Rogers Cadenhead, Jesse Liberty

C++

w 24 godziny

SAMS

Helion

Tytuł oryginału: Sams Teach Yourself C++ in 24 Hours, 6th Edition

Tłumaczenie: Robert Górczyński

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

ISBN: 978-83-283-3551-6

Authorized translation from the English language edition: SAMS TEACH YOURSELF C++ IN 24 HOURS, 6TH EDITION; ISBN 0672337460; by Rogers Cadenhead and Jesse Liberty; published by Pearson Education, Inc, publishing as SAMS Publishing.

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/cpp246.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/cpp246
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

CZĘŚĆ IV ZAAWANSOWANY C++

CZĘŚĆ V DZIEDZICZENIE I POLIMORFIZM

Godzina 2.

Organizacja elementów programu

W trakcie tej godziny nauczysz się:

- **dlaczego warto używać języka C++;**
- **w jaki sposób są zorganizowane programy C++;**
- **dlaczego komentarze ułatwiają zrozumienie sposobu działania programu;**
- **do czego można wykorzystać funkcje.**

Wprawdzie ma już prawie 40 lat, ale język programowania C++ starzeje się znacznie wolniej niż inne rzeczy pochodzące z końca lat 70. ubiegłego wieku. W przeciwieństwie do disco, embarga na dostawy ropy naftowej, włochatych dywanów i kolorowych lodówek C++ nadal się cieszy popularnością i pozostaje światowej klasy językiem programowania.

Powodem zaskakującej długowieczności tego języka jest fakt, że C++ umożliwia utworzenie szybko wykonywanych programów za pomocą niewielkiej ilości kodu, który może być uruchamiany w różnych środowiskach informatycznych. Obecne narzędzia programistyczne C++ pozwalają na tworzenie skomplikowanych i potężnych aplikacji w środowiskach komercyjnych, biznesowych i *open source*.

Dlaczego warto używać C++?

Od czasu pierwszych komputerów języki programowania przebyły długą drogę. C++ jest uznawany za ewolucyjne usprawnienie języka programowania o nazwie C, który pojawił się w 1972 r.

Na początku programiści używali najbardziej prymitywnych poleceń komputera: języka maszynowego. Te polecenia zapisywane były jako długie ciągi zer i jedynek. Dlatego wymyślono tzw. asemblery zamieniające instrukcje maszynowe na czytelne dla człowieka i łatwiejsze do zapamiętania mnemoniki, takie jak `ADD` czy `MOV`.

Polecenia tworzące program komputerowy są określane mianem jego **kodu źródłowego**.

Z czasem pojawiły się języki wyższego poziomu, takie jak BASIC czy COBOL. Te języki umożliwiały tworzenie programów za pomocą języka bliższego rzeczywistym słowom

i zdaniom, np. `Let Average = 0.366`. Te instrukcje były tłumaczone z powrotem na język maszynowy przez narzędzia nazywane **interpreterami** lub **kompilatorami**.

Język interpretowany tłumaczy program bezpośrednio podczas odczytywania kolejnych wierszy i działa na poszczególnych poleceniach.

Z kolei język kompilowany tłumaczy program na tzw. **kod obiektowy** w trakcie procesu nazywanego **kompilacją**. Wspomniany kod jest przechowywany w pliku obiektowym. Następnie linker przekształca ten plik obiektowy na program wykonywalny, który może być uruchomiony przez system operacyjny.

Ponieważ interpretery bezpośrednio odczytują kod źródłowy programu i wykonują go na bieżąco, są łatwiejsze w użyciu dla programistów. Kompilatory wymagają przeprowadzenia dodatkowych niewygodnych kroków kompilacji programów i dołączania do nich niezbędnych bibliotek. Jednak zaletą tego podejścia jest to, że programy skompilowane są wykonywane znacznie szybciej niż uruchamiane przez interpretera.

Przez wiele lat głównym celem programistów było tworzenie krótkich fragmentów kodu, które mogłyby być szybko wykonywane. Programy musiały być małe, ponieważ pamięć komputera była niezwykle droga. Ponadto musiały być szybkie, ponieważ moc obliczeniowa komputerów również była kosztowna. Gdy komputery stały się tańsze, szybsze i zaoferowały znacznie potężniejsze możliwości, a koszt pamięci operacyjnej i masowej znacznie spadł, wymienione wcześniej priorytety utraciły swoją wagę.

Obecnie największym kosztem w programowaniu jest praca programisty. Nowoczesne języki programowania, takie jak C++, pozwalają na szybsze tworzenie doskonale napisanych i łatwych w późniejszej obsłudze programów, które mogą być rozbudowywane i usprawniane.

Style programowania

W toku ewolucji powstawały nowe języki, aby umożliwić stosowanie różnych stylów programowania.

W stylu programowania proceduralnego program był traktowany jako seria procedur działających na danych. Programowanie strukturalne zostało wprowadzone w celu dostarczenia systematycznego podejścia do organizacji tych procedur i zarządzania ogromnymi ilościami danych.

