helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

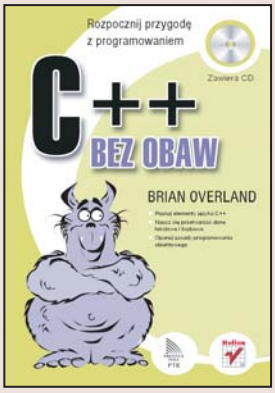
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

C++ bez obaw



Autor: Brian Overland
Tłumaczenie: Łukasz Suma
ISBN: 83-246-0410-3
Tytuł oryginału: C++ Without Fear: A Beginners Guide That Makes You Feel Smart
Format: B5, stron: 520

Rozpocznij przygodę z programowaniem

- Poznaj elementy języka C++.
- Naucz się przetwarzać dane tekstowe i liczbowe.
- Opanuj zasady programowania obiektowego.

C++ to język programowania wykorzystywany do wielu zadań. Programiści stosują go do tworzenia aplikacji użytkowych, gier, a nawet części systemów operacyjnych. Może więc wydawać się, że opanowanie zasad programowania w tym języku przekracza możliwości przeciętnego człowieka. Tak jednak nie jest — programowanie to nie wiedza tajemna, dostępna jedynie dla wybranych, a programiści znający C++ nie są nadludźmi. Z odpowiednią pomocą można bez kłopotów zdobyć wiedzę o wszystkich sztuczkach i ciekawostkach C++, a nauka programowania w tym języku nie powinna sprawić Ci problemów.

Książka „C++ bez obaw" może być Twoim asystentem, gdy będziesz się uczył programowania w języku C++. Dzięki niej przekonasz się, że opanowanie tej sztuki leży w Twoim zasięgu. Znajdziesz w niej omówienie elementów C++ zilustrowane praktycznymi przykładami. Dowiesz się, z czego zbudowany jest program w C++ i jak sprawić, aby kod źródłowy stał się aplikacją możliwą do uruchomienia na Twoim komputerze. Przeczytasz o przetwarzaniu danych, operacjach na plikach, sortowaniu, obliczeniach matematycznych i programowaniu obiektowym. Poznasz wiele przydatnych zagadnień związanych z tworzeniem oprogramowania.

- Struktura programu w C++
- Wyświetlanie danych na ekranie
- Typy danych
- Pętle i instrukcje warunkowe
- Korzystanie ze wskaźników
- Operacje na plikach
- Programowanie obiektowe

Poszerz swoje możliwości, programuj w języku C++

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Wstęp 13
Po co nowa książka o języku C++? 13
Co nowego jest jeszcze w tej książce? 14
Wiele ścieżek nauczania. Co jest dla Ciebie najlepsze? 15
A co, jeśli masz już pewne podstawowe wiadomości na temat programowania? 16
Czego w tej książce nie ma? 16
Dlaczego warto zaczynać naukę języka C++? 17
Aby rozpocząć… 18
Sztuczki i chwyty. Na co powinieneś uważać? 19

Rozdział 1. Twoje pierwsze programy w C++ 21
Myśleć jak programista 21
Komputery robią tylko to, co im każesz 21
Określanie, co ma robić program 22
Pisanie odpowiednich wyrażeń C++ 23
Przegląd kilku specjalistycznych definicji 24
Co wyróżnia język C++? 27
Ciekawostka — Co z Javą i C#? 28
Tworzenie programu w języku C++ 29
Wprowadzanie wyrażeń tworzących program 30
Budowanie programu (kompilowanie i łączenie) 30
Testowanie programu 31
Poprawianie w miarę potrzeby 32
Instalowanie Twojego własnego kompilatora 33
Przykład 1.1. Wyświetlanie komunikatu 33
Jeśli używasz środowiska RHIDE 34
Jeśli używasz Microsoft Visual Studio 35
Jak to działa? 36
Ćwiczenia 37
Ciekawostka — Co z wyrażeniami #include i using? 38
Przechodzenie do kolejnej linii wyświetlania 39
Przykład 1.2. Wyświetlanie wielu linii 39
Jak to działa? 40
Ćwiczenia 41
Ciekawostka — Co to jest łańcuch tekstowy? 42
Przechowywanie danych — zmienne w C++ 42
Wprowadzenie do typów danych 44
Ciekawostka — Dlaczego podwójna precyzja zamiast pojedynczej? 45

Przykład 1.3. Konwertowanie temperatur ... 46
Jak to działa? ... 47
Wariacje na temat przykładu ... 49
Ćwiczenia ... 51
Słowo na temat nazw zmiennych i słów kluczowych ... 51
Ćwiczenie ... 52
Podsumowanie rozdziału 1. ... 52

Rozdział 2. Decyzje, decyzje ... 55
Najpierw jednak kilka słów na temat typów danych ... 55
Ciekawostka — Dla programistów znających C ... 59
Podejmowanie decyzji w programach ... 59
Instrukcje if oraz if-else ... 60
Ciekawostka — Po co dwa operatory (= i ==)? ... 62
Przykład 2.1. Nieparzysty czy parzysty? ... 63
Jak to działa? ... 64
Optymalizowanie programu ... 65
Ćwiczenie ... 66
Podstawy działania pętli ... 66
Ciekawostka — Nieskończone zapętlenie ... 70
Przykład 2.2. Wyświetlanie liczb od 1 do N ... 70
Jak to działa? ... 71
Ćwiczenia ... 72
Prawda i fałsz w C++ ... 72
Ciekawostka — Typ danych bool ... 73
Operator inkrementacji (++) ... 74
Polecenia kontra wyrażenia ... 75
Wprowadzenie do algebry Boole'a (układów logicznych) ... 76
Ciekawostka — Czym jest wartość „prawda" („true")? ... 78
Przykład 2.3. Sprawdzanie wieku ludzi ... 78
Jak to działa? ... 79
Ćwiczenie ... 79
Wprowadzenie do biblioteki math ... 79
Przykład 2.4. Sprawdzanie, czy wartość jest liczbą pierwszą ... 80
Jak to działa? ... 82
Optymalizowanie programu ... 83
Ćwiczenie ... 83
Podsumowanie rozdziału 2. ... 84

Rozdział 3. Wygodna i wszechstronna instrukcja for ... 87
Pętle opracowane w celu zliczania ... 88
Wprowadzenie do pętli for ... 89
Worek z przykładami ... 90
Ciekawostka — Czy pętla for zawsze zachowuje się tak samo jak while? ... 91
Przykład 3.1. Wyświetlanie kolejnych liczb naturalnych od 1 do N przy użyciu pętli for ... 92
Jak to działa? ... 92
Ćwiczenie ... 93
Bloki poleceń w pętli for ... 93
Deklarowanie zmiennych sterujących pętlą „w locie" ... 94
Przykład 3.2. Sprawdzanie, czy wartość jest liczbą pierwszą, z wykorzystaniem pętli for ... 95
Jak to działa? ... 96
Ćwiczenie ... 98
Porównanie języków — instrukcja for w języku Basic ... 99
Podsumowanie rozdziału 3. ... 100

Rozdział 4. Funkcje — wiele jest wywoływanych 103
Pojęcie funkcji 103
Wywołania funkcji i przepływ sterowania w programie 105
Podstawy korzystania z funkcji 107
Krok pierwszy: deklaracja (prototyp) funkcji 108
Krok drugi: definicja funkcji 109
Krok trzeci: wywołanie funkcji 110
Przykład 4.1. Funkcja wyznaczająca liczbę trójkątną 111
Jak to działa? 112
Optymalizowanie programu 113
Ćwiczenia 114
Przykład 4.2. Funkcja sprawdzająca, czy podana wartość jest liczbą pierwszą 115
Jak to działa? 116
Ćwiczenia 118
Zmienne lokalne i globalne 118
Funkcje rekurencyjne 120
Przykład 4.3. Największy wspólny dzielnik (NWD) 121
Jak to działa? 124
Ćwiczenia 125
Przykład 4.4. Rozkładanie liczb na czynniki pierwsze 125
Jak to działa? 127
Ciekawostka — Dla matematycznych oszołomów 129
Ćwiczenia 130
Przykład 4.5. Generator liczb losowych 130
Jak to działa? 131
Ćwiczenie 133
Podsumowanie rozdziału 4. 133

Rozdział 5. Tablice — znamy ich liczbę 135
Pierwsze spojrzenie na tablice w języku C++ 135
Inicjalizowanie tablic 137
Indeksowanie rozpoczynające się od zera 138
Ciekawostka — Dlaczego używać indeksów rozpoczynających się od zera? 139
Przykład 5.1. Wyświetlanie elementów 139
Jak to działa? 140
Ćwiczenia 141
Przykład 5.2. Jak bardzo losowe są liczby losowe? 141
Jak to działa? 144
Ćwiczenia 145
Łańcuchy znakowe i tablice łańcuchów znakowych 146
Przykład 5.3. Krupier nr 1 147
Jak to działa? 149
Ćwiczenie 150
Przykład 5.4. Krupier nr 2 150
Jak to działa? 151
Ćwiczenie 153
Przykład 5.5. Krupier nr 3 153
Jak to działa? 155
Optymalizowanie programu 156
Ćwiczenie 157
Słowo do dociekliwych 158
Tablice dwuwymiarowe — witamy w świecie macierzy 159
Podsumowanie rozdziału 5. 160

Rozdział 6. Wskaźniki — sposób na uchwycenie danych 163
Pojęcie wskaźnika 164
Ciekawostka — Jak naprawdę wyglądają adresy? 165
Deklarowanie i używanie wskaźników 167
Przykład 6.1. Funkcja podwajająca 169
Jak to działa? 170
Ćwiczenia 172
Zamiana wartości, czyli kolejna funkcja wykorzystująca wskaźniki 172
Przykład 6.2. Program sortujący zawartość tablicy 174
Jak to działa? 177
Ćwiczenia 179
Arytmetyka wskaźników 179
Wskaźniki i przetwarzanie elementów tablic 182
Przykład 6.3. Zerowanie elementów tablicy 184
Jak to działa? 184
Optymalizowanie programu 185
Ćwiczenia 187
Podsumowanie rozdziału 6. 187

Rozdział 7. Łańcuchy znakowe — analizowanie tekstu 189
Przechowywanie tekstu na komputerze 190
Ciekawostka — W jaki sposób komputer tłumaczy kody programów? 190
Mam tekst, czy go nie mam? 192
Funkcje pozwalające na działania na łańcuchach znakowych 193
Przykład 7.1. Budowanie łańcuchów znakowych 195
Jak to działa? 196
Ćwiczenia 198
Ciekawostka — Co z tymi sekwencjami specjalnymi? 198
Pobieranie tekstu z wejścia 199
Przykład 7.2. Pobieranie liczby 202
Jak to działa? 203
Ćwiczenie 204
Przykład 7.3. Konwertowanie tekstów na wielkie litery 204
Jak to działa? 205
Ćwiczenia 206
Pojedyncze znaki kontra łańcuchy znakowe 206
Przykład 7.4. Analizowanie danych wejściowych 208
Jak to działa? 209
Ćwiczenia 212
Nowa klasa łańcuchów znakowych w C++ 213
Zapewnienie możliwości obsługi klasy string 214
Deklarowanie i inicjalizowanie zmiennych typu string 215
Korzystanie ze zmiennych typu string 215
Wejście i wyjście 217
Przykład 7.5. Budowanie łańcuchów znakowych za pomocą zmiennych typu string 217
Jak to działa? 218
Ćwiczenia 219
Inne operacje na danych typu string 219
Podsumowanie rozdziału 7. 221

Rozdział 8. Pliki — elektroniczne magazyny 225
Wprowadzenie do obiektów strumieni plików 226
Jak odwoływać się do plików na dysku? 228
Przykład 8.1. Zapisywanie tekstu w pliku 229
Jak to działa? 230
Ćwiczenia 231

Przykład 8.2. Wyświetlanie zawartości pliku tekstowego 232
Jak to działa? 233
Ćwiczenia 234
Pliki tekstowe kontra pliki „binarne” 234
Ciekawostka — Czy pliki „binarne” są w rzeczywistości bardziej binarne od tekstowych? 237
Wprowadzenie do operacji binarnych 238
Przykład 8.3. Bezpośredni zapis danych do pliku 240
Jak to działa? 242
Ćwiczenia 243
Przykład 8.4. Bezpośredni odczyt danych z pliku 243
Jak to działa? 245
Ćwiczenia 245
Podsumowanie rozdziału 8. 246

Rozdział 9. Niektóre z zaawansowanych technik programistycznych 249
Argumenty linii poleceń 250
Przykład 9.1. Wyświetlanie zawartości pliku o nazwie podanej w linii poleceń 251
Jak to działa? 253
Ćwiczenia 253
Przeładowanie funkcji 254
Ciekawostka — Przeładowanie i OOPS 255
Przykład 9.2. Wyświetlanie różnych rodzajów tablic 256
Jak to działa? 257
Ćwiczenie 258
Pętla do-while 258
Wyrażenie switch-case 260
Korzystanie z wielu modułów 261
Obsługa wyjątków 265
Wyjątkom powiedz: „Dzień dobry!” 265
Obsługa wyjątków: próba pierwsza 266
Wprowadzenie do obsługi wyjątków za pomocą instrukcji try-catch 267
Przykład 9.3. Obsługa wyjątków przy obliczaniu NWD 270
Jak to działa? 271
Ćwiczenie 272
Ciekawostka — Czy można korzystać z wielu bloków try-catch? 272
Podsumowanie rozdziału 9. 273

