

W PROSTOCIE TKWI SIŁA



wydanie VII

C++

dla
bystrzaków



Wprowadzenie
do języka C++

Programowanie
funkcjonalne

Używanie klas
nadrzędnych
i dziedziczenia

septem
septem.pl

Helion 

Stephen R. Davis

autor bestsellera *C# dla bystrzaków*

Tytuł oryginału: C++ For Dummies, 7th edition

Tłumaczenie: Michał Sternik

ISBN: 978-83-283-5987-1

Original English language edition Copyright © 2014 by John Wiley & Sons, Inc., Hoboken, New Jersey
All rights reserved including the right of reproduction in whole or in part in any form. This translation published by arrangement with John Wiley & Sons, Inc.

Oryginalne angielskie wydanie Copyright © 2014 by John Wiley & Sons, Inc., Hoboken, New Jersey
Wszelkie prawa, włączając prawo do reprodukcji całości lub części w jakiegokolwiek formie, zarezerwowane.
Tłumaczenie opublikowane na mocy porozumienia z John Wiley & Sons, Inc.

Translation copyright © 2019 by Helion S.A.

Wiley, the Wiley Publishing logo, For Dummies, Dla Bystrzaków, the Dummies Man logo, Dummies.com, Making Everything Easier, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and/or other countries. Used by permission.
All other trademarks are the property of their respective owners.

Wiley, the Wiley Publishing logo, For Dummies, Dla Bystrzaków, the Dummies Man logo, Dummies.com, Making Everything Easier, i związana z tym szata graficzna są markami handlowymi John Wiley & Sons, Inc. i/lub firm stowarzyszonych w Stanach Zjednoczonych i/lub innych krajach. Wykorzystywane na podstawie licencji.
Wszystkie pozostałe marki handlowe są własnością poszczególnych właścicieli.

Media and software compilation copyright © 2014 by John Wiley & Sons, Inc. All rights reserved.

Autor oraz HELION SA 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 HELION SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://dlabystrzakow.pl/user/opinie/cppby7>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/cppby7.zip>

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 231 22 19, 32 230 98 63

e-mail: dlabystrzakow@dlabystrzakow.pl

WWW: <http://dlabystrzakow.pl>

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

O autorze	15
Podziękowania od autora	17
Wprowadzenie	19

CZĘŚĆ I: PODSTAWY PROGRAMOWANIA W C++

25

ROZDZIAŁ 1: Pierwszy program w C++	27
Pojęcia dotyczące języka C++	28
Instalacja Code::Blocks	29
Windows	29
Ubuntu Linux	32
Macintosh	34
Tworzenie pierwszego programu w C++	37
Tworzenie projektu	38
Wprowadzanie kodu w języku C++	39
Korzystanie ze ściągi	41
Budowanie programu	42
Uruchamianie programu	43
Przeglądanie programu z komentarzami	44
Analiza struktury programów w C++	45
Opatrywanie kodu źródłowego komentarzami	45
Tworzenie programów z instrukcji C++	46
Deklaracja zmiennych	47
Generowanie wyjścia programu	48
Obliczanie wyrażeń	48
Przechowywanie wyników wyrażenia	48
Pozostała część programu Conversion	49

ROZDZIAŁ 2:	Stałe deklarowanie zmiennych	51
	Deklaracja zmiennych	52
	Deklarowanie różnych typów zmiennych	52
	Przegląd ograniczeń liczb całkowitych w C++	53
	Rozwiązywanie problemu zaokrągleń	54
	Ograniczenia liczb zmiennoprzecinkowych	55
	Deklarowanie typów zmiennych	57
	Typy stałych	58
	Zakres typów liczbowych	59
	Znaki specjalne	60
	Szerokie ładunki na autostradzie typu Char	61
	Czy te obliczenia są logiczne?	62
	Wyrażenia mieszane	63
	Deklaracje automatyczne	64
ROZDZIAŁ 3:	Operacje matematyczne	67
	Wykonywanie prostych obliczeń binarnych	68
	Dekomponowanie wyrażeń	69
	Określanie kolejności operacji	69
	Wykonywanie operacji jednoargumentowych	70
	Korzystanie z operatorów przypisania	72
ROZDZIAŁ 4:	Operacje logiczne	75
	Po co wykonujemy operacje logiczne?	76
	Korzystanie z prostych operatorów logicznych	76
	Przechowywanie wartości logicznych	77
	Używanie logicznych zmiennych typu int	79
	Zawiłości wykonywania operacji logicznych na zmiennych zmiennoprzecinkowych	79
	Wyrażanie liczb binarnych	81
	System liczb dziesiętnych	81
	Inne systemy liczbowe	82
	System liczb binarnych	82
	Wykonywanie bitowych operacji logicznych	84
	Operatory jednobitowe	84
	Używanie operatorów bitowych	86
	Prosty test	86

ROZDZIAŁ 5:	Sterowanie przepływem programu	89
	Sterowanie przepływem programu za pomocą instrukcji rozgałęzienia	90
	Wykonywanie pętli	92
	Wykonywanie pętli, gdy warunek jest prawdziwy	92
	Wykorzystanie autoinkrementacji i autodekrementacji	94
	Pętla for	95
	Unikanie budzącej strach pętli nieskończonej	98
	Pętla for zależna od zakresu	98
	Instrukcje zarządzania pętlami	99
	Zagnieżdżanie instrukcji sterujących	102
	Przełączanie na inny temat?	104

CZĘŚĆ II: PROGRAMOWANIE W C++ Z WYKORZYSTANIEM FUNKCJI 107

ROZDZIAŁ 6:	Tworzenie funkcji	109
	Pisanie i używanie funkcji	110
	Definiowanie funkcji	112
	Definiowanie funkcji sumujSekwencje()	112
	Wywoływanie funkcji sumujSekwencje()	113
	Dziel i rządź	113
	Funkcje w szczegółach	113
	Funkcje proste	114
	Funkcje z argumentami	115
	Przeciążanie funkcji	117
	Definiowanie prototypów funkcji	119
	Argumenty domyślne	120
	Przekazywanie przez wartość i przekazywanie przez referencję	122
	Typy przechowywania zmiennych	123
ROZDZIAŁ 7:	Przechowywanie sekwencji w tablicach	125
	Szeregowanie argumentów w tablicach	126
	Korzystanie z tablic	127
	Inicjowanie tablic	130
	Sięganie poza zakres tablicy	131
	Pętle bazujące na zakresach	131
	Definiowanie i używanie tablic w tablicach	132
	Korzystanie z tablic znaków	133
	Tworzenie tablicy znaków	133
	Tworzenie ciągu znaków	134

	Manipulowanie ciągami znaków	136
	Niektóre funkcje biblioteki standardowej	138
	Miejsce dla ciągów złożonych z szerokich znaków	139
ROZDZIAŁ 8:	Wskaźniki w C++	141
	Rozmiar zmiennej	141
	Czym jest adres?	143
	Operatory adresu	143
	Używanie wskaźników	145
	Używanie różnych typów wskaźników	146
	Przekazywanie wskaźników do funkcji	147
	Przekazywanie przez wartość	147
	Przekazywanie wartości wskaźnikowych	148
	Przekazywanie przez referencję	148
	Zmienne stałe	149
	Korzystanie ze sterty	151
	Ograniczony zasięg	151
	Analiza problemu zakresu ważności	152
	Użycie sterty	153
ROZDZIAŁ 9:	Drugie spojrzenie na wskaźniki w C++	155
	Definiowanie operacji na wskaźnikach	155
	Użycie wskaźników w tablicach	156
	Użycie operatorów do adresu tablicy	158
	Operacje wskaźnikowe na ciągach znaków	159
	Uzasadnienie użycia wskaźników	161
	Stosowanie operatorów do wskaźników innych typów niż char	161
	Porównanie wskaźnika z tablicą	162
	Wskaźnik null	164
	Deklaracja i użycie tablic wskaźników	165
	Wykorzystanie tablic ciągów znaków	166
	Dostęp do argumentów funkcji main()	168
ROZDZIAŁ 10:	Preprocesor C++	173
	Co to jest preprocesor?	173
	Dołączanie plików	174
	#Definiowanie	177
	A może by tak nie definiować?	180
	Wyliczeniowy typ danych	181
	Dołączanie warunkowe z użyciem instrukcji #if	183
	Obiekty zdefiniowane wewnątrz	184
	Typedef	186

CZĘŚĆ III: WPROWADZENIE DO KLAS 187

ROZDZIAŁ 11: Programowanie obiektowe 189

Abstrakcyjne kuchenki mikrofalowe	189
Przygotowanie funkcjonalnego nachos	190
Przygotowywanie nachos w sposób obiektowy	191
Klasyfikacja kuchenki mikrofalowej	191
Po co klasyfikujemy przedmioty?	192

ROZDZIAŁ 12: Klasy w C++ 195

Wprowadzenie do klas	195
Kształt klasy	196
Dostęp do elementów klasy	197
Aktywacja obiektów	197
Symulowanie rzeczywistych obiektów	198
Po co zwracać sobie głowę funkcjami składowymi?	198
Dodawanie funkcji składowej	199
Wywoływanie funkcji składowych	200
Dostęp do pól z funkcji składowej	202
Ustalanie zakresu	203
Definiowanie funkcji składowych w klasie	204
Umieszczanie funkcji składowych poza klasą	206
Przeciążanie funkcji składowych	208

ROZDZIAŁ 13: Wskazywanie na obiekty 211

Deklarowanie tablic obiektów	211
Deklarowanie wskaźników do obiektów	212
Dereferencja wskaźnika obiektu	213
Wskazywanie w kierunku strzałek	214
Przekazywanie obiektów do funkcji	214
Wywoływanie funkcji z wartością obiektu	215
Wywoływanie funkcji ze wskaźnikiem na obiekt	216
Wywoływanie funkcji za pomocą operatora referencji	217
Po co przejmować się wskaźnikami i referencjami?	218
Zwracanie do sterty	219
Alokowanie obiektów ze sterty	220
Kiedy kompilator alokuje za Ciebie pamięć?	220
Listy	221
Wykonywanie innych operacji na listach	222
Wykorzystanie list w programie DaneListyJed	223
Promyk nadziei: lista kontenerów w bibliotece C++	226

ROZDZIAŁ 14:	Chronione elementy składowe klasy	227
	Ochrona składowych	227
	Dlaczego potrzebujesz składowych chronionych?	228
	Jak działają składowe chronione?	228
	Po co chronić składowe klasy?	230
	Ochrona wewnętrznego stanu klasy	230
	Używanie klasy z ograniczonym interfejsem	231
	Dostęp do elementów chronionych z zewnątrz	231
ROZDZIAŁ 15:	„Po co mnie tworzysz, skoro za chwilę chcesz mnie zniszczyć?”	235
	Tworzenie obiektów	236
	Korzystanie z konstruktorów	237
	Konstruowanie pojedynczego obiektu	237
	Konstruowanie wielu obiektów	238
	Konstruowanie dupleksu	239
	Dekonstrukcja obiektu	241
	Do czego służy destruktor?	241
	Praca z destruktorami	242
ROZDZIAŁ 16:	Argumenty przekazywane przez konstruktor	247
	Wyposażenie konstruktorów w argumenty	248
	Korzystanie z konstruktora	248
	Przeciążanie konstruktora	250
	Automatyczne konstruktory domyślne	253
	Konstruowanie składowych klasy	255
	Konstruowanie składowej typu złożonego	255
	Konstruowanie składowej, która jest stałą	261
	Rekonstrukcja kolejności konstruowania	261
	Kolejność konstrukcji obiektów lokalnych	262
	Obiekty statyczne są konstruowane tylko raz	262
	Wszystkie obiekty globalne są konstruowane przed funkcją main()	263
	Obiekty globalne są konstruowane bez określonej kolejności	264
	Składowe są konstruowane w kolejności, w jakiej zostały zadeklarowane	265
	Destruktry niszczą obiekty w kolejności odwrotnej do kolejności ich tworzenia	265
	Konstruowanie tablic	265
	Konstruktory jako forma konwersji	266