Główną ideą programowania strukturalnego jest „dziel i rządź”. Program komputerowy może być uważany za zestaw zadań. Każde zbyt skomplikowane zadanie, aby można było je łatwo opisać, jest rozbijane na zestaw mniejszych zadań składowych, aż do momentu gdy wszystkie zadania są wystarczająco łatwe do zrozumienia.

Przykładowo przyjmujemy założenie, że otrzymałeś od wydawcy polecenie utworzenia programu monitorującego średnie wynagrodzenie zespołu składającego się z niezwykle utalentowanych i charyzmatycznych autorów książek informatycznych. Tego rodzaju zadanie można jednak podzielić na następujące podzadania:

1. Obliczenie, ile zarabiają poszczególni autorzy.
2. Obliczenie liczby autorów pracujących dla wydawcy.

3. Zsumowanie wszystkich pensji.
4. Podzielenie tej sumy przez liczbę autorów.

Sumowanie pensji można podzielić na następujące kroki:

1. Odczytanie danych dotyczących każdego autora.
2. Odwołanie się do danych dotyczących honorarium i zaliczek pobranych przez danego autora.
3. Potrącenie kosztu porannej kawy, okularów korekcyjnych i osteoterapii.
4. Dodanie pensji do naliczanej sumy.
5. Przejście do danych dotyczących następnego autora.

Z kolei uzyskanie danych na temat autora można rozbić na:

1. Otwarcie pliku pracowników.
2. Przejście do właściwych danych.
3. Odczytanie danych z dysku.

Wprawdzie programowanie strukturalne było powszechnie stosowane, ale ograniczenia takiej metody programowania objawiły się aż nazbyt jasno. Oddzielenie danych od zadań przeprowadzających operacje na tych danych staje się coraz trudniejsze do wykonania i późniejszej obsługi wraz ze wzrostem ilości danych. Im więcej rzeczy trzeba zrobić z użyciem danych, tym bardziej zawiły staje się program.

Podczas stosowania tego rodzaju podejścia programiści bardzo często wyważali otwarte drzwi i tworzyli nowe rozwiązania dla starych problemów, zamiast opracowywać programy przeznaczone do wielokrotnego zastosowania. Idea wielokrotnego użycia polega na takim utworzeniu komponentów programu, aby można było je stosować w innych programach, gdy zachodzi potrzeba. Tego rodzaju podejście jest znane z rzeczywistości, gdzie urządzenia są budowane z poszczególnych elementów, które są już wcześniej wyprodukowane i przeznaczone do wykonywania określonych zadań. Przykładowo inżynier tworzący rower nie musi zupełnie od zera opracowywać układu hamulcowego. Zamiast tego w swoim projekcie może użyć istniejącego układu i tym samym wykorzystać zalety płynące z już znanej funkcjonalności.

Po raz pierwszy podejście oparte na komponentach stało się dostępne dla programistów wraz z wprowadzeniem programowania zorientowanego obiektowo.

C++ i programowanie zorientowane obiektowo

Język C++ pomógł w spopularyzowaniu rewolucyjnego stylu programowania zorientowanego obiektowo dzięki akronimowi **OOP** (ang. *object-oriented programming*).

Istotą programowania zorientowanego obiektowo jest potraktowanie danych i procedur działających na tych danych jako pojedynczego obiektu. Powstaje w ten sposób samodzielna jednostka wraz z własną tożsamością i cechami charakterystycznymi.

Język C++ obsługuje programowanie zorientowane obiektowo i obejmuje swym działaniem trzy podstawowe koncepcje takiego stylu programowania: hermetyzację, dziedziczenie oraz polimorfizm.

Hermetyzacja

Gdy wspomniany wcześniej inżynier pracuje nad nowym rowerem, łączy ze sobą dostępne komponenty, takie jak ramę, kierownicę, koła, przednie światło (kolorowe dodatki umieszczane w szprychach są opcjonalne). Każdy komponent ma określone właściwości i charakteryzuje się konkretnym zachowaniem. Inżynier może użyć komponent, np. przednią lampę, bez zastanawiania się nad szczegółami sposobu jego działania, jeśli tylko wie, do czego on służy.

W tym celu komponent (tutaj: wspomniana lampa) musi być samozawierający się. To oznacza, że musi być jedną doskonale przygotowaną rzeczą, która całkowicie wykonuje stawiane przed nią zadanie. Właściwość samozawierania się jest nazywana **hermetyzacją**.

Wszystkie właściwości przykładowej lampy są hermetyzowane w obiekcie lampy. Nie są porozrzucane po całym rowerze.