Rozdział 10. Orientowanie się na obiekty 275
Dlaczego orientować się obiektowo? 276
Parser łańcuchów znakowych 277
Obiekty kontra klasy 279
Inny przykład: klasa Ulamek 279
Tworzenie i niszczenie obiektów 280
Dziedziczenie 281
Tworzenie wspólnych interfejsów 283
Polimorfizm — prawdziwa niezależność obiektów 285
Polimorfizm i funkcje wirtualne 287
Ciekawostka — Polimorfizm i tradycyjne języki programowania 288
A co z wielokrotnym wykorzystywaniem kodu? 289
Podsumowanie rozdziału 10. 291

Rozdział 11. Klasa Ulamek 293
Punkt — prosta klasa 293
Ciekawostka — Dla programistów używających języka C — struktury i klasy 295
Teren prywatny — tylko dla swoich, czyli rzecz o ochronie danych 296
Przykład 11.1. Testowanie klasy Punkt 298
Jak to działa? 299
Ćwiczenia 300
Wprowadzenie klasy Ulamek 300
Funkcje inline 303
Znajdowanie największego wspólnego dzielnika 305
Znajdowanie najmniejszej wspólnej wielokrotności 307
Przykład 11.2. Funkcje pomocnicze klasy Ulamek 308
Jak to działa? 309
Ćwiczenia 311
Przykład 11.3. Testowanie klasy Ulamek 311
Jak to działa? 313
Ćwiczenie 314
Ciekawostka — Nowy rodzaj dyrektywy #include? 314
Przykład 11.4. Arytmetyka ułamków — dodawanie i mnożenie 315
Jak to działa? 317
Ćwiczenia 318
Podsumowanie rozdziału 11. 319

Rozdział 12. Konstruktory — jeśli już budujesz... 321
Wprowadzenie do konstruktorów 321
Definiowanie wielu konstruktorów (przeładowanie) 323
Konstruktor domyślny… i związane z nim ostrzeżenie 324
Ciekawostka — Czy w kwestii domyślnego konstruktora język C++ bawi się z Tobą w kotka i myszkę? 326
Przykład 12.1. Konstruktory klasy Punkt 327
Jak to działa? 328
Ćwiczenia 328
Przykład 12.2. Konstruktory klasy Ulamek 328
Jak to działa? 330
Ćwiczenia 331
Referencje do zmiennych i argumenty przekazywane przez referencje (&) 331
Konstruktor kopiujący 333
Ciekawostka — Konstruktor kopiujący i referencje 335
Przykład 12.3. Konstruktor kopiujący klasy Ulamek 336
Jak to działa? 338
Ćwiczenia 338
Podsumowanie rozdziału 12. 339

Rozdział 13. Funkcje operatorowe — zrób to z klasą 341
Wprowadzenie do funkcji operatorów działających na klasach 341
Globalne funkcje operatorowe 344
Poprawienie wydajności działania kodu za pomocą referencji 346
Przykład 13.1. Operatory klasy Punkt 349
Jak to działa? 350
Ćwiczenia 351
Przykład 13.2. Operatory klasy Ulamek 352
Jak to działa? 354
Ćwiczenia 355
Praca z innymi typami danych 355
Funkcja operatora przypisania (=) dla klasy Ulamek 356

Funkcja operatora porównania (==) dla klasy Ulamek ... 358
Ciekawostka — Co z tym typem boolowskim (bool)? ... 359
Funkcja wyświetlająca dla klasy Ulamek ... 359
Przykład 13.3. Kompletna klasa Ulamek ... 361
Jak to działa? ... 364
Ćwiczenia ... 366
Podsumowanie rozdziału 13. ... 366

Rozdział 14. Czym jest „new”? Klasa ParserTekstu ... 369
Operator new ... 370
Obiekty i operator new ... 371
Tworzenie tablic dynamicznych ... 373
Ciekawostka — Radzenie sobie z problemami związanymi z alokacją pamięci ... 375
Przykład 14.1. Dynamiczne przydzielanie pamięci w akcji ... 376
Jak to działa? ... 376
Ćwiczenie ... 377
Projektowanie parsera (analizatora leksykalnego) ... 377
Przykład 14.2. Klasa ParserTekstu ... 382
Jak to działa? ... 384
Poprawianie kodu ... 386
Ćwiczenia ... 387
Podsumowanie rozdziału 14. ... 388

Rozdział 15. Czym jest „this”? Klasa String ... 391
Wprowadzenie do klasy String ... 392
Wprowadzenie do destruktorów klasy ... 393
Przykład 15.1. Prosta klasa String ... 394
Jak to działa? ... 396
Ćwiczenia ... 398
„Głębokie” kopiowanie i konstruktor kopiujący ... 398
Słowo kluczowe this ... 401
Tworzenie operatora przypisania ... 402
Tworzenie funkcji konkatenacji (łączenia) ... 405
Przykład 15.2. Kompletna klasa String ... 407
Jak to działa? ... 409
Ćwiczenia ... 410
Podsumowanie rozdziału 15. ... 411

Rozdział 16. Dziedziczenie. Cóż to za spadek? ... 413
Dziedziczenie, czyli przyjemne z pożytecznym ... 414
Ciekawostka — Dlaczego nazwy klas bazowych poprzedzone są słowem „public”? ... 417
Przykład 16.1. Klasa UlamekZmp ... 418
Jak to działa? ... 421
Ćwiczenia ... 421
Problemy związane z klasą UlamekZmp ... 422
Konstruktory domyślne klas pochodnych ... 423
Konstruktory kopiujące dla klas pochodnych ... 424
Funkcja operatora przypisania dla klas pochodnych ... 424
Dodawanie brakujących konstruktorów ... 424
Rozwiązywanie konfliktów typów z klasą bazową ... 425
Przykład 16.2. Kompletna klasa UlamekZmp ... 426
Jak to działa? ... 426
Ćwiczenia ... 427

Przykład 16.3. Klasa UlamekPrawidl 427
Jak to działa? 429
Ćwiczenia 431
Składniki prywatne (private) i chronione (protected) 432
Przykład 16.4. Elementy składowe będące obiektami — klasa UlamekJedn 434
Jak to działa? 436
Ćwiczenie 438
Podsumowanie rozdziału 16. 438

Rozdział 17. Polimorfizm — niezależność obiektów 441
Inna metoda opracowania klasy UlamekZmp 442
Funkcje wirtualne idą z odsieczą! 444
Ciekawostka — Jaka jest cena za stosowanie funkcji wirtualnych? 445
Przykład 17.1. Zmieniona klasa UlamekZmp 447
Jak to działa? 448
Poprawianie kodu 449
Ćwiczenie 451
„Czysta wirtualność" i inne skomplikowane zagadnienia 451
Klasy abstrakcyjne i interfejsy 453
Dlaczego obiekt cout nie jest prawdziwie polimorficzny? 454
Przykład 17.2. Prawdziwy polimorfizm — klasa Drukowalny 456
Jak to działa? 458
Ćwiczenie 459
Słowo (lub dwa słowa) na zakończenie 460
Słowo na zakończenie zakończenia 461
Podsumowanie rozdziału 17. 463

Dodatek A Operatory oferowane przez język C++ 465

Dodatek B Wbudowane typy danych należące do języka C++ 469

Dodatek C Podsumowanie składni wyrażeń w języku C++ 471
Stałe dosłowne 471
Podstawowa składnia wyrażeń 472
Podstawowa składnia poleceń 473
Struktury kontrolne 473
Specjalne polecenia kontrolne 476
Deklaracje danych 477
Deklaracje funkcji 477
Deklaracje klas 478

Dodatek D Kody znaków ASCII 481

Dodatek E Często używane funkcje biblioteczne 483
Funkcje umożliwiające działania na łańcuchach znakowych 483
Funkcje do konwersji danych 485
Funkcje wykonujące operacje na pojedynczych znakach 486
Funkcje matematyczne 487
Funkcje związane z wyznaczaniem wartości losowych 488

Dodatek F Słownik trudnych terminów używanych w tej książce 489

Skorowidz 501

Rozdział 1.

Twoje pierwsze programy w C++

Naprawdę nie ma się czego obawiać w programowaniu w języku C++! Podobnie jak wszystkie inne języki programowania stanowi on sposób na określenie logicznych i precyzyjnych wskazówek dla komputera, które dotyczą jego działania. Kod C++ możesz oczywiście dowolnie skomplikować, lecz najprostszą metodą nauczenia się tego języka jest rozwiązywanie pewnych podstawowych zadań programistycznych. Jest to też sposób, z którego będziemy korzystać w tej książce.

W kilku pierwszych podrozdziałach dokonam przeglądu podstawowych koncepcji związanych z programowaniem. Jeśli pisałeś już jakieś programy, korzystając z innych języków, możesz pominąć tę część rozdziału lub jedynie pobieżnie się z nią zapoznać. Jeżeli jednak zechcesz się wgłębić w tę treść, obiecuję nie zanudzić Cię na śmierć.

Myśleć jak programista

Programowanie może nie przypominać czynności, które zwykłeś wykonywać na co dzień. W ogólnym zarysie chodzi w nim o to, że powinieneś wydawać komputerowi pewne instrukcje, robiąc to w bardzo logiczny i usystematyzowany sposób.

Komputery robią tylko to, co im każesz

Komputery wykonują tylko te czynności, których od nich wymagasz — to najważniejsza zasada w tej książce, która przyda Ci się, szczególnie jeśli nie miałeś dotąd żadnych doświadczeń z programowaniem. Korzystając z języka programowania, takiego jak C++, Visual Basic, Pascal czy FORTRAN, określasz listę zadań do wykonania i ta właśnie lista stanowi **program**.

Prowadziłem kiedyś praktyczne zajęcia z programowania w położonym w stanie Waszyngton mieście Tacoma, które znane jest chyba tylko z tego, że stanowi najbardziej stresujące miasto Ameryki. Jednym z moich studentów był mały człowieczek noszący słomkowy kapelusz i zniszczone ubrania. Każdego dnia odwiedzał mnie ze stosem egzemplarzy codziennego zestawienia wyników gonitw konnych i przekonywał, że dzięki wpisaniu wszystkich tych informacji do komputera i przewidywaniu za jego pomocą numerów koni wygrywających wyścigi możemy szybko stać się milionerami.

Niestety, sprawy nie mają się tak świetnie i żadna maszyna elektroniczna nie jest w stanie podołać tego typu zadaniu. Oczywiście, każdy komputer potrzebuje informacji, które nazywane są **danymi** dla programu. Musi jednak dodatkowo wiedzieć, co począć z tymi informacjami. Instrukcje opisujące operacje, które mają być przeprowadzane na danych (cele tych działań opiszę w dalszej części rozdziału), nazywane są **kodem** programu.

Określanie, co ma robić program

Aby komputer wykonał dla Ciebie jakiekolwiek zadanie, musisz dokładnie zdefiniować, co właściwie ma on robić.

Jak dotąd miałeś prawdopodobnie okazję uruchamiać na swoim komputerze programy napisane przez innych ludzi, takich jak Bill Gates i jego kumple. Byłeś więc do tej pory tak zwanym **użytkownikiem końcowym**, nazywanym też w skrócie po prostu **użytkownikiem**.

Pisząc własne programy, wzniesiesz się teraz na wyższy szczebel w hierarchii komputerowców. Od teraz to Ty będziesz decydował, co program ma robić. To Ty będziesz sprawiał, że coś się będzie działo.

Pamiętać jednak musisz, że komputer jest skończonym, choć genialnym idiotą i to w dużo większym stopniu niż Dustin Hoffman w filmie *Rain Man*. Nigdy sam nie przewidzi, czego od niego chcesz. Nigdy też nie będzie w stanie podejmować samodzielnych decyzji. Jest do bólu dosłowny i bardzo dokładnie będzie przeprowadzał operacje, które każesz mu wykonać, niezależnie od tego, jak głupie by one nie były. Z tego też powodu musisz być niezwykle skrupulatny w zapisywaniu instrukcji określających działania, o które Ci chodzi.

Może się zdarzyć, że wydasz komputerowi polecenie, które będzie się wydawało zupełnie zrozumiałe dla przeciętnego człowieka, takie jak na przykład „Zmień daną temperaturę wyrażoną w stopniach Celsjusza na temperaturę w skali Fahrenheita". Niestety, nawet to jest dla maszyny zbyt ogólną instrukcją. Musisz zatem określić swe wymagania w bardziej precyzyjny sposób, podając komputerowi kolejne kroki działania, podobne do wymienionych poniżej:

1. Wyświetl komunikat: „Wprowadź temperaturę w stopniach Celsjusza".
2. Pobierz liczbę wprowadzoną za pomocą klawiatury i zapisz ją w zmiennej o nazwie `ctemp`.

3. Skonwertuj temperaturę na skalę Fahrenheita, korzystając z następującego wzoru: `ftemp = (ctemp * 1.8) + 32`.
4. Wyświetl komunikat: „Temperatura w skali Fahrenheita wynosi:”.
5. Wyświetl wartość przechowywaną w zmiennej `ftemp`.

Jeśli to proste zadanie wymaga od Ciebie przejścia przez tak skomplikowaną procedurę, możesz zacząć się zastanawiać nad tym, po co w ogóle trudzić się w ten sposób. Odpowiedź jest prosta: raz napisany program można uruchamiać wielokrotnie. I chociaż pisanie programu zajmuje nieraz sporo czasu, będzie się on zwykle wykonywał z szybkością błyskawicy.

Pisanie odpowiednich wyrażeń C++

Po dokładnym sprecyzowaniu zadania, które program ma wykonywać krok po kroku, powinieneś przystąpić do zapisania odpowiednich instrukcji języka C++. Polecenie jest w C++, ogólnie rzecz biorąc, ekwiwalentem zdania zapisanego w języku polskim — może zatem określać jedną lub większą ilość operacji lub tworzyć jakąś daną, jak przekonasz się, czytając dalszą część niniejszego rozdziału.

