# EFEKTYWNY C

## WPROWADZENIE DO PROFESJONALNEGO PROGRAMOWANIA

ROBERT C. SEACORD

Helion

no starch press

Tytuł oryginału: Effective C: An Introduction to Professional C Programming

Tłumaczenie: Piotr Pilch

ISBN: 978-83-283-8343-2



Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

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

Printed in Poland.

# Spis treści

# 1

# Wprowadzenie do języka C

W ROZDZIALE UTWORZYSZ W JĘZYKU C SWÓJ PIERWSZY PROGRAM, CZYLI TRADYCYJNY PROGRAM WYŚWIETLAJĄCY KOMUNIKAT „WITAJ, ŚWIECIE!". ZAPREZENTUJĘ RÓŻNE ASPEKTY TAKIEGO PROSTEGO programu, a także przybliżę kompilowanie i uruchamianie go. Dalej zajmę się wybranymi opcjami edytora i kompilatora oraz wspomnę o typowych kwestiach związanych z możliwościami przenoszenia, z którymi szybko zaznajomisz się podczas pisania kodu w języku C.

## Tworzenie pierwszego programu C

Najlepszym sposobem nauki programowania w języku C jest rozpoczęcie pisania programów C. Tradycyjny program, od którego zaczniesz, wyświetla komunikat Witaj, świecie!

Aby napisać taki program, potrzebujesz edytora tekstu lub *zintegrowanego środowiska programistycznego IDE* (*Integrated Development Environment*). Dostępnych jest wiele do wyboru, ale na razie otwórz swój ulubiony edytor, a innymi możliwościami zajmiemy się dalej w tym rozdziale.

W edytorze tekstu wprowadź poniższy kod programu (listing 1.1).

*Listing 1.1. Program hello.c*

```
#include <stdio.h>
#include <stdlib.h>
int main(void) { ❶
```

```
puts("Witaj, świecie!"); ❷
return EXIT_SUCCESS; ❸
} ❹
```

Wkrótce bardziej szczegółowo zostanie przeanalizowany każdy wiersz kodu tego programu. Na razie zapisz go jako plik o nazwie *hello.c*. Rozszerzenie pliku *.c* wskazuje, że zawiera on kod źródłowy języka C.

**UWAGA** *Jeśli zakupiłeś wersję elektroniczną książki, za pomocą operacji wycinania i wklejania umieść kod programu w edytorze. Korzystaj z tych operacji, gdy tylko to możliwe, ponieważ w ten sposób wyeliminujesz błędy pojawiające się podczas przepisywania.*

## Kompilowanie i uruchamianie programu

Teraz niezbędne jest skompilowanie i uruchomienie programu, co obejmuje dwa osobne kroki. Dostępnych do wyboru jest wiele kompilatorów języka C. Polecenie służące do kompilowania programu zależy od używanego kompilatora. W systemie Linux oraz w innych uniksowych systemach operacyjnych możesz uruchomić kompilator za pomocą polecenia `cc`. Aby skompilować program, w wierszu poleceń wpisz polecenie `cc`, a po nim podaj nazwę pliku do skompilowania:

```
% cc hello.c
```

**UWAGA** *Polecenia te są powiązane z systemem Linux (oraz z innymi uniksowymi systemami operacyjnymi). Inne kompilatory obecne w innych systemach operacyjnych będą wymagać uruchamiania w odmienny sposób. Sprawdź dokumentację konkretnego kompilatora.*

Jeśli poprawnie wprowadziłeś kod programu, polecenie kompilatora utworzy nowy plik o nazwie *a.out* w tym samym katalogu, w którym znajduje się plik z kodem źródłowym. Sprawdź zawartość katalogu za pomocą polecenia `ls`. Powinien być widoczny następujący wynik:

```
% ls
a.out hello.c
```

Plik *a.out* to program wykonywalny, który możesz uruchomić z poziomu wiersza poleceń:

```
% ./a.out
Witaj, świecie!
```

Jeśli wszystko się powiedzie, program powinien wyświetlić w oknie terminala komunikat `Witaj, świecie!`. W przeciwnym razie porównaj kod programu z listingu 1.1 z kodem swojego programu, aby mieć pewność, że niczym się nie różnią.

Polecenie `cc` oferuje wiele flag oraz opcji kompilatora. Przykładowo flaga `-o` *`plik`* umożliwia nadanie plikowi wykonywalnemu łatwej do zapamiętania nazwy zamiast nazwy *a.out*. Następujące polecenie uruchamiające kompilator powoduje nadanie plikowi nazwy *`hello`*:

```
% cc -o hello hello.c
% ./hello
Witaj, świecie!
```

Pora przeanalizować kolejne wiersze kodu programu *hello.c*.

