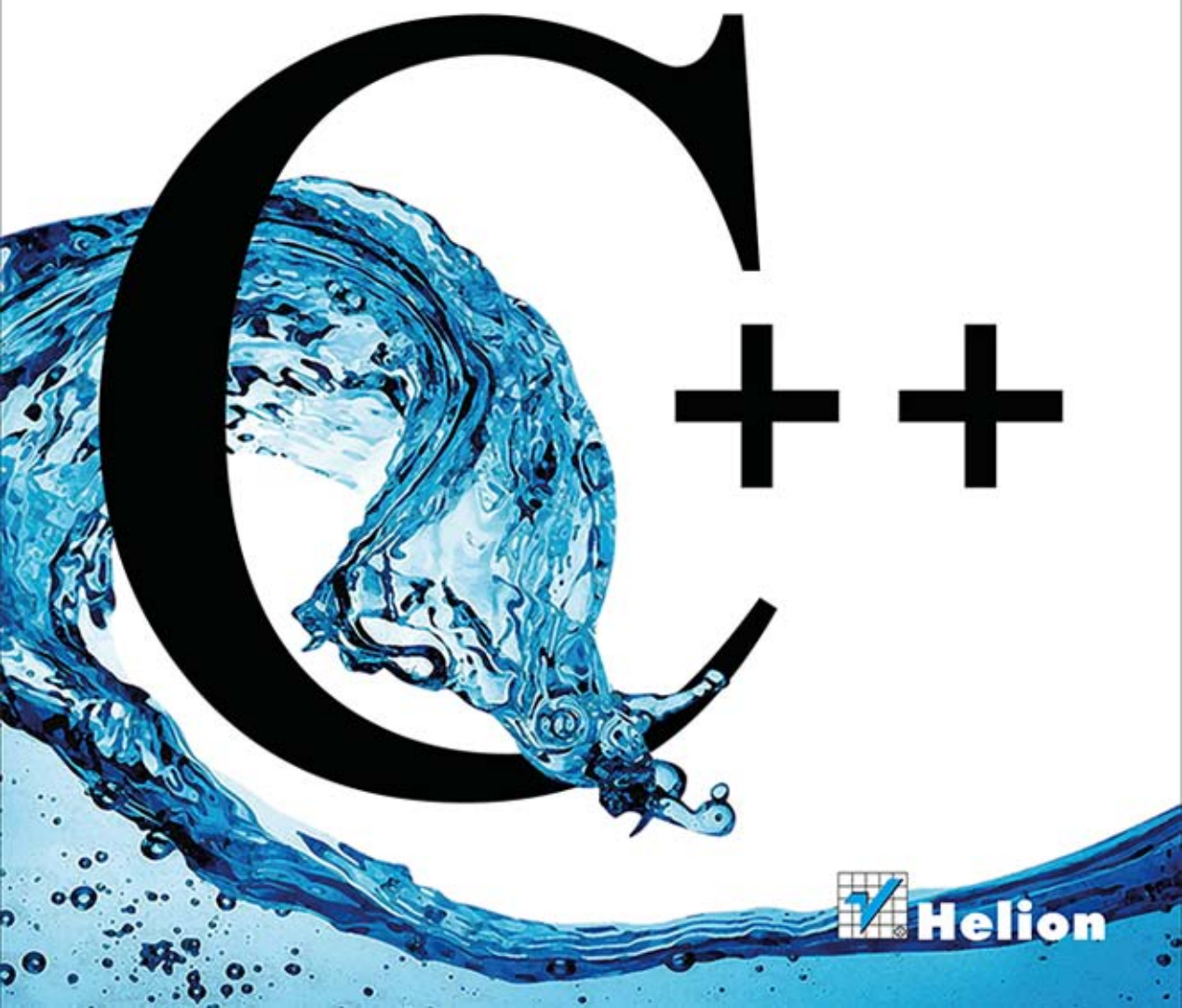


ALEX ALLAIN

JUŻ DZIŚ NAUCZ SIĘ PROGRAMOWAĆ!