Powiedzmy, że chcesz, aby Twój program wykonywał następujące działania:

1. Wyświetlał komunikat: „Temperatura w skali Fahrenheita wynosi:”.
2. Wyświetlał wartość zmiennej `ftemp`.

Kroki te możesz przełożyć na następujące wyrażenia języka C++:

```
cout << "Temperatura w skali Fahrenheita wynosi: ";
cout << ftemp;
```

Pamiętaj, że celem programowania jest wykorzystanie komputera do przeprowadzania serii określonych operacji. Komputery rozumieją jednak jedynie swój własny wewnętrzny język — **kod maszynowy** — który zapisywany jest w postaci ciągów wielu zer i jedynek. W latach 50. zeszłego wieku programiści zapisywali poszczególne instrukcje za pomocą kodu maszynowego, jest to jednak niezwykle trudne i zajmuje bardzo dużo czasu.

Aby ułatwić to zadanie, inżynierowie rozwijający oprogramowanie komputerowe opracowali języki programowania, takie jak FORTRAN, Basic i C, które umożliwiły ludziom tworzenie programów w sposób nieco zbliżony do pisania tekstów w języku angielskim.

Pisanie programu możesz rozpocząć od utworzenia tak zwanego **pseudokodu** — w książce tej będę nieraz korzystał z tej właśnie metody. Pseudokod jest bardzo podobny do języka naturalnego, jednak pozwala na opisanie działań wykonywanych przez program w ścisły sposób, odzwierciedlający przepływ sterowania w programie. Poniżej znajdziesz przykładowy program zapisany w postaci pseudokodu.

Jeżeli a jest większe niż b,

    Wyświetl „a jest większe niż b.”.

W innym przypadku

    Wyświetl „a nie jest większe niż b.”.

Po utworzeniu pseudokodu będziesz już bardzo blisko napisania programu w języku C++. Wszystko, co będziesz musiał w tym celu zrobić, to odnaleźć odpowiednie wyrażenia języka C++ odpowiadające określonym działaniom zapisanym w pseudokodzie i zastąpić ten pseudokod właściwymi poleceniami przy zachowaniu reguł składni języka.

```
if (a > b)
   cout << "a jest większe niż b.";
else
   cout << "a nie jest większe niż b.";
```

Przewagą zapisu za pomocą języka programowania jest to, że podlega on zasadom, które nie dopuszczają żadnych niejednoznaczności. Wyrażenia C++ są tak precyzyjne, że mogą być przetłumaczone na ciąg zer i jedynek kodu maszynowego, bez konieczności jakiegokolwiek zgadywania intencji piszącego.

Nie powinno być dla Ciebie żadnym zaskoczeniem, że języki programowania mają ścisłe reguły określające składnię wyrażeń. Reguły te są bardziej konsekwentne i zwykle również dużo prostsze niż zasady rządzące naturalnymi językami ludzkimi. Od czasu do czasu będę podawał Ci te reguły. Oto na przykład składnia wyrażenia `if-else`:

```
if (warunek)
   polecenie
else
   polecenie
```

Słowa wyróżnione w powyższym zapisie pogrubioną czcionką to **słowa kluczowe**, które muszą się znaleźć w programie dokładnie w takiej postaci, w jakiej zostały zapisane. Słowa zapisane zwykłą czcionką, nazywane również **słowami zastępczymi**, reprezentują elementy, które zależą od Ciebie.

Aplikacja, która tłumaczy wyrażenia języka C++ na kod maszynowy, nazywana jest **kompilatorem**. Na temat kompilatorów dużo więcej informacji znajdziesz w podrozdziale „Budowanie programu w C++”. Najpierw jednak omówmy niektóre kluczowe definicje.

Przegląd kilku specjalistycznych definicji

Chciałbym za wszelką cenę uniknąć specjalistycznego żargonu, ale bądźmy szczerzy — kiedy zaczynasz uczyć się programowania, wchodzisz w świat, który wymaga od Ciebie używania zupełnie nowej terminologii. To trochę tak jak z przysłowiowym wejściem między wrony… Poniżej znajdziesz zatem szereg definicji, które okażą się niezbędne, abyś mógł przetrwać w tym trudnym świecie.

aplikacja

Jest dokładnie tym samym co program, ale widziany z punktu widzenia użytkownika. Aplikacja to program, który jest uruchamiany przez użytkownika w celu wykonania określonego zadania. Edytor tekstu jest zatem aplikacją, podobnie jak przeglądarka internetowa czy program dostępu do bazy danych. Jak przekonasz się niebawem, nawet **kompilator** (patrz niżej) jest aplikacją, jednak jest to aplikacja bardzo szczególnego rodzaju, ponieważ jest ona używana przez programistów. Aby uprościć sprawę — Twój program będzie aplikacją, gdy zostanie napisany, zbudowany oraz przetestowany.

dane

To informacje przechowywane przez program, które mają być przetwarzane lub wyświetlane w czasie jego działania. W najprostszym ujęciu dane zawierają słowa lub liczby, choć mogą też tworzyć bardziej skomplikowane i interesujące struktury danych, zwane „klasami" i „obiektami".

kod

To kolejny synonim słowa „program", ale akcent położony jest tu na punkt widzenia programisty. Kod jest serią wyrażeń zapisanych zgodnie z zasadami składni, używanymi przy tworzeniu programu. Określenie to może się odnosić zarówno do **kodu maszynowego**, czyli ciągu zer i jedynek, jak i **kodu źródłowego**, a więc zestawu instrukcji języka C++. Używanie terminu „kod" wiąże się z czasami, gdy wszyscy programiści pisali swoje programy wyłącznie w kodzie maszynowym. Każda z instrukcji maszynowych jest zapisywana w postaci niepowtarzalnej kombinacji zer i jedynek, dzięki czemu stanowi kod definiujący odpowiednią akcję podejmowaną przez komputer. Pomimo korzystania z języków wysokiego poziomu, takich jak C++, Java, FORTRAN czy Visual Basic, programiści w dalszym ciągu używali słowa „kod". Więcej informacji na ten temat znajdziesz w definicji terminu **kod źródłowy**.

Słowo „kod" bywa również czasami używane do odróżniania pasywnych informacji występujących w programie, czyli jego danych, od części programu, która jest odpowiedzialna za przeprowadzanie działań, czyli właśnie kodu.

kod maszynowy

To naturalny język procesora (CPU). W tym języku każda instrukcja procesora składa się z unikatowej kombinacji (lub **kodu**) zer i jedynek. W dalszym ciągu możliwe jest programowanie w kodzie maszynowym, jednak wymaga to dobrej znajomości poszczególnych kodów instrukcji oraz posiadania dużej wiedzy na temat architektury jednostki centralnej, czyli wszystkiego tego, co nie należy do tematyki niniejszej książki.

Języki programowania, takie jak C++, pozwalają na pisanie programów w sposób podobny do tworzenia tekstów w języku angielskim, ale na tyle precyzyjnie, iż mogą one być tłumaczone na kod maszynowy. Język C++ oferuje również sporo ciekawych możliwości i ułatwień.

kod źródłowy

To program napisany w języku wysokiego poziomu, takim jak właśnie C++. Kod źródłowy składa się z szeregu wyrażeń języka C++, które stanowią właściwy program. Kod taki musi zostać przetłumaczony na kod maszynowy, zanim będzie mógł być wykonany przez komputer.

Kod maszynowy, jak już pisałem, zawiera tylko zera i jedynki, ale jest zwykle zapisywany za pomocą kodu szesnastkowego, zwanego też heksadecymalnym, który stanowi ciąg liczb o podstawie 16. Kod maszynowy wygląda zatem mniej więcej tak:

```
08 A7 C3 9E 58 6C 77 90
```

Trochę trudno stwierdzić, jakie zadanie wykonuje ten kod, prawda? Dopóki nie poznasz kodów wszystkich instrukcji, program taki będzie dla Ciebie zupełnie niezrozumiały i to właśnie stanowi powód, dla którego bardzo niewielu ludzi używa w obecnych czasach kodu maszynowego do tworzenia programów. Kod źródłowy, dla odmiany, przynajmniej w pewnej mierze wykazuje podobieństwo do języka angielskiego, a więc i do innych języków naturalnych. Fragment kodu może na przykład wyglądać tak:

```
if(wynagrodzenie < 0)
   wyswietl_komunikat_bledu();
```

kompilator

Jest to translator języka zamieniający wyrażenia C++ (czyli kod języka C++) na program w postaci kodu maszynowego. Działanie to jest niezbędne, ponieważ sam komputer, a dokładniej jego jednostka centralna (*CPU* — ang. *Central Processing Unit*) jest w stanie zrozumieć jedynie kod maszynowy.

polecenie

Stanowi przeważnie jedną linię programu C++. Wyrażenie w C++ odpowiada z grubsza zdaniu w języku naturalnym, takim jak na przykład język angielski. C++ dopuszcza stosowanie bardzo skomplikowanych struktur językowych składających się z jednego lub z wielu mniejszych wyrażeń, co również przypomina zasady tworzenia zdań w języku angielskim. Większość poleceń języka C++ powoduje wykonanie pojedynczego działania, ale istnieją również takie, które przeprowadzają całe serie operacji.

program

Jest serią instrukcji, które ma wykonać komputer w stosunku do danych wejściowych. Tak jak wspomniałem wcześniej, napisanie programu może zająć sporo czasu, jednak po ukończeniu tej pracy okazuje się zwykle, że wykonuje się on niezwykle szybko i można go nieustannie uruchamiać.

użytkownik

Osoba uruchamiająca program, czyli człowiek, który korzysta z komputera w celu wykonania jakiegoś pożytecznego zadania, takiego jak edytowanie pliku tekstowego, przeczytanie wiadomości e-mail, przejrzenie stron WWW czy też przelanie jakieś kwoty pieniędzy z konta bankowego. Bardziej oficjalna nazwa użytkownika to **użytkownik końcowy**.

W czasie, gdy pracowałem w firmie Microsoft, użytkownik był osobą powodującą większość istniejących na świecie problemów, był jednak również człowiekiem, który płacił wszystkie rachunki i generował wszystkie zyski przedsiębiorstwa. Rozpoczynając projektowanie poważnych programów, musisz bardzo uważnie przyjrzeć się potrzebom użytkownika i spróbować przewidzieć wszystkie potencjalne kłopoty, które mogą mu się przytrafić w związku z korzystaniem z Twojej aplikacji.

Mimo że jako programiście będzie Ci się zdarzać patrzeć z góry na użytkowników, musisz mieć świadomość, że pierwszym użytkownikiem aplikacji jest zawsze… sam programista! Po napisaniu programu będziesz bowiem prawdopodobnie pierwszą osobą, a nieraz i jedyną osobą, która uruchomi program i sprawdzi jego działanie. Pamiętaj więc, że będąc programistą, jesteś też jednocześnie normalnym użytkownikiem.

Co wyróżnia język C++?

Większość rzeczy, które właśnie napisałem o C++, odnosi się równie dobrze do innych języków programowania, takich jak Pascal, Java, FORTRAN czy Basic. Wszystkie one są **językami wysokiego poziomu**, co oznacza, że nie mają bardzo bliskiego związku z kodem maszynowym, lecz używają słów kluczowych, takich jak `if` i `while`, przynajmniej ogólnie przypominających słowa należące do języka angielskiego.

Jeśli jednak wszystkie te języki umożliwiają wykonanie dokładnie tego samego zadania, polegającego na pisaniu programów w prostszy sposób niż przy wykorzystaniu kodu maszynowego, dlaczego jest ich aż tak wiele?

Każdy z wymienionych tu języków został opracowany w nieco innym celu. Basic, na przykład, zaprojektowano w taki sposób, aby był łatwy w nauce i używaniu. Dzięki temu możliwe jest bardzo swobodne traktowanie składni tego języka, co z kolei prowadzi nieraz do dość niezrozumiałych i błędnych zachowań programów napisanych za jego pomocą. Mimo to firma Microsoft opracowała środowisko Visual Basic, które jest bardzo potężnym i wygodnym narzędziem pozwalającym na szybkie tworzenie aplikacji przeznaczonych dla systemu Windows.

Pascal został zaprojektowany z myślą o środowiskach akademickich, w których wykorzystywany jest w celu nauczania sposobów rozwiązywania złożonych problemów programistycznych. O ile Basic jest językiem szybkim i z bardzo swobodnymi regułami składni, tak język Pascal cechują niezwykła szczegółowość i masa wyrażeń o bardzo skomplikowanej składni. Pascal jest bardzo dobrym językiem, jednak większość programistów woli korzystać z narzędzi, które pozwalają na większą swobodę i nie posiadają tak wielu ograniczeń.

C został początkowo opracowane w celu tworzenia systemów operacyjnych. Podczas gdy jego składnia narzuca konieczność stosowania znacznie bardziej sztywnych struktur językowych (dzięki czemu zapobiega też powstawaniu nieprzewidzianych błędów) niż Basic, jest to równocześnie niezwykle jasny język, który umożliwia używanie skrótów i pozwala na tworzenie bardziej zwięzłych programów. Prosta i jednocześnie

zrozumiała składnia C przez lata zdobyła niezwykłą popularność wśród programistów. Inną zaletą C jest fakt, że język ten nakłada bardzo niewiele ograniczeń, dzięki czemu prawie wszystko, co da się zrobić za pomocą kodu maszynowego, można też osiągnąć przy wykorzystaniu instrukcji języka C.

Dobrze, ale co to ma wspólnego z C++?