## Dyrektywy preprocesora

W pierwszych dwóch wierszach programu *hello.c* użyto dyrektywy preprocesora `#include`, której działanie polega na zastępowaniu jej zawartością określonego pliku dokładnie w tym samym położeniu. Dołączono pliki nagłówkowe `<stdio.h>` i `<stdlib.h>` w celu uzyskania dostępu do zadeklarowanych w nich funkcji, które mogą być następnie wywoływane z poziomu programu. Funkcję `puts` zadeklarowano w pliku nagłówkowym `<stdio.h>`, a makro `EXIT_SUCCESS` zdefiniowano w pliku nagłówkowym `<stdlib.h>`. Jak wskazują nazwy plików, plik nagłówkowy `<stdio.h>` zawiera deklaracje standardowych funkcji operacji wejścia-wyjścia języka C, natomiast plik nagłówkowy `<stdlib.h>` obejmuje deklaracje ogólnych funkcji narzędziowych. Niezbędne jest dołączenie deklaracji wszystkich funkcji biblioteki używanych w programie.

## Funkcja main

Główna część kodu programu z listingu 1.1 rozpoczyna się od następującego wiersza ❶:

```
int main(void) {
```

W tym wierszu definiowana jest funkcja `main` wywoływana podczas uruchamiania programu. Funkcja definiuje główny punkt wejścia programu wykonywanego w środowisku hostowanym, gdy program wywoływany jest z poziomu wiersza poleceń lub innego programu. W języku C zdefiniowano dwa możliwe środowiska wykonywania, czyli *zwykłe* i *hostowane*. Środowisko zwykłe może nie zapewniać systemu operacyjnego. Używane jest zwykle w przypadku programowania na potrzeby systemów wbudowanych. Takie implementacje oferują minimalny zestaw funkcji biblioteki. Nazwa i typ funkcji wywoływanej podczas uruchamiania programu są definiowane w ramach implementacji. W książce przyjęto wykorzystanie głównie środowiska hostowanego.

Funkcję `main` zdefiniowano w celu zwrócenia wartości typu `int`. Słowo kluczowe `void` umieszczono wewnątrz nawiasów okrągłych, aby wskazać, że funkcja nie akceptuje argumentów. Typ `int` to typ całkowitoliczbowy ze znakiem, który może zostać zastosowany do reprezentowania zarówno dodatnich, jak i ujemnych wartości całkowitych, a także zera. Programy napisane w języku C, podobnie do innych języków proceduralnych, składają się z procedur (określanych mianem *funkcji*), które mogą akceptować argumenty i zwracać wartości. Każda funkcja to jednostka robocza wielokrotnego użycia, która może być wywoływana w programie tak często, jak to konieczne. W omawianym przypadku wartość zwracana przez funkcję `main` wskazuje, czy program pomyślnie zakończył pracę. Faktyczne działanie realizowane przez tę konkretną funkcję ❷ to wyświetlenie komunikatu `Witaj, świecie!`:

```
puts(„Witaj, świecie!");
```

Funkcja `puts` to funkcja standardowej biblioteki języka C. Zapisuje ona argument łańcucha w standardowym wyjściu `stdout`, które zwykle reprezentuje konsolę lub okno terminala. Funkcja dołącza do danych wyjściowych znak nowego wiersza. `„Witaj, świecie!"` to literał łańcuchowy przypominający łańcuch tylko do odczytu. Wywołanie tej funkcji powoduje wyświetlenie w terminalu komunikatu `Witaj, świecie!`.

Po zakończeniu pracy programu wskazane będzie zakończenie go. Umożliwia to umieszczona w obrębie funkcji `main` instrukcja `return` ❸, która zwraca wartość całkowitą do środowiska hosta. Inna opcja to wywołanie skryptu:

```
return EXIT_SUCCESS;
```

`EXIT_SUCCESS` to przypominające obiekt makro, które przeważnie rozwijane jest do wartości 0. Makro jest zwykle definiowane w następujący sposób:

```
#define EXIT_SUCCESS 0
```

Każde wystąpienie `EXIT_SUCCESS` zastępowane jest zerem zwracanym następnie do środowiska hosta z poziomu wywołania funkcji `main`. Skrypt wywołujący program może dalej sprawdzić jego status w celu stwierdzenia, czy wywołanie się powiodło. Wartość zwracana przez początkowe wywołanie funkcji `main` odpowiada wywołaniu funkcji `exit` standardowej biblioteki języka C z wartością zwracaną przez funkcję `main` jako jej argument.