ROZDZIAŁ 17:	Konstruktory kopiujący i przenoszący	269
	Kopiowanie obiektu	269
	Do czego służy konstruktor kopiujący?	270
	Korzystanie z konstruktora kopiującego	270
	Automatyczny konstruktor kopiujący	272
	Kopiowanie głębokie i kopiowanie płytkie	273
	Obiekty tymczasowe	277
	Trwałe unikanie obiektów tymczasowych	279
	Konstruktor przenoszący	280

ROZDZIAŁ 18:	Składowe statyczne: czy miękczacz tkanin może pomóc?	283
	Definiowanie składowej statycznej	284
	Do czego służą składowe statyczne?	284
	Używanie składowych statycznych	284
	Odwoływanie się do składowych statycznych	285
	Przypadki użycia składowych statycznych	287
	Deklarowanie statycznych funkcji składowych	287
	O czym to w ogóle jest?	290

CZĘŚĆ IV: DZIEDZICZENIE **293**

ROZDZIAŁ 19:	Dziedziczenie	295
	Czy potrzebuję dziedziczenia?	297
	Jak dziedziczy klasa?	297
	Używanie podklasy	299
	Konstruowanie podklasy	299
	Niszczenie podklasy	301
	Dziedziczenie konstruktorów	301
	Relacja MA	302

ROZDZIAŁ 20:	Wirtualne funkcje składowe. Czy są realne?	303
	Do czego jest potrzebny polimorfizm?	306
	Jak działa polimorfizm?	307
	Kiedy funkcja nie jest wirtualna?	308
	Wirtualne rozważania	310

ROZDZIAŁ 21:	Faktoryzacja klas	313
	Faktoryzacja	314
	Implementacja klas abstrakcyjnych	318
	Opisywanie koncepcji klasy abstrakcyjnej	318
	Tworzenie klasy konkretnej z klasy abstrakcyjnej	320
	Przekazywanie klas abstrakcyjnych	320

CZĘŚĆ V: BEZPIECZEŃSTWO

ROZDZIAŁ 22:	Czy zaakceptujesz nowy operator przypisania?	325
	Porównanie operatorów z funkcjami	326
	Wstawianie nowego operatora	327
	Tworzenie płytkich kopii to poważny problem	327
	Przeciążanie operatora przypisania	329
	Przeciążenie operatora indeksu	333
	Konstruktor i operator przenoszący	334
ROZDZIAŁ 23:	Strumienie wejścia-wyjścia	337
	Jak działa strumień wejścia-wyjścia?	338
	Domyślne obiekty strumienia	338
	Strumienie wejścia-wyjścia	340
	Tryby otwierania	341
	Hej, pliku, w jakim jesteś stanie?	342
	Czy możesz mi pokazać przykład?	343
	Inne metody klas obsługi strumieni	346
	Bezpośrednie czytanie i zapisywanie strumieni	348
	Zarządzanie formatem	350
	O co chodzi z endl?	351
	Umieszczanie wskaźnika w pliku	352
	Korzystanie z podklas klasy stringstream	353
	Manipulowanie manipulatorami	356
ROZDZIAŁ 24:	Obsługa błędów — wyjątki	359
	Uzasadnienie dla nowego mechanizmu obsługi błędów	361
	Działanie mechanizmu wyjątków	362
	Czym można rzucać?	365
	Uwaga, przechodzę!	367

ROZDZIAŁ 25:	Dziedziczenie wielokrotne	369
	Mechanizm dziedziczenia wielokrotnego	370
	Jak sobie poradzić z niejasnościami dotyczącymi dziedziczenia?	371
	Dziedziczenie wirtualne	372
	Konstruowanie obiektów dziedziczenia wielokrotnego	378
	Wyrażanie sprzecznych opinii	379
ROZDZIAŁ 26:	Szablony w C++	381
	Uogólnianie funkcji do szablonu	382
	Szablony klas	384
	Wskazówki dotyczące korzystania z szablonów	387
	Instancje szablonów zewnętrznych	388
	Implementowanie listy inicjalizacyjnej	388
ROZDZIAŁ 27:	Standaryzacja w bibliotece STL	391
	Kontener typu string	392
	Iterowanie po listach	397
	Przeglądanie listy	398
	Operacje na całej liście	400
	Czy możesz mi pokazać przykład?	400
ROZDZIAŁ 28:	Pisanie bezpiecznego kodu	403
	Motywy działania hakerów	404
	Na czym polega wstrzyknięcie kodu?	406
	Przykładowe wstrzyknięcie SQL	406
	Unikanie wstrzykiwania kodu	408
	Przepełnianie bufora dla zabawy i zysku	409
	Czy mogę zobaczyć przykład?	410
	Hakerskie wykorzystanie przepełnienia bufora	412
	Unikanie przepełnienia bufora — pierwsza próba	416
	Unikanie przepełnienia bufora — druga próba	418
	Kolejna zaleta klasy string	420
	Dlaczego nie zawsze należy używać klasy string?	422
	CZĘŚĆ VI: DEKALOGI	425
ROZDZIAŁ 29:	Dziesięć sposobów na zapobieganie błędom	427
	Włączaj wszystkie ostrzeżenia i komunikaty o błędach	427
	Koduj w jasnym i spójnym stylu	428
	Ograniczaj widoczność	429
	Dodawaj komentarze do kodu w trakcie pisania	430

Wykonuj krok po kroku każdą ścieżkę przynajmniej raz	431
Unikaj przeciążania operatorów	431
Zarządzaj stertą systematycznie	431
Używaj wyjątków do obsługi błędów	432
Deklaruj wirtualne destruktory	432
Unikaj dziedziczenia wielokrotnego	434

ROZDZIAŁ 30: Dziesięć sposobów na ochronę programów przed hakerami435

Nie zakładaj niczego w odniesieniu do danych wprowadzanych przez użytkowników	436
Obsługuj awarie z wdziękiem	437
Zapisuj logi programu	438
Postępuj zgodnie z dobrymi zasadami wytwarzania oprogramowania	439
Wdrożenie kontroli wersji	440
Bezpiecznie uwierzytniaj użytkowników	441
Zarządzaj zdalnymi sesjami	444
Zaciemniaj swój kod	445
Podpisz swój kod za pomocą certyfikatu cyfrowego	448
Użyj bezpiecznego szyfrowania wszędzie tam, gdzie jest to konieczne	449

CZĘŚĆ VII: DODATKI451

Skorowidz453

- ▶▶ Podstawy C++.
- ▶▶ Instalowanie *Code::Blocks* w systemach Windows, Ubuntu Linux lub Macintosh OS X.
- ▶▶ Tworzenie pierwszego programu w C++.
- ▶▶ Uruchamianie programu.

Rozdział **1**

Pierwszy program w C++

O to jesteśmy: tylko Ty i ja. Nie zwlekajmy, czas zacząć. Na początek przedstawię kilka podstawowych pojęć.

Komputer to niesamowicie szybka, ale zarazem niezwykle głupia maszyna. Komputer może zrobić wszystko, co mu powiesz (w granicach rozsądku), ale robi **dokładnie** to, co mu powiesz — nic więcej ani nic mniej.

Zła wiadomość dla nas, ludzi, jest taka, że komputery nie rozumieją niestety żadnego rozsądnego języka ludzkiego — nie mówią też po polsku. No dobrze, wiem, co powiesz: „Widziałem komputery, które rozumiały polski”. Właściwie jednak widziałeś komputer wykonujący **program**, który potrafił zrozumieć język polski.

Komputery rozumieją język, który nazywamy **językiem komputerowym** lub **językiem maszynowym**. Człowiek mógłby posługiwać się językiem maszynowym, ale byłoby to bardzo trudne. Dlatego komputery i ludzie spotykają się pośrodku i używają języków pośrednich, takich jak C++. Ludzie rozumieją C++ (poniekąd), a C++ może zostać przekonwertowany na język maszynowy, aby mógł go zrozumieć komputer.

Pojęcia dotyczące języka C++

Program w C++ jest plikiem tekstowym zawierającym sekwencję komend C++ zgodnych z gramatyką języka C++. Ten plik tekstowy jest określany jako **plik źródłowy** (prawdopodobnie dlatego, że jest źródłem wszystkich frustracji). Pliki źródłowe w C++ zwykle zawierają rozszerzenie `.CPP`, podobnie jak pliki Adobe Acrobat zawierają rozszerzenie `.PDF`, a pliki wsadowe MS-DOS (pamiętasz je?) mają rozszerzenie `.BAT`.

Celem programowania w C++ jest napisanie sekwencji poleceń, które można przekształcić na program w języku maszynowym. Dopiero on faktycznie robi to, co chcemy. Ta konwersja nazywana jest *kompilacją* i jest zadaniem *kompilatora*. Kod maszynowy, który napisałeś, musi być połączony z pewnymi instrukcjami instalacji i usuwania oraz niektórymi standardowymi procedurami bibliotecznymi w procesie zwanym *łączeniem* lub *konsolidacją* (ang. *linking*). Kompilacja razem z konsolidacją to *budowanie*. Powstałe pliki **wykonywalne maszynowo** mają w systemie Windows rozszerzenie `.EXE`. W systemie Linux lub w komputerach Macintosh nie zawierają one żadnego konkretnego rozszerzenia.

To brzmi dość prosto — co w tym trudnego? Zatem czytaj dalej.

Aby napisać program, potrzebujesz dwóch specjalistycznych programów komputerowych. Jeden (edytor) służy do pisania kodu podczas tworzenia pliku źródłowego `.CPP`. Drugi (kompilator) konwertuje plik źródłowy na plik wykonywalny, który faktycznie wykonuje polecenia (otwórz arkusz kalkulacyjny, wydawaj nieprzyzwoite odgłosy, odbijaj nadlatujące asteroidy czy cokolwiek innego).

Obecnie twórcy narzędzi zazwyczaj łączą kompilator i edytor w jeden pakiet, nazywany **zintegrowanym środowiskiem programistycznym**. W takim środowisku, po zakończeniu wprowadzania poleceń, które tworzą program, wystarczy kliknąć jeden przycisk, aby utworzyć plik wykonywalny.

Na szczęście istnieją darmowe środowiska dla języka C++. W tej książce używam jednego z nich — Code::Blocks. Ten edytor działa z wieloma różnymi kompilatorami, ale wersja Code::Blocks w połączeniu z kompilatorem GNU gcc użytym do napisania tej książki jest dostępna dla systemów Windows i Macintosh oraz dla różnych wersji systemu Linux — tak jak zostało to opisane w punkcie niniejszego rozdziału poświęconym instalacji.

Code::Blocks jest darmową aplikacją. Jednak jeśli chcesz, możesz wesprzeć jej dalszy rozwój. Zachęcam do wpłacenia niewielkiej darowizny na ten cel. Wcale nie **musisz** płacić, żeby używać Code::Blocks, ale możesz mieć swój wkład, jeśli chcesz. Szczegółowe informacje można znaleźć w witrynie Code::Blocks.

Przetestowałem programy w tej książce za pomocą Code::Blocks 13.12, który jest wyposażony w gcc w wersji 4.7.1. Ta wersja gcc implementuje większość dodatków ze standardu C++ 2011.



Możesz użyć innych wersji gcc lub nawet innych kompilatorów, ale mogą one nie implementować pełnego standardu 2011. Z tego powodu rozszerzenia 2011 są oznaczone ikoną „11” widoczną tutaj.



Użyty w tej książce kompilator gcc nie implementuje żadnego z rozszerzeń dodanych w standardzie C++ 2014, ale uwzględniłem je tam, gdzie ma to zastosowanie, ponieważ są już dostępne nowsze wersje.

Muszę się do czegoś przyznać: ta książka jest w pewnym sensie ukierunkowana na Windowsa. Wszystkie programy w książce przetestowałem w systemach Windows 2000/XP/Vista/7/8, Ubuntu Linux i Macintosh OS X. Różnice występujące pomiędzy systemami operacyjnymi zazaczyłem w tekście.