C++ obsługuje hermetyzację poprzez tworzenie typów zdefiniowanych przez użytkownika, zwanych **klasami**. Po stworzeniu dobrze zdefiniowana klasa działa jako spójna całość — jest używana jako jednostka lub w ogóle nie jest używana. Wewnętrzne działanie klasy powinno być ukryte. Użytkownicy dobrze zdefiniowanych klas nie muszą wiedzieć, w jaki sposób one działają; muszą jedynie umieć z nich korzystać. O tym, jak tworzyć klasy, dowiesz się z godziny 8.

Dziedziczenie i ponowne wykorzystanie

Teraz dowiesz się nieco więcej o inżynierze, który pracuje nad naszym nowym rowerem. Przyjmujemy założenie, że to jest Jan Kowalski. Janek chce jak najszybciej wprowadzić na rynek nowy rower, ponieważ narobił sobie ogromnych długów hazardowych u osób, które są znane ze swojej niecierpliwości.

Skoro czas go goni, Janek rozpoczyna projektowanie nowego roweru od wykorzystania już istniejącego i usprawnia go nowymi świetnymi dodatkami, takimi jak uchwyt na kubek i licznik. Przygotowany przez niego rower jest przedstawiany jako pojazd wraz z dodatkowymi funkcjami. Janek ponownie wykorzystał wszystkie funkcje zwykłego roweru i dodał nowe, aby w ten sposób rozszerzyć możliwości oferowane przez dwukołowy pojazd.

C++ obsługuje tego rodzaju ponowne wykorzystanie przez koncepcję o nazwie **dziedziczenie**. Można dzięki niemu deklarować nowe typy będące rozszerzeniem już istniejących typów. Mówi się, że nowa podklasa wywodzi się z istniejącego typu i czasem nazywa się ją typem pochodnym. Rower opracowany przez Janka wywodzi się ze zwykłego roweru i tym samym dziedziczy po nim wszystkie możliwości, choć jednocześnie można w nim dodać nowe funkcje, jeśli zachodzi potrzeba. Dziedziczenie i jego zastosowania w języku C++ zostaną omówione w godzinie 16.

Polimorfizm

Ostatnią cechą, która ma przyciągnąć nabywców nowego roweru Janka, jest odmienne zachowanie sygnału dźwiękowego w zależności od sposobu jego użycia. Zamiast dźwięku podobnego do odgłosu kaczki pojazd wydaje dźwięk znany z klaksonu samochodowego, gdy przycisk zostanie lekko naciśnięty. Natomiast mocne naciśnięcie przycisku powo-

duje wydanie dźwięku przypominającego syrenę. Klakson wykonuje tutaj właściwe zadanie i dobiera dźwięk w zależności od sposobu użycia przycisku przez rowerzystę.

C++ sprawia, że różne obiekty „robią odpowiednie rzeczy" poprzez mechanizm zwany *polimorfizmem funkcji* i *polimorfizmem klas*. Pojęcie „polimorfizm" oznacza, że ta sama rzecz może przybierać wiele form. Dokładne omówienie polimorfizmu znajdziesz w godzinie 17.

Ucząc się języka C++, poznasz pełny zakres programowania zorientowanego obiektowo. Wymienione powyżej koncepcje staną się dla Ciebie jasne, zanim ukończysz wszystkie 24 godziny przedstawione w tej książce i rozpoczniesz tworzenie własnych programów w C++.

Uwaga! Z tej książki nie dowiesz się, jak projektować rowery lub jak wyjść z długów hazardowych.

Poszczególne elementy programu

Program utworzony w godzinie 1., *Motto.cpp*, składał się z prostego frameworka przedstawiającego typowy program w języku C++. W listingu 2.1 ponownie zaprezentowałem program *Motto.cpp*, aby móc go tutaj szczegółowo omówić.

LISTING 2.1. Pełny kod źródłowy programu Motto.cpp

```
1:  #include <iostream>
2:
3:  int main()
4:  {
5:      std::cout << "Solidum petit in profundis!\n";
6:      return 0;
7:  }
```

Podczas wpisywania kodu programu w edytorze takim jak Netbeans pamiętaj o pominięciu numerów wierszy w listingu. Zamieściłem je tylko po to, aby z poziomu tekstu móc łatwiej odwoływać się do poszczególnych poleceń kodu źródłowego.

Po uruchomieniu programu powoduje on wygenerowanie motta Aarhus University:

```
Solidum petit in profundis!
```

W wierszu 1. kodu przedstawionego w listingu 2.1 następuje dołączenie pliku o nazwie *iostream* do kodu źródłowego. Ten wiersz powoduje, że kompilator działa tak, jakby cała zawartość wymienionego pliku została umieszczona w miejscu wiersza 1. programu *Motto.cpp*.

Dyrektywy preprocesora

Pierwszą operacją kompilatora C++ jest wywołanie innego narzędzia o nazwie preprocesor analizującego kod źródłowy. To jest przeprowadzane automatycznie w trakcie każdego uruchomienia kompilatora.