Ostatni wiersz omawianego kodu programu ❹ zawiera domykający nawias klamrowy }, który domyka blok kodu rozpoczęty deklaracją funkcji `main`:

```
int main(void) {
// ---fragment kodu---
}
```

Otwierający nawias klamrowy możesz zostać wstawiony w tym samym wierszu co deklaracja lub w osobnym wierszu w następujący sposób:

```
int main(void)
{
// ---fragment kodu---
}
```

Taka decyzja jest podyktowana wyłącznie kwestiami stylu, ponieważ białe znaki (w tym znaki nowego wiersza) nie mają przeważnie żadnego znaczenia składniowego. W książce otwierający nawias klamrowy umieszczam zwykle w wierszu deklaracji funkcji, gdyż jest to bardziej zwięzłe pod względem stylistycznym.

## Sprawdzanie wartości zwracanych funkcji

Funkcje będą często zwracać wartość, która jest wynikiem obliczenia albo wskazuje, czy funkcja pomyślnie zakończyła swoje zadanie. Przykładowo funkcja `puts` użyta w programie wyświetlającym komunikat `Witaj, świecie!` pobiera łańcuch do pokazania i zwraca wartość typu `int`. Funkcja ta zwraca wartość makra `EOF` (ujemna liczba całkowita), jeśli wystąpi błąd zapisu. W przeciwnym razie zwracana jest nieujemna wartość całkowita.

Choć mało prawdopodobne jest, że w przypadku omawianego prostego programu działanie funkcji `puts` zakończy się niepowodzeniem i zwróceniem wartości `EOF`, jest to możliwe. Ponieważ wywołanie funkcji `puts` może się nie powieść i spowodować zwrócenie wartości `EOF`, oznacza to, że Twój pierwszy program C zawiera błąd lub przynajmniej może zostać ulepszony w następujący sposób:

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
  if (puts(„Witaj, świecie!") == EOF) {
    return EXIT_FAILURE;
    // umieszczony tutaj kod nie zostanie wykonany
  }
  return EXIT_SUCCESS;
  // umieszczony tutaj kod nie zostanie wykonany
}
```

Zmodyfikowana wersja przykładowego programu sprawdza, czy wywołanie funkcji `puts` zwraca wartość `EOF` wskazującą błąd zapisu. Jeśli funkcja zwróci tę wartość, program zwraca wartość makra `EXIT_FAILURE` (jego wynikiem jest wartość różna od zera). W przeciwnym razie funkcja powiodła się, a program zwraca wartość `EXIT_SUCCESS` (musi mieć wartość 0). Skrypt wywołujący program może następnie sprawdzić jego status, aby określić, czy jego działanie zakończyło się powodzeniem. Kod znajdujący się za instrukcją `return` to „martwy" kod, który nigdy nie zostanie wykonany. W zmodyfikowanym kodzie programu wskazano go za pomocą komentarzy jednowierszowych. Wszystko, co znajduje się za znakami //, jest ignorowane przez kompilator.

## Sformatowane dane wyjściowe

Funkcja `puts` zapewnia przyjemny i prosty sposób zapisywania łańcucha w wyjściu standardowym `stdout`, ale ostatecznie niezbędne będzie wyświetlenie danych wyjściowych sformatowanych za pomocą funkcji `printf` (na przykład w celu pokazania argumentów innych niż łańcuchy). Funkcja ta pobiera łańcuch formatu definiujący sposób formatowania danych wyjściowych, po którym następuje zmienna liczba argumentów będących faktycznymi wartościami do wyświetlenia. Aby na przykład użyć funkcji `printf` do wyświetlenia komunikatu `Witaj, świecie!`, możesz zastosować następujący wiersz kodu:

```
printf(„%s\n", „Witaj, świecie!");
```

Pierwszy argument to łańcuch formatu `„%s\n"`, gdzie `%s` to specyfikacja konwersji nakazująca funkcji `printf` wczytanie drugiego argumentu (literał łańcuchowy) i wyświetlenie go w wyjściu standardowym `stdout`, a `\n` to alfabetyczna sekwencja wyjścia używana do reprezentowania znaków niegraficznych, która nakazuje funkcji wstawienie nowego wiersza za łańcuchem. Bez sekwencji nowego wiersza kolejne wyświetlane znaki (prawdopodobnie identyfikujące symbole wiersza poleceń) pojawiłyby się w tym samym wierszu. Wywołanie funkcji `printf` zwraca następujące dane wyjściowe:

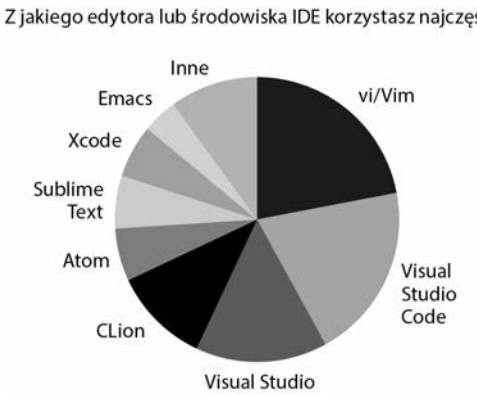
```
Witaj, świecie!
```

Staraj się nie przekazywać funkcji `printf` danych wprowadzanych przez użytkownika jako części pierwszego argumentu, ponieważ może to spowodować lukę w zabezpieczeniach związaną z formatowanymi danymi wyjściowymi (Seacord, 2013 r.).

Najprostszą metodą zwrócenia łańcucha jest zastosowanie funkcji `puts` we wcześniej zaprezentowany sposób. Jeżeli w zmodyfikowanej wersji programu z komunikatem `Witaj, świecie!` skorzystasz z funkcji `printf`, a nie z funkcji `puts`, okaże się, że program przestanie działać, ponieważ funkcja `printf` zwraca status inaczej niż funkcja `puts`. Jeśli zadziała pomyślnie, funkcja `printf` zwróci liczbę wyświetlanych znaków. W przypadku wystąpienia błędu danych wyjściowych lub błędu kodowania funkcja zwróci wartość ujemną. Możesz spróbować zmodyfikowania przykładowego programu, aby w ramach ćwiczenia zastosować funkcję `printf`.

# Edytory i zintegrowane środowiska programistyczne

Do tworzenia programów w języku C możesz użyć różnych edytorów i środowisk IDE. Na rysunku 1.1 pokazano większość stosowanych edytorów. Dane pochodzą z ankiety przeprowadzonej w 2018 r. przez firmę JetBrains.




*Rysunek 1.1. Zastosowanie edytorów i środowisk IDE*

Dokładny zestaw narzędzi zależy od używanego systemu operacyjnego. W książce skoncentrowano się na systemach Linux, Windows i macOS, gdyż są to najpopularniejsze platformy, dla których projektuje się oprogramowanie.

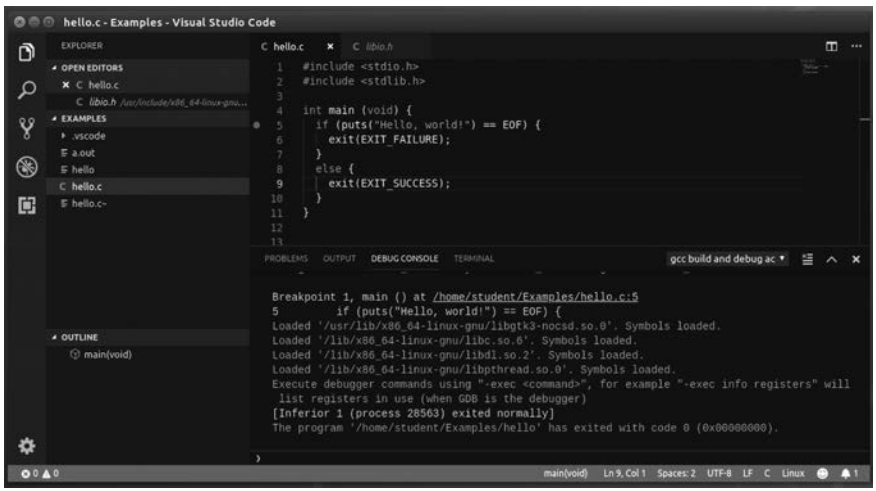
W systemie Microsoft Windows oczywistą opcją wyboru jest środowisko IDE Visual Studio firmy Microsoft (*https://visualstudio.microsoft.com/*). Dostępne jest ono w trzech wersjach: Community, Professional i Enterprise. Zaletą wersji Community jest bezpłatność, natomiast w przypadku dwóch pozostałych za opłatą oferowane są dodatkowe możliwości. W książce użyto wyłącznie wersji Community.

W przypadku systemu Linux wybór jest mniej oczywisty. Dostępne są następujące narzędzia: Vim, Emacs, Visual Studio Code i Eclipse. Vim to edytor używany przez wielu projektantów i zaawansowanych użytkowników. Jest to bazujący na narzędziu vi edytor tekstu utworzony przez Billa Joya w latach 70. ubiegłego wieku dla wersji systemu Unix. Edytor Vim dziedziczy skróty klawiaturowe narzędzia vi, ale też zapewnia funkcjonalność i możliwości rozszerzania nieobecne w tym oryginalnym narzędziu. Masz możliwość opcjonalnego zainstalowania dodatków edytora Vim, takich jak YouCompleteMe (*https://github.com/Valloric/YouCompleteMe/*) lub deoplete (*https://github.com/Shougo/deoplete.nvim/*). Zapewniają one dla programowania w języku C wbudowane uzupełnianie jego kodu.