Podstawowa różnica między C a C++ polega na tym, że drugi z nich jest dodatkowo wyposażony w możliwości **programowania zorientowanego obiektowo**. Ta metoda programowania została opracowana specjalnie dla potrzeb tworzenia złożonych systemów, takich jak graficzne interfejsy użytkownika oraz środowiska sieciowe. Jako programista korzystający z języka zorientowanego obiektowo będziesz pytał przede wszystkim o następujące kwestie:

1. Jakie są podstawowe rodzaje danych (czyli informacji) występujących w problemie, który ma zostać rozwiązany?
2. Jakie operacje mają być zdefiniowane dla każdego rodzaju danych?
3. W jaki sposób poszczególne obiekty danych oddziałują na siebie nawzajem?



Co z Javą i C#?

Gdy pod koniec lat 80. ubiegłego wieku zaczynało się intensywnie rozwijać programowanie zorientowane obiektowo, podjęto kilka niezależnych prób opracowania zorientowanej obiektowo wersji języka C. Bjarne Stroustrup powołał do życia pierwszy język tego typu, który zyskał szeroką akceptację w środowisku programistycznym. Język ten, czyli właśnie C++, jest nadal bardzo powszechnie stosowany, czego dowodem jest chociażby istnienie tej książki.

C++ nie jest jednak ostatnim słowem, jakie zostało powiedziane w kwestii projektowania zorientowanej obiektowo wersji języka C. Dwa nowe języki — Java i C# — są tak bardzo zbliżone do C i C++, że noszą miano „opartych na C", ale każdy z nich jest nieco inny.

Istnieje szereg różnic pomiędzy tymi trzema językami. C++ został opracowany w taki sposób, aby w dużej mierze zapewnić wsteczną zgodność z C, a większość programistycznych trików możliwych w C (a wśród nich i takie, których stosowanie jest zdecydowanie odradzane przez dzisiejszych guru informatyki) nadal działa w języku C++, jednak są zupełnie nie do pomyślenia w Javie czy C#.

Java i C# to doskonałe narzędzia przeznaczone do tworzenia aplikacji, jednak w żadnym przypadku nie nadające się do pisania systemów operacyjnych. Mimo że przejęły one znaczną część składni języków C i C++, nie pozwalają na przykład na uzyskanie dostępu do dowolnych adresów pamięci operacyjnej. Niektórzy ludzie uważają również, że stanowią one czystsze implementacje idei programowania obiektowego niż C++.

Wbrew tym opiniom różnice pomiędzy składnią Javy i C# nie są zbyt wielkie. Powołana do życia przez firmę Sun Microsystems Java jest językiem niezależnym od platformy, zaś C# został opracowany przez Microsoft dla potrzeb zaprojektowanej przez tę firmę platformy .NET. Główna różnica wiąże się więc tak naprawdę ze stosowaną platformą i używanymi dodatkowymi bibliotekami funkcji.

Mimo że kilka razy wspomniałem tu o C++ jako o języku używanym do pisania systemów operacyjnych, możesz, oczywiście, bez przeszkód stosować go do tworzenia dowolnych aplikacji biznesowych, gier komputerowych i programów do prywatnego użytku. Język ten oferuje większą swobodę niż spora część pozostałych języków programowania, w tym również swobodę w generowaniu błędów niskiego poziomu. To właśnie dlatego próbuję pokierować Cię tak, abyś ominął te wszystkie ewentualne pułapki.

Dobrą informacją będzie dla Ciebie z pewnością również to, że gdy nauczysz się już C++, przejście do programowania w Javie czy C# powinno okazać się bardzo łatwą sprawą. C++ jest także łatwiejszym do opanowania językiem dla osób, które mają już jakieś doświadczenie w programowaniu w C.

Zauważyłem, że w procesie uczenia się programowania zorientowanego obiektowo bardzo pomaga wcześniejsze solidne opanowanie składni podstawowych wyrażeń. Z tego też powodu nie koncentruję się na orientacji obiektowej aż do rozdziału 10.

Wprowadzam jednak pewne obiekty — czyli fragmenty danych, z którymi łączą się określone operacje — już na początku tej książki. W tym rozdziale używam na przykład obiektu danych `cout`, który nie stanowi elementu języka C. W klasycznym C w celu wyświetlenia informacji na ekranie musiałbyś wywoływać specjalną funkcję składającą się ze zdefiniowanej wcześniej serii wyrażeń. Korzystając z obiektu `cout`, wysyłasz po prostu dane do obiektu, który — w dosłownym znaczeniu — **wie, w jaki sposób** wyświetlić te informacje.

Zamiast więc zastanawiać się: „Wywołam funkcję, która wyświetli ten tekst na ekranie”, powinieneś raczej pomyśleć: „Prześlę ten tekst do obiektu `cout`, który reprezentuje wyjście konsoli, i niech już on martwi się o to, jak wyświetlić moje dane”.

Taki sposób działania pozwala na o wiele sprawniejsze przeprowadzanie wszelkiego typu operacji z kilku powodów, z których niektóre są bardziej, a inne mniej oczywiste. W szczególnym przypadku korzystania z obiektu `cout` (czyli obiektu wyjścia konsoli) objawia się to tym, że obiekt ten wie, jak poradzić sobie z wyświetlaniem różnych rodzajów danych i — co jeszcze istotniejsze — wiedza ta może być poszerzona o dowolne nowe typy danych, jakie tylko zechcesz utworzyć. Programowanie zorientowane obiektowo nie narzuca na Ciebie zatem ograniczeń związanych z określonym zestawem typów i formatów danych, jak miało to miejsce w przypadku starego podejścia stosowanego w języku C.

Odpowiedź na pytanie, co tak naprawdę oznacza przesyłanie poleceń do obiektu i czym różni się ono od stylu używanego w klasycznym programowaniu strukturalnym, jest jednym z głównych tematów tej książki i osią, wokół której krąży treść jej całej drugiej połowy.

Tworzenie programu w języku C++

Pisanie programu jest w rzeczywistości tylko pierwszym krokiem w procesie tworzenia aplikacji. W kolejnych punktach opiszę pokrótce wszystkie etapy, przez które musisz przejść, aby opracować prawdziwy program.

Wprowadzanie wyrażeń tworzących program

Aby napisać program w języku C++, a właściwie również w każdym innym języku programowania, musisz dysponować jakąś metodą wprowadzania wyrażeń składających się na aplikację. Istnieje co najmniej kilka sposobów wykonania tego zadania, a zaliczyć można do nich na przykład takie:

- Możesz skorzystać z edytora tekstowego, takiego jak Microsoft Word lub Notatnik, który jest aplikacją rozprowadzaną wraz z systemem operacyjnym. W istocie dosłownie każdy edytor okaże się wystarczający. Korzystając z tego sposobu, musisz pamiętać o konieczności zapisywania dokumentów — czy raczej plików źródłowych — w czystym formacie tekstowym.
- Możesz wprowadzić tekst, używając zintegrowanego środowiska programisty (*IDE* — ang. *Integrated Development Environment*). Środowisko takie składa się z edytora tekstowego połączonego z innymi przydatnymi narzędziami programistycznymi. Przykładem zintegrowanego środowiska programisty jest Microsoft Visual Studio.

Po wprowadzeniu wyrażeń tworzących program i sprawdzeniu, czy nie występują w nim jakieś błędy, możesz przystąpić do zbudowania programu.

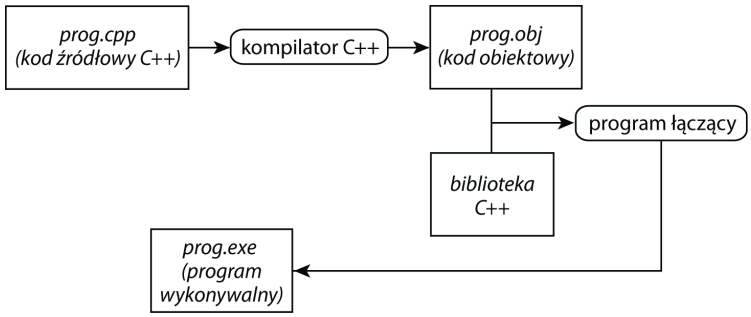
Budowanie programu (kompilowanie i łączenie)

Budowanie programu jest procesem polegającym na skonwertowaniu Twojego kodu źródłowego, czyli wyrażeń języka C++, na aplikację, która może zostać uruchomiona. Jeśli program jest poprawnie napisany, czynność ta jest równie prosta jak naciśnięcie jednego klawisza funkcyjnego. Proces ten w rzeczywistości składa się z dwóch oddzielnych kroków.

Pierwszym z nich jest **kompilacja** programu, która polega na przetłumaczeniu wyrażeń C++ na kod maszynowy, zwany też kodem obiektowym. Jeśli krok ten zostanie pomyślnie przeprowadzony, w kolejnym etapie uruchamiany jest **linker**, czyli **program łączący** lub — jeszcze inaczej — **konsolidujący**, którego zadaniem jest połączenie otrzymanego kodu maszynowego z kodem pochodzącym z biblioteki funkcji C++.

Biblioteka C++, nazywana również w technicznych kręgach biblioteką czasu wykonania, zawiera funkcje, które wywołujesz w celu wykonania typowych czynności. W języku C termin **funkcja** stanowi właściwie synonim słowa „procedura”. W bibliotece znajduje się na przykład standardowa funkcja `sqrt` (co stanowi skrót od słów pierwiastek kwadratowy — ang. *square root*), która została opracowana po to, abyś nie musiał za każdym razem samodzielnie obliczać pierwiastka kwadratowego liczby. Zawiera ona również procedury, których zadaniem jest przesyłanie danych do monitora i które umożliwiają odczytywanie i zapisywanie plików danych na dysku Twojego komputera.

Umieszczony na sąsiedniej stronie rysunek przedstawia sposób działania procesu budowania aplikacji. Pamiętaj, że gdy korzystasz ze zintegrowanego środowiska programisty, czynności te są przez nie wykonywane automatycznie; Twoim zadaniem jest jedynie naciśnięcie odpowiedniego klawisza funkcyjnego.



Jeżeli operacja budowania zostanie zakończona sukcesem, możesz sobie pogratulować. Oznaczać to bowiem będzie, że ani kompilator, ani program łączący nie znalazły żadnych błędów. Ale czy będzie to jednocześnie oznaczać koniec Twojej pracy? Niezupełnie. Kompilator wyłapuje błędy gramatyczne, a więc błędy składni, istnieje jednak bardzo wiele błędów, których nie jest w stanie znaleźć.

Rozważ następującą analogię. Załóżmy, że dane jest poniższe zdanie:

Księżyc jest zrobiony zielonego sera.

Zdanie to nie jest poprawną wypowiedzią w języku polskim. Aby je poprawić, powinieneś dodać w odpowiednim miejscu literę „z". Zdanie przyjmie wtedy postać:

Księżyc jest zrobiony z zielonego sera.

W zdaniu tym nie występują już żadne błędy składniowe. Poprawność gramatyczna nie oznacza jednak jeszcze, że zdanie jest poprawne w szerszym kontekście, bo czy jest ono prawdziwe w sensie logicznym? Oczywiście, nie jest. Aby uczynić podane powyżej zdanie prawdziwym, będziesz musiał w odpowiednim jego miejscu dodać słowo „nie".

Księżyc nie jest zrobiony z zielonego sera.

Z językami programowania jest podobnie. Kompilator C++ sprawdza, czy Twój program jest poprawny składniowo, i jeśli tak nie jest, zwraca informację na temat wiersza, w którym pojawił się błąd. Ważniejszym pytaniem jest jednak to, czy **program będzie się zachowywał właściwie we wszystkich przypadkach, gdy zostanie uruchomiony**, a odpowiedź na to pytanie nie jest już taka oczywista. I ta właśnie kwestia prowadzi nas do następnego kroku tworzenia aplikacji.

Testowanie programu

Po zakończonym sukcesem zbudowaniu programu powinieneś uruchomić go kilkukrotnie w celu sprawdzenia, czy wykonuje on dokładnie te operacje, do których został powołany. W przypadku poważnej aplikacji, a więc programu, który będzie udostępniany lub sprzedawany innym użytkownikom, będziesz musiał przetestować to wielokrotnie. Możesz sobie nie zdawać z tego sprawy, ale duże firmy informatyczne mają w rzeczywistości całe oddziały, których jedynym zadaniem jest właśnie testowanie oprogramowania utworzonego przez przedsiębiorstwo.

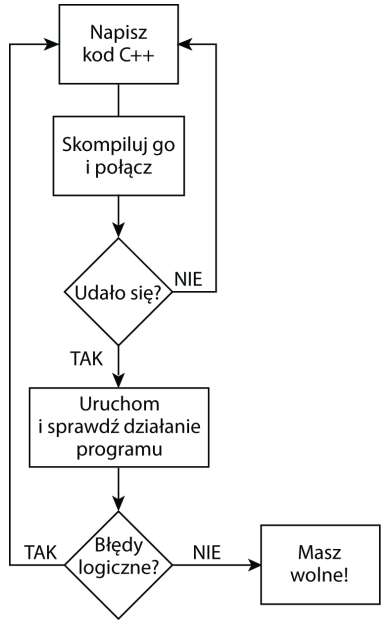
Błędy, których będziesz szukał na tym etapie, nazywane są błędami **logiki programu**. Błędy te będą występowały, gdy napiszesz poprawny składniowo program (co oznacza, że nie będzie na przykład w nim brakowało żadnych przecinków), ale z pewnych względów program ten nie będzie działał tak, jakbyś sobie tego życzył.