PRZEWODNIK DLA POCZĄTKUJĄCYCH



Tytuł oryginału: Jumping into C++

Tłumaczenie: Ireneusz Jakóbiak

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 jakiegokolwiek 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

Część I. Wskocz w C++ 13

Podziękowania 14

Rozdział 1. Wprowadzenie. Konfiguracja środowiska programistycznego 15

Czym jest język programowania? 15

Słyszałem o języku, który nazywa się C. Jaka jest różnica między nim a C++? 15

Czy powinienem znać C, aby nauczyć się C++? 16

Czy żeby zostać programistą, muszę znać matematykę? 16

Terminologia 16

 Programowanie 16

 Plik wykonywalny 16

Edycja i kompilowanie plików źródłowych 17

Uwaga na temat przykładowych kodów źródłowych 17

Windows 17

 Krok 1. Pobierz Code::Blocks 18

 Krok 2. Zainstaluj Code::Blocks 18

 Krok 3. Uruchom Code::Blocks 18

 Rozwiązywanie problemów 21

 Czym właściwie jest Code::Blocks? 23

Macintosh 23

 Xcode 23

 Instalowanie Xcode 24

 Uruchamianie Xcode 24

 Tworzenie pierwszego programu C++ w Xcode 24

 Instalowanie Xcode 4 28

 Uruchamianie Xcode 28

 Tworzenie pierwszego programu C++ w Xcode 28

 Rozwiązywanie problemów 31

Linux 33

 Krok 1. Instalowanie g++ 33

 Krok 2. Uruchomienie g++ 34

 Krok 3. Uruchomienie programu 34

 Krok 4. Konfigurowanie edytora tekstowego 35

 Konfigurowanie nano 35

 Korzystanie z nano 36

Rozdział 2. Podstawy C++	39
Wprowadzenie do języka C++	39
Najprostszy program w C++	39
Co się dzieje, jeżeli nie możesz zobaczyć swojego programu?	41
Podstawowa struktura programu w C++	42
Komentowanie programów	42
Specyfika myślenia programisty. Tworzenie kodu wielokrotnego użycia	43
Kilka słów na temat radości i bólu praktyki	44
Sprawdź się	44
Zadania praktyczne	45
Rozdział 3. Interakcja z użytkownikiem. Zapisywanie informacji w zmiennych	47
Deklarowanie zmiennych w C++	47
Korzystanie ze zmiennych	48
Co zrobić, gdy program błyskawicznie kończy działanie?	48
Zmiana wartości zmiennych oraz ich porównywanie	49
Skrócone zapisy na dodawanie i odejmowanie jedynki	50
Poprawne i niepoprawne użycie zmiennych	51
Najczęściej popełniane błędy podczas deklarowania zmiennych w C++	51
Rozróżnianie wielkości liter	52
Nazwy zmiennych	53
Przechowywanie łańcuchów tekstowych	53
No dobrze, rozumiem już łańcuchy tekstowe, ale co z pozostałymi typami?	55
Mały sekret liczb zmiennoprzecinkowych	56
Mały sekret liczb całkowitych	57
Sprawdź się	57
Zadania praktyczne	58
Rozdział 4. Instrukcje warunkowe	59
Podstawowa składnia instrukcji if	59
Wyrażenia	60
Czym jest prawda?	61
Typ bool	61
Instrukcja else	62
Instrukcje else-if	62
Porównywanie łańcuchów tekstowych	63
Więcej interesujących warunków budowanych za pomocą operatorów logicznych	64
Logiczne nie	64
Logiczne ORAZ	65
Logiczne LUB	65
Łączenie wyrażień	66
Przykładowe wyrażenia logiczne	67
Sprawdź się	67
Zadania praktyczne	68

Rozdział 5. Pętle	69
Pętla while	69
Najczęściej popełniany błąd	70
Pętla for	70
Inicjalizacja zmiennej	71
Warunek pętli	71
Aktualizacja zmiennej	71
Pętla do-while	72
Kontrolowanie przebiegu pętli	73
Pętle zagnieżdżone	75
Wybór właściwego rodzaju pętli	76
Pętla for	76
Pętla while	76
Pętla do-while	77
Sprawdź się	78
Zadania praktyczne	78
Rozdział 6. Funkcje	81
Składnia funkcji	81
Zmienne lokalne i zmienne globalne	83
Zmienne lokalne	83
Zmienne globalne	84
Ostrzeżenie dotyczące zmiennych globalnych	85
Przygotowanie funkcji do użycia	86
Definicja i deklaracja funkcji	86
Przykład użycia prototypu funkcji	87
Rozbijanie programu na funkcje	88
Kiedy wciąż na nowo powtarzasz ten sam kod	88
Kiedy chcesz, żeby kod był łatwiejszy do czytania	88
Nazywanie i przeładowywanie funkcji	89
Podsumowanie wiadomości o funkcjach	89
Sprawdź się	90
Zadania praktyczne	90
Rozdział 7. Instrukcje switch case oraz typ wyliczeniowy	91
Porównanie instrukcji switch case z if	93
Tworzenie prostych typów za pomocą wyliczeń	94
Sprawdź się	96
Zadania praktyczne	96
Rozdział 8. Dodawanie do programu elementu losowości	97
Uzyskiwanie liczb losowych w C++	98
Błędy i losowość	100
Sprawdź się	101
Zadania praktyczne	102