GNU Emacs to darmowy edytor tekstu oferujący możliwości rozszerzania i dostosowywania. Zasadniczo jest to interpreter języka Emacs Lisp, czyli dialektu języka programowania Lisp z rozszerzeniami obsługującymi edycję tekstu, choć nigdy nie uznałem tego za problem. Niemal cały kod, jaki dotąd napisałem w języku C, był modyfikowany za pomocą edytora Emacs.

*Visual Studio Code* (*VS Code*) to ulepszony edytor kodu obsługujący takie działania programistyczne jak debugowanie, uruchamianie zadań i kontrola wersji (zostało to omówione w rozdziale 11.). Edytor zapewnia po prostu narzędzia, jakich projektant potrzebuje do szybkiego zrealizowania cyklu złożonego z pisania kodu, kompilowania i debugowania. Edytor VS Code działa w systemach macOS, Linux i Windows. Oferowany jest za darmo do celów prywatnych lub odpłatnie w przypadku zastosowań komercyjnych. Instrukcje instalacji dostępne

są dla systemu Linux oraz innych platform[4]. Kiedy używasz systemu Windows, najprawdopodobniej skorzystasz z narzędzia Microsoft Visual Studio. Na rysunku 1.2 pokazano edytor Visual Studio Code użyty w systemie Ubuntu do napisania programu z listingu 1.1, który wyświetla komunikat `Witaj, świecie!`. W oknie konsoli debugowania możesz zauważyć, że zgodnie z oczekiwaniami program zakończył działania z kodem statusu 0.




*Rysunek 1.2. Edytor Visual Studio Code uruchomiony w systemie Ubuntu*

# Kompilatory

Dostępnych jest wiele kompilatorów języka C, dlatego nie będę tutaj omawiać ich wszystkich. Różne kompilatory implementują odmienne wersje standardu języka C. Wiele kompilatorów dla systemów wbudowanych obsługuje jedynie standard C89/C90. Popularne kompilatory dla systemów Linux i Windows działają intensywniej, aby zapewnić obsługę nowszych wersji standardu języka C z uwzględnieniem obsługi standardu C2x.

## GNU Compiler Collection

Pakiet *GCC* (*GNU Compiler Collection*) obejmuje interfejsy dla języków C, C++ i Objective-C, a także dla innych języków (*https://gcc.gnu.org/*). Programowanie z wykorzystaniem tego pakietu wiąże się z realizowaniem odpowiednio zdefiniowanego planu prac projektowych zgodnego z wytycznymi komitetu sterującego rozwojem pakietu GCC.

[4] Instrukcje instalacji w systemie Linux są dostępne na stronie *Visual Studio Code on Linux* pod adresem *https://code.visualstudio.com/docs/setup/linux/*.

Pakiet GCC został zaadaptowany jako standardowy kompilator dla systemów linuksowych, choć dostępne są również wersje dla systemów Microsoft Windows i macOS oraz innych platform. Instalowanie pakietu GCC w systemie Linux nie stanowi problemu. Przykładowo następujące polecenie powinno zainstalować pakiet GCC 8 w systemie Ubuntu:

```
% sudo apt-get install gcc-8
```

Aby sprawdzić, jakiej używasz wersji pakietu GCC, użyj następującego polecenia:

```
% gcc—version
gcc (Ubuntu 8.3.0-6ubuntu1~18.04) 8.3.0
This is free software; see the source for copying conditions. There is NO
Warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Fedora to idealny system do projektowania, jeśli zamierzasz tworzyć oprogramowanie dla dystrybucji Red Hat Enterprise systemu Linux. Do zainstalowania pakietu GCC w systemie Fedora może zostać użyte następujące polecenie:

```
% sudo dnf install gcc
```

## Clang

*Clang* to kolejny popularny kompilator (*https://clang.llvm.org/*). Zainstalowanie go w systemie Linux jest prostym zadaniem. Przykładowo następujące polecenie powinno zainstalować kompilator Clang w systemie Ubuntu:

```
% sudo apt-get install clang
```

Aby sprawdzić, z jakiej wersji kompilatora Clang korzystasz, zastosuj następujące polecenie:

```
% clang --version
clang version 6.0.0-1ubuntu2 (tags/RELEASE_600/final)
Target: x86_64-pc-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
```

## Microsoft Visual Studio

*Microsoft Visual Studio* to najpopularniejsze narzędzie programistyczne dla systemu Windows. Obejmuje ono zarówno samo środowisko IDE, jak i kompilator. Podczas pisania książki, najnowszą wersją narzędzia Visual Studio była wersja 2019

(*https://visualstudio.microsoft.com/downloads/*). Narzędzie oferowane jest razem z narzędziem Visual C++ 2019, które zawiera kompilatory języków C i C++.

Opcje narzędzia Visual Studio możesz ustawić na stronach właściwości projektu. Upewnij się, że na karcie *Advanced* poniżej sekcji *C/C++* włączona jest opcja *Compile as C Code* (`/TC`), a nie opcja *Compile as C++ Code* (`/TP`). Po wpisaniu w nazwie pliku rozszerzenia *.c* domyślnie jest on kompilowany przy użyciu opcji `/TC`. W przypadku zastosowania nazwy pliku z rozszerzeniem *.cpp*, *.cxx* lub jednego z kilku innych rozszerzeń plik zostanie skompilowany za pomocą opcji `/TP`.

# Przenośność

Każda implementacja kompilatora C różni się choć trochę. Kompilatory nieustannie się rozwijają, dlatego na przykład kompilator, taki jak GCC, może zapewniać pełną obsługę standardu C17, ale w przypadku prac zmierzających do wspierania standardu C2x mogą zostać zaimplementowane tylko niektóre jego elementy. W konsekwencji kompilatory obsługują pełne spektrum wersji standardu języka C (w tym wersje przejściowe). Ogólny rozwój implementacji języka C jest powolny. Wiele kompilatorów pozostaje w tyle w stosunku do tempa rozwoju standardu języka C.

Programy napisane w języku C mogą być uznawane za *ściśle zgodne*, jeśli korzystają wyłącznie z tych elementów języka i biblioteki, które określono w standardzie. Takie programy mają cechować się maksymalną przenośnością. Jednakże z powodu skali zachowań charakteryzujących implementacje żaden program utworzony w języku C nie jest ściśle zgodny ani taki nie będzie (i prawdopodobnie nie powinien być). Standard języka C umożliwia pisanie programów *zgodnych*, które mogą być zależne od elementów języka i biblioteki pozbawionych opcji przenośności.

Częstą praktyką jest tworzenie kodu pod kątem pojedynczej implementacji referencyjnej, a czasami kilku implementacji (zależnie od tego, na jakich platformach planowane jest wdrożenie kodu). Standard C Standard jest tym, co zapewnia, że implementacje te nie będą się zbytnio różnić. Ponadto umożliwia on jednoczesne uwzględnienie kilku platform bez konieczności poznawania każdorazowo nowego języka.

W załączniku Annex J dokumentów standardu języka C wymieniono pięć następujących rodzajów problemów związanych z przenośnością, są to:

- zachowanie zdefiniowane w implementacji,
- zachowanie nieokreślone,
- zachowanie niezdefiniowane,
- zachowanie powiązane z ustawieniami regionalnymi,
- wspólne rozszerzenia.

W trakcie poznawania języka C napotkasz przykłady wszystkich pięciu rodzajów zachowań, dlatego ważne jest dokładne zrozumienie ich znaczenia.

## Zachowanie zdefiniowane w implementacji

*Zachowanie zdefiniowane w implementacji* to zachowanie programu, które nie jest określone przez standard języka C. Ponadto może ono powodować różne rezultaty dla poszczególnych implementacji, ale w obrębie jednej implementacji zapewnia spójne i udokumentowane zachowanie. Przykładem zachowania zdefiniowanego w implementacji jest liczba bitów w bajcie.