Logiczne błędy programu mogą być dużo bardziej nieuchwytne niż błędy składni. Wyobraź sobie, że program wyświetla niewłaściwy wynik lub nagle zatrzymuje swoje działanie bez żadnego wyraźnego powodu. Które wyrażenie spowodowało problem? Odpowiedź nie zawsze jest oczywista. Powodów może być mnóstwo. Jednym z nich może być to, że przyjąłeś pewne założenie, które jest prawdziwe w niektórych przypadkach, lecz w innych nie. Proces testowania programu i znajdowania źródła problemów nazywany jest **debugowaniem** lub nieco mniej ściśle — **uruchamianiem programu**.

Poprawianie w miarę potrzeby

Jeżeli program działa jak należy, masz wolne. Jeśli jednak w programie występują opisane wyżej błędy logiczne, musisz określić źródło problemu, wrócić do etapu pisania i dokonać odpowiednich zmian w kodzie źródłowym C++, a następnie ponownie przebudować program.

W przypadku bardziej skomplikowanych programów konieczne może się okazać wielokrotne przejście przez ten cykl. Program taki może wymagać bardzo wielu testów w celu sprawdzenia, czy działa zgodnie z założeniami we wszystkich możliwych przypadkach. Dopóki nie przeprowadzisz wszystkich niezbędnych testów i poprawek, aplikacja nie może być naprawdę uważana za ukończoną. Jednak w przypadku prostszych programów będziesz zwykle mógł ograniczyć się do niewielkiej liczby testów.



Instalowanie Twojego własnego kompilatora

Jeśli masz swój własny kompilator języka C++, możesz go wykorzystać w celu kompilowania i uruchamiania przykładów, które znajdują się w tej książce. Najlepsze wyniki otrzymasz, używając możliwie najnowszej wersji C++, ale przykłady napisane są w taki sposób, aby działały pod różnymi kompilatorami tego języka.

Możesz również zainstalować kompilator C++ GNU, który znajduje się na płycie CD-ROM dołączonej do niniejszej książki. Jest on darmowym programem typu shareware przeznaczonym do działania w środowiskach MS-DOS. W systemie Windows możesz uruchomić ten kompilator, otwierając najpierw okno poleceń MS-DOS. Masz prawo do swobodnego udostępniania każdego programu opracowanego przy użyciu kompilatora GNU. Na płycie CD-ROM znajduje się również środowisko programisty RHIDE, z którego możesz korzystać do pisania i testowania programów.

Aby zainstalować kompilator, powinieneś odnaleźć plik *PRZECZYTAJ.TXT* znajdujący się w głównym katalogu płyty. CD-ROM zawiera również katalog *Przykładowe kody i odpowiedzi*. W jego podkatalogach umieszczono odpowiedzi do wszystkich ćwiczeń znajdujących się w tej książce.



Płyta CD zawiera także źródłowe pliki kompilatora. Nie musisz ich jednak instalować, przynajmniej dopóki nie stwierdzisz, że jest Ci to do czegoś potrzebne.

Przykład 1.1. Wyświetlanie komunikatu

Aby zacząć programować, otwórz nowy plik źródłowy i wprowadź kod zamieszczony poniżej. Jeżeli używasz środowiska programisty RHIDE, wybierz w tym celu polecenie *New* z menu *File*. Następnie wpisz kod źródłowy dokładnie tak, jak zostało to przedstawione w poniższym przykładzie. Jeśli korzystasz ze środowiska Microsoft Visual Studio, postępuj według wskazówek, które zamieściłem w punkcie „Jeśli używasz Microsoft Visual Studio” znajdującym się na następnej stronie.



Niektóre czynności w Visual Studio będziesz musiał przeprowadzić w nieco inny sposób w RHIDE. Nie powinieneś się więc spodziewać, że kod od razu skompiluje się poprawnie w środowisku VS, dopóki nie zapoznasz się z uwagami zamieszczonymi we wspomnianym wyżej punkcie.

wyswietlanie1.cpp

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

int main(){
   cout << "Nie bój się, C++ jest tutaj!";
   return 0;
}
```

Piąta linia kodu, rozpoczynająca się od nazwy obiektu `cout`, nie musi zawierać dokładnie takiej liczby spacji jak w przedstawionym przykładzie. Również przestrzenie oddzielające poszczególne słowa i znaki, takie jak <<, mogą składać się z dowolnej ilości znaków spacji.

Musisz jednak zwrócić uwagę na kilka szczegółów, z których pierwszym jest sprawa używania wielkich i małych liter. C++ wymaga ścisłej konsekwencji w stosowaniu znaków, dlatego nie powinieneś pisać wielką literą niczego poza tekstami ujętymi w znaki cudzysłowu. Pamiętaj także o umieszczeniu znaku średnika (;) na końcu drugiej, piątej i szóstej linii kodu.

Po wprowadzeniu programu zapisz plik pod nazwą *wyswietlanie1.cpp*, skompiluj go i uruchom. W efekcie tych działań powinieneś zobaczyć na ekranie następujący tekst, jeśli oczywiście nie popełniłeś nigdzie błędu:

```
Nie bój się, C++ jest tutaj!
```

Jeśli używasz środowiska RHIDE

Jeżeli zainstalowałeś bezpłatne środowisko typu shareware, które opisane zostało w poprzednim podrozdziale, i chcesz za jego pomocą skompilować i uruchomić program, oto sposób, z którego powinieneś skorzystać:

1. Zapisz program pod nazwą *wyswietlanie1.cpp*, jeśli dotąd tego nie zrobiłeś. Tak naprawdę możesz wybrać dowolną nazwę pliku, ważne jest, abyś nadał mu rozszerzenie *cpp*. W tym celu w środowisku RHIDE wybierz polecenie *Save* z menu *File*.
2. Naciśnij klawisz *F9*, aby zbudować program.
3. Brak jakiegokolwiek komunikatu o błędzie oznaczać będzie, że program został poprawnie skompilowany i zlinkowany. Gratulacje! Jeśli jednak pojawiły się jakieś komunikaty o błędach, to albo niewłaściwie zainstalowałeś kompilator, albo źle wpisałeś jakąś część kodu przykładu. Wróć do niego i sprawdź, czy poprawnie wprowadziłeś każdy znak, w tym również znaki przestankowe, dokładnie tak, jak zostało to pokazane w przykładzie.
4. Po udanym skompilowaniu przykładu przetestuj go, opuszczając najpierw środowisko RHIDE. W tym celu z menu *File* wybierz polecenie *DOS Shell*.
5. Po przejściu do linii poleceń systemu DOS napisz nazwę programu i naciśnij klawisz *Enter*:

   ```
   wyswietlanie1
   ```

6. Po zakończeniu testowania programu wprowadź poniższe polecenie i naciśnij klawisz *Enter*:

   ```
   exit
   ```



Masz również możliwość uruchamiania programów bezpośrednio w środowisku RHIDE, ale po zakończeniu ich wykonywania sterowanie zostanie natychmiast przekazane z powrotem do RHIDE, z czego wynika, że dopóki Twój program sam nie będzie wstrzymywał swego działania, nie będziesz mógł przekonać się, co pojawia się na wyjściu konsoli. Właśnie dlatego zalecam Ci korzystanie z polecenia *DOS Shell*.

Jeśli używasz Microsoft Visual Studio

Jeżeli do pisania swoich programów używasz środowiska Microsoft Visual Studio, będziesz jeszcze musiał wykonać kilka dodatkowych czynności. Visual Studio stanowi doskonałe narzędzie do pisania programów, ale opracowane zostało głównie z myślą o tworzeniu poważnych aplikacji dla systemu Windows, nie zaś prostych programów. Jednak pisaniem właśnie tych ostatnich musisz się zająć, jeśli C++ jest dla Ciebie czymś zupełnie nowym.

Aby utworzyć program w Visual Studio, będziesz najpierw musiał wybrać odpowiedni typ projektu. **Projekt** w Visual Studio oznacza zbiór plików, które razem tworzą program.

1. Korzystając z menu *File*, wybierz polecenie *New*. Możesz też kliknąć przycisk *New Project*, który powinien być widoczny na pasku narzędzi mniej więcej w połowie szerokości ekranu.
2. W oknie dialogowym pojawiającym się na ekranie wybierz pozycję *Console Application* jako typ projektu i wprowadź nazwę programu, którą w naszym przypadku będzie *wyswietlanie1*, a następnie kliknij przycisk *OK*.
3. Jeśli plik *wyswietlanie1.cpp* nie zostanie otwarty w oknie programu, odszukaj go wśród nazw plików widocznych w lewej części okna i dwukrotnie kliknij jego nazwę.

Przed wprowadzeniem jakiegokolwiek kodu C++ usuń całą zawartość pliku *wyswietlanie1.cpp* z wyjątkiem następującej linii:

```
#include "stdafx.h"
```

Wyrażenie to musi znajdować się w każdej aplikacji uruchamianej w konsoli, czyli w każdym programie nieokienkowym opracowanym za pomocą Visual Studio. Jeżeli czytając tę książkę i korzystając z zawartych w niej przykładów, masz zamiar korzystać ze środowiska Visual Studio, pamiętaj, aby zawsze wstawiać tę linię na początku każdego programu.

Kod pliku *wyswietlanie1.cpp* powinien zatem wyglądać jak poniżej (wraz z dodaną na początku linią, która wyróżniona tu została pogrubioną czcionką):

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main() {
   cout << "Nie bój się, C++ jest tutaj!";
   return 0;
}
```

Aby zbudować program, powinieneś jedynie nacisnąć klawisz *F7*. Spowoduje to uruchomienie zarówno kompilatora, jak i programu łączącego.

Jeśli kod został poprawnie skompilowany i połączony — gratuluję! Oznacza to, że znajdujesz się na dobrej drodze. Jeśli podczas budowania pojawiły się jakieś błędy, powinieneś powrócić do edycji pliku źródłowego i upewnić się, że dokładnie i bezbłędnie wprowadziłeś każdą linię kodu.

Aby uruchomić program, naciśnij kombinację klawiszy *Ctrl+F5*. Istnieją co prawda inne metody uruchamiania programu w środowisku Visual Studio, jednak wymieniony tu sposób stanowi jedyny, który pozwala uniknąć problemu polegającego na tym, że okno MS-DOS pojawia się na ekranie na krótką chwilę i natychmiast znika. Wybranie skrótu *Ctrl+F5* (uruchomienie programu bez debugowania) powoduje wykonanie programu i wyświetlenie na ekranie tekstu `Press any key to continue` (naciśnij dowolny klawisz, aby kontynuować), dzięki czemu masz szansę, aby przyjrzeć się danym, które pojawiły się na wyjściu konsoli.



Jak to działa?

Uwierz w to albo nie, ale ten prosty program zawiera tak naprawdę tylko jedną instrukcję. O pozostałych zawartych w nim wyrażeniach możesz od teraz myśleć jak o składnikach pewnego szablonu, czyli o elementach, które musisz umieścić w każdym programie, ale które możesz spokojnie zignorować przy analizie jego kodu. Jeśli jednak interesują Cię szczegóły, w kolejnej ramce „Na marginesie" znajdziesz dokładniejsze omówienie znaczenia dyrektywy `#include`.

W kodzie znajdującym się poniżej za pomocą pogrubionej czcionki wyróżniono standardowe, niezbędne do działania programu elementy. Na razie nie musisz się martwić tym, dlaczego są one konieczne, po prostu używaj ich. Pomiędzy nawiasami klamrowymi ({}) powinieneś umieścić właściwe linie programu, który w naszym przypadku zawiera tak naprawdę tylko jedną instrukcję.

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

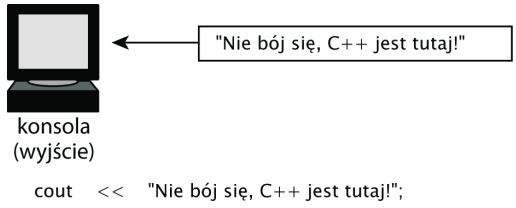
int main() {
   Tutaj_wprowadź_swoje_wyrażenia!
   return 0;
}
```

Jak już wspomniałem, nasz program zawiera tylko jedną prawdziwą instrukcję, którą wprowadziłeś w miejsce piątej linii w znajdującym się powyżej kodzie.

```
cout << "Nie bój się, C++ jest tutaj!";
```

Ale co to jest `cout`? To **obiekt**, pojęcie, które szerzej będę omawiał w drugiej połowie tej książki. Na razie jednak wszystkim, co trzeba Ci na ten temat wiedzieć, jest to, że `cout` oznacza po prostu „wyjście konsoli". Innymi słowy, obiekt ten reprezentuje ekran Twojego komputera. Gdy wyślesz coś na ekran, zostanie to na nim wyświetlone, dokładnie tak, jak można się tego spodziewać.

W języku C++ wyświetlasz dane, korzystając z obiektu `cout` oraz lewostronnego operatora „strumienia” (<<) pokazującego przepływ danych z wartości, którą w naszym przypadku jest łańcuch tekstowy `"Nie bój się, C++ jest tutaj!"`, do konsoli. Używając obiektu `cout`, nigdy nie popełnisz błędu, jeśli tylko należycie zapamiętasz metodę jego działania, korzystając z takiego właśnie sposobu.



Nie zapomnij o średniku (;). Każde polecenie w języku C++ musi być zakończone znakiem średnika, a od reguły tej istnieje tylko kilka wyjątków.

Ze względów technicznych obiekt `cout` musi się zawsze znajdować po lewej stronie, niezależnie od miejsca, w którym będziesz go używał. Dane przepływają w takim przypadku w lewą stronę i powinieneś korzystać ze „strzałek” — tworzonych z dwóch połączonych znaków mniejszości — wskazujących w tym kierunku.

W zamieszczonej poniżej tabeli zebrano inne proste przykłady wykorzystania obiektu `cout`.

Polecenie	Działanie
`cout << "Czy jesteś fanem C++?"`	Wyświetla tekst „Czy jesteś fanem C++?”
`cout << "Halo!"`	Wyświetla tekst „Halo!”
`cout << "Hej tam, na pokładzie!"`	Wyświetla tekst „Hej tam, na pokładzie!”



ĆWICZENIA

Ćwiczenie 1.1.1. Napisz program, który będzie wyświetlał komunikat: „Zabierz się za programowanie!”. Jeśli chcesz, możesz pracować na bazie tego samego pliku źródłowego, który używany był w omawianym wcześniej przykładzie, i zmienić go w celu wykonania zadania. (Wskazówka: zmodyfikuj tylko tekst znajdujący się w cudzysłowie, poza tym wykorzystaj dokładnie ten sam kod programu).

Ćwiczenie 1.1.2. Napisz program, który będzie wyświetlał Twoje imię i nazwisko.



Co z wyrażeniami #include i using?

Napisałem, że pierwszą „prawdziwą” linią programu jest tak naprawdę piąta linia kodu. Co więc oznacza pierwsza linia kodu?