Pierwszym znakiem w wierszu 1. jest # oznaczający, że ten wiersz stanowi polecenie przeznaczone dla preprocesora. Tego rodzaju polecenia są nazywane **dyrektywami preprocesora**. Zadanie preprocesora polega na odczytaniu kodu źródłowego, wyszukaniu dyrektyw oraz zmodyfikowaniu kodu zgodnie ze znalezionymi dyrektywami. Dopiero tak zmodyfikowany kod źródłowy jest przekazywany kompilatorowi.

Preprocesor działa więc jak redaktor sprawdzający kod tuż przed jego kompilacją. Każda dyrektywa jest poleceniem dla wspomnianego redaktora.

Dyrektywa `#include` nakazuje preprocesorowi dołączenie w miejscu tej dyrektywy całej zawartości wskazanego pliku. Jak się dowiedziałeś podczas godziny 1., język C++ zawiera standardową bibliotekę kodu źródłowego, którą można używać we własnych programach, aby uzyskać dostęp do pewnych użytecznych funkcjonalności. Kod umieszczony w pliku *iostream* jest odpowiedzialny za wykonywanie zadań związanych z operacjami wejścia-wyjścia, takich jak wyświetlanie informacji na ekranie oraz pobieranie danych wejściowych od użytkownika.

Znaki nawiasu ostrego wokół nazwy pliku nakazują preprocesorowi szukanie tego pliku w standardowych lokalizacjach. Z powodu użycia nawiasów ostrych preprocesor będzie szukał pliku *iostream* w katalogu zawierającym pliki nagłówkowe dla tego kompilatora. Te pliki są również określane mianem **plików dołączanych**, ponieważ są dołączane do kodu źródłowego programu.

Pełna zawartość pliku *iostream* zajmie miejsce wiersza 1.

Uwaga

> Nazwy plików nagłówkowych tradycyjnie mają rozszerzenie *.h*, były nazywane również *plikami h* i mogły być używane w dyrektywie o postaci `include <iostream.h>`.
>
> Nowoczesne kompilatory nie wymagają tego rozszerzenia, ale jeśli odwołujesz się do plików z jego użyciem, dyrektywa nadal będzie działała w celu zapewnienia wstecznej zgodności. W tej książce pomijam rozszerzenia *.h* w nazwach plików nagłówkowych.

Zawartość pliku *iostream* jest używana przez znajdujące się w wierszu 5. polecenie `cout`, które wyświetla informacje na ekranie.

W omawianym tutaj kodzie źródłowym nie ma więcej żadnych dyrektyw, więc to kompilator zajmuje się obsługiwaniem pozostałej części pliku *Motto.cpp*.

Kod źródłowy wiersz po wierszu

Wiersz 3. rozpoczyna rzeczywisty program od deklaracji funkcji o nazwie `main()`. **Funkcja** jest blokiem kodu wykonującym jedną lub więcej operacji. Funkcja wykonuje pewne zadanie, a po zakończeniu jej działania następuje powrót do miejsca, w którym została wywołana.

Funkcję `main()` ma każdy program C++. Gdy program rozpoczyna działanie, jest ona wywoływana automatycznie.

Po zakończeniu działania, każda funkcja w języku C++ musi zwracać pewnego rodzaju wartość. W przypadku funkcji `main()` wartością zwrotną zawsze jest liczba całkowita. Do zdefiniowania liczby całkowitej używamy słowa kluczowego `int`.

Podobnie jak inne bloki kodu w programie C++, polecenia definiujące funkcję są umieszczane wewnątrz nawiasów klamrowych. Dlatego też blok każdej funkcji rozpoczyna się od nawiasu otwierającego {, a kończy nawiasem zamykającym }.

W pliku kodu źródłowego *Motto.cpp* otwierający i zamykający nawias klamrowy należący do funkcji `main()` znajdują się odpowiednio w wierszach 4. i 7. Wszystkie polecenia umieszczone między nawiasami otwierającym i zamykającym są częścią funkcji.

W wierszu 5. mamy polecenie `cout` używane do wyświetlenia komunikatu na ekranie. Na początku tego polecenia znajduje się prefiks `std::` informujący kompilator o konieczności użycia standardowej biblioteki języka C++ odpowiedzialnej za obsługę wejścia-wyjścia. Szczegóły dotyczące sposobu działania tego polecenia są zbyt skomplikowane, aby je wyjaśnić w tej lekcji, i gdybym zaczął je teraz przedstawiać, prawdopodobnie wyrzuciłbyś tę książkę przez okno. Aby zapewnić bezpieczeństwo Twojemu otoczeniu, powrócę do tego tematu w dalszej części książki. Teraz potraktuj `std::cout` jako nazwę obiektu odpowiadającego za obsługę danych wyjściowych generowanych przez program. Z kolei `std::cin` potraktuj jako obiekt zapewniający obsługę danych wejściowych programu wprowadzanych przez użytkownika.