Zachowania zdefiniowane w implementacji są przeważnie nieszkodliwe, ale mogą powodować problemy przy przenoszeniu do innych implementacji. Gdy tylko to możliwe, unikaj pisania kodu zależnego od zachowań zdefiniowanych w implementacji, które zmieniają się dla poszczególnych implementacji języka C, jakich możesz używać do kompilowania kodu. W załączniku Annex J.3 standardu C Standard znajduje się kompletna lista zachowań zdefiniowanych w implementacji. Deklaracja `static_assert` omówiona w rozdziale 11. umożliwia dokumentowanie zależności od tego rodzaju zachowań. W książce umieszczono odpowiednią uwagę, gdy kod będzie zawierać zachowanie zdefiniowane w implementacji.

## Zachowanie, którego nie określono

*Zachowanie nieokreślone* to zachowanie programu, w przypadku którego standard zapewnia co najmniej dwie opcje. Standard nie narzuca żadnych wymagań dotyczących tego, jaka opcja jest wybierana w danym przypadku. Każde wykonanie danego wyrażenia może oznaczać inne wyniki lub zapewniać inną wartość niż wcześniejsze wykonanie tego samego wyrażenia. Przykładem zachowania nieokreślonego jest sposób przechowywania parametrów funkcji, który może się zmieniać dla poszczególnych wywołań funkcji w obrębie tego samego programu. Unikaj pisania kodu zależnego od zachowań nieokreślonych wymienionych w załączniku Annex J.1 standardu języka C.

## Zachowanie niezdefiniowane

*Zachowanie niezdefiniowane* to zachowanie niedookreślone przez standard języka C lub, wyrażając to bardziej opisowo, „zachowanie powiązane z użyciem nieprzenośnej lub zawierającej błąd konstrukcji programu albo danych z błędem, w przypadku których standard nie wymusza żadnych wymagań". Przykłady zachowania niezdefiniowanego obejmują przepełnienie spowodowane liczbą całkowitą ze znakiem oraz użycie operatora dereferencji dla niepoprawnej wartości wskaźnika. Kod cechujący się zachowaniem niezdefiniowanym często zawiera błąd, ale jest to mniej oczywiste. Zachowania niezdefiniowane są identyfikowane w standardzie w następujących sytuacjach:

- w przypadku naruszenia wymagania identyfikowanego przez określenie „powinno" lub „nie powinno", gdy pojawia się ono poza ograniczeniem,
- w przypadku wyraźnego zidentyfikowania zachowania za pomocą słów „zachowanie niezdefiniowane",
- przez pominięcie jawnej definicji zachowania.

Pierwsze dwa rodzaje zachowań niezdefiniowanych są często określane mianem *jawnych zachowań niezdefiniowanych*. Z kolei trzeci rodzaj zachowań identyfikuje się jako *niejawne zachowania niezdefiniowane*. W przypadku tych trzech rodzajów nie ma różnicy dotyczącej ich znaczenia. Wszystkie opisują zachowania, które są niezdefiniowane. Załącznik Annex J.2, *Undefined behavior*, standardu C Standard zawiera listę jawnych zachowań niezdefiniowanych języka C.

Projektanci często mylnie interpretują zachowania niezdefiniowane jako błędy lub pominięcia w standardzie języka C, ale decyzja dotycząca zaklasyfikowania zachowania jako niezdefiniowanego jest *zamierzona* i *przemyślana*. Zachowania są klasyfikowane jako niezdefiniowane przez komitet rozwijający standard języka C w celu zapewnienia następujących rzeczy:

- zagwarantowania implementującemu tego, że będą występować błędy programu, które są trudne do zdiagnozowania,
- uniknięcia konieczności definiowania niejasnych, skrajnych przypadków, które powodowałyby faworyzowanie jednej strategii implementacji względem innej,
- zidentyfikowania obszarów możliwego, zgodnego rozszerzenia języka, w których implementujący może rozszerzyć język przez zapewnienie definicji oficjalnego zachowania niezdefiniowanego.

Powyższe trzy powody są naprawdę dość różne, ale wszystkie uznaje się za związane z przenośnością. Przykłady wszystkich trzech powodów zostaną zaprezentowane w odpowiednim miejscu, dalej w książce. Kompilatory (implementacje) mają możliwość realizowania następujących rzeczy:

- całkowitego ignorowania zachowania niezdefiniowanego, co może spowodować nieprzewidywalne wyniki,
- działania w udokumentowany sposób charakterystyczny dla środowiska (z przeprowadzaniem diagnostyki lub bez niej),
- zakończenia translacji lub wykonywania (z przeprowadzaniem diagnostyki).