```
#include <iostream>
```

Jest to przykład **dyrektywy preprocesora** C++, czyli ogólnej instrukcji przeznaczonej dla kompilatora języka. Dyrektywa w postaci:

```
#include <nazwa_pliku>
```

powoduje załadowanie deklaracji i definicji stanowiących fragment standardowej biblioteki funkcji języka C++. Bez tego wyrażenia nie mógłbyś używać obiektu `cout`.

Jeśli zdarzyło Ci się korzystać ze starszych wersji C++ i C, możesz się zastanawiać, dlaczego wraz z nazwą pliku nie jest podane jego rozszerzenie, takie jak *h*. Nazwa pliku `iostream` określa **wirtualny** plik nagłówkowy, który zawiera informacje w formie prekompilowanej.

Jeżeli dopiero zaczynasz swoją przygodę z C++, zapamiętaj tylko, że musisz umieszczać dyrektywę `#include`, aby umożliwić sobie korzystanie z określonych fragmentów biblioteki standardowej. W dalszej części, gdy zaczniemy używać funkcji matematycznych, takich jak `sqrt` (pierwiastek kwadratowy), będziesz musiał wprowadzić do pliku dyrektywę włączającą plik nagłówkowy biblioteki funkcji matematycznych:

```
#include <math.h>
```

Czy jest to dodatkowa praca? Tak. Czy C++ mógłby zostać opracowany w taki sposób, aby uniknąć tej konieczności? Może. Istnienie plików nagłówkowych spowodowane jest rozgraniczeniem między samym językiem C a standardową biblioteką czasu wykonania. Doświadczeni programiści C i C++ czasami unikają używania bibliotek lub modyfikują je w celu dostosowania funkcji do swoich potrzeb. Funkcje i obiekty biblioteczne — chociaż konieczne dla początkujących — traktowane są dokładnie w taki sam sposób jak funkcje definiowane przez użytkownika, co oznacza, że muszą być deklarowane, o czym więcej dowiesz się z lektury rozdziału 4. I to właśnie jest głównym zadaniem plików nagłówkowych — uwolnienie Cię od konieczności samodzielnego deklarowania czegokolwiek.

Powinieneś również korzystać z wyrażenia `using`. Pozwala Ci to na bezpośrednie korzystanie z obiektów, takich jak `std::cout`. Bez tego wyrażenia odwołania do obiektów musiałyby mieć następującą formę:

```
std::cout << "Nie bój się, C++ jest tutaj!";
```

Z obiektu `cout` i jego kuzyna, `cin`, będziemy korzystać w tej książce bardzo często, po dobnie jak z innego symbolu należącego do przestrzeni nazw `std`, którym jest `endl`. Zdecydowanie prościej będzie więc nam umieścić wyrażenie `using` na początku każdego programu niż za każdym razem używać pełnych nazw obiektów.

Przechodzenie do kolejnej linii wyświetlania

Zauważyłeś pewnie, że nasz napisany w C++ program nie spowodował automatycznego przejścia do następnej linii tekstu na ekranie. Aby to zrobić, musisz sam wyświetlić odpowiedni znak **nowej linii**, a jeżeli sobie to odpuścisz, cały tekst pojawi się w tym samym wierszu. Wyjątkiem będzie tu sytuacja, gdy mimo braku jakiegokolwiek znaku rozpoczęcia nowej linii, tekst będzie automatycznie zawijany po wypełnieniu bieżącej linii; żałosny wynik tego działania rzadko jednak będzie spełniał Twoje oczekiwania.

Jedną z metod wyświetlenia nowej linii jest skorzystanie z predefiniowanej stałej `endl`. Podobnie jak obiekt `cout`, stała `endl` także należy do przestrzeni nazw `std`:

```
std::cout << "Nie bój się, C++ jest tutaj!" << std::endl;
```

Umieszczenie tego wyrażenia na początku programu:

```
using namespace std;
```

uwolni Cię od konieczności żmudnego podawania przestrzeni nazw, do której należą `cout` i `endl`. Będziesz zatem mógł wprowadzić wyświetlające tekst na ekranie wyrażenie w prostszej postaci:

```
cout << "Nie bój się, C++ jest tutaj!" << endl;
```



Nazwa `endl` jest skrótem od angielskich słów „end line” oznaczających koniec linii, powinna być zatem czytana jako „end ELL”, nie zaś „end jeden”, jak mogłoby Ci się wydawać.

Innym sposobem przejścia do nowej linii jest wstawienie w odpowiednim miejscu znaków `\n`. Znaki te stanowią sekwencję, która w C++ jest interpretowana w szczególny, niedosłowny sposób. Dzięki temu wprowadzenie poniższego wyrażenia odniesie taki sam skutek jak poprzednio:

```
cout << "Nie bój się, C++ jest tutaj!\n";
```

Przykład 1.2. Wyświetlanie wielu linii

Program opisany w tym podrozdziale wyświetla komunikat w kilku kolejnych wierszach. Pamiętaj, aby w czasie wpisywania kodów programów zwracać baczną uwagę na wielkość używanych liter. Tylko zmiana wielkości liter w tekstach ujętych w cudzysłów nie będzie miała znaczenia dla poprawności kompilacji i program w dalszym ciągu będzie działał właściwie.

wyswietlanie2.cpp

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

int main() {
   cout << "Jestem Blaxxon," << endl;
   cout << "komputer o boskiej mocy." << endl;
   cout << "Bój się mnie!" << endl;

   return 0;
}
```

Zapisz program w pliku *wyswietlanie2.cpp*, a następnie skompiluj go i uruchom w taki sam sposób, jak robiłeś to w przypadku programu z przykładu 1.1.



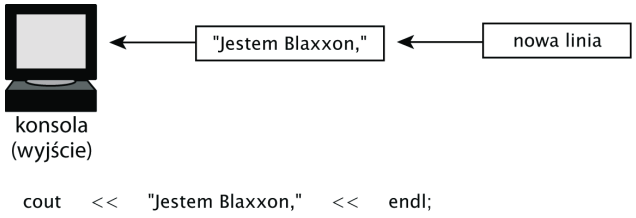
Jak to działa?

Ten przykład jest bardzo podobny do pierwszego przedstawionego przeze mnie kodu. Różnica polega na tym, że wykorzystuje on znaki nowej linii. Bez nich program wyświetlałby następujący tekst:

```
Jestem Blaxxon,komputer o boskiej mocy.Bój się mnie!
```

co z pewnością nie jest efektem, jakiego mógłbyś sobie życzyć.

Sposób działania poszczególnych wyrażeń znajdujących się w tym programie przedstawia ogólnie następujący rysunek:



Korzystając z tej metody, możesz wyświetlić dowolną liczbę oddzielnych elementów. Pamiętaj jednak, że nie pojawią się one w nowych liniach, jeśli nie użyjesz stałej oznaczającej przejście do nowej linii (`endl`). Za pomocą jednego wyrażenia możesz wysłać do konsoli dowolną ilość elementów. Poniższa instrukcja:

```
cout << "To jest " << "fajny " << "program w C++.";
```

spowoduje na przykład wyświetlenie na ekranie następującego tekstu:

```
To jest fajny program w C++.
```

Możesz też wstawić w to zdanie znak nowej linii, jak pokazano niżej:

```
cout << "To jest " << endl << "program w C++.";
```

czego efektem będzie pojawienie się na ekranie następującego tekstu:

```
To jest
program w C++
```

W przykładzie tym, podobnie jak w poprzednim, program zwraca wartość. Zwracanie wartości jest procesem polegającym na zwrotnym wysyłaniu sygnału, w tym przypadku do systemu operacyjnego lub środowiska programisty. Wartość można zwrócić, korzystając z wyrażenia `return`:

```
return 0;
```

Wartością zwracaną przez funkcję `main` jest kod przekazywany systemowi operacyjnemu. Wartość `0` oznacza poprawne zakończenie działania programu. Wszystkie przykłady w tej książce zwracają oczywiście wartość `0`.



Wartości zwracane są o wiele bardziej przydatne w przypadku innych rodzajów funkcji, o których dowiesz się z lektury rozdziału 4. Zwracanie wartości w funkcji `main` jest jedną z tych denerwujących rzeczy, które wydają się początkowo kompletnie bezużyteczne, ale które po prostu **musisz robić**. Zdarza się jednak, że niektóre programy zwracają wartości inne od `0` w celu poinformowania o wystąpieniu określonego problemu. Na razie przyjmij, że instrukcja powodująca zwracanie `0` przez funkcję `main` jest jedną z tych rzeczy, które musisz wstawić do programu, aby mieć pewność, że jest on poprawny. „Dlaczego muszę to robić?", spyta dziecko. „Bo tak powiedziałem", odpowie ojciec.



ĆWICZENIA

Ćwiczenie 1.2.1. Usuń znaki nowej linii z przykładu zaprezentowanego w tym podrozdziale, ale umieść dodatkowe znaki spacji, aby żadne ze słów występujących w zdaniu nie było sklejone z innym. (Wskazówka: pamiętaj, że C++ nie dodaje automatycznie żadnych pustych znaków pomiędzy wyjściowymi łańcuchami tekstowymi). Wynikiem tego działania powinien być następujący tekst:

```
Jestem Blaxxon, komputer o boskiej mocy. Bój się mnie!
```

Ćwiczenie 1.2.2. Zmień przykładowy kod w taki sposób, aby wyświetlał dodatkowe puste wiersze między każdymi dwoma liniami tekstu. Innymi słowy, powinieneś sprawić, aby wynik działania programu był wyświetlany z podwójnym odstępem między wierszami. (Wskazówka: wyświetl dwa znaki nowej linii po każdym fragmencie łańcucha tekstowego).



Co to jest łańcuch tekstowy?

Od samego początku używałem wyłącznie tekstów umieszczonych w cudzysłowie, tak jak na przykład w następującym wyrażeniu:

```
cout << "Nie bój się, C++ jest tutaj!";
```

Wszystko, co znajduje się poza znakami cudzysłowu, jest częścią składni C++. To, co umieszczono między nimi, stanowi dane.

Tak naprawdę wszystkie dane przechowywane w Twoim komputerze mają ostatecznie postać numeryczną. Zależnie jednak od tego, w jaki sposób dane te są używane, mogą być interpretowane jako łańcuchy możliwych do wydrukowania znaków tekstowych. Z tym przypadkiem mamy tu właśnie do czynienia.

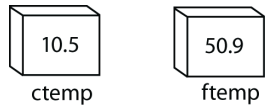
Mogłeś gdzieś kiedyś usłyszeć o kodzie ASCII. Ciąg `"Nie bój się, C++ jest tutaj!"` jest przykładem danych wykorzystujących ten kod. Znaki `'N'`, `'i'`, `'e'`, `' '`, `'b'` i kolejne są przechowywane w postaci oddzielnych bajtów, z których każdy reprezentuje pojedynczy znak możliwy do wydrukowania lub wyświetlenia na ekranie komputera.

W rozdziale 7. napiszę dużo więcej na temat tego typu danych. Ważną rzeczą, którą powinieneś teraz zapamiętać, jest to, że tekst znajdujący się w cudzysłowie jest traktowany jako dana, w przeciwieństwie do komend języka. Dana tego typu uznawana jest za **łańcuch tekstowy** (**znakowy**), który w żargonie informatycznym bywa również określany angielskim terminem **string**.

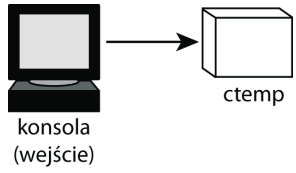
Przechowywanie danych — zmienne w C++

Język C++ nie byłby zbyt użyteczny, gdyby wszystko, co możesz dzięki niemu zrobić, ograniczało się do wyświetlania głupich komunikatów na ekranie komputera. Zadaniem programu jest bowiem przeważnie pobranie skądś nowych danych — „skądś” oznacza zwykle z wejścia konsoli, czyli od użytkownika końcowego — a następnie przeprowadzenie na nich jakichś interesujących działań.

Operacje te wymagają **zmiennych** stanowiących miejsca, w których możesz umieszczać dane. O zmiennych możesz myśleć jako o magicznych pudełkach przechowujących wartości. Program w czasie swojego działania może wedle potrzeby odczytywać, zapisywać lub zmieniać te wartości. Zaprezentowany niżej przykład korzysta ze zmiennych o nazwach `ctemp` i `ftemp` w celu przechowywania wartości temperatur wyrażonych odpowiednio w skali Celsjusza i w skali Fahrenheita.



