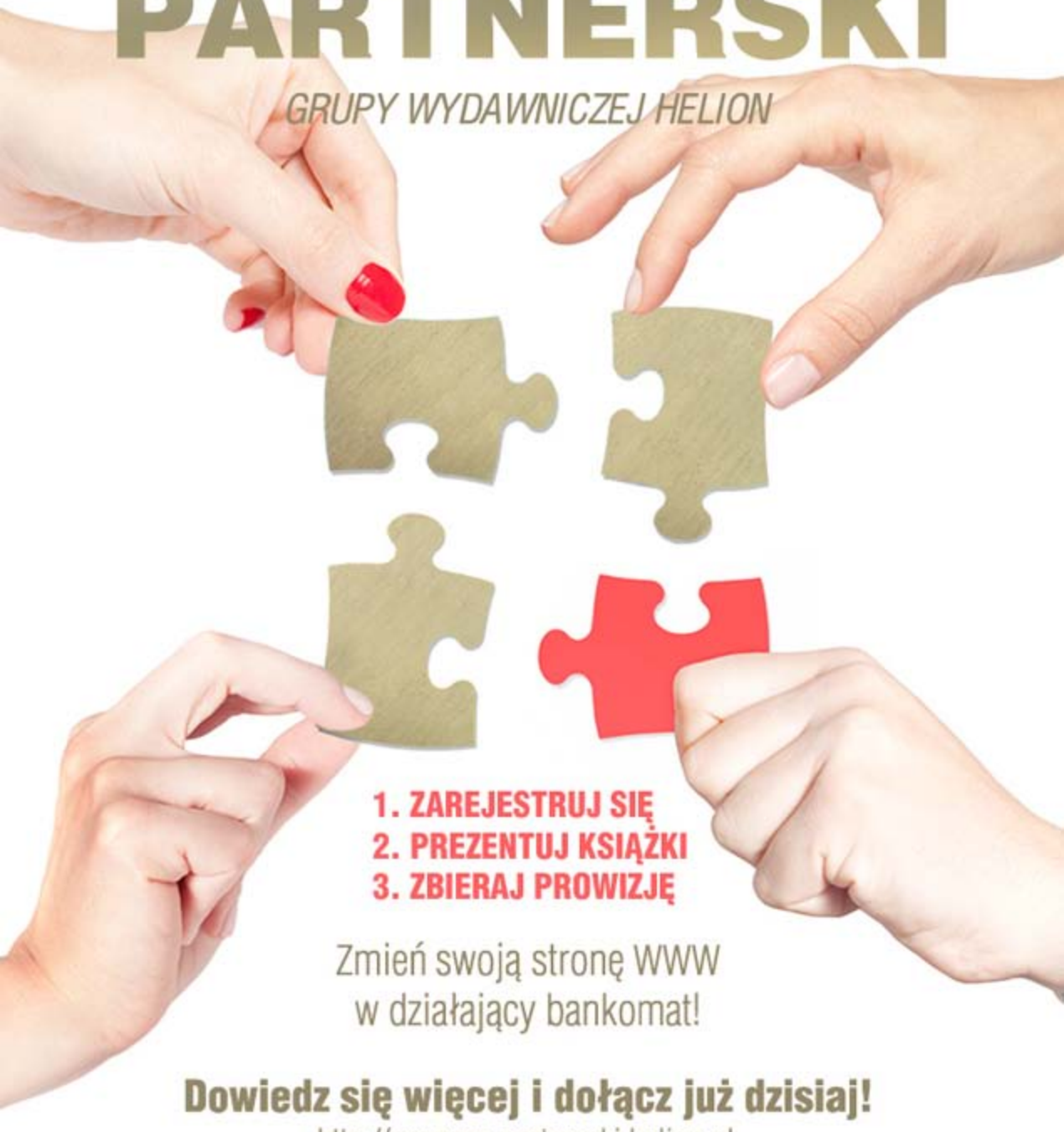
X

Z

Kup książkę Poleć książkę

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl

GRUPA WYDAWNICZA
Helion SA

JUŻ DZIŚ NAUCZ SIĘ PROGRAMOWAĆ!

Pomimo ogromnej konkurencji ze strony Javy oraz platformy .NET język C++ wciąż jest niezastąpiony w wielu zadaniach. Sprawdza się wyśmienicie, gdy konieczna jest pełna kontrola nad sprzętem oraz możliwie najwyższa wydajność. Jeżeli chcesz się przekonać, jak wykorzystać jego potencjał, trafiłeś na doskonałą książkę!

Opanowanie tego języka nie jest tak trudne, jak myślisz. Dzięki temu podręcznikowi bez problemu rozpoczniesz przygodę z C++. W trakcie lektury przygotujesz Twoje środowisko pracy (niezależnie od tego, czy korzystasz z Linuksa, Mac OS, czy Windowsa), poznasz składnię języka i jego elementy. Z kolejnych rozdziałów dowiesz się, jak deklarować zmienne i przechowywać w nich dane oraz jak używać instrukcji warunkowych. Część druga tej książki została poświęcona zagadnieniom związanym z pracą z danymi. Poznasz możliwości tablic, struktur oraz popracujesz ze wskaźnikami. Na koniec zajmiesz się technikami programowania obiektowego oraz debugowaniem kodu. To świetny podręcznik, dzięki któremu nawet laik może zacząć programować w C++. Warto spróbować!

Dzięki tej książce:

- przygotujesz środowisko pracy
- poznasz składnię i elementy języka C++
- opanujesz tablice, struktury i wskaźniki
- poznasz techniki programowania obiektowego
- stworzysz swój pierwszy program w C++

helion.pl
księgarnia internetowa

Helion

Nr katalogowy: 19659

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ

KOD KORZYŚCI

ISBN 978-83-246-8920-0

cena: 67,00 zł

9 788324 689200