Rozdział 9. Co zrobić, kiedy... nie wiesz, co robić?	103
Krótka dygresja na temat wydajności i bezpieczeństwa kodu	106
Co robić, kiedy nie znasz algorytmu?	107
Zadania praktyczne	110
Część II. Praca z danymi	111
Rozdział 10. Tablice	113
Podstawowa składnia tablic	113
Przykładowe zastosowania tablic	114
Przechowywanie zamówień w tablicach	114
Odwzorowanie siatek w tablicach wielowymiarowych	115
Korzystanie z tablic	115
Tablice i pętle	115
Przekazywanie tablic do funkcji	116
Wypadnięcie poza ostatni element tablicy	118
Sortowanie tablic	118
Sprawdź się	122
Zadania praktyczne	123
Rozdział 11. Struktury	125
Wiązanie wielu wartości	125
Składnia	125
Przekazywanie struktur	127
Sprawdź się	129
Zadania praktyczne	130
Rozdział 12. Wprowadzenie do wskaźników	131
Zapomnij o wszystkim, co do tej pory słyszałeś	131
No dobrze, czym są wskaźniki? Dlaczego powinny mnie obchodzić?	131
Czym jest pamięć komputera?	132
Zmienne a adresy	133
Uwaga na temat nazewnictwa	133
Organizacja pamięci	134
Nieprawidłowe wskaźniki	135
Pamięć i tablice	136
Pozostałe zalety i wady wskaźników	136
Sprawdź się	137
Zadania praktyczne	137
Rozdział 13. Korzystanie ze wskaźników	139
Składnia wskaźników	139
Deklarowanie wskaźnika	139

Otrzymywanie adresu zmiennej za pomocą wskaźnika	140
Użycie wskaźnika	140
Niezainicjalizowane wskaźniki i wartość NULL	143
Wskaźniki i funkcje	144
Referencje	146
Referencje a wskaźniki	147
Sprawdź się	148
Zadania praktyczne	148
Rozdział 14. Dynamiczna alokacja pamięci	151
Pozyskiwanie pamięci za pomocą instrukcji new	151
Brak pamięci	152
Referencje i dynamiczna alokacja	152
Wskaźniki i tablice	152
Tablice wielowymiarowe	155
Arytmetyka wskaźników	155
Zrozumieć tablice dwuwymiarowe	156
Wskaźniki do wskaźników	157
Wskaźniki do wskaźników i tablic dwuwymiarowych	159
Oswajanie wskaźników	160
Sprawdź się	160
Zadania praktyczne	161
Rozdział 15. Wprowadzenie do struktur danych: listy powiązane	163
Wskaźniki i struktury	165
Tworzenie listy powiązanej	166
Pierwszy przebieg	167
Drugi przebieg	168
Przeglądanie listy powiązanej	169
Oswajanie list powiązanych	170
Tablice a listy powiązane	171
Sprawdź się	173
Zadania praktyczne	175
Rozdział 16. Rekurencja	177
Jak postrzegać rekurencję?	177
Rekurencja i struktury danych	179
Pętle i rekurencja	181
Stos	183
Zaleta stosu	185
Wady rekurencji	185
Debugowanie przepełnienia stosu	186
Wydajność	188
Oswajanie rekurencji	188
Sprawdź się	188
Zadania praktyczne	189