W jaki sposób wartości mogą znaleźć się w zmiennych? Jedną z metod pobierania danych jest odczytywanie wejścia konsoli. W C++ możesz tego dokonać, korzystając z obiektu `cin`, który — ogólnie rzecz biorąc — reprezentuje właśnie wejście konsoli. Wraz z obiektem `cin` powinieneś używać operatora strumienia, który przedstawia przepływ danych ku prawej stronie.



Oto, co stanie się, w efekcie działania tego polecenia. W rzeczywistości proces ten jest nieco bardziej skomplikowany i wiąże się z nim sprawdzanie bufora wejścia, ale punkty znajdujące się poniżej opisują w ogólny sposób wszystkie operacje, które mają znaczenie w przypadku naszego prostego programu:

1. Wykonywanie programu zostaje zawieszone i oczekuje on na wprowadzenie przez użytkownika liczby.
2. Użytkownik wpisuje liczbę i naciska klawisz *Enter*.
3. Liczba zostaje zaakceptowana i umieszczona (w naszym przypadku) w zmiennej o nazwie `ctemp`.
4. Program kończy swoje działanie.

Jeśli więc dobrze zastanowisz się, co staje się w odpowiedzi na poniższą instrukcję, zobaczysz, że dzieje się całkiem sporo:

```
cin >> ctemp;
```

Zanim jednak będziesz mógł skorzystać ze zmiennej w C++, będziesz musiał ją zadeklarować. Zasada ta ma charakter bezwzględny i jest jedną z cech, które odróżniają C++ od innych języków, takich jak na przykład Basic, który jest dość niedbały w tej kwestii i nie wymaga żadnych deklaracji. Całe pokolenia programistów pracujących w Basicu potrzaskały głowy o swoje terminale, gdy dowiedziały się, że masa błędów w ich programach wynikała ze swobody, z jaką język Basic traktuje kwestie deklarowania zmiennych.



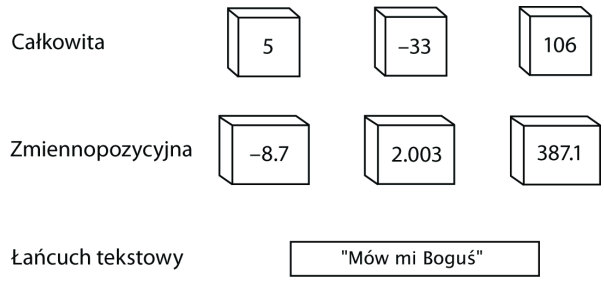
W języku C++ musisz deklarować każdą zmienną, zanim będziesz mógł jej użyć.

Aby zadeklarować zmienną, będziesz najpierw musiał wiedzieć, z jakiego typu danych skorzystać. Kwestia ta ma podstawowe znaczenie w C++, podobnie jak w większości innych języków programowania.

Wprowadzenie do typów danych

Zmienna jest czymś, o czym możesz myśleć jako o magicznym pudełku, w którym umieszczasz informacje lub raczej **dane**. Ale jakiego rodzaju dane mogą się w nim znajdować?

Wszystkie dane są w ostateczności zapisywane przez komputer w postaci cyfrowej, jednak w przypadku każdej z nich stosowany jest jeden z trzech podstawowych formatów: liczby całkowitej (ang. *integer*), liczby zmiennopozycyjnej czy też zmiennoprzecinkowej (ang. *floating-point*) lub łańcucha tekstowego (ang. *text string*).



Istnieje kilka różnic pomiędzy formatem liczb całkowitych i zmiennopozycyjnych. W naszym przypadku, czyli w przypadku osób rozpoczynających naukę programowania, zasada jest jednak prosta:



> Jeżeli zachodzi konieczność przechowywania liczb posiadających część ułamkową, powinieneś skorzystać ze zmiennej zmiennopozycyjnej. W innym przypadku należy użyć zmiennej całkowitej.

Głównym zmiennopozycyjnym typem danych jest w C++ typ `double`. Nazwa ta może się wydawać dość dziwna, ale jest to skrót od angielskiego wyrażenia *double-precision floating point* (liczba zmiennopozycyjna podwójnej precyzji). Istnieje również typ liczb zmiennopozycyjnych pojedynczej precyzji, czyli `float`, ale nie jest on zbyt często używany. Gdy potrzebujesz możliwości zapisywania liczb posiadających części ułamkowe, najlepsze efekty uzyskasz, jeżeli zastosujesz typ `double`, który zapewni Ci równocześnie najmniejsze błędy przechowywanych wartości.

Deklaracja zmiennej typu `double` ma podaną poniżej składnię. Zwróć uwagę na to, że również ta deklaracja zakończona jest znakiem średnika (`;`), tak jak większość wyrażeń w języku C++.

```
double nazwa_zmiennej;
```

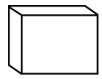
Za pomocą jednej deklaracji możesz też utworzyć wiele zmiennych.

```
double nazwa_zmiennej1, nazwa_zmiennej2, ...;
```

Podane niżej wyrażenie spowoduje na przykład utworzenie zmiennej `double` o nazwie `aZmp`:

```
double aZmp;
```

Deklaracja ta powołuje do życia zmienną typu `double`.



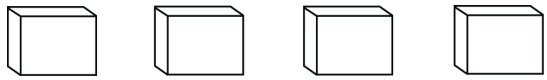
Kolejne wyrażenie wykorzystujące nieco bardziej skomplikowaną składnię pozwala na zadeklarowanie czterech zmiennych `double` o nazwach `a`, `b`, `c` i `kwota`:

```
double b, c, d, kwota;
```

Efekt tej deklaracji jest dokładnie taki sam jak w przypadku zestawu poniższych deklaracji:

```
double b;
double c;
double d;
double kwota;
```

Wynikiem tych deklaracji jest utworzenie czterech zmiennych typu `double`.





Dlaczego podwójna precyzja zamiast pojedynczej?

Zmienne podwójnej precyzji są bardzo podobne do zmiennych pojedynczej precyzji, z tą różnicą, że te pierwsze są lepsze. Podwójna precyzja pozwala na korzystanie z szerszego zakresu liczb i cechuje się większą dokładnością. Tak, to prawda, przy używaniu zmiennych zmiennopozycyjnych możliwa jest utrata dokładności. Do tematu tego powrócę jeszcze w kolejnych rozdziałach tej książki.

Większa dokładność zmiennych podwójnej precyzji jest powodem, dla którego jest to preferowany typ zmiennopozycyjny w języku C++. Przed przeprowadzeniem jakichkolwiek operacji zmiennopozycyjnych wszystkie wartości są konwertowane na liczby podwójnej precyzji, oczywiście tylko w takich sytuacjach, jeżeli nie miały tej formy do tej pory. Również stałe zmiennopozycyjne są w języku C++ przechowywane z podwójną precyzją, chyba że wyraźnie określisz, iż ma być inaczej, na przykład poprzez zapis `12.5F` zamiast zwykłego `12.5`.

Liczby podwójnej precyzji mają tylko jedną wadę — wymagają większej ilości pamięci; w środowiskach komputerów osobistych zapisywane są na przykład za pomocą ośmiu bajtów, w przeciwieństwie do zwykłych wartości zmiennopozycyjnych, które wykorzystują tylko cztery bajty. Nie ma to oczywiście żadnego znaczenia w przypadku prostych programów, ponieważ koprocesory matematyczne są w stanie wykonywać bezpośrednie operacje na danych ośmiobajtowych. Może się to stać problemem w przypadku dużych zbiorów danych zapisanych w postaci wartości podwójnej precyzji, które mają być przechowywane na dysku. Wtedy i tylko wtedy powinieneś zastanowić się nad skorzystaniem z typu zmiennopozycyjnego pojedynczej precyzji, czyli z typu `float`.

Przykład 1.3. Konwertowanie temperatur

Ty z pewnością nie masz zbyt często tego typu problemów, ale każdy obywatel Stanów Zjednoczonych, który przyjeżdża do Europy czy choćby podróżuje po Kanadzie, musi przeliczać w myślach podawane w mediach wartości temperatury wyrażone w stopniach Celsjusza na stosowaną w USA skalę Fahrenheita. Przydatny w takich sytuacjach okazuje się niewielki komputer lub chociaż porządny kalkulator, gdyż takie urządzenia zdecydowanie najlepiej sprawdzają się przy wykonywaniu tego typu zadań.

Oto wzór, który pozwala na przeliczanie temperatury. Znak gwiazdki (*) wykorzystany do „połączenia" dwóch wartości oznacza działanie mnożenia.

```
Fahrenheit = (Celsjusz * 1.8) + 32
```

Dobry program nie będzie jednak jedynie przeliczał jednej wartości temperatury i na tym poprzestawał. Gdyby tak miało być, zdecydowanie prościej byłoby skorzystać z narzędzia Kalkulator oferowanego przez system Windows! Nie, prawdziwie użyteczny program powinien pobierać **dowolną** wartość wejściową określoną w skali Celsjusza i przeliczać ją na odpowiadającą mu liczbę w skali Fahrenheita. Zadanie to wymaga zastosowania pewnych nowych dla Ciebie możliwości:

- pobrania danej wejściowej od użytkownika,
- przechowania jej w zmiennej.

Oto cały program, który spełnia do zadanie. Otwórz nowy plik źródłowy, wprowadź poniższy kod, zapisz plik pod nazwą *konwersja1.cpp*, a następnie skompiluj go i uruchom.

konwersja1.cpp

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

int main() {
     double  ctemp, ftemp;

     cout << "Wprowadź temperaturę wyrażoną w skali Celsjusza i naciśnij klawisz
     Enter: ";
     cin >> ctemp;
     ftemp = (ctemp * 1.8) + 32;
     cout << "Temperatura w skali Fahrenheita wynosi: " << ftemp;

     return 0;
}
```

Program będzie znacznie prostszy do zrozumienia, gdy dodasz do niego komentarze, które w C++ oznacza się podwójnym znakiem ukośnika (//). Komentarze stanowią elementy kodu, które są zupełnie ignorowane przez kompilator języka, co oznacza, że nie mają one żadnego wpływu na sposób działania programu, bardzo przydają się jednak czytającym go ludziom. Poniżej znajdziesz ten sam kod, lecz z wprowadzonymi komentarzami.

konwersja2.cpp

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

int main() {
     double  ctemp, ftemp;

// Prośba o wprowadzenie wejściowej wartości temperatury i zapisanie jej w zmiennej ctemp
// (temperatura w skali Celsjusza).

     cout << "Wprowadź temperaturę wyrażoną w skali Celsjusza i naciśnij klawisz
Enter: ";
     cin >> ctemp;

// Obliczenie wartości zmiennej ftemp (temperatura w skali Fahrenheita) oraz wyświetlenie jej na ekranie.

     ftemp = (ctemp * 1.8) + 32;
     cout << "Temperatura w skali Fahrenheita wynosi: " << ftemp;

     return 0;
}
```

Ta skomentowana wersja kodu jest co prawda dużo prostsza do odczytania i zrozumienia przez człowieka, jednak napisanie jej zajmuje więcej czasu. Wprowadzając kody przykładów znajdujące się w dalszej części tej książki, zawsze będziesz spokojnie mógł pominąć komentarze lub dodać je później. Pamiętaj jednak o podstawowej zasadzie dotyczącej komentarzy:



Kod C++ zaczynający się od dwóch znaków ukośnika (//) stanowi komentarz i jako taki jest zupełnie ignorowany przez kompilator języka aż do samego końca linii, w której się znajduje.

Wprowadzanie komentarzy nie jest wymagane, ale stanowi bardzo dobrą praktykę, szczególnie jeśli ktokolwiek, w tym również Ty sam, ma w późniejszym czasie przeglądać napisany przez Ciebie kod C++.

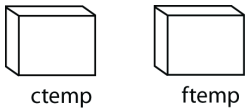


Jak to działa?

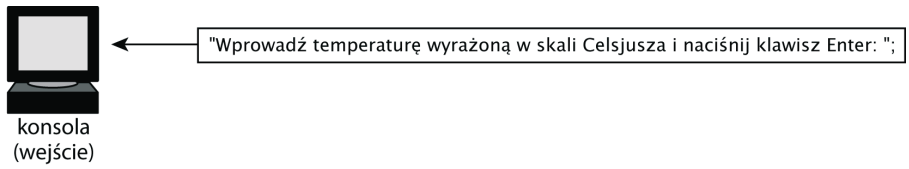
Pierwsze wyrażenie znajdujące się w funkcji `main` to deklaracja zmiennych typu `double` o nazwach `ctemp` i `ftemp`, które przechowują odpowiednie wartości temperatur w skalach Celsjusza i Fahrenheita.

```
double ctemp, ftemp;
```

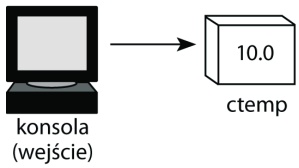
Daje nam to dwa miejsca, w których będziemy mogli przechowywać liczby. Dzięki temu, że zmienne zostały zadeklarowane jako `double`, można w nich umieszczać wartości ułamkowe. Pamiętaj, że `double` oznacza daną zmiennopozycyjną podwójnej precyzji.



Kolejne dwa wyrażenia zachęcają użytkownika do wprowadzenia danej i powodują umieszczenie jej w zmiennej `ctemp`. Załóżmy, że osoba korzystająca z programu podała liczbę `10`. Oznaczać to będzie, że w zmiennej `ctemp` znajdzie się wartość `10.0`.




```
cout << "Wprowadź temperaturę wyrażoną w skali Celsjusza i naciśnij klawisz Enter: ";
```




```
cin >> ctemp
```

We własnych programach możesz korzystać z podobnych wyrażeń w celu wyświetlania komunikatu z prośbą o podanie danych i pobierania do zmiennych wartości wejściowych. Zachęta ta bywa bardzo potrzebna, ponieważ bez niej użytkownik może nie wiedzieć, co i kiedy powinien zrobić.