Po odwołaniu do `std::cout` w wierszu 5. znajdują się znaki `<<`, które nazywamy **operatorem przekierowania danych wyjściowych**. Wspomniany **operator** to znaki umieszczone w wierszu kodu — przeprowadza on określoną operację w reakcji na pewnego rodzaju informacje. Zadanie operatora `<<` polega na wyświetleniu danych znajdujących się po tym operatorze. W wierszu 5. tekst `"Solidum petit in profundis!\n"` został ujęty w cudzysłów. Dlatego też podany ciąg tekstowy będzie wyświetlony wraz ze znakiem specjalnym `\n`, który oznacza znak nowego wiersza. Ewentualne kolejne dane wyjściowe programu zostaną umieszczone w nowym wierszu.

W wierszu 6. program zwraca wartość 0. Ta wartość jest otrzymywana przez system operacyjny po zakończeniu działania programu. Zwykle program zwraca wartość 0, aby wskazać, że jego działanie zakończyło się sukcesem. Każda inna wartość wskazuje na pewnego rodzaju niepowodzenie.

Nawias zamykający w wierszu 7. kończy funkcję `main()`, co jednocześnie powoduje zakończenie działania programu. Wszystkie tworzone przez Ciebie programy będą używały tego prostego frameworka zaprezentowanego w omawianym programie.

Komentarze

Gdy piszesz program, wówczas to, co chcesz osiągnąć, zawsze jest jasne i oczywiste. Jednak gdy do niego wracasz po upływie pewnego czasu w celu usunięcia błędu lub dodania nowej funkcji, kod może się okazać całkiem niezrozumiały.

Aby sobie z tym poradzić, a także by pomóc innym w zrozumieniu Twojego kodu, powinieneś używać komentarzy. **Komentarze** są tekstem całkowicie ignorowanym przez kompilator, mogą natomiast informować czytającego o tym, co robisz w danym punkcie programu.

Komentarze w C++ występują w dwóch odmianach: jako komentarze umieszczone w pojedynczym wierszu oraz komentarze umieszczone w wielu wierszach. Komentarz umieszczony w jednym wierszu jest oznaczony za pomocą podwójnego ukośnika (//), który informuje kompilator, by zignorował wszystko, co po nim następuje aż do końca wiersza. Poniżej przedstawiłem przykład tego rodzaju komentarza.

```
// Kolejny wiersz to prowizorka (brr!).
```

Komentarze umieszczone w wielu wierszach są oznaczane za pomocą ukośnika i gwiazdki (/*). Takie oznaczenie informuje kompilator, by zignorował wszystko to, co jest zawarte pomiędzy znakami /* oraz */. Wymienione znaki mogą się znajdować w tym samym wierszu lub między nimi mogą się znajdować inne wiersze. Każdemu znakowi /* musi odpowiadać zamykający komentarz znak */, w przeciwnym razie kompilator zgłosi błąd. Poniżej przedstawiłem przykład tego rodzaju komentarza.

```
/* Ten fragment programu nie działa zbyt dobrze. Pamiętaj o konieczności
   poprawienia tego kodu przed wydaniem aplikacji — ewentualnie znajdź ofiarę,
   którą będzie można obarczyć winą za problem. Dobrą kandydaturą na kozła
   ofiarnego jest tutaj nasz nowy pracownik, Janek. */
```

W powyższym komentarzu tekst ma ustawiony lewy margines, aby zapewnić lepszą czytelność. Oczywiście to nie jest wymagane. Ponieważ kompilator ignoruje wszystko, co jest umieszczone między znakami /* i */, w komentarzu możesz umieścić cokolwiek: listę zakupów, listy miłosne, nieopowiedziane nikomu wcześniej tajemnice z Twojego życia itd.

Ostrzeżenie

> Trzeba koniecznie pamiętać, że nie wolno zagnieżdżać w sobie komentarzy wielowierszowych. Jeżeli użyjesz znaków /* rozpoczynających komentarz, a następnie kilka wierszy później ponownie umieścisz znaki /*, wtedy pierwszy napotkany ciąg */ zakończy wszystkie komentarze wielowierszowe, a drugi ciąg tekstowy */ spowoduje wygenerowanie komunikatu błędu przez kompilator. Większość edytorów programistycznych dla C++ wyświetla komentarze innym kolorem, aby wyraźnie pokazać ich początek i koniec.

Kolejny utworzony tutaj projekt będzie zawierał oba rodzaje komentarzy. W swoich programach umieszczaj wiele komentarzy. Im więcej czasu poświęcisz na pisanie komentarzy wyjaśniających sposób działania kodu źródłowego, tym łatwiejsze będzie zrozumienie tego kodu, gdy powrócisz do niego po kilku tygodniach, miesiącach lub nawet latach.

Funkcje

Funkcja `main()` nie jest zwykłą funkcją, ponieważ jest wywoływana automatycznie po rozpoczęciu działania programu.