Rozdział 17. Drzewa binarne	191
Konwencje nazewnictwa	193
Implementacja drzew binarnych	194
Wstawianie węzła do drzewa	194
Przeszukiwanie drzewa	196
Niszczenie drzewa	197
Usuwanie węzła z drzewa	199
Praktyczne zastosowanie drzew binarnych	206
Koszt tworzenia drzew i map	207
Sprawdź się	208
Zadania praktyczne	208
Rozdział 18. Standardowa biblioteka szablonów	211
Wektor — tablica o zmiennych rozmiarach	212
Przekazywanie wektorów do metod	213
Inne właściwości wektorów	213
Mapy	214
Iteratory	215
Sprawdzanie, czy wartość znajduje się w mapie	217
Oswajanie biblioteki STL	218
Więcej informacji o STL	219
Sprawdź się	219
Zadania praktyczne	220
Rozdział 19. Więcej o łańcuchach tekstowych	221
Wczytywanie łańcuchów tekstowych	221
Długość łańcucha i dostęp do jego elementów	223
Wyszukiwanie i podłańcuchy	224
Przekazywanie łańcucha przez referencję	225
Szerzenie się const	226
Const i STL	227
Sprawdź się	228
Zadania praktyczne	229
Rozdział 20. Debugowanie w Code::Blocks	231
Zaczynamy	232
Wstrzymywanie działania programu	233
Debugowanie awarii	239
Zagłębienie do zawieszonych programów	242
Modyfikowanie zmiennych	245
Podsumowanie	246
Zadania praktyczne	246
Zadanie nr 1. Problem z wykładnikiem	246
Zadanie nr 2. Problem z dodawaniem liczb	247
Zadanie nr 3. Problem z ciągiem Fibonacciego	247
Zadanie nr 4. Problem z odczytywaniem i wyświetlaniem listy	248

Część III. Tworzenie większych programów	249
Rozdział 21. Rozbijanie programów na mniejsze części	251
Proces kompilacji w języku C++	251
Przetwarzanie wstępne	252
Kompilacja	253
Konsolidacja	253
Dlaczego kompilacja i konsolidacja przebiegają oddzielnie?	254
Jak rozbić program na wiele plików?	254
Krok 1. Oddzielanie deklaracji od definicji	255
Krok 2. Określenie, które funkcje powinny być wspólne	255
Krok 3. Przeniesienie wspólnych funkcji do nowych plików	255
Przykładowy program	256
Pozostałe zasady pracy z plikami nagłówkowymi	259
Praca z wieloma plikami źródłowymi w środowisku programistycznym	260
Sprawdź się	262
Zadania praktyczne	264
Rozdział 22. Wprowadzenie do projektowania programów	265
Powielony kod	265
Założenia dotyczące przechowywania danych	266
Projekt i komentarze	268
Sprawdź się	269
Rozdział 23. Ukrywanie reprezentacji struktur danych	271
Użycie funkcji w celu ukrycia układu struktury	272
Deklaracja metody i składnia wywołania	273
Przeniesienie definicji funkcji poza strukturę	274
Sprawdź się	275
Zadania praktyczne	275
Rozdział 24. Klasa	277
Ukrywanie sposobu przechowywania danych	278
Deklarowanie instancji klasy	279
Odpowiedzialności klasy	280
Co tak naprawdę znaczy private?	281
Podsumowanie	281
Sprawdź się	281
Zadania praktyczne	282
Rozdział 25. Cykl życia klasy	283
Konstruowanie obiektu	283
Co się stanie, jeśli nie utworzysz konstruktora?	285
Inicjalizacja składowych klasy	286
Użycie listy inicjalizacyjnej do pól stałych	287

Niszczenie obiektu	288
Niszczenie podczas usuwania	290
Niszczenie przy wyjściu poza zakres	290
Niszczenie przez inny destruktor	291
Kopiowanie klas	291
Operator przypisania	292
Konstruktor kopiujący	295
Pełna lista metod generowanych przez kompilator	296
Całkowite zapobieganie kopiowaniu	296
Sprawdź się	297
Zadania praktyczne	298
Rozdział 26. Dziedziczenie i polimorfizm	299
Dziedziczenie w C++	300
Pozostałe zastosowania oraz nieprawidłowe użycia dziedziczenia	304
Dziedziczenie, konstruowanie obiektów oraz ich niszczenie	304
Polimorfizm i dziedziczenie obiektów	306
Problem przycinania	308
Dzielenie kodu z podklasami	309
Dane chronione	309
Dane obejmujące całą klasę	310
W jaki sposób zaimplementowany jest polimorfizm?	311
Sprawdź się	313
Zadania praktyczne	314
Rozdział 27. Przestrzenie nazw	317
Kiedy stosować instrukcję using namespace	319
Kiedy należy utworzyć przestrzeń nazw?	319
Sprawdź się	320
Zadania praktyczne	320
Rozdział 28. Plikowe operacje wejścia-wyjścia	321
Podstawy plikowych operacji wejścia-wyjścia	321
Czytanie z plików	321
Formaty plików	323
Koniec pliku	324
Zapisywanie plików	326
Tworzenie nowych plików	327
Pozycja pliku	327
Pobieranie argumentów z wiersza poleceń	330
Obsługa argumentów liczbowych	332
Pliki binarne	332
Praca z plikami binarnymi	334
Konwersja na typ char*	335
Przykład binarnych operacji we/wy	335