Dodatkowo w tym rozdziale zamieściłem instrukcje instalacji dla każdego z trzech wymienionych powyżej systemów operacyjnych. Wersje Code::Blocks i gcc są dostępne dla innych odmian systemu Linux i innych wersji systemu operacyjnego Macintosh. Programy powinny także z nimi współpracować.



OSTRZEŻENIE

Pakiet Code::Blocks/gcc generuje programy 32-bitowe. Nie umożliwia jednak łatwego tworzenia programów „okienkowych”. Programy w tej książce uruchamiane są z wiersza poleceń i wyświetlają wynik w wierszu poleceń. Mimo że wydaje się to nudne, stanowczo zalecam, abyś najpierw zapoznał się z przykładami z tej książki, aby nauczyć się C++, zanim zajmiesz się tworzeniem aplikacji okienkowych. C++ i programowanie w systemie Windows to dwie różne rzeczy i (z uwagi na Twoje zdrowie psychiczne) powinny jako takie pozostać w Twojej głowie.

Postępuj zgodnie z procedurą opisaną w następnym podrozdziale, aby zainstalować Code::Blocks, a następnie zbuduj swój pierwszy program w C++. Zadaniem tego programu jest konwersja temperatury wprowadzonej przez użytkownika ze stopni Celsjusza na stopnie Fahrenheita.

Instalacja Code::Blocks

Ze strony <http://www.codeblocks.org/downloads/binaries> możesz pobrać najnowszą wersję środowiska Code::Blocks dla systemów: Windows, Ubuntu Linux i Macintosh OS X. Postępuj zgodnie z zamieszczonymi poniżej instrukcjami instalacji dotyczącymi Twojego systemu operacyjnego.

Windows

Środowisko Code::Blocks jest dostępne w łatwym do zainstalowania, skompresowanym pliku wykonywalnym, który jest zgodny ze wszystkimi wersjami systemu Windows począwszy od wersji Windows 2000. Oto sposób instalacji środowiska:



WSKAZÓWKA

1. **Pobierz plik wykonywalny `codeblocks-13.12.mingw-setup.exe` ze strony <http://www.codeblocks.org/downloads/source/5>.**

Zapisz plik wykonywalny na pulpicie lub w innym miejscu, które można łatwo znaleźć.

Wspomniany plik wykonywalny zawiera wersję kompilatora GCC 4.71. Nie jest to najnowsza wersja GCC, ale właśnie tej wersji użyłem podczas pisania tej książki.

W chwili powstawania jej tłumaczenia najnowszą dostępną wersją Code::Blocks jest wersja 17.12. Można ją pobrać ze strony

<http://www.codeblocks.org/downloads/binaries>. Ostatnia dostępna wersja dla Mac OS X to 13.12. To tej wersji będziemy używać.

2. **Po zakończeniu pobierania kliknij dwukrotnie ikonę programu.**
3. **W zależności od używanej wersji systemu Windows może pojawić się wyskakujące okienko „Niezidentyfikowany program chce uzyskać dostęp do komputera”. Jeśli tak się stanie, kliknij *Zezwól*, aby rozpocząć instalację.**
4. **Po zamknięciu wszystkich aplikacji zewnętrznych, tak jak zostanie to wyświetlone w oknie dialogowym *Welcome to the CodeBlocks Setup Wizard* (witaj w instalatorze CodeBlocks), kliknij przycisk *Next* (dalej).**
5. **Przeczytaj *Umowę licencyjną użytkownika końcowego* (powszechnie znaną jako *EULA*), a następnie kliknij przycisk *I Agree*, jeśli zgadzasz się z jej postanowieniami.**

Nie masz dużego wyboru — jeśli nie zaakceptujesz licencji, pakiet się nie zainstaluje. Załóżmy jednak, że się zgodzisz. Instalator Code::Blocks otworzy okno dialogowe z opcjami instalacji. Opcje domyślne są dla nas wystarczające.

6. **Kliknij przycisk *Next*.**

Program instalacyjny pozwala zainstalować tylko niektóre podzbiory funkcjonalności. Musisz wybrać przynajmniej instalację domyślną i pakiet *MinGW Compiler Suite*. Najlepszym wyjściem jest zainstalowanie wszystkich opcji, tak jak jest to zaznaczone domyślnie.

Jeśli nie ma opcji *MinGW Compiler Suite*, prawdopodobnie pobrałeś wersję Code::Blocks, która nie zawiera gcc. Ta wersja nie będzie działać poprawnie.

7. **Po kolejnym kliknięciu przycisku *Next* pojawi się zapytanie o folder instalacji; kliknij *Install*, aby zaakceptować domyślny folder docelowy.**

Instalator Code::Blocks rozpocznie kopiowanie wszystkich potrzebnych plików na Twój dysk twardy. Po zakończeniu zapyta: *Do you want to run Code::Blocks now?* (Czy chcesz teraz uruchomić Code::Blocks?).

8. **Kliknij *Yes* (lub *Tak*), aby uruchomić Code::Blocks.**

Code::Blocks zapyta teraz, którego kompilatora użyć. Domyślny kompilator to GNU GCC. To jest właściwy wybór.

9. **Z poziomu okna głównego Code::Blocks wybierz *Settings/Compiler*.**



OSTRZEŻENIE

10. Wybierz zakładkę *Compiler Flags* (flagi kompilatora).

11. Upewnij się, że wybrano następujące trzy flagi (patrz rysunek 1.1):

- *Enable All Compiler Warnings* (włącz wszystkie ostrzeżenia kompilatora);
- *Have g++ Follow the Coming C++0x ISO C++ Language Standard* (niech g++ postępuje zgodnie z nadchodzącym standardem C++ 0x ISO C++);
- *Have g++ Follow the C++11 ISO C++ Language Standard* (niech g++ postępuje zgodnie ze standardem C++11 ISO C++).

Standard C++ 2011 pierwotnie miał być standardem C++ 2008 lub 2009. Ponieważ nie było to jasne, standard zyskał popularność pod nazwą 0x. Standard nie został całkowicie zaakceptowany do 2011 roku. W środowisku kompilatora gcc języki C++ 0x i C++ 11 odnoszą się do tego samego standardu.



RYСУNEK 1.1.

Upewnij się, że ustawiona jest flaga *Enable All Compiler Warnings*

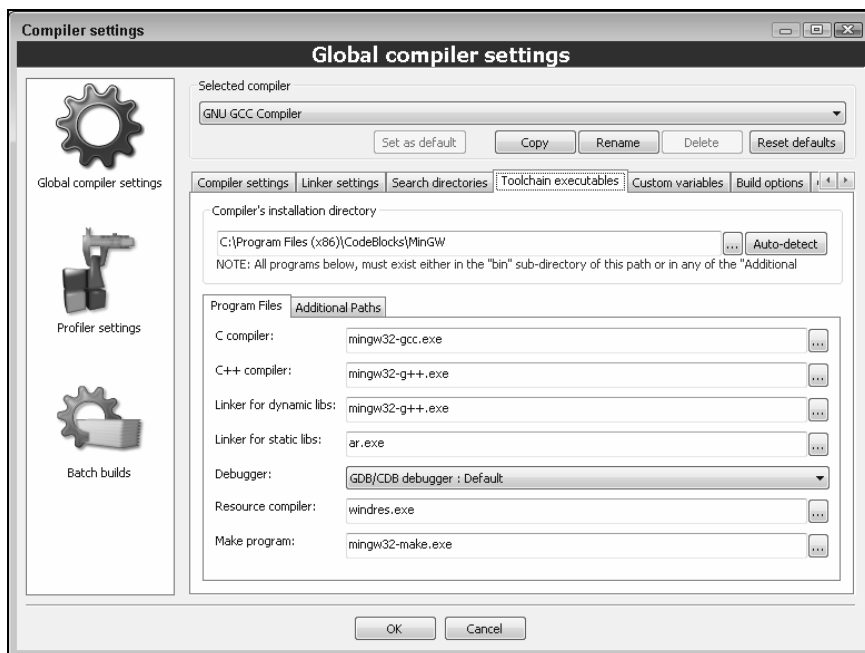
12. Wybierz zakładkę *Toolchain Executables* (pliki wykonywalne zestawu narzędzi). Upewnij się, że wygląda tak, jak pokazano na rysunku 1.2.

Domyślną lokalizacją kompilatora gcc jest podkatalog *MinGW\bin* katalogu *Code::Blocks*.

Jeśli domyślna lokalizacja jest pusta, *Code::Blocks* nie będzie wiedział, gdzie jest kompilator gcc, i nie będzie mógł budować programów. Upewnij się, że pobrałeś wersję *Code::Blocks*, która zawiera gcc, oraz że zaznaczyłeś właściwą opcję podczas instalacji. Jeśli używasz kompilatora gcc zainstalowanego wcześniej, powinieneś wskazać programowi *Code::Blocks* miejsce na dysku twardym, w którym się on znajduje.



OSTRZEŻENIE



RYSUNEK 1.2.
Upewnij się,
że katalog
instalacyjny
kompilatora jest
poprawny

13. Zamknij okno dialogowe ustawień — *Compiler Settings*.
14. Kliknij *Next* w oknie dialogowym instalatora Code::Blocks, a następnie kliknij *Finish* (zakończ), aby zakończyć program instalacyjny.

Program instalacyjny zakończy działanie.

Ubuntu Linux

W systemie Linux instalacja jest procesem dwuetapowym, ponieważ Code::Blocks dla Linuksa nie zawiera gcc. Najpierw trzeba zainstalować gcc, a następnie Code::↪Blocks.

Instalacja gcc

Kompilator gcc dla Linuksa jest łatwo dostępny. Aby go zainstalować, uruchom terminal i wykonaj następujące czynności:

1. Wprowadź następujące polecenia z wiersza poleceń:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install g++
```

Standardowa dystrybucja Ubuntu Linux zawiera kompilator GNU C, ale nie obejmuje rozszerzeń C++. W szczególności nie zawiera rozszerzeń standardu C++ 2011. Pierwsze dwa polecenia aktualizują narzędzia, które już masz. Trzecie polecenie instaluje C++.

2. Wprowadź następujące polecenie z wiersza poleceń:

```
gcc --version
```

Mój Ubuntu 18.04 pobrał GNU C++ w wersji 7.4.0. Na potrzeby przykładów z tej książki wersja 4.7.1 jest wystarczająca. Jeśli masz wcześniejszą wersję, niektóre funkcje C++ 2011 mogą nie działać poprawnie. Poza tym wszystko powinno działać jak należy.

Jeśli używasz Debiana, polecenia są takie same. Jeśli używasz Red Hata, zamiast programu `apt-get` zastosuj program `yum`. Polecenie instalacji dla Red Hata wygląda następująco:

```
sudo yum install g++
```



WSKAZÓWKA

Instalacja Code::Blocks

Wersja Code::Blocks gotowa do użycia w Ubuntu jest dostępna w *Ubuntu Software Center* (Centrum Oprogramowania Ubuntu). Na szczęście dla wszystkich zainteresowanych wiele innych wersji systemu Linux zawiera coś podobnego do Centrum Oprogramowania. Aby zainstalować Code::Blocks, wykonaj poniższe czynności:

1. Kliknij ikonę Centrum Oprogramowania na pulpicie Ubuntu.

2. Wybierz Code::Blocks z listy dostępnych oprogramowania.

Rozpocznie się proces instalacji.

Code::Blocks przeszukuje dysk twardy w poszukiwaniu kompilatora C++. Powinien być w stanie go znaleźć bez problemu, ale jeśli tego nie zrobi, wykonaj poniższe kroki.

3. Uruchom Code::Blocks.

4. Wybierz *Settings/Compiler* (ustawienia kompilatora).

5. Wybierz zakładkę *Compiler Flags* (flagi kompilatora).

6. Upewnij się, że wybrałeś następujące trzy flagi (patrz rysunek 1.1):

- *Enable All Compiler Warnings* (włącz wszystkie ostrzeżenia kompilatora);
- *Have g++ Follow the Coming C++0x ISO C++ Language Standard* (niech g++ postępuje zgodnie z nadchodzącym standardem C++ 0x ISO C++);
- *Have g++ Follow the C++11 ISO C++ Language Standard* (niech g++ postępuje zgodnie ze standardem C++11 ISO C++).

7. Wybierz zakładkę *Toolchain Executables* (pliki wykonywalne zestawu narzędzi).

8. Kliknij przycisk

9. Przejdź do `/usr`, chyba że zainstalowałeś kompilator gcc w innym miejscu niż domyślna lokalizacja `/usr/bin`.

10. W formularzu, w polu *C compiler* należy wpisać *gcc*, w polu *C++ compiler* należy wybrać *g++*, w polu *Linker for dynamic libs* również należy wybrać *g++*.
11. Wybierz *OK*, aby zamknąć okno dialogowe.

Macintosh

Wersja Code::Blocks dla komputerów Macintosh używa dystrybucji Xcode firmy Apple jako kompilatora. Z tego powodu podzieliłem instalację na trzy części.

Instalacja Xcode

Potrzebne jest darmowe środowisko programistyczne oferowane przez Apple, czyli *Xcode*. Aby je zainstalować, wykonaj poniższe czynności:

1. Otwórz przeglądarkę *Safari* i przejdź na stronę <http://developer.apple.com>.
2. Kliknij *Download Xcode* (pobierz Xcode), aby pobrać najnowszą wersję programu.

Spowoduje to otwarcie okna dialogowego pobierania Xcode, pokazanego na rysunku 1.3.



RYСУNEK 1.3.
Okno dialogowe pobierania Xcode pozwala zainstalować Xcode za darmo

3. Kliknij przycisk *Free*, aby zmienił się na *Install App* (zainstaluj aplikację).
4. Możesz zostać poproszony o wprowadzenie hasła systemowego (którym logujesz się podczas uruchamiania komputera Mac).

Ikona zmieni się na *Installing*.

Pobieranie i instalacja zajmuje trochę czasu (w chwili pisania przeze mnie tego tekstu archiwum z Xcode miało nieco ponad 2 GB).

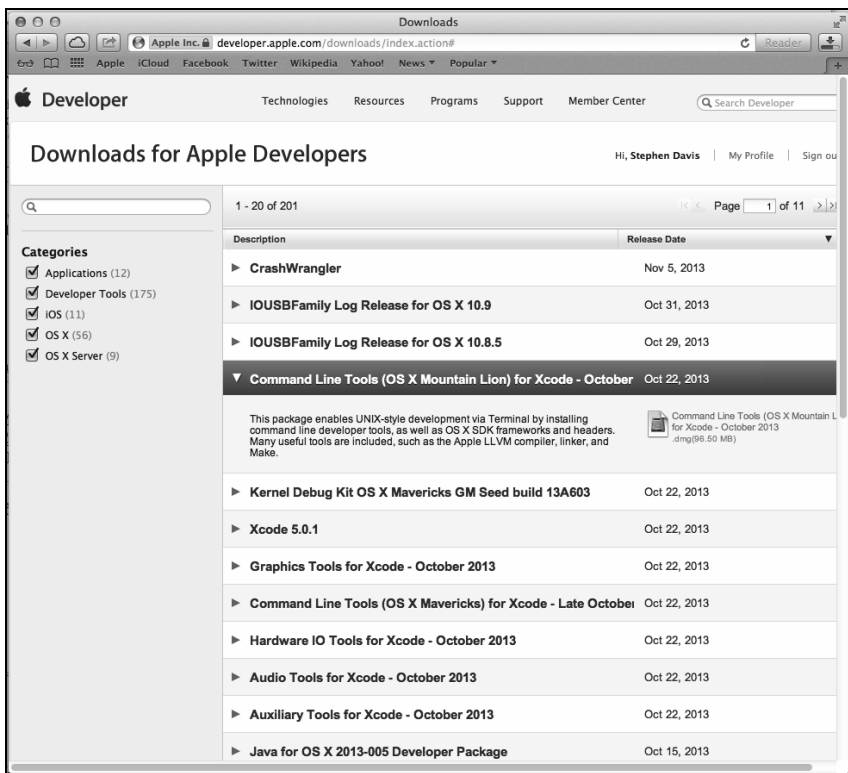
Instalowanie narzędzi wiersza poleceń

Można by się spodziewać, że tak duży program jak Xcode ma wszystko, czego potrzebuje. Nic bardziej mylnego. Do uzyskania działającego kompilatora gcc na komputerze Macintosh potrzebny jest jeszcze jeden pakiet firmy Apple. Aby zainstalować narzędzia wiersza poleceń dla Xcode, postępuj zgodnie z poniższą procedurą:

1. Otwórz przeglądarkę Safari i przejdź na stronę <http://developer.apple.com/downloads>.

Możesz zostać poproszony o zarejestrowanie identyfikatora *Apple Developer ID*. Bez obaw, to nic nie kosztuje.

2. Wyszukaj narzędzia wiersza poleceń dla Xcode. Wybierz aplikację pokazaną na rysunku 1.4. Następnie kliknij ikonę *Download*.



RYSUNEK 1.4.
Aby przygotować kompilator dla macintosha, musisz zainstalować zarówno Xcode, jak i Command Line Tools dla Xcode

3. Kliknij dwukrotnie pobrany pakiet mpkg, aby go zainstalować.
4. Zaakceptuj wszystkie wartości domyślne.

Instalacja powinna zakończyć się powodzeniem.

Instalacja Code::Blocks

Teraz możesz dokończyć instalację, pobierając i instalując edytor Code::Blocks:

1. Otwórz przeglądarkę Safari i przejdź na stronę www.codeblocks.org/downloads.
2. Kliknij *Downloads/Binaries* (pobierz/binaria).
3. Kliknij *Mac OS X*.
4. Wybierz jedno ze źródeł plików, aby pobrać najnowszą wersję.

W chwili powstawania tego tekstu najnowsza dostępna wersja to *CodeBlocks-13.12-mac.zip*.

5. Zainstaluj pobrany plik *Zip* w folderze *Applications*.

Jeśli nigdy nie instalowałeś aplikacji z witryny innej firmy, zanim będziesz mógł to zrobić, być może będziesz musiał wykonać poniższe dodatkowe operacje:

- a) kliknąć *System Preferences* (preferencje systemowe);
- b) kliknąć *Security and Privacy* (bezpieczeństwo i prywatność);
- c) kliknąć kłódkę w lewym dolnym rogu okna dialogowego, aby zezwolić na zmiany;
- d) w formularzu *Allow applications downloaded from* (zezwalaj na pobieranie aplikacji z) kliknąć opcję *Anywhere* (skądkolwiek), tak jak pokazano na rysunku 1.5.



RYСУNEK 1.5.

Przed instalacją Code::Blocks w Twoim systemie Macintosh trzeba zainstalować inne aplikacje

Po zakończeniu instalacji Code::Blocks możesz powrócić do tego okna dialogowego i przywrócić domyślną wartość ustawień (*Mac App Store*).

6. Kliknij dwukrotnie ikonę Code::Blocks.

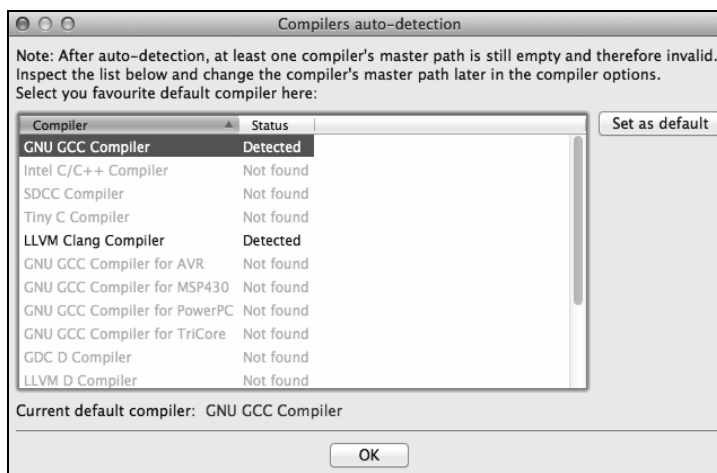
Przy pierwszym uruchomieniu system Mac OS zapyta Cię, czy na pewno chcesz go otworzyć.

7. Wybierz *Don't Warn Me When Opening Applications on This Disk Image* (nie ostrzegaj mnie przy otwieraniu aplikacji na obrazie dysku) i kliknij *Open* (otwórz).

Code::Blocks powinien się uruchomić i znaleźć zainstalowany kompilator gcc wraz z narzędziami wiersza polecenia.

8. Wybierz kompilator gcc, jak pokazano na rysunku 1.6. Kliknij *Set as Default* (ustaw jako domyślny). Aby kontynuować uruchamianie Code::Blocks, kliknij *OK*.

Code::Blocks otworzy stronę z banerem, a następnie wyświetli menu w górnej części okna dialogowego.



RYSUNEK 1.6.
Code::Blocks przy pierwszym uruchomieniu automatycznie znajduje zainstalowany kompilator gcc

9. Wybierz *Settings/Compiler* (ustawienia kompilatora), następnie kliknij opcję *Have g++ Follow the Coming C++ 0x ISO C++ Language Standard* (niech g++ postępuje zgodnie z nadchodzącym standardem C++ 0x ISO C++). Kliknij *OK*, aby zamknąć okno dialogowe.

Teraz jesteś gotowy do utworzenia pierwszego programu w C++.

Tworzenie pierwszego programu w C++

W tym podrozdziale utworzysz swój pierwszy program w C++. Kod C++ wprowadzimy w pliku o nazwie `CONVERT.CPP`, a następnie skonwertujemy ten kod na program wykonywalny.

Tworzenie projektu

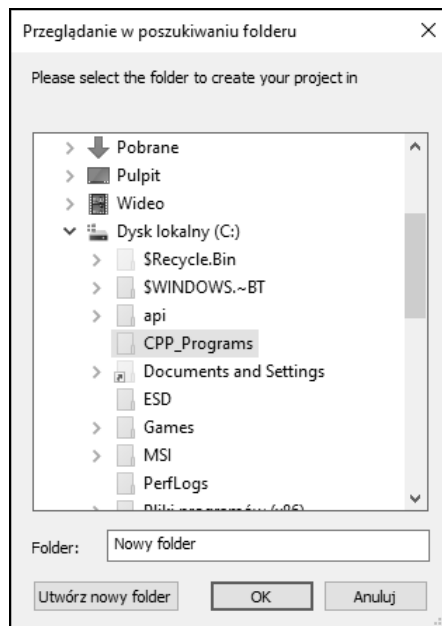
Pierwszym krokiem na drodze do stworzenia programu w C++ jest opracowanie projektu. Projekt w Code::Blocks definiuje nazwy plików źródłowych *.CPP*, które mają zostać dołączone, oraz typ programu do utworzenia. Większość programów w książce składa się z jednego pliku źródłowego i jest uruchamiana z wiersza poleceń.

1. **Uruchom środowisko Code::Blocks.**
2. **W oknie Code::Blocks wybierz *File/New/Project* (plik/nowy/projekt).**
3. **Wybierz ikonę *Console Application* (aplikacji konsoli), a następnie kliknij przycisk *Go* (przejdź).**
4. **W następnym oknie dialogowym wybierz C++ jako język, którego chcesz używać. Kliknij *Next* (następny).**

Code::Blocks i gcc umożliwiają również pisanie programów w języku C.

5. **W polu *Folder to Build Project In* (folder do budowy projektu) wybierz ikonę**
6. **W systemie Windows kliknij *Ten Komputer*, a następnie dysk C:.**
W systemach Linux i Macintosh możesz wybrać Pulpit.
7. **Wybierz przycisk *Utwórz nowy folder* w lewym dolnym rogu ekranu.**
8. **Nazwij nowy folder *CPP_Programs*.**

Wynik powinien wyglądać tak, jak pokazano na rysunku 1.7.

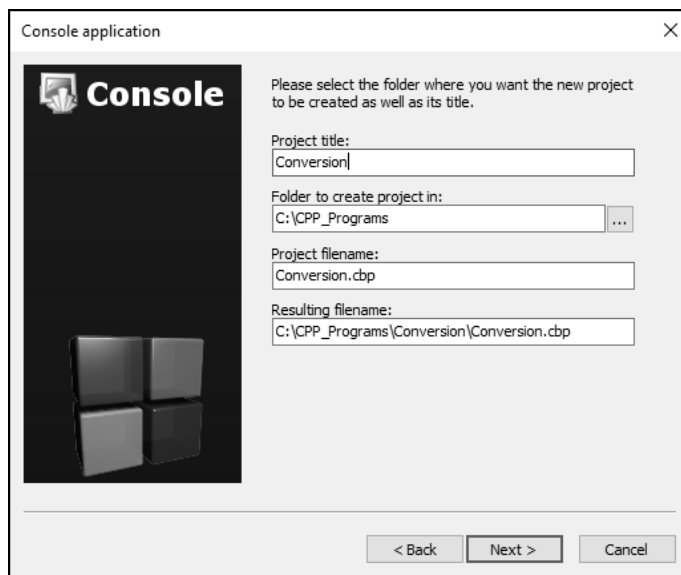


RYSUNEK 1.7.

W systemie Windows umieść swój projekt w folderze C:\CPP_Programs

9. W polu *Project title* (tytuł projektu) wpisz nazwę projektu, w tym przypadku *Conversion*.

Ekran wynikowy w systemie Windows pokazano na rysunku 1.8. Wersje dla systemów Linux i Macintosh nie niczym się od niego różnią, z wyjątkiem ścieżki.



RYSUNEK 1.8.
Tworzenie projektu
Conversion dla
pierwszego programu

10. Kliknij *Next*.

Następne okno dialogowe pozwala na modyfikację nazwy pliku i ścieżki, gdzie będą zapisywały się skompilowane pliki programu, zarówno wersji do testowania, jak i ostatecznej. Możesz zostawić domyślne wartości.

11. Kliknij przycisk *Finish* (zakończ), aby utworzyć projekt *Conversion*.

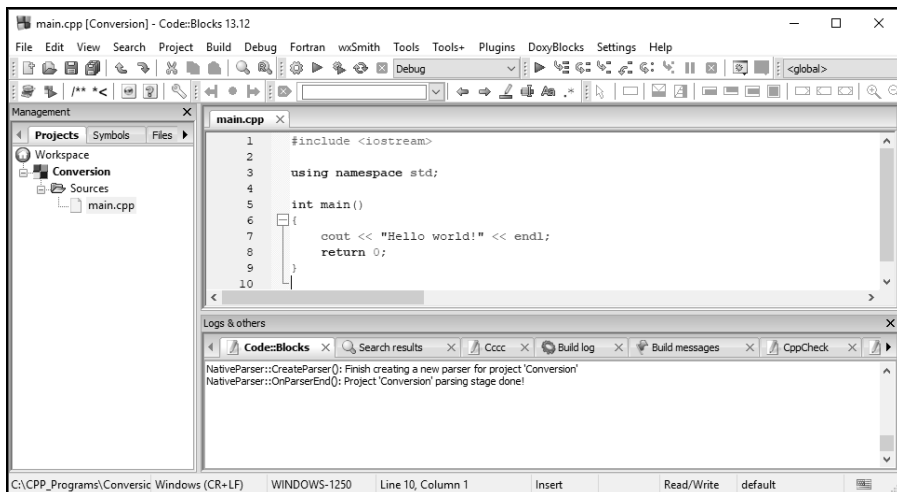
Wprowadzanie kodu w języku C++

Projekt *Conversion*, który stworzył Code::Blocks, składa się z pojedynczego pliku *main.cpp*, wyświetlającego komunikat Hello, world. Teraz wprowadź kod programu:

1. Kliknij dwukrotnie plik *main.cpp*, który znajduje się w oknie dialogowym *Management* (zarządzanie) po lewej stronie. Plik jest zagnieżdżony w węzłach *Conversion/Sources*.

Jak pokazano na rysunku 1.9, Code::Blocks otwiera w edytorze kodu domyślny program *main.cpp*.

RYSUNEK 1.9.
Okno dialogowe
Management
wyświetla strukturę
katalogów wszystkich
dostępnych
programów



2. Zastąp zawartość pliku `main.cpp` kodem podanym poniżej.

Nie przejmuj się zbytnio wcięciami lub odstępami — nie ma znaczenia, czy dana linia jest wcięta na dwie spacje, czy na trzy. Tak samo nie ma znaczenia, czy między słowami jest jedna spacja, czy są dwie spacje. Musisz się upewnić, że wszystkie litery są małe, ponieważ kompilator C++ rozróżnia wielkość liter.

Możesz użyć ściągiawki dostępnej w plikach na stronie www.dummies.com/extra/plusplus, tak jak to zostało opisane w następnym punkcie.



WSKAZÓWKA