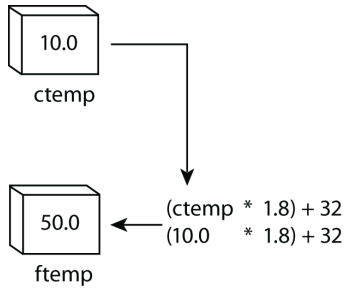
Mimo że liczba została podana w postaci `10`, przechowywana jest jako wartość `10.0`. W czysto matematycznym sensie liczby `10` i `10.0` są sobie równe, jednak w przypadku C++ zapis `10.0` oznacza, że wartość jest przechowywana w postaci zmiennopozycyjnej, nie zaś jako liczba całkowita. To z kolei ma dość istotne konsekwencje, które wytłumaczę Ci w następnym rozdziale.

Kolejne wyrażenie przeprowadza właściwą konwersję, wykorzystując w tym celu wartość zapisaną w zmiennej `ctemp` do obliczenia wartości `ftemp`:

```
ftemp = (ctemp * 1.8) + 32;
```

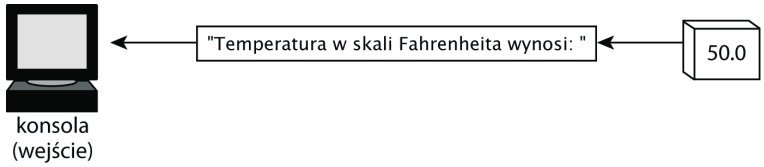
Instrukcja ta zawiera **przypisanie**. Wartość znajdująca się z prawej strony znaku przypisania (=) jest obliczana, a następnie kopiowana do zmiennej widocznej z lewej strony tego znaku. Wyrażenie to jest jednym z najbardziej powszechnych poleceń występujących w programach C++.

Załóżmy ponownie, że wartością podaną przez użytkownika jest liczba 10. Poniższy diagram przedstawia sposób, w jaki dane będą przepływały w programie.




```
ftemp = (ctemp * 1.8) + 32 ;
```

Na koniec program wyświetla wynik swojego działania. W naszym przypadku będzie to liczba 50.




```
cout << "Temperatura w skali Fahrenheita wynosi: " << ftemp ;
```



Wariacje na temat przykładu

Jeśli uważnie przyjrzysz się ostatniemu przykładowi, zaczniesz się zapewne zastanawiać, czy na pewno potrzebne jest deklarowanie dwóch zmiennych i czy tego samego efektu nie dałoby się uzyskać, korzystając tylko z jednej.

Tak się składa, że jest to możliwe. Witamy w świecie optymalizacji. Kolejna wersja przykładu stanowi poprawkę poprzedniej, a różnica między nimi polega na wyeliminowaniu zmiennej ftemp oraz połączeniu kroków konwersji i wyświetlania wyniku w jedno działanie. Sposób ten nie zawsze będzie się równie dobrze sprawdzał, gdyż w bardziej skomplikowanych programach konieczne może się okazać przechowanie obliczonej wartości temperatury w celu dalszych obliczeń. Jednak tutaj działa to wprost idealnie.

konwersja3.cpp

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

int main() {
```

```
// Deklaracja ctemp jako zmiennej zmiennopozycyjnej.

    double  ctemp;

// Prośba o wprowadzenie wejściowej wartości temperatury i zapisanie jej w zmiennej ctemp
// (temperatura w skali Celsjusza).

    cout << "Wprowadź temperaturę wyrażoną w skali Celsjusza i naciśnij klawisz
Enter: ";
    cin >> ctemp;

// Obliczenie wartości zmiennej ftemp (temperatura w skali Fahrenheita) oraz wyświetlenie jej na ekranie.

    cout << "Temperatura w skali Fahrenheita wynosi: " << (ctemp * 1.8) + 32;

    return 0;
}
```

Czy w tym momencie zauważyłeś już pewien schemat działania wiążący się z pisaniem programów? W przypadku najprostszych aplikacji jest on zwykle następujący:

1. Deklaracja zmiennych.
2. Pobranie danych wejściowych od użytkownika po wyświetleniu odpowiedniej zachęty.
3. Przeprowadzenie właściwych obliczeń i wyświetlenie otrzymanych wyników.

Następny program spełnia na przykład zupełnie inne zadanie, ale jego kod powinien Ci się wydawać dość znajomy. Aplikacja ta prosi o podanie liczby, a następnie wyświetla kwadrat jej wartości, czyli liczbę przemnożoną przez samą siebie. Wyrażenia są podobne do instrukcji wchodzących w skład poprzedniego przykładu. Jedyną różnicą jest tutaj tak naprawdę nazwa zmiennej (`n`) oraz konkretne działanie, które jest wykonywane przez program (`n * n`).

kwadrat.cpp

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

int main() {

// Deklaracja n jako zmiennej zmiennopozycyjnej.

    double n;

// Prośba o podanie wartości n i zapisanie tej wartości w zmiennej.

    cout << "Wprowadź liczbę i naciśnij klawisz Enter: ";
    cin >> n;

// Obliczenie i wyświetlenie kwadratu liczby.

    cout << "Kwadrat liczby wynosi: " << n * n;

    return 0;
}
```



ĆWICZENIA

Ćwiczenie 1.3.1. Zmień kod przykładowego programu w taki sposób, aby przeprowadzał działanie odwrotne, to znaczy konwertował wejściową wartość temperatury podaną w skali Fahrenheita (`ftemp`) na liczbę wyrażoną w skali Celsjusza (`ctemp`), a następnie wyświetlał uzyskany wynik. (Wskazówka: wzór na odwrotną konwersję jest następujący: `ctemp = (ftemp - 32) / 1.8`).

Ćwiczenie 1.3.2. Napisz program zmieniający temperaturę ze skali Fahrenheita na wartość w skali Celsjusza, tak aby korzystał on z tylko jednej zmiennej. Zadanie polega tu na optymalizacji programu z ćwiczenia 1.3.1.

Ćwiczenie 1.3.3. Napisz program, który pobiera od użytkownika pewną wartość, umieszcza ją w zmiennej `n`, a następnie wyświetla sześcian, czyli trzecią potęgę (`n * n * n`) podanej liczby. Pamiętaj, aby w wyrażeniu wyświetlającym obliczoną wartość użyć słowa „sześcian” zamiast „kwadrat”.

Ćwiczenie 1.3.4. Zmień kod przykładu *kwadrat.cpp*, korzystając ze zmiennej o nazwie `liczba` zamiast `n`. Upewnij się, że zastąpiłeś nazwę `n` nową we wszystkich miejscach, w których się ona pojawiła.

Słowo na temat nazw zmiennych i słów kluczowych

W tym rozdziale występowały zmienne `ctemp`, `ftemp` oraz `n`. W ćwiczeniu 1.3.4 zasugerowałem Ci możliwość zastąpienia zmiennej `n` zmienną o nazwie `liczba`, która będzie działać należycie, jeśli tylko konsekwentnie podmienisz dokładnie wszystkie wystąpienia nazwy `n` znajdujące się w programie. A zatem `liczba` może być równie dobrą nazwą jak używana wcześniej nazwa `n`.

Mogłem oczywiście wybrać inną z bezliku możliwych nazw zmiennych zamiast nazwy `n` czy `liczba`. Mógłbym na przykład nadać zmiennej nazwę taką jak `terminator2003`, `robotZabojca` czy `WojewodaKatowicki`.

Jakie nazwy są zatem dopuszczalne, a których nie możesz używać w przypadku zmiennych? Odpowiedź jest prosta: możesz korzystać z każdej nazwy, jaka tylko przyjdzie Ci do głowy, pod warunkiem że będzie ona spełniała następujące wymagania:

- Pierwszy znak nazwy musi być literą. Nie może być cyfrą. Technicznie rzecz biorąc, pierwszym znakiem nazwy zmiennej może być znak podkreślenia (_), jednak biblioteki C++ używają takiej właśnie konwencji wewnętrznej, lepiej więc będzie unikać nazw rozpoczynających się w ten sposób.
- Pozostała część nazwy może składać się z liter, cyfr lub znaków podkreślenia (_).
- Musisz unikać słów, które mają już swoje predefiniowane znaczenie w C++.

Te słowa o specjalnym zastosowaniu w C++ noszą nazwę **słów kluczowych**. Wyróżniamy tu na przykład nazwy standardowych typów danych języka C++, takich jak `int`, `float` i `double`. Do innych należą słowa `if`, `else`, `while`, `do`, `switch` oraz `class`.

Nie musisz teraz siadać i uczyć się wszystkich słów kluczowych języka C++, mimo że wiele podręczników programowania sugeruje Ci takie właśnie działanie! Powinieneś jedynie wiedzieć, że próba użycia nazwy zastrzeżonej dla któregoś ze słów kluczowych C++ spowoduje wygenerowanie przez kompilator odpowiedniego komunikatu błędu, w którym będzie mowa o konflikcie nazw lub błędzie składni. W takim przypadku będziesz po prostu musiał spróbować skorzystać z innej nazwy.



ĆWICZENIE

Ćwiczenie 1.3.5. Które ze słów znajdujących się na poniższej liście są poprawnymi nazwami zmiennych w języku C++, a które nie mogą nimi być? Sprawdź to, korzystając z podanych przed chwilą reguł.

```
x
x1
PanZloWcielone
UlicaMoczymordki16
16UlicaMoczymordki
Robot_Robercik
Robot+Robercik
CoDoJasnej???
kwota
licznik2
Licznik2piec
5licznik
main
main2
```

Podsumowanie rozdziału 1.

Oto najważniejsze rzeczy, które pojawiły się w tym rozdziale:

- Tworzenie programu zaczyna się od napisania kodu źródłowego C++. Składa się on z wyrażeń języka C++, które przypominają nieco zdania zapisane w języku angielskim. Kod maszynowy, dla odmiany, jest zupełnie niezrozumiały dla zwykłego śmiertelnika, dopóki mozolnie nie sprawdzi on znaczenia każdej kolejnej kombinacji zer i jedynek. Zanim jednak program będzie mógł zostać uruchomiony, musi zostać przetłumaczony na kod maszynowy, który jest jedynym zapisem zrozumiałym dla komputera.
- Proces tłumaczenia wyrażeń języka C++ na kod maszynowy nosi nazwę **kompilacji**.

- Program po skompilowaniu musi zostać połączony ze standardowymi funkcjami przechowywanymi w bibliotece C++. Proces ten nosi nazwę **linkowania**, **łączenia** lub **konsolidacji**. Po pomyślnym przeprowadzeniu tej operacji otrzymasz działający program.
- Na szczęście, jeśli dysponujesz środowiskiem programisty, proces kompilowania i łączenia aplikacji, zwany też **budowaniem** programu, może być zautomatyzowany, dzięki czemu Twoim jedynym zadaniem na tym etapie będzie naciśnięcie odpowiedniego klawisza funkcyjnego.
- Prosty program napisany w języku C++ ma następującą formę:

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

int main() {
   Tutaj_wprowadź_swoje_wyrażenia!
   return 0;
}
```

- Do wyświetlania danych na ekranie możesz wykorzystać obiekt `cout`, tak jak zostało to przedstawione w poniższym przykładzie:

```
cout << "Nie bój się, C++ jest tutaj!";
```

- Aby wyświetlić na ekranie tekst i przejść do nowej linii, powinieneś skorzystać z obiektu `cout` i znaku nowej linii (`endl`), tak jak zostało to zrobione w następującej instrukcji:

```
cout << "Nie bój się, C++ jest tutaj!" << endl;
```

- Niemal każde wyrażenie w języku C++ kończy się znakiem średnika (`;`). Jedynym wyjątkiem są tu dyrektywy preprocesora, po których nie należy stawiać tego znaku.
- Dwa znaki ukośnika (`//`) wskazują, że znajdujący się po nich tekst stanowi komentarz. Wszystkie znaki wchodzące w jego skład aż do końca linii będą zatem ignorowane przez kompilator i stanowić będą jedynie pomocny opis programu przeznaczony dla osób, które podejmą próbę zrozumienia jego działania lub będą odpowiedzialne za konserwację kodu i naprawianie w nim błędów.
- Każda zmienna musi być zadeklarowana przed pierwszym użyciem. Oto przykład:

```
double x; // Deklaracja x jako zmiennej typu zmiennopozycyjnego.
```

- Zmienne, które mogą mieć wartość ułamkową, powinny być deklarowane jako dane typu `double`, czyli wartości zmiennopozycyjne podwójnej precyzji. Typ zmiennopozycyjny pojedynczej precyzji (`float`) powinien być używany jedynie w sytuacjach, gdy będzie istniała konieczność zapisywania na dysku dużych zbiorów danych zmiennopozycyjnych, a więc w sytuacjach, kiedy oszczędność pamięci masowej ma duże znaczenie.

- Aby pobrać dane wejściowe z klawiatury i umieścić je w zmiennej, możesz skorzystać z obiektu `cin`. Przykładem może tu być następująca instrukcja:

```
cin >> x;
```

- Daną możesz umieścić w zmiennej, korzystając z operatora przypisania (**=**). Operator ten powoduje obliczenie wartości wyrażenia znajdującego się po prawej stronie znaku równości (**=**) i zapisanie tej wartości w zmiennej, która figuruje po jego lewej stronie. Przykładem przypisania może być następujące polecenie:

```
x = y * 2; // Przemnożenie wartości y przez liczbę 2 i umieszczenie wyniku w zmiennej x.
```