Przechowywanie klas w pliku	336
Czytanie z pliku	338
Sprawdź się	340
Zadania praktyczne	341
Rozdział 29. Szablony w C++	343
Szablony funkcji	343
Inferencja typów	345
Kacze typowanie	345
Szablony klas	346
Wskazówki dotyczące pracy z szablonami	347
Szablony i pliki nagłówkowe	349
Podsumowanie informacji o szablonach	349
Interpretacja komunikatów o błędach w szablonach	350
Sprawdź się	353
Zadania praktyczne	354
Części IV. Zagadnienia rozmaite	355
Rozdział 30. Formatowanie danych wyjściowych za pomocą iomanip	357
Rozwiązywanie problemów związanych z odstępami	357
Określanie szerokości pola za pomocą instrukcji setw	357
Zmiana znaku dopełniającego	358
Trwała zmiana ustawień	358
Korzystanie ze znajomości iomanip	359
Wyświetlanie liczb	360
Określanie precyzji wyświetlanych liczb za pomocą instrukcji setprecision	361
A co z pieniędzmi?	361
Wyświetlanie liczb o różnych podstawach	362
Rozdział 31. Wyjątki i raportowanie błędów	363
Zwalnianie zasobów po wystąpieniu wyjątku	364
Ręczne czyszczenie zasobów w bloku catch	365
Zgłaszanie wyjątków	366
Specyfikacja wyjątków	367
Korzyści płynące z wyjątków	368
Nieprawidłowe użycie wyjątków	369
Podsumowanie informacji o wyjątkach	370
Rozdział 32. Końcowe przemyślenia	371
Rozwiązanie testu z rozdziału 2.	372
Rozwiązanie testu z rozdziału 3.	373
Rozwiązanie testu z rozdziału 4.	374

Rozwiązanie testu z rozdziału 5.	374
Rozwiązanie testu z rozdziału 6.	375
Rozwiązanie testu z rozdziału 7.	375
Rozwiązanie testu z rozdziału 8.	376
Rozwiązanie testu z rozdziału 10.	377
Rozwiązanie testu z rozdziału 11.	377
Rozwiązanie testu z rozdziału 12.	378
Rozwiązanie testu z rozdziału 13.	379
Rozwiązanie testu z rozdziału 14.	380
Rozwiązanie testu z rozdziału 15.	381
Rozwiązanie testu z rozdziału 16.	382
Rozwiązanie testu z rozdziału 17.	383
Rozwiązanie testu z rozdziału 18.	384
Rozwiązanie testu z rozdziału 19.	385
Rozwiązanie testu z rozdziału 21.	385
Rozwiązanie testu z rozdziału 22.	386
Rozwiązanie testu z rozdziału 23.	386
Rozwiązanie testu z rozdziału 24.	387
Rozwiązanie testu z rozdziału 25.	387
Rozwiązanie testu z rozdziału 26.	389
Rozwiązanie testu z rozdziału 27.	390
Rozwiązanie testu z rozdziału 28.	391
Rozwiązanie testu z rozdziału 29.	391

Skorowidz	393
------------------------	------------

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 niezłe, 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.

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 `tojestliczba`. 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

pojane; na przykład 3,1415 zostanie odczytane jako 3). Po uruchomieniu programu spróbuj wpisać ciąg liter albo liczbę dziesiętną. Reakcja programu będzie zależała od wprowadzonych danych, ale w żadnym przypadku nie będzie pożądana. Na razie nie zajmujemy 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 cudzysłowu, 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 następującej 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.