Program jest wykonywany „wiersz po wierszu” — w kolejności występowania poleceń w kodzie źródłowym, począwszy od początku funkcji `main()`. Po napotkaniu wywołania funkcji, działanie programu „rozgałęzia” się w celu wykonania danej funkcji. Gdy funkcja zakończy działanie, następuje powrót do wiersza kodu znajdującego się bezpośrednio po wierszu, w którym funkcja została wywołana. Funkcja może, ale nie musi zwrócić wartość. Wyjątkiem jest tutaj funkcja `main()`, która zawsze zwraca liczbę całkowitą.

Funkcja składa się z nagłówka oraz części głównej. Nagłówek zbudowany jest z wymienionych poniżej elementów:

- zwracanego typu danych,
- nazwy funkcji,
- parametrów otrzymywanych przez funkcję.

Nazwa funkcji to krótki identyfikator opisujący jej przeznaczenie.

Jeżeli funkcja nie zwraca wartości, wówczas używa typu danych `void`. Śpieszę tutaj z wyjaśnieniem, że typ `void` nie jest bez znaczenia. Po prostu oznacza „nic”, podobnie jak gwiazdy w Kosmosie znajdują się w ogromnych odległościach nicości nazywanej „pustką” (ang. *the void*).

Argumenty to przekazywane funkcji dane kontrolujące jej sposób działania. Te argumenty są otrzymywane przez funkcję w postaci **parametrów**. Funkcja może mieć zero, jeden lub więcej parametrów. Kolejny utworzony przez nas program ma funkcję o nazwie `add()` obliczającą sumę dwóch liczb. Poniżej przedstawiłem deklarację tego rodzaju funkcji.

```
int add(int x, int y)
{
    // Miejsce na polecenia tworzące tę funkcję.
}
```

Parametry są umieszczone w nawiasie, na rozdzielonej przecinkami liście elementów. W powyższym przypadku parametrami są liczby całkowite `x` i `y`.

Nazwa funkcji, jej parametry i kolejność tych parametrów nosi nazwę **deklaracji** pozwalającej na unikatowe zidentyfikowanie danej funkcji.

Funkcja bez parametrów ma pusty nawias po nazwie, jak pokazałem poniżej.

```
int getServerStatus()
{
    // Miejsce na polecenia tworzące tę funkcję.
}
```

Nazwa funkcji nie może zawierać spacji, więc w przypadku nazwy takiej jak `getServer`
`↪Status()` pierwsza litera każdego słowa, począwszy od drugiego, jest zapisywana dużą literą. Tego rodzaju konwencja jest powszechnie stosowana przez programistów C++ i została przyjęta w tej książce.

Treść funkcji składa się z otwierającego nawiasu klamrowego, żadnego lub więcej poleceń oraz zamykającego nawiasu klamrowego. Funkcja zwracająca wartość używa polecenia `return`, podobnie jak to mogłeś zobaczyć we wcześniejszym programie *Motto.cpp*.

```
return 0;
```

Polecenie `return` kończy działanie funkcji. Jeżeli w funkcji nie umieścisz przynajmniej jednego polecenia `return`, automatycznie będzie zwrócona wartość `void`. W takim przypadku konieczne jest podanie `void` jako typu wartości zwrotnej funkcji.

Użycie argumentów wraz z funkcją

Przedstawiony w listingu 2.2 program *Calculator.cpp* zawiera wspomnianą wcześniej funkcję `add()`, która oblicza sumę dwóch liczb i wyświetla tę wartość. Ten program pokazuje, jak utworzyć funkcję pobierającą dwa argumenty w postaci liczb całkowitych, która to funkcja zwróci wartość również w postaci liczby całkowitej.

LISTING 2.2. Pełny kod źródłowy programu Calculator.cpp

```
 1: #include <iostream>
 2:
 3: int add(int x, int y)
 4: {
 5:     // Dodanie liczb x oraz y, a następnie zwrócenie obliczonej sumy.
 6:     std::cout << "Uruchomienie kalkulatora...\n";
 7:     return (x+y);
 8: }
 9:
10: int main()
11: {
12:     /* Ten program wywołuje funkcję add() w celu obliczenia
13:        i wyświetlenia sumy dwóch różnych zbiorów liczb. Funkcja
14:        add() nic nie robi, dopóki nie zostanie wywołana przez
15:        odpowiednie polecenie w funkcji main(). */
16:     std::cout << "Ile wynosi suma liczb 867 + 5309?\n";
17:     std::cout << "Suma wynosi " << add(867, 5309) << ".\n\n";
18:     std::cout << "Ile wynosi suma liczb 777 + 9311?\n";
19:     std::cout << "Suma wynosi " << add(777, 9311) << ".\n";
20:     return 0;
21: }
```

Omawiany program powoduje wygenerowanie poniższych danych wyjściowych.

```
Ile wynosi suma liczb 867 + 5309?
Uruchomienie kalkulatora...
Suma wynosi 6176.

Ile wynosi suma liczb 777 + 9311?
Uruchomienie kalkulatora...
Suma wynosi 10088.
```

Program *Calculator.cpp* zawiera komentarz jednowierszowy w wierszu 5. i wielowierszowy w wierszach od 12. do 15. Wszystkie komentarze zostają zignorowane przez kompilator.

