helion.pl

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. Leksykon kieszonkowy



Autorzy: Peter Prinz, Ulla Kirch-Prinz
Tłumaczenie: Piotr Imiela
ISBN: 83-7361-083-9
Tytuł oryginału: C Pocket Reference
Format: B5, stron: 164

Książka „C. Leksykon kieszonkowy" składa się z dwóch części: zwięzłego opisu języka C oraz tematycznie ułożonego przewodnika po bibliotece standardowej. Opis języka został przedstawiony na podstawie standardu ANSI i zawiera wprowadzone później rozszerzenia. Dołączony skorowidz ułatwia szybkie znalezienie informacji na temat funkcji, typów i innych interesujących elementów składni.

Książka przedstawia:

- Podstawy języka C
- Typy danych
- Wyrażenia i operatory
- Instrukcje języka C
- Deklaracje
- Funkcje
- Dyrektywy preprocesora
- Biblioteka standardowa

Gdy zawiedzie Cię pamięć i zapomnisz składni danej instrukcji, „C. Leksykon kieszonkowy" przyjdzie Ci z pomocą. Dzięki tej książce szybko znajdziesz potrzebne informacje, bez konieczności wertowania setek stron podręczników.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Wprowadzenie **7**

Podstawy **9**

Struktura programu w języku C 9
Zbiory znaków 11
Identyfikatory 13
Rodzaje i zasięg identyfikatorów 15

Podstawowe typy **17**

Typy całkowite 17
Typy rzeczywiste i zespolone 19
Typ void 22

Stałe **23**

Stałe całkowite 23
Stałe zmiennopozycyjne 25
Stałe znakowe i literały napisowe 26

Wyrażenia i operatory **28**

Operatory arytmetyczne 30
Operatory przypisania 32
Operatory relacji 34
Operatory logiczne 35
Operatory bitowe 36
Operatory dostępu do pamięci 38
Pozostałe operatory 39

Konwersje typów **42**

Promocja całkowita 42
Zwykłe konwersje arytmetyczne 43
Konwersja typów w przypadku przypisań i wskaźników 44

Instrukcje **45**
Bloki i instrukcje wyrażeniowe 45
Skoki 46
Pętle 49
Skoki bezwarunkowe 52

Deklaracje **54**
Ogólna składnia i przykłady 55
Deklaracje złożone 56

Zmienne **57**
Klasy pamięci 57
Inicjalizacja 58

Typy pochodne **60**
Typy wyliczeniowe 60
Struktury, unie i pola bitowe 61
Tablice 67
Wskaźniki 70
Kwalifikatory typów oraz definicje typów 74

Funkcje **77**
Prototypy funkcji 78
Definicje funkcji 79
Wywołania funkcji 82
Funkcje o zmiennej liczbie argumentów 83

Łączenie identyfikatorów **86**

Dyrektywy preprocesora **87**

Biblioteka standardowa **97**

Standardowe pliki nagłówkowe **97**

Wejście i wyjście **98**
Obsługa błędów w funkcjach wejścia-wyjścia 100
Ogólne funkcje dostępu do plików 101
Plikowe funkcje wejścia-wyjścia 104

Zakresy liczb oraz ich klasyfikacja *114*
Zakresy wartości typów całkowitych 114
Zakres i precyzja typów zmiennopozycyjnych rzeczywistych 116
Klasyfikacja liczb zmiennopozycyjnych 116

Funkcje matematyczne *119*
Funkcje matematyczne dla typów całkowitych 119
Funkcje matematyczne dla typów rzeczywistych 120
Optymalizacja efektywności wykonywania 123
Funkcje matematyczne dla typów zmiennopozycyjnych zespolonych 124
Makroinstrukcje niezależne od typów 125
Obsługa błędów w funkcjach matematycznych 126
Środowisko zmiennopozycyjne 127

Klasyfikacja znaków i zmiana ich wielkości *131*

Obsługa łańcuchów *133*
Konwersja pomiędzy łańcuchami i wartościami liczbowymi 136
Przekształcenia znaków wielobajtowych 138

Wyszukiwanie i sortowanie *140*

Obsługa bloków pamięci *141*

Dynamiczne zarządzanie pamięcią *142*

Czas i data *143*

Sterowanie procesami *144*
Komunikacja z systemem operacyjnym 145
Sygnały 146
Skoki nielokalne 148
Obsługa błędów w funkcjach systemowych 149

Ustawienia narodowe *150*

Skorowidz *155*

C.
Leksykon kieszonkowy

Wprowadzenie

Język programowania C został opracowany w latach siedemdziesiątych przez Dennisa Ritchiego z Bell Labs (w Murray Hill, w stanie New Jersey) podczas przygotowywania implementacji systemu operacyjnego Unix, przeznaczonej dla komputera DEC PDP-11. Korzenie języka C sięgają nieposiadającego typów języka programowania BCPL (ang. *Basic Combined Programming Language*, którego autorem był M. Richards) oraz języka B (opracowanego przez K. Thompsona). W 1978 roku Brian Kernighan i Dennis Ritchie przedstawili pierwszy dostępny publicznie opis języka C, znany obecnie jako standard K&R.

C jest językiem o wysokim stopniu przenośności, ukierunkowanym na architekturę współczesnych komputerów. Obecnie język sam w sobie jest stosunkowo mały i posiada niewiele elementów związanych ze sprzętem. Nie zawiera na przykład instrukcji wejścia-wyjścia ani technik zarządzania pamięcią. Funkcje realizujące te zadania są dostępne w obszernej standardowej bibliotece języka C.

Konstrukcja języka C posiada istotne zalety:

- kod źródłowy jest w dużym stopniu przenośny,
- kod maszynowy jest efektywny,
- kompilatory języka C są dostępne dla wszystkich istniejących systemów.

Pierwsza część leksykonu opisuje język C, natomiast jego drugą część poświęcono standardowej bibliotece języka C. Język został opisany na podstawie standardu ISO X3.159. Standard ten odpowiada międzynarodowemu standardowi ISO/IEC 9899, który został przyjęty przez Międzynarodową Organizację Normalizacyjną w 1990 roku, a następnie został poprawiony w latach 1995 i 1999. Standard ISO/IEC 9899 można zamówić za pośrednictwem internetowej witryny ANSI, znajdującej się pod adresem *http://webstore.ansi.org*.

Standard ustanowiony w 1995 roku jest obecnie obsługiwany przez wszystkie popularne kompilatory języka C. Natomiast w wielu kompilatorach języka C nie zostały jeszcze zaimplementowane nowe rozszerzenia języka, zdefiniowane w opublikowanej w 1999 roku wersji standardu (nazywanej w skrócie „ANSI C99"), w związku z czym zostały one w niniejszej książce w specjalny sposób oznaczone. Nowe typy, funkcje i makroinstrukcje, wprowadzone przez standard ANSI C99, oznaczono gwiazdką umieszczoną w nawiasie (*).

Konwencje typograficzne

W książce stosowane są następujące konwencje typograficzne:

Kursywa
: Jest stosowana przy wprowadzaniu nowych pojęć oraz do wyróżniania nazw plików.

`Czcionka o stałej szerokości znaków`
: Jest używana w kodzie programów w języku C oraz w nazwach funkcji i dyrektyw.

`Kursywa o stałej szerokości znaków`
: Oznacza wymienne elementy w obrębie składni kodu.

`Pogrubiona czcionka o stałej szerokości znaków`
: Wykorzystywana jest w celu zwrócenia szczególnej uwagi na wyróżnione fragmenty kodu.

Podstawy