*	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:


```
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 << " = " <<
        <math>pierwszy\_argument * drugi\_argument</math> << endl;
    cout << pierwszy_argument << " + " << drugi_argument << " = " <<
        <math>pierwszy\_argument + drugi\_argument</math> << endl;
    cout << pierwszy_argument << " / " << drugi_argument << " = " <<
        <math>pierwszy\_argument / drugi\_argument</math> << endl;
    cout << pierwszy_argument << " - " << drugi_argument << " = " <<
        <math>pierwszy\_argument - drugi\_argument</math> << 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.

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 zmienno-przecinkowym 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 pojawiają 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 zachowywał 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 nieznaną. 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 klawisz?

Rozróżnianie wielkości liter

Mamy teraz dobrą okazję do porozmawiania o kolejnym ważnym zagadnieniu, przez które łatwo możesz się pogubić. Jest nim **rozzróżnienie 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.

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 beużyteczne. Kiedy tworzysz program, myślisz, że kod, który piszesz, jest dość czytywisty — 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 danych¹.

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.

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.

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 (choćby użytkownik nadal musi naciśnąć *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 następczej 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 rozumiałeś 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

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 pokazany³.

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 $x \times 10^y$. 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>.

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

Liczyby 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 zaliczyłybyś 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. ==

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 `getline`.
 - 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 wielojęzyczności, 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.

Skorowidz

A

abstrakcja funkcyjna, 267–268
Access Violation, 240
adres pamięci, 132, 155
aktualizacja zmiennej, 71
algorytm, 103, 107–109
 mieszający, 206
alokacja pamięci, 135
argumenty
 funkcji, 41, 83
 w wierszu poleceń, 331
arytmetyka wskaźników, 156
automatyczne wcinanie tekstu, 17

B

bajt, 55
bit, 55
błędy
 kompilacji, 42
 kompilatora, 22
 podczas deklarowania zmiennych, 51–53

C

C++ a C, 15–16
ciało instrukcji, 59
Code::Blocks, 17, 23
cykl życia klasy, 283
czysty tekst, 17

D

dane
 wejściowe, 47
 wielowymiarowe, 115

debugger, 231
debugowanie, 246
 awarii, 239–241
 modyfikowanie zmiennych, 245
 przepełnienia, 186
 w Code::Blocks
 włamanie się do działającego programu, 243
 wstrzymywanie działania programu, 233–239
 zawieszenie programu, 242–245
definicja metody, 674
definiowanie funkcji, 86
deklarowanie
 funkcji, 87
 instancji klasy, 279
 klasy, 278
 metody, 273
 wskaźnika, 139–140
 zmiennych, 47
dekrementacja, 50
delimiter, 222
dereferencja wskaźnika, 141
destruktor, 288–291
 czyszczący po obiektach, 365
dołączanie do pliku, 326
drzewo
 binarne, 192–193
 zastosowanie, 206
 puste, 193
 zrównoważone, 192
dynamiczna alokacja pamięci, 151
dyrektywy preprocesora, 252
dziedziczenie, 300–306
 a konstruowanie obiektów i ich niszczenie,
 304–306
dzielenie
kodu z podklasami, 309
programu na wiele plików, 254–259

E

edytor, 17
endl, 41
enkapsulacja, 281
EOF, 325

F

fixup, 254
format pliku, 323
formatowanie danych wyjściowych, 357–362