Żadna z wymienionych opcji nie jest doskonała (a zwłaszcza pierwsza), dlatego najlepiej unikać zachowań niezdefiniowanych, z wyjątkiem sytuacji, gdy implementacja określa takie zachowania jako zdefiniowane w celu umożliwienia zastosowania rozszerzenia języka[5].

## Zachowanie powiązane z ustawieniami regionalnymi i wspólne rozszerzenia

*Zachowanie powiązane z ustawieniami regionalnymi* zależy od lokalnych konwencji danej narodowości, kultury i języka, jakie są dokumentowane przez każdą implementację. Wspólne rozszerzenia są powszechnie wykorzystywane w wielu systemach, ale może nie być możliwe przenoszenie ich do wszystkich implementacji.

[5] Czasami kompilatory oferują tryb `pedantic`, który może ułatwić powiadamianie programisty o takich kwestiach związanych z przenośnością.

# Podsumowanie

W rozdziale dowiedziałeś się, jak napisać prosty program w języku C, a następnie skompilować go i uruchomić. Zaprezentowano kilka edytorów i interaktywnych środowisk programistycznych, a także kilka kompilatorów, które mogą zostać użyte do tworzenia programów C w systemach Windows, Linux i macOS. Ogólnie rzecz ujmując, powinno się korzystać z nowszych wersji kompilatorów i innych narzędzi, ponieważ obsługują one zwykle nowsze elementy języka programowania C, a ponadto zapewniają lepsze możliwości diagnozowania i optymalizacji. Może nie być potrzeby sięgania po nowsze wersje kompilatorów, jeśli spowodują, że nie będzie działać utworzony kod, lub przygotowujesz się do wdrożenia kodu i chcesz uniknąć zbędnych modyfikacji w już sprawdzonej aplikacji. Rozdział zakończono omówieniem kwestii przenośności programów napisanych w języku C.

W kolejnych rozdziałach zajmiemy się konkretnymi elementami języka C i jego biblioteki. W następnym rozdziale zaczniemy od obiektów, funkcji i typów.


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

## C: KOD, KTÓRY PRZETRWA PRÓBĘ CZASU!

C jest systemowym językiem programowania, więc zapewnia maksymalną wydajność sprzętu bazowego. Program napisany w C współpracuje bezpośrednio z warstwą sprzętową, co daje pełną kontrolę nad jego wykonywaniem, a sam kod jest krótki, prosty i działa szybko. Równocześnie można korzystać z zalet programowania wysokiego poziomu. W efekcie C od dwóch dekad pozostaje jednym z najpopularniejszych języków programowania. Trzeba jednak pamiętać o bardzo ważnym szczególe: programista, który używa C, musi wiedzieć, co robi.

Ta książka stanowi przystępne wprowadzenie do tworzenia w języku C profesjonalnego kodu wysokiej jakości. Jest adresowana do każdego, kto chce szybko opanować umiejętność pisania poprawnego, przenośnego i bezpiecznego kodu. Ułatwia również zrozumienie kluczowych zagadnień związanych z programowaniem w C, dzięki czemu wkrótce będziesz tworzyć programy, rozwiązywać problemy i budować działające systemy. Omówiono tu także tematykę debugowania, testowania i analizy kodu C. Sporo miejsca poświęcono dobrym praktykom programowania, dzięki którym łatwiej jest tworzyć poprawny i bezpieczny kod. Poszczególne rozdziały zostały uzupełnione zwięzłymi przykładami kodu i ćwiczeniami pozwalającymi utrwalić prezentowaną treść.

### W książce:

- struktura programu napisanego w C, typy podstawowe, operatory, zmienne i funkcje
- przepływ sterowania programu i dynamiczna alokacja pamięci
- kodowanie i typy znaków
- operacje wejścia-wyjścia i standardowe strumienie C
- wykorzystywanie preprocesora
- testowanie, debugowanie i analiza kodu C

**Robert C. Seacord** jest dyrektorem ds. technicznych w firmie NCC Group. Zajmuje się szkoleniami z zakresu tworzenia bezpiecznego kodu w C, C++ i w innych językach. Jest również członkiem międzynarodowej grupy roboczej standaryzacji ISO/IEC JTC1/SC22/WG14. Autor kilku dobrze przyjętych książek i licznych artykułów w czasopismach branżowych.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl



KOD KORZYŚCI
*Sięgnij po więcej!* ▶
ISBN 978-83-283-8343-2
9 788328 383432
INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 69,00 zł
no starch press