Zdefiniowana w wierszach od 3. do 8. funkcja `add()` pobiera dwa parametry w postaci liczb całkowitych o nazwach `x` i `y`, a następnie oblicza ich sumę w poleceniu `return`.

Wykonywanie programu rozpoczyna się w funkcji `main()`. Pierwsze polecenie w wierszu 16. używa obiektu `std::cout` i operatora przekierowania `<<` w celu wyświetlenia komunikatu `"Ile wynosi suma liczb 867 + 5309?` i znaku nowego wiersza.

W kolejnym wierszu mamy wyświetlenie ciągu tekstowego `Suma wynosi` i wywołanie funkcji `add()` wraz z argumentami 867 i 5309. Wykonywanie programu odbywa się teraz w funkcji `add()`, na co wskazuje komunikat `Uruchomienie kalkulatora...` wyświetlony w danych wyjściowych.

Zwrócona przez funkcję wartość w postaci liczby całkowitej zostaje wyświetlona wraz z dwoma znakami nowego wiersza.

W wierszach 18. i 19. omówiony powyżej proces zostaje powtórzony, ale tym razem dla kolejnego zbioru liczb.

Element `(x+y)` nosi nazwę wyrażenia. Więcej informacji na temat tworzenia wyrażeń matematycznych znajdziesz w godzinie 4.

Podsumowanie

Z tej lekcji dowiedziałeś się, jak język C++ ewoluował z innych stylów języków komputerowych i stosuje metodologię określaną mianem programowania zorientowanego obiektowo. Ta metodologia odniosła tak ogromny sukces w informatyce, że język C++ dzisiaj nadal pozostaje równie nowoczesny, co w 1979 r., gdy został opracowany.

Żałuję, że moja czupryna z czasów studenckich równie dobrze nie przetrwała próby czasu. Zamiast tego pozostała jedynie na zdjęciach, które moi przyjaciele udostępniają w serwisie Facebook.

W dwóch programach opracowanych w trakcie tej lekcji wykorzystałeś trzy fragmenty programu w języku C++: dyrektywy preprocesora, komentarze i funkcje.

Wszystkie kolejne programy, które utworzysz, będą stosowały ten sam prosty framework użyty w programach *Motto.cpp* i *Calculator.cpp*. Będą jedynie znacznie bardziej skomplikowane z powodu wykorzystania większej liczby funkcji, niezależnie od tego, czy opracujesz je zupełnie od początku, czy też będziesz wywoływać funkcje z plików nagłówkowych dołączonych za pomocą dyrektyw `#include`.

Pytania i odpowiedzi

P.: Do czego służy znak # w programie C++?

O.: Ten znak oznacza dyrektywę dla preprocesora uruchamianego w trakcie działania kompilatora. Przykładowo dyrektywa `#include` powoduje odczytanie pliku, którego nazwa została umieszczona po niej. Kompilator nigdy nie ma styczności z dyrektywą. Zawartość tego pliku jest traktowana tak samo, jakby została wpisana w kodzie źródłowym w miejscu dyrektywy.

P.: Jaka jest różnica między komentarzami w stylu // i /*?

O.: Komentarz rozpoczynający się od podwójnego ukośnika (//) wygasa wraz z końcem wiersza, w którym został umieszczony. Z kolei komentarz rozpoczynający się od ukośnika i gwiazdki (/*) jest komentarzem wielowierszowym, rozciąga się aż do końcowego znacznika komentarza (*/). Należy pamiętać, że nawet koniec funkcji nie powoduje zakończenia komentarza wielowierszowego. Jeśli nie chcesz otrzymać błędu w trakcie kompilacji, nie zapominaj o umieszczeniu znacznika zamykającego komentarz.

P.: Jaka jest różnica między argumentami i parametrami funkcji?

O.: Te dwa pojęcia są dwiema stronami tego samego procesu zachodzącego w trakcie wywoływania funkcji. Argument to informacje przekazywane funkcji. Z kolei parametry to te same informacje otrzymane przez funkcję. Wywołujesz funkcję wraz z argumentami. Natomiast wewnątrz funkcji te argumenty są otrzymywane w postaci parametrów.

P.: Co to jest prowizorka?

O.: Prowizorka to tymczasowe rozwiązanie problemu, które później ma być zastąpione czymś lepszym. To pojęcie zostało spopularyzowane przez techników marynarki USA, programistów i inżynierów, a następnie rozprzestrzeniło się w innych profesjach.

W programie komputerowym prowizorka to kod źródłowy, który wprawdzie działa, ale jeśli poświęcić mu więcej czasu, to można utworzyć go znacznie lepiej. Prowizorka z reguły pozostaje w użyciu znacznie dłużej, niż można oczekiwać.