 za pomocą iomanip, 357–362
funkcja, 39, 81
 atoi, 332
 getline, 54–55, 221–222
 main, 39
 rand, 98
 setf, 359
 setfill, 358
 setiosflags, 362
 setprecision, 361
 setw, 357
 srand, 98, 100

G

g++, 33
getter, 280
głowa listy, 168

H

hermetyzacja, 281

I

implementacja
 drzew binarnych, 194
 polimorfizmu, 311–313
include guard, 259
indeks tablicy, 114
inferencja typów, 345
inicjalizacja
 składowych klasy, 286–287
 zmiennej, 52
inicjowanie przy pozyskaniu zasobu, 291
inkrementacja, 50
input file stream, 321

instancja wersji szablonu, 343
instrukcja
 break, 73–74
 cin, 48
 cin.get (), 41, 48
 cin.ignore (), 48
 continue, 74
 cout, 41
 delete, 289–290
 else, 62
 else-if, 62–63
 include, 40
 new, 151, 153
 switch case, 91–94
 using, 319
 using namespace std, 40, 317, 319
 warunkowa if, 59–60
interfejs użytkownika, 279
iterator, 215–217
iterowanie, 169

J

język programowania, 15

K

kacze typowanie, 345–346
katalog roboczy programu, 322
klasa, 277–281, 283
 czysto wirtualna, 301
 fstream, 328
 string, 53, 223
kod źródłowy, 16
komentarze, 42, 268
komentowanie kodu, 268
komentowanie programów, 42
kompilacja, 251, 253
kompilator, 17
komunikaty o błędach w szablonach, 350–353
konfiguracja środowiska programistycznego
 Linux, 33–38
 Macintosh, 23–33
 Windows, 17–28
konsolidacja, 253–254
konstruktor, 283–285
 klasy ofstream, 327
 kopiujący, 295–296

konstruowanie obiektu, 283–285
 kontrakt funkcji, 180
 kontrolowanie przebiegu pętli, 73–74
 kopiowanie klas, 291–292
 korzystanie ze zmiennych, 48
 koszt tworzenia drzew i map, 207

L

liczby

całkowite, 57
 losowe w C++, 98–100
 a błędy, 100
 pseudolosowe, 97
 zmiennoprzecinkowe, 56

licznik pętli, 71

lista

inicjalizacyjna, 287
 metod generowanych przez kompilator, 296
 powiązana, 166, 170, 179
 a tablica, 171–173

lukier składniowy, 156

ł

łańcuchy tekstowe, 53–55, 221
 wczytywanie, 221–223

M

makro, 252
 mapa, 206–207, 218
 mapa (STL), 214–215
 metoda, 213, 272–275
 chroniona, 310
 cin, 221
 clear, 325
 eof, 325
 fail, 325
 fill, 359
 find, 224
 find (STL), 217
 is_open, 322
 length, 223
 prywatna, 309
 publiczna, 309
 push_back, 214
 read, 338
 rfind, 224

seekg, 328
 seekp, 328
 size, 223
 statyczna, 310–311
 substr, 224
 tellg, 327
 tellp, 327
 w STL, 215
 wirtualna, 301, 312
 write, 334, 336
 modyfikatory dostępu, 278, 309

N

nadklasa, 300
 nano, 35–38
 nawiasy klamrowe, 40
 nazwy zmiennych, 53
 nazywanie funkcji, 89
 niszczenie
 drzewa, 197–199
 obektu, 288–291

O

obiekt, 279
 obiekt cout, 41
 obsługa argumentów liczbowych, 332
 odczytywanie informacji z plików, 321–326
 oddzielna kompilacja, 254
 odpowiedzialności klasy, 280
 odwijanie stosu, 365
 ojciec w drzewie binarnym, 192
 operacja dzielenia modulo, 99
 operator
 adresu, 140
 boolowski, 64
 dekrementacji, 50
 inkrementacji, 50
 logiczny, 64
 kolejność wykonywania działań, 66–67
 LUB, 65
 łączenie wyrażeń, 66
 negacja, 64
 ORAZ, 65
 odejmowania, mnożenia i dzielenia, 50
 porównania, 49
 kolejność wykonywania działań, 67

operator
 przypisania, 49, 292–295
 relacyjny, 61
 umożliwiający dodanie dowolnej wartości
 do zmiennej, 50
 wstawiania, 41
 organizacja pamięci, 134
 otrzymywanie adresu zmiennej
 za pomocą wskaźnika, 140
 output file stream, 321

P

pamięć
 dostępna, 134
 komputera, 132–133
 pętla, 69
 do-while, 72–73, 77
 for, 70–72, 76
 nieskończona, 69
 wewnętrzna, 75
 while, 69–70, 76
 zagnieżdżona, 75–76
 zewnętrzna, 75
 piksel, 265
 plik
 binarny, 332–340
 nagłówkowy, 258–259, 321
 obiektowy, 253
 tekstowy, 333
 wykonywalny, 16
 plikowe operacje wejścia-wyjścia, 321
 płytka kopia wskaźnika, 292
 pobieranie argumentów z wiersza poleceń, 330–332
 poddrzewa, 192
 podklasa, 300
 polimorfizm, 303, 311–313
 a dziedziczenia obiektów, 306–308
 porównywanie łańcuchów tekstowych, 63–64
 pozycja pliku, 327–330
 praca z wieloma plikami źródłowymi