```
//  
// Conversion — Program do konwersji temperatury ze  
// Stopni Celsjusza na stopnie Fahrenheita:  
// Fahrenheit = Celsjusz * (212 - 32)/100 + 32  
//  
#include <cstdlib>  
#include <cstring>  
#include <iostream>  
using namespace std;  
int main(int nNumberOfArgs, char* pszArgs[])  
{  
    // wprowadź temperaturę w stopniach Celsjusza  
    int celsius;  
    cout << "Wprowadź temperaturę w stopniach Celsjusza:";  
    cin >> celsius;  
    // dokonaj konwersji stopni Celsjusza  
    // na stopnie Fahrenheita  
    int factor;  
    factor = 212 - 32;  
    // użyj przelicznika do przeliczenia stopni Celsjusza  
    // na stopnie Fahrenheita  
    int fahrenheit;  
    fahrenheit = factor * celsius/100 + 32;
```

```

// wyświetl wyniki (a następnie znak przejścia do nowego wiersza)
cout << "Wartość w stopniach Fahrenheita to:";
cout << fahrenheit << endl;
// Aby pozwolić użytkownikowi zobaczyć wyniki programu,
// poczekaj, aż użytkownik będzie gotowy przed zakończeniem programu.
cout << "Naciśnij Enter, aby kontynuować..." << endl;
cin.ignore(10, '\n');
cin.get();
return 0;
}

```

3. Wybierz *File/Save*, aby zapisać plik źródłowy.

Może nie wydaje Ci się to ekscytujące, ale właśnie stworzyłeś swój pierwszy program C++!

Korzystanie ze ściągi

Wszystkie programy w książce są dostępne online wraz z plikami projektu do ich tworzenia. Aby z nich skorzystać, musisz je pobrać i zainstalować. Następnie postępuj zgodnie z poniższą procedurą:



WSKAZÓWKA

Poniższe instrukcje dotyczą systemu Windows. Czynności, które należy wykonać dla Linuksa czy Macintosha, są bardzo podobne.

1. **Otwórz przeglądarkę internetową.**
2. **Przejdź na stronę** <https://www.dummies.com/store/product/C-For-Dummies-6th-Edition.↪productCd-0470317264,navId-322468,descCd-DOWNLOAD.html>.
3. **Kliknij link do C++ Code files.**
Może wyświetlić się okno dialogowe z pytaniem, gdzie zapisać pobrany plik.
4. **Kliknij *Zapisz plik*.**
System operacyjny skopiuje plik *CPP_programs.zip* do wskazanej w poprzednim kroku lokalizacji lub do domyślnej lokalizacji pobierania.
5. **Kliknij prawym przyciskiem myszy plik *CPP_programs.zip* i wybierz *Otwórz*.**
Otworzy się okno dialogowe zawierające pojedynczy katalog *CPP_Programs*.
6. **Skopiuj ten folder na dysk C.**
Spowoduje to skopiowanie wszystkich plików źródłowych użytych w książce do katalogu *C:\CPP_Programs*.



OSTRZEŻENIE

Możesz umieścić folder *CPP_Programs* w innym miejscu. Pamiętaj, aby nie umieszczać plików źródłowych w katalogu, do którego ścieżka zawiera spację. W systemie Windows uważaj, żeby nie umieszczać plików, których używa Code::Blocks, w folderach takich jak *Moje dokumenty* czy *Pulpit*, ponieważ ścieżka do nich zawiera spację.

Są dwa sposoby korzystania z tych plików. Jednym z nich jest utworzenie programu ręcznie poprzez wykonanie procedury opisanej w tej książce. W razie popełnienia błędu możesz skopiować i wkleić kod z pobranych plików do swojego programu. Jest to sposób zalecany.

Drugie podejście polega na korzystaniu bezpośrednio z pobranych źródeł i pliku projektu:

1. Kliknij dwukrotnie plik *AllPrograms.workspace* w folderze *C:\CPP_Programs*.

Rozszerzenie *workspace* charakteryzuje plik, który odwołuje się do jednego lub większej liczby projektów. Plik *AllPrograms.workspace* zawiera odwołania do wszystkich projektów z tej książki.

2. Kliknij prawym przyciskiem myszy projekt *Conversion* w oknie dialogowym *Management* po lewej stronie. Następnie z wyświetlonego menu kontekstowego wybierz *Activate project*.

Program, który jest aktywny, ma pogrubioną etykietę. W tym przypadku jest to projekt *Conversion*. Gdy wydajesz polecenie budowania — *Build*, *Code::Blocks* zawsze tworzy aktywny projekt.

3. Aby otworzyć plik w edytorze, kliknij dwukrotnie plik *main.cpp*.

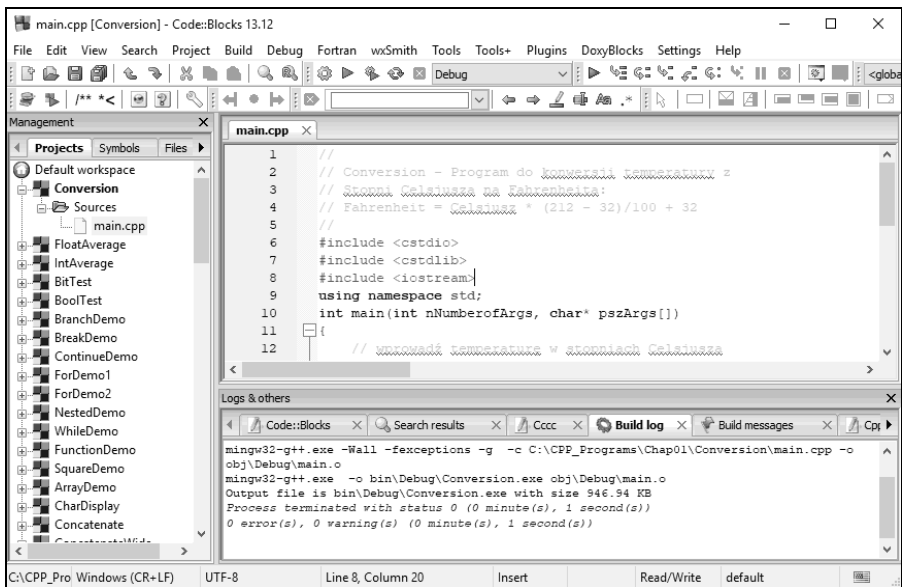
Problem z tym podejściem polega na tym, że jeśli sam nie wprowadzasz kodu w C++, nie uczysz się języka zbyt skutecznie.

Budowanie programu

Po zapisaniu pliku źródłowego w C++ na dysku twardym nadszedł czas na wygenerowanie wykonywalnych instrukcji maszynowych.

Aby zbudować program *Conversion*, wybierz polecenie *Build/Build* z paska poleceń lub wciśnij *Ctrl+F9*. Niemal natychmiast *Code::Blocks* rozpocznie proces kompilacji programu. Jeśli wszystko pójdzie dobrze, w prawym dolnym oknie dialogowym wyświetli się komunikat z liczbą błędów i liczbą ostrzeżeń równymi 0. Przykład przedstawiono na rysunku 1.10.

Jeśli *Code::Blocks* wykryje jakikolwiek błąd w programie C++ — a błędy kodowania są tak samo powszechne jak piasek na plaży — poinformuje Cię o tym za pomocą stosownego komunikatu. Bez wątplenia napotkasz w przyszłości wiele ostrzeżeń i komunikatów o błędach. Błędy są prawdopodobne nawet podczas pisania tak prostego programu jak *Conversion.cpp*. Aby zobaczyć, jak wygląda proces zgłaszania błędów, zamień w wierszu 16. instrukcję `cin >> celsius;` na `cin >>> celsius;`.



RYSUNEK 1.10.
Code::Blocks buduje program Conversion

Być może dla Ciebie i dla mnie jest to błąd „do wybaczenia”, ale nie dla kompilatora C++. Wybierz *Build/Build*, aby uruchomić proces kompilacji. Prawie natychmiast Code::Blocks umieści czerwony kwadrat obok wiersza z błędem. Komunikat error: expected primary-expression before '>' token w zakładce *Build messages* może Ci nic nie mówić. Usuń dodatkowy znak > i ponownie skompiluj, aby pozbyć się tego błędu.



WSKAZÓWKA

Prawdopodobnie uważasz, że komunikat o błędzie wygenerowany przez przykład jest tajemniczy. Daj sobie trochę czasu — programujesz dopiero jakieś 30 minut. Z biegiem czasu będziesz znacznie lepiej rozumiał komunikaty o błędach generowane przez Code::Blocks i gcc.



OSTRZEŻENIE

Tym razem Code::Blocks potrafił wskazać błąd, ale nie zawsze mu się to udaje. Czasami błąd zostaje zauważony dopiero w którymś z kolejnych wierszy. Zatem jeśli wiersz oznaczony jako błędny wygląda dobrze, to aby znaleźć błąd, sprawdź wiersze poprzednie.

Uruchamianie programu

Nadszedł czas, aby sprawdzić, jak działa Twoje nowe dzieło, czyli uruchomić program. Teraz uruchomimy plik programu *Conversion* i wprowadzimy dane wejściowe, aby zobaczyć, jak świetnie działa.

Aby uruchomić program *Conversion* w Code::Blocks w systemie Windows, wybierz *Build/Build and Run* lub naciśnij *F9*. Jeśli coś się zmieniło, program ponownie zostanie skompilowany, a następnie, jeśli kompilacja się powiedzie, uruchomi się.

Na początek otworzy się okno dialogowe proszące o podanie temperatury w stopniach Celsjusza. Wprowadź znaną temperaturę, np. 100 stopni. Po naciśnięciu klawisza *Enter* program zwróci temperaturę 212 stopni Fahrenheita, która jest równoważna temperaturze 100 stopni Celsjusza. Oto przykład:

```
Wprowadź temperaturę w stopniach Celsjusza:100
Wartość w stopniach Fahrenheita to:212
Naciśnij Enter, aby kontynuować...
```

Wiadomość Naciśnij Enter, aby kontynuować... daje możliwość przeczytania wpisanego tekstu, zanim zniknie. Naciśnij *Enter*, a okno dialogowe (wraz z zawartością) zniknie. Gratulacje! Właśnie napisałeś, zbudowałeś i uruchomiłeś swój pierwszy program w C++.

Zauważ, że Code::Blocks nie jest przeznaczony do tworzenia programów okienkowych, takich jak te używane w systemie Windows. Napisanie aplikacji okienkowej dla Windows przy użyciu Code::Blocks nie jest łatwe, ale możliwe. Znacznie prościej jest stworzyć aplikację okienkową w Visual Studio.

Programy okienkowe Windows pokazują użytkownikowi dane w sposób czytelny wizualnie. Są one estetycznie rozmieszczone w oknach na ekranie. *Conversion.exe* to 32-bitowy program, który jest wykonywany w systemie Windows, ale nie jest programem Windows w sensie wizualnym.

Nie przejmuj się, jeśli nie wiesz, co oznacza program 32-bitowy. Jak wspomniałem wcześniej, ta książka nie dotyczy pisania programów okienkowych w systemie Windows. Programy C++, które piszesz w tej książce, mają *interfejs wiersza poleceń* wykonywalny w oknie wiersza poleceń.

Jeśli jesteś początkującym programistą aplikacji okienkowych dla Windowsa, nie przejmuj się — nie zmarnowałeś pieniędzy. Nauczenie się C++ jest niezbędne przed przystąpieniem do pisania programów dla Windowsa. Moim zdaniem powinieneś uczyć się programowania w następującej kolejności: najpierw C++, potem Windows.

Przeglądanie programu z komentarzami

Wprowadzanie danych do czyjegós programu jest równie ekscytujące jak oglądanie kogoś, kto jeździ samochodem. Aby było to ekscytujące, musisz sam zająć miejsce za kierownicą. Programy są trochę podobne do samochodów. Wszystkie samochody są w zasadzie takie same, z małymi różnicami. Być może francuskie samochody różnią się trochę bardziej od innych, ale wyjątek potwierdza regułę. Samochody są stworzone według tego samego wzoru — kierownica z przodu, fotel pod kierowcą, dach nad kierowcą i tak dalej.

Podobnie wszystkie programy C++ mają wspólny wzorzec. Ten wzorzec jest już obecny w naszym pierwszym programie. Aby znaleźć elementy wspólne dla wszystkich programów, spróbujmy spojrzeć na program *Conversion*.

Analiza struktury programów w C++

Każdy program w C++ opisany w tej książce opiera się na tej samej podstawowej strukturze. Wygląda ona podobnie jak w tym przykładzie:

```
//  
// Szablon — zapewnia szablon do użycia jako punkt wyjścia  
//  
// następujące dołączone pliki definiują większość  
// funkcji potrzebnych każdemu programowi  
#include <cstdlib>  
#include <cstdliblib>  
#include <iostream>  
using namespace std;  
int main(int nNumberOfArgs, char* pszArgs[])  
{  
    // Twój kod C++ zaczyna się tutaj  
    // aby pozwolić użytkownikowi zobaczyć wyniki programu,  
    // poczekaj przed zakończeniem programu, aż użytkownik będzie gotowy  
    cout<< "Naciśnij Enter, aby kontynuować..." <<endl;  
    cin.ignore(10, '\n');  
    cin.get();  
    return 0;  
}
```

Bez wdawania się we wszystkie nudne szczegóły: wykonywanie programu zaczyna się od kodu zawartego między znakami otwarcia i zamknięcia nawiasu klamrowego, bezpośrednio po wierszu definiującym funkcję `main()`.

Skopiowałem ten kod do pliku o nazwie *Template.cpp*, znajdującego się w głównym folderze *CPP_Programs*.

Opatrywanie kodu źródłowego komentarzami

Pierwsze kilka wierszy w programie *Conversion* wydaje się zwykłym tekstem. Albo ten tekst był przeznaczony dla ludzkich oczu, albo kompilator C++ jest o wiele mądrzejszy, niż mi się wydawało. Te sześć pierwszych wierszy to **komentarze**. Pisząc segment kodu, programista może opisać to, co robi lub myśli, i umieścić to w komentarzu przed nim. Kompilator ignoruje komentarze. Programiści (przynajmniej ci dobrzy) tego nie robią.

Komentarz w C++ zaczyna się podwójnym ukośnikiem (//) i kończy znakiem przejścia do nowego wiersza. W komentarzu możesz umieścić dowolne znaki. Komentarz nie ma ograniczenia długości, ale zazwyczaj nie dodaje się komentarzy, których długość przekracza 80 znaków. W dawnych czasach — przymiotnik „dawnych” jest tu pojęciem względnym — ekrany komputerów były ograniczone do 80 znaków szerokości. Niektóre drukarki nadal mają ustawione 80 znaków jako domyślną długość wiersza wydruku. Obecnie utrzymywanie pojedynczej linii krótszej niż 80 znaków jest po prostu dobrą praktyką. Z tych samych powodów co zawsze: tekst jest łatwiejszy do odczytania, a oczy mniej się męczą.

W czasach maszyn do pisania znak końca wiersza był znany jako *powrót karetki* (ang. *carriage return*). Znak przejścia do nowego wiersza to znak kończący wiersz kodu.



OSTRZEŻENIE

C++ pozwala na stosowanie innego sposobu wprowadzania komentarzy. Polega on na umieszczeniu komentarza pomiędzy znakami /* i */. Wszystkie znaki między tymi dwoma znacznikami są ignorowane przez kompilator. Taka forma komentarzy nie jest jednak powszechnie używana w C++.

Istnienie w C++ (lub jakimkolwiek innym języku programowania) polecenia, które jest specjalnie ignorowane przez komputer, może wydawać się dziwne. Jednak wszystkie języki programowania mają jakiś sposób dodawania komentarzy. Wyjaśnienie tego, co działo się w umyśle programisty, gdy pisał kod, jest bardzo ważne. Myśli programisty mogą nie być oczywiste dla jego kolegi po fachu, który próbuje zastosować lub zmodyfikować program. Nawet ten sam programista, który napisał program, gdy spojrzy na kod kilka miesięcy później i nie znajdzie żadnej wskazówki, może sobie nie przypomnieć, co miał na myśli, tworząc go.

Tworzenie programów z instrukcji C++

Wszystkie programy C++ bazują na tak zwanych instrukcjach C++. W tym punkcie omówiono instrukcje składające się na strukturę programu użyte w programie *Conversion*.

Instrukcja to pojedynczy zestaw poleceń. Prawie wszystkie instrukcje C++, poza komentarzami, kończą się średnikiem (jeszcze jeden wyjątek opisany został w rozdziale 10.). Wykonywanie programu w C++ rozpoczyna się od pierwszej instrukcji po otwierającym nawiasie klamrowym. Kolejne instrukcje są wykonywane po kolei, tak jak zostały wymienione w listingu.

Podczas przeglądania kodu możesz zobaczyć, że w całym programie pojawiają się spacje, tabulatory i znaki nowego wiersza. Po każdej instrukcji występuje nowy wiersz. Wszystkie te znaki nazywane są *białymi znakami* (ang. *whitespace*), ponieważ nie widać ich na monitorze.



WSKAZÓWKA

Białe znaki możesz umieszczać w dowolnym miejscu w swoim programie. Zwiększa to czytelność kodu. Nie możesz jednak umieszczać ich w środku wyrazu:


```
Widzisz o c
o mi chodzi?
```

Chociaż kompilator C++ ignoruje białe znaki, jest bardzo czuły na wielkość liter: `pełnapredkosc` i `PełnaPredkosc` to dwie różne zmienne, które nie mają ze sobą nic wspólnego. Słowo kluczowe `int` jest zrozumiałe, ale kompilator C++ nie ma pojęcia, czym jest `INT`. Wiesz już, co miałem na myśli, mówiąc o szybkich, ale głupich komputerach?

Deklaracja zmiennych

Wiersz kodu zawierający `int celsius;` to deklaracja. *Deklaracja* jest instrukcją, która definiuje zmienną. Zmienna to kontener do przechowywania wartości określonego typu. Zmienna zawiera *wartość*, taką jak liczba lub znak.

Termin *zmienna* wywodzi się z formuł algebry następującego typu:

```
x = 10
y = 3 * x
```

Zmienna `y` jest w drugim wyrażeniu ustawiona na 3 razy `x`, ale co to jest `x`? Zmienna `x` działa jako kontener przechowujący wartość. W tym przypadku wartość `x` wynosi 10. Równie dobrze `x` może równać się 20, 30 lub `-1`. Drugie sformułowanie ma sens bez względu na wartość `x`.

W algebrze nie trzeba rozpoczynać od instrukcji takiej jak `x = 10`. Za to w C++ programista musi zdefiniować zmienną `x`, zanim będzie mógł z niej skorzystać.

W C++ definicja zmiennej składa się z typu i nazwy. Zmienna zdefiniowana w wierszu 11. nazywa się `celsius` i zadeklarowano, że zawiera liczbę typu `int`, czyli liczbę całkowitą (ang. *integer*) (nigdy się nie dowiemy, dlaczego w tym przypadku nie użyto po prostu *integer* zamiast *int*; to jedna z tych rzeczy, z którymi trzeba po prostu żyć).

Dla kompilatora C++ nazwa zmiennej nie ma szczególnego znaczenia. Zmienna musi rozpoczynać się od dużej lub małej litery z zakresu od A do Z lub od znaku podkreślenia (`_`). Wszystkie kolejne znaki muszą być małymi lub dużymi literami, cyframi od 0 do 9 lub znakami podkreślenia. Nazwy zmiennych mogą być dowolnie długie.



WSKAZÓWKA

Konwencja mówi, że nazwy zmiennych powinny się zaczynać od małej litery, a każde nowe słowo w zmiennej powinno się zaczynać od dużej litery, na przykład `mojaZmienna`.



WSKAZÓWKA

Staraj się nazywać zmienne krótko, ale opisowo. Unikaj takich nazw jak `x`, ponieważ widząc `x`, możemy się nie domyślić jego znaczenia. Nazwa zmiennej taka jak `dlugoscSegmentuLinii` dużo lepiej opisuje, co może oznaczać wartość zmiennej.

Generowanie wyjścia programu

Wiersze zaczynające się od `cout` i `cin` są znane jako instrukcje wejścia-wyjścia. Często posługujemy się skrótem „instrukcje I/O” (ang. *input/output*) (programiści uwielbiają skróty i akronimy, tak jak wszyscy inżynierowie).

Pierwsza instrukcja I/O mówi: „Przełącz frazę Wprowadź temperaturę w stopniach Celsjusza: do `cout`” (wymawiane jak angielskie „see-out” — widoczne na zewnątrz). `cout` to w C++ nazwa standardowego urządzenia wyjściowego. W tym przypadku standardowym urządzeniem wyjściowym C++ jest Twój monitor, a dokładniej okno konsoli, w której uruchamiasz program.

Następny wiersz opisuje odwrotne działanie. Mówi: „Pobierz wartość z urządzenia wejściowego do C++ i zapisz ją w zmiennej `celsius` typu `int`”. Standardowym urządzeniem wejściowym w C++ jest zwykle klawiatura. Mamy tutaj analogię C++ do wspomnianego wcześniej wzoru algebraicznego $x = 10$. W pozostałej części programu wartość zmiennej `celsius` to wartość liczbową, którą wprowadził użytkownik.

Obliczanie wyrażeń

Wszystkie programy, oprócz tych najbardziej podstawowych, wykonują jakieś obliczenia. W C++ wyrażenie jest instrukcją, która wykonuje obliczenia. Mówiąc inaczej, wyrażenie jest instrukcją, która ma wartość. Za to *operator* to polecenie, które generuje wartość.

Na przykład w programie *Conversion* — w szczególności w dwóch wierszach oznaczonych jako wyrażenie obliczeniowe — program deklaruje zmienną, a następnie przypisuje jej wartość wynikającą z obliczeń. To polecenie oblicza różnicę liczb 212 i 32; operatorem jest znak minus (-), a całe wyrażenie to 212-32.

Przechowywanie wyników wyrażenia

Język mówiony może być bardzo niejednoznaczny. Jedną z takich niejednoznaczności jest termin *równy*. Zwrot *równa się* może oznaczać, że dwie rzeczy mają tę samą wartość, na przykład „złotówka jest równa stu groszom”. Równość może także oznaczać przypisanie, jak w matematyce, gdy mówisz, że y jest równe 3 razy x .

Aby uniknąć niejednoznaczności, programiści C++ znak równości (=) nazywają **operatorem przypisania**. Mówi on: „wyniki na prawo od znaku przypisania wyrażenia przechowuj w zmiennej po lewej stronie”. Programiści powiedzą: „zmiennej `factor` przypisujemy wartość 212 minus 32.” W skrócie można powiedzieć: „do `factor` podstaw 212 minus 32”.



OSTRZEŻENIE

Nigdy nie mów: „factor jest równy 212 minus 32”. Możesz usłyszeć takie sformułowanie od jakiegoś leniucha, ale my nie jesteśmy leniuchami.

Pozostała część programu Conversion

Drugie wyrażenie w programie Conversion jest nieco bardziej skomplikowane niż pierwsze. To wyrażenie również używa symboli matematycznych — * do mnożenia, / do dzielenia i + do dodawania. Jednak w tym przypadku obliczenia są wykonywane na zmiennych, a nie na stałych.

Wartość zawarta w zmiennej o nazwie factor (która, nawiasem mówiąc, została obliczona jako wynik $212 - 32$) jest mnożona przez wartość zawartą w zmiennej celsius (która została wprowadzona z klawiatury). Wynik jest dzielony przez 100 i sumowany z 32. Końcowy wynik wyrażenia jest przypisany do zmiennej całkowitej fahrenheit.

Kolejne dwa polecenia wyświetlają na standardowym urządzeniu wyjściowym napis: *Wartość w stopniach Fahrenheita to:*, a następnie wartość zmiennej fahrenheit. Wszystko to dzieje się tak szybko, że użytkownik nie wie, że zostało wykonanych tyle różnych czynności.

Ostatnie trzy instrukcje przypominają użytkownikowi o naciśnięciu klawisza *Enter*, a następnie czekają, aż to zrobi. Jest to konieczne, ponieważ w niektórych systemach program może wyświetlić wyniki, a następnie zamknąć okno dialogowe konsoli tak szybko, że nawet nie wiadomo, czy coś się stało.



WSKAZÓWKĄ

W wielu systemach możesz pominąć te trzy wiersze — `Code::Blocks` i tak zachowa otwarte okno dialogowe, dopóki nie naciśniesz *Enter*. Mimo wszystko dodanie tych wierszy kodu nie zaszkodzi.

Końcowa instrukcja `return 0;` zwraca sterowanie do systemu operacyjnego.

Skorowidz

A

- adres, 143
 - tablicy, 158
- algorytm MD5, 443
- alokowanie, 220
 - tablicy obiektów, 220
- ANSI, 140
- argumenty funkcji, 114, 115
 - domyślne, 120
 - main(), 168
 - przekazywane przez konstruktor, 247
- autodekrementacja, 94
- autoinkrementacja, 94, 179
- automatyczny konstruktor
 - domyślny, 253
 - kopiujący, 272

B

- bezpieczeństwo, 323, 403
- bezpieczne
 - kodowanie, 403
 - szyfrowanie, 449
- biblioteka
 - standardowa, 138, 381
 - szablonów, STL, 391, 392
- błędy
 - sposoby zapobiegania, 427
- botnet, 405
- budowanie programu, 42
- bufor, 345
 - przepełnienie, 409, 412, 416, 418

C

- certyfiat cyfrowy, 448
- ciasteczko, cookie, 444
- ciąg znaków, string of characters, 135, 136, 166
- Code::Blocks
 - instalacja w systemie
 - Macintosh, 34
 - Ubuntu Linux, 32
 - Windows, 29
 - kompilator gcc 4.7.1, 28

D

- deassembler, 414
- definiowanie
 - funkcji, 112
 - funkcji składowych, 204
 - prototypów funkcji, 119
 - składowej statycznej, 284
- deklaracje
 - automatyczne, 64
 - wyprzedzające, 232
- deklarowanie
 - destruktorów, 432
 - rozmiaru tablicy, 127
 - statycznych funkcji składowych, 287
 - tablic obiektów, 211
 - tablic wskaźników, 165
 - wskaźników do obiektów, 212
 - zmiennych, 47, 52, 57
- dekomponowanie wyrażeń, 69
- dekonstrukcja obiektu, 241
- dereferencja wskaźnika obiektu, 213
- destruktor, 241, 265, 432

- dodawanie funkcji składowej, 199
- dołączanie
 - plików, 174
 - warunkowe, 183
- dostęp
 - do argumentów, 170
 - do argumentów funkcji main(), 168
 - do elementów chronionych, 231
 - do elementów klasy, 197
 - do pól, 202
- dupleks, 239
- dyrektywy preprocesora, 185
- działanie mechanizmu wyjątków, 362
- dziedziczenie, 293, 295
 - konstruktorów, 301
 - wielokrotne, 369, 371, 434
 - wirtualne, 372
- dostęp do pól, 202
- otwarte, 206
- poza klasą, 206
- przeciążanie, 208
- statyczne, 287
- wirtualne, 303
- wywoływanie, 200
- szablony, 382
- sterowania formatem strumienia, 357
- trygonometryczne, 175
- tworzenie, 109
- używanie, 110
- wirtualne, 308
- wywoływanie, 113
 - z wartością obiektu, 215
 - za pomocą operatora referencji, 217
 - ze wskaźnikiem na obiekt, 216
- z argumentami, 115, 117

F

- faktoryzacja klas, 313
- flagi formatu strumienia wejścia-wyjścia, 350
- funkcja, 109
 - getline(), 137, 346, 355, 421
 - main(), 112, 117, 152
 - dostęp do argumentów, 168
 - operator*(), 399
 - operator=(), 327
 - operator>>(), 338
- funkcje
 - argumenty, 114
 - argumenty domyślne, 120
 - biblioteki standardowej, 138
 - czysto wirtualne, 318
 - definiowanie, 112
 - definiowanie prototypów, 119
 - deklaracja, 112
 - obsługi ciągów znaków, 139
 - proste, 114
 - przeciążanie, 117
 - przekazywanie
 - obiektów, 214
 - przez referencję, 121, 148
 - przez wartość, 121, 147
 - wartości wskaźnikowych, 148
 - składowe, 198–200
 - definiowanie w klasie, 204
 - dodawanie, 199

G

- generowanie wyjścia programu, 48

H

- haker, 404
- hakowanie, 414
- haszowanie, 443
- hierarchia klas, 370

I

- implementowanie listy inicjalizacyjnej, 388
- inicjowanie tablic, 130
- inkrementacja wskaźnika, 161
- instalacja Code::Blocks, 29
- instancja, 320
- instancjonowanie szablonów, 383
 - zewnętrznych, 388
- instrukcja, 46
 - #error, 185
 - #define, 177
 - #if, 183
 - catch, 367
 - for, 95
 - if, 90
 - switch, 104, 166
 - while, 92

instrukcje sterowania przepływem, 89
interfejs ograniczony, 231

K

klasa, 195
 fstream, 342
 ifstream, 340
 istream, 346
 ofstream, 340
 ostream, 346
 string, 420, 422
 metody, 393
 stringstream, 353
klasy
 abstrakcyjne, 318
 przekazywanie, 320
 dostęp do elementów, 197
 dziedziczenie, 297
 faktoryzacja, 313
 konkretne, 320
 ochrona składowych, 227
 ochrona wewnętrznego stanu, 230
 ograniczony interfejs, 231
 szablony, 384
klasyfikacja, 191
kod
 bezpieczeństwo, 403
 podpisywanie, 448
 zaciemnianie, 445
kolejność operacji, 69
komentarze, 44, 45, 430
komunikaty o błędach, 427
konstruktor, 237, 248
 argumenty, 248
 automatyczny, 253
 domyślny, 253
 dziedziczenie, 301