Kosmonauci z promu Apollo 13 utworzyli prowizorkę wszech czasów — składający się z taśmy klejącej i skarpetek system, który filtrował dwutlenek węgla z powietrza i pomógł załodze w powrocie na Ziemię.

Pierwsze znane użycie pojęcia prowizorki jest datowane na 1962 r. i przypisywane Jacksonowi W. Granholmowi. W artykule zamieszczonym w magazynie „Datamation" użył on eleganckiej definicji, która przetrwała próbę czasu: „Źle dobrany zestaw niedopasowanych części tworzących niepewną całość".

Warsztaty

Skoro poznałeś fragmenty tworzące program C++, poświęć teraz nieco czasu i odpowiedz na zamieszczone poniżej pytania, a także wykonaj kilka ćwiczeń, aby upewnić się o przyswojeniu materiału omówionego w tej lekcji.

Quiz

1. Jakiego typu danych jest wartość zwrotna funkcji `main()`?
   a) `void`.
   b) `int`.
   c) Ta funkcja nie zwraca wartości.

2. Na czym polega działanie nawiasów w programie C++?
   a) Wskazuje początek i koniec funkcji.
   b) Wskazuje początek i koniec programu.
   c) Zwiększa możliwości programu.
3. Który z wymienionych poniżej fragmentów nie zalicza się do definicji funkcji?
   a) Jej nazwa.
   b) Jej argumenty.
   c) Jej typ wartości zwrotnej.

Odpowiedzi

1. **B**. Wartością zwrotną funkcji `main()` jest liczba całkowita, czyli typ `int`.
2. **A**. Nawias oznacza początek i koniec funkcji, a także innych bloków kodu, które poznasz w kolejnych godzinach.
3. **C**. Definicja funkcji składa się z jej nazwy, parametrów i dokładnej kolejności tych parametrów. Nie zawiera typu wartości zwrotnej.

Ćwiczenia

1. Zmodyfikuj program *Motto.cpp* w taki sposób, aby motto Aarhus University było wyświetlane za pomocą funkcji.
2. Zmodyfikuj program *Calculator.cpp* w taki sposób, aby obliczał sumę trzech liczb całkowitych w funkcji `add()`. Dodaj więc trzecią liczbę z i wywołaj nową wersję funkcji wraz z dwoma zbiorami składającymi się z trzech liczb.

Odpowiedzi do ćwiczeń znajdziesz w witrynie tej książki pod adresem: *http://workbench.cadenhead.org/book/cpp-24-hours/*.

Skorowidz

F

H

I

K

L

Ł

M

N

O

P

R

S

T

U

W

Z

Notatki

C++ powstał w 1979 roku i doskonale przetrwał próbę czasu, a dziś oferuje programistom potężne możliwości i ogromną elastyczność. Co więcej, istnieją języki programowania, których twórcy inspirowali się właśnie C++; jednym z nich jest Java. Oznacza to, że opanowanie C++ daje wiele korzyści: ułatwia zrozumienie innych języków programowania, a przede wszystkim pozwala na tworzenie aplikacji praktycznie dla wszystkich platform, od komputerów i serwerów, przez urządzenia mobilne i konsole do gier, aż po komputery typu mainframe.

Dzięki tej książce w ciągu 24 godzinnych lekcji poznasz podstawy programowania w C++ i szybko zaczniesz tworzyć w pełni funkcjonalne aplikacje. Najpierw zapoznasz się z instalacją i użyciem kompilatora, później dowiesz się, jak debugować tworzony kod, wreszcie poznasz nowości wprowadzone w standardzie C++14. Dowiesz się, jak zarządzać wejściem i wyjściem oraz jak pracować z pętlami i tablicami. Nauczysz się programowania zorientowanego obiektowo i zobaczysz, jak wykorzystywać szablony. Każda lekcja kończy się zestawem pytań i odpowiedzi, warsztatami oraz ćwiczeniami do samodzielnego wykonania.

Najważniejsze zagadnienia:

- instalacja i korzystanie z kompilatora C++ na platformach: Windows, MacOS i Linux
- podstawowe koncepcje C++, takie jak funkcje i klasy
- wyrażenia lambda, wskaźniki i przeciążanie operatorów
- dziedziczenie i polimorfizm
- nowe funkcje języka wprowadzone w standardzie C++14

Rogers Cadenhead jest pisarzem i programistą. Tworzy Drudge Retort oraz inne popularne witryny internetowe.

Jesse Liberty jest autorem wielu książek dotyczących programowania. Pełni funkcję prezesa spółki Liberty Associates, Inc., która zajmuje się tworzeniem oprogramowania, świadczeniem usług konsultingowych oraz szkoleniami.

C++.
Programowanie na miarę wyobraźni!

Helion

SAMS

księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900

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

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

ISBN 978-83-283-3551-6

9 788328 335516

Informatyka w najlepszym wydaniu

cena: 79,00 zł