 Code::Blocks, 260
 g++, 261
 Xcode, 261–262
 w środowisku programistycznym, 260
 preprocesor, 252–253
 private, 278, 281
 programowanie, 16
 rozwiązywanie problemów, 103–109

projektowanie
 od dołu do góry, 107
 od góry do dołu, 107
 protected, 309
 prototyp funkcji, 86–88
 przeciążanie funkcji, 89
 przeglądanie listy powiązanej, 169–170
 przekazywanie łańcucha przez referencję, 225–226
 przekazywanie
 struktur, 127–129
 tablic do funkcji, 116–117
 wektorów do metod, 213
 przeładowywanie funkcji, 89
 przepelnienie stosu, 186–187
 przestrzeń nazw, 317–320
 przesunięcie, 114
 przeszukiwanie drzewa, 196–197
 przetwarzanie wstępne, 252–253
 przycinanie, 308–309
 przypadek bazowy funkcji, 178
 public, 278

R

RAII, 291
 ramka stosu, 183–184
 raportowanie błędów, 363–370
 referencja, 146–147
 a dynamiczna alokacja pamięci, 152
 rekurencja, 177, 188
 a pętli, 181–183, 188
 a struktury danych, 179–181
 ogonowa, 182
 wzajemna, 187
 zalety i wady, 185
 rekursja, 177
 rozróżnienie wielkości liter, 52
 rzutowanie
 reinterpret_cast, 335
 static_cast, 335
 typu zmiennej, 335

S

separator, 222
 setter, 280
 składnia
 C++, 39–43
 funkcji, 81–82

struktury, 125–127
 tablic, 113–114
 wektora, 212
 wskaźników, 139–140
 składowa
 klasy, 278
 prywatna, 278–279
 publiczna, 278–279
 skrócone wartościowanie, 65
 słowo kluczowe
 const, 225–228
 delete, 151
 enum, 94
 false, 61
 new, 151, 285
 sizeof, 156
 template, 344
 true, 61
 typename, 344
 sortowanie
 przez wstawianie, 122
 tablic, 118–122
 specyfikacja wyjątków, 367
 sprawdzanie, czy wartość znajduje się w mapie, 217
 standardowa biblioteka szablonów, Patrz STL
 sterta, 134
 STL, 211, 218–219, 343
 stos, 134, 183–185
 strażnik include, 259
 struktura, 125, 146
 struktura danych, 163
 a wskaźniki, 165
 strumień, 321
 symbole debugowania, 232
 synowie w drzewie binarnym, 192
 szablony, 343, 347–350
 funkcji, 343–346
 klas, 346–347

Ś

średnik, 40

T

tabela
 ASCII, 56
 metod wirtualnych, 312

tablica, 113, 163
 a listy powiązane, 171–173
 a pamięć, 136
 a pętla, 115–116
 dwuwymiarowa, 115, 156
 wypadnięcie poza ostatni element tablicy, 118
 zastosowania, 114–115
 tworzenie
 instancji, 346
 listy powiązanej, 166–169
 nowych plików, 327
 typ danych
 ifstream, 321
 ofstream, 321, 326
 typ wyczeniowy, 94–95
 typy zmiennych, 47

U

ukrywanie sposobu przechowywania danych
 przy pomocy klas, 278
 usuwanie węzła z drzewa, 199–206
 użycie
 listy inicjalizacyjnej do pól stałych, 287
 wskaźnika, 140–143

V

vtable, 312

W

wartość NULL, 143, 152
 warunek pętli, 71
 wektor (STL), 212–214
 węzły drzewa binarnego, 192
 właściciel pamięci, 134
 wskaźnik, 131–133, 135–136, 139–143
 a funkcje, 144–146
 a referencja, 147
 a tablica, 152–155
 do wskaźników, 157–160
 do wskaźników i tablic dwuwymiarowych, 159
 this, 294
 wycieki pamięci, 134
 wyjątki, 363–370
 wykładnik, 56
 wykomentowanie kodu, 43

wyrażenie, 60
 fałszywe, 61
 prawdziwe, 61
wyrównywanie, 254
wyróżnianie składni, 17

X

Xcode, 23
Xcode 4, 28

Z

zagnieżdżanie przestrzeni nazw, 318
zakres zmiennej, 83
zapisywanie plików, 326
zapobieganie przed kopiowaniem, 296–297
zasięg zmiennej, 83

zastosowanie funkcji, 88
zgłaszanie wyjątków, 366–367
ziarno, 97
zintegrowane środowisko programistyczne, 17
zmiana wartości zmiennych, 49
zmienna, 47, 133
 globalna, 84–86
 lokalna, 83–84
 pętli, 71
 statyczna, 310–311
 typu bool, 61–62
 typu char, 47, 56
 typu double, 47, 56
 typu float, 55
 typu int, 47
zwalnianie pamięci, 151

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, traficie 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 deklorować 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



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>

sięgnij po WIĘCEJ



KOD KORBUSC

ISBN 978-83-246-8920-0



9 788324 689200

Cena: 67,00 zł

Informatyka w najlepszym wydaniu