 jako forma konwersji, 266
 kopiujący, 269
 kopiujący automatyczny, 272
 przeciążanie, 250
 przenoszący, 280, 334
konstruowanie
 składowych, 265
 tablic, 265
 dupleksu, 239

obiektów
 dziedziczenia wielokrotnego, 378
 globalnych, 263
 lokalnych, 261
 stacycznych, 262
podklasy, 299
pojedynczego obiektu, 237
składowych klasy, 255
 stałych, 260
 typu złożonego, 255
wielu obiektów, 238
kontener, 226
 list, 397
 typu string, 392
kontrola wersji, 440
konwersja, 266
kopiowanie
 głębokie, 273
 obiektu, 269
 płytkie, 273

L

liczba
 całkowita, 53
 rzeczywista, 54
linker, 174
lista inicjalizacyjna, 388
listy, 221
 iterowanie, 397
 operacje, 222, 400
 przeglądanie, 398
logi, 438
lokalizacja, localization, 140

Ł

łańcuch znaków, 125, 134

M

manipulatory, 356, 357
manipulowanie ciągami znaków, 136
mechanizm
 dziedziczenia wielokrotnego, 370
 obsługi błędów, 361

metody
 klas obsługi strumieni, 346
 klasy string, 393
 klasy-szablonu list, 397

N

nazewnictwo, 64
NTBS, null-terminated byte string, 135

O

obiekt, 197
 cerr, 339
 cin, 339
 clog, 339
 cout, 339
 endl, 351
 wcerr, 339
 wcin, 339
 wclog, 339
 wcout, 339

obiekty
 dekonstrukcja, 241
 dziedziczenia wielokrotnego, 378
 globalne, 263
 lokalne, 261
 statyczne, 262
 strumieni wejścia-wyjścia, 339
 tworzenie, 236
 tymczasowe, 277

obliczenia binarne, 68

obsługa
 awarii, 437
 błędów, 359, 361, 432
 strumieni, 346

ochrona
 programów, 435
 składowych, 227

ograniczenie widoczności składowych, 428

operacje
 jednoargumentowe, 70
 logiczne, 75
 na zmiennych typu int, 79
 na zmiennych
 zmiennoprzecinkowych, 79
 matematyczne, 67
 na listach, 222, 400

określanie kolejności, 69
wskaźnikowe, 159

operator
 !, 76
 !=, 76
 *(), 399
 *, 143, 157
 ::, 203
 ^, 84
 |, 84
 ||, 76
 ~, 84
 <<(), 338
 <=, 76
 =(), 327, 431
 ==, 76
 ->, 214
 >=, 76
 >>(), 338
 AND (&), 84, 85, 143
 indeksu, 333
 NOT (~), 84
 OR (|), 84, 85
 przenoszący, 334
 przypisania, 48, 72, 325, 329
 referencji, 217
 XOR, 86
 zakresu, 203

operatory
 adresu, 143
 bitowe, 84, 86
 jednobitowe, 84
 matematyczne, 68
 wskaźnikowe, 143
 wstawiania, 327
 wyodrębniania, 327

ostrzeżenia, 427

otwieranie plików, 340

P

pętla, loop, 89, 92
 do ... while, 94
 for, 95
 for each, 98
 while, 92

- pętle
 - bazujące na zakresach, 131
 - nieskończone, 98
 - zagnieżdżone, 102
- piaskownica nazw, 176
- pierwszy program, 37
- plik iostream, 338
- pliki
 - otwieranie, 340
 - stałe, 340
 - tryby otwierania, 341
 - ze wskaźnikami, 352
 - źródłowe, 28
- płytkie kopie, 327
- podklasy, 299
 - konstruowanie, 299
 - niszczenie, 301
- podpisywanie kodu, 448
- polecenie, *Patrz instrukcja*
- polimorfizm, 306
- pomoc systemu operacyjnego, 423
- preprocesor C++, 173
 - stałe, 184
- program
 - ConstructSeparateID, 257
 - DaneListyJed, 223
 - DemoOperatoraPrzenoszaczego, 335
 - DemoOperatoraPrzypisania, 329, 333
 - DisplayString, 159
 - DziedziczenieWielokrotne, 370
 - DziedziczenieWirtualne, 376
 - FaktoryzacjaDziedziczeniaWielokrotnego, 373
 - KaskadaWyjątków, 363
 - KonstruktorKopiujący, 270
 - KonstruktorWywołującySieNawzajem, 252
 - KonstruktorZArgumentami, 248
 - KonstruktorZWartDomyslnymi, 251
 - KopiaGłęboka, 276
 - KopiujPlik, 348
 - MacroConfusion, 178
 - MójWektor, 389
 - MoveCopy, 280
 - NiePrzepełnianieBufora2, 418
 - NiestandardowaKlasaWyjatków, 365
 - OtwartaKlasaOszczednosci, 205
 - PłytkieKopiowanie, 274
 - PrzeciazKonstruktor, 250
 - PrzeciazNadpisz, 304
 - PrzepelnienieBufora, 410
 - STLListaStudentow, 400
 - STLString, 395
 - StringStreamDemo, 353
 - StrumienWejscowy, 343
 - StrumienWyjscia, 341
 - StworzSkadowe, 239
 - SzablonMax, 382
 - SzablonWektora, 384
 - WolajSkadoweStatyczne, 288
 - WyjatekSilni, 359
 - WywolajFunkcjeSkadowa, 200
- programowanie obiektowe, 189, 199
- prototyp funkcji, 119
- przeciążanie
 - funkcji, 117
 - funkcji składowych, 208
 - konstruktor, 250
 - operator, 325, 431
 - indeksu, 333
 - przypisania, 329
- przeglądanie
 - listy, 398
 - programu, 44
- przekazywanie
 - obiektów do funkcji, 214
 - przez referencję, 121, 148
 - przez wartość, 121, 147
 - wartości wskaźnikowych, 148
 - wskaźników do funkcji, 147
- przełącznik, switch, 89
- przepełnianie bufora, 345, 409, 412, 416, 418
- przestrzeń nazw, 176

R

- referencja, 148, 218
- rekonstrukcja kolejności konstruowania, 261
- relacja
 - JEST, 313
 - MA, 302
- rozgałęzienie, branch, 89, 90
- rozmiar zmiennej, 141

S

- składowe
 - chronione, 228
 - statyczne, 283
 - definiowanie, 284
 - odwołania, 285
 - użycie, 287
 - używanie, 284
- słowo kluczowe
 - break, 100
 - case, 104
 - catch, 359
 - class, 196
 - const, 149, 249
 - continue, 99
 - default, 254
 - delete, 153, 245, 254
 - endl, 79
 - enum, 181
 - final, 309
 - friend, 231
 - inline, 180
 - new, 153
 - nullptr, 222, 331
 - operator, 326
 - override, 309
 - protected, 227, 230
 - public, 196, 227, 299
 - signed, 57
 - template, 383
 - this, 202, 290
 - throw, 359, 365
 - try, 359
 - typedef, 186
 - unsigned, 57
 - virtual, 307
 - void, 112
- SQL, Structured Query Language, 406
- stała, 58
- stałe preprocesora, 184
- standard ANSI, 140
- sterowanie przepływem, 89
- sterta, 151, 153, 219, 431
- STL, Standard Template Library, 392
- struktura programu, 45

- strumienie
 - czytanie, 348
 - wejścia-wyjścia, 337, 340
 - flagi formatu, 350
 - metody, 346
 - zapisywanie, 348
- system liczb
 - binarnych, 82
 - liczb dziesiętnych, 81
- szablony, 381
 - funkcji, 382
 - klas, 384
 - zewnętrzne, 388
- szybkość obliczeniowa, 56
- szyfrowanie, 449

Ś

- ścieżka do pliku, 61

T

- tabela prawdy
 - dla operatora AND, 85
 - dla operatora OR, 85
 - dla operatora XOR, 86
- tablica, array, 125
 - ciągów znaków, 166
 - inicjowanie, 130
 - konstruowanie, 265
 - obiektów, 211
 - alokowanie, 220
 - deklarowanie, 211
 - rozmiar, 127
 - szeregowanie argumentów, 126
 - użycie wskaźników, 156
 - w tablicy, 132
 - wskaźników, 165
 - zakres, 131
 - znaków, 133
- tryby otwierania, 341
- tworzenie
 - ciągu znaków, 134
 - funkcji, 109
 - instancji, 320
 - obiektów, 236

- płytkich kopii, 327
- projektu, 38
- tablicy znaków, 133
- typ danych, 52
 - bool, 57, 62
 - char, 57, 61
 - char string, 57
 - double, 56
 - float, 54, 57
 - int, 53, 54, 57
 - long double, 57
 - long int, 57
 - long long int, 57
 - short int, 57
 - string, 140
- typy
 - całkowite
 - bez znaku, 57
 - ze znakiem, 57
 - przechowywania zmiennych, 123
 - stałych, 58
 - wyliczeniowe, 181
 - znakowe, 62

U

- uruchamianie programu, 43
- uwierzytelnianie użytkownika, 441
- użycie
 - klasy string, 422
 - podklasy, 299
 - składowych statycznych, 284, 287
 - tablic wskaźników, 165
 - wskaźników, 161

W

- wartości logiczne, 77
- wirtualne
 - destruktory, 432
 - funkcje składowe, 303
- wprowadzanie kodu, 39
- wskaźnik, 141, 155, 218
 - null, 164
 - nullptr, 164

- wskaźniki
 - dereferencja, 213
 - do obiektów, 212
 - inkrementacja, 161
 - porównanie z tablicą, 162
 - użycie, 145, 161
 - w pliku, 352
 - w tablicach, 156, 165
 - wstrzyknięcie kodu, 406, 408
 - wyjątki, 359
 - wyliczeniowy typ danych, 181
 - wyrażanie liczb binarnych, 81
 - wyrażenia, 48
 - dekomponowanie, 69
 - mieszane, 63
 - wywoływanie funkcji, 113
 - składowych, 200
 - z wartością obiektu, 215
 - za pomocą operatora referencji, 217
 - ze wskaźnikiem na obiekt, 216

Z

- zaciemnianie kodu, 445
- zakres, 203
 - tablicy, 131
 - typów liczbowych, 59
 - ważności, 152
- zaokrąglanie liczb całkowitych, 53
- zapobieganie błędom, 427
- zarządzanie
 - formatem, 350
 - pętlami, 99
 - stertą, 431
 - zdalnymi sesjami, 444
- zasady wytwarzania oprogramowania, 439
- zasięg, 151
- zdalne sesje, 444
- zintegrowane środowisko programistyczne, 28
- zmienne, 47
 - automatyczne, 130
 - deklarowanie, 52
 - globalne, 123
 - lokalne, 123
 - logiczne, 62
 - rozmiar, 141

zmienne

stałe, 149

typu, 52

bool, 57, 62

char, 57, 61

char string, 57

double, 56

float, 54, 57

int, 53, 54, 57

long double, 57

long int, 57

long long int, 57

short int, 57

string, 140

zmiennoprzecinkowe, 54

znak

"\n", 352

#, 178, 185

::, 203

ampersand (&), 132

CR, 342

końca wiersza, 342

lewego ukośnika, 61

LF, 342

null, 159, 135

pusty, null character, 134

tyldy, 242

znaki

specjalne, 60

wieloznaczne *.* , 170

zwarcia logiczne, 80

PROGRAM PARTNERSKI

— GRUPY 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
Helion 

Język C++ od podstaw

Bez względu na to, czy jesteś początkującym czy średnio zaawansowanym programistą, który chce doszlifować swoje umiejętności, dzięki *C++ dla bystrzaków* możesz zostać sprawnym programistą. Ten podręcznik, zaktualizowany, uwzględniający zmiany z rozszerzenia do języka C++ z 2014 roku, pomoże Ci zrozumieć klasy, dziedziczenie, pokaże, jak sprawić, by Twój program wykonywał się w sposób bezpieczny — i wiele więcej.

W książce:

- Twój pierwszy program
- Operatory matematyczne i logiczne
- Programowanie obiektowe
- Obsługa błędów i obrona przed hakerami

Jonathan Landaw

jest autorem wielu bestsellerów i artykułów. Programowaniem zajmuje się od ponad 30 lat, obecnie pracuje w korporacji Booz Allen Hamilton dla Departamentu Obrony Stanów Zjednoczonych.

dla
bystrzaków

Zamówienia telefoniczne:



0 801 339900



0 601 339900

septem
septem.pl

Sprawdź najnowsze promocje:
• <http://dlabystrzakow.pl/promocje>
Książki najchętniej czytane:
• <http://dlabystrzakow.pl/bestsellery>
Zamów informacje o nowościach:
• <http://dlabystrzakow.pl/nowosci>

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

Helion

Cena 69,00 zł

ISBN 978-83-283-5987-1



9 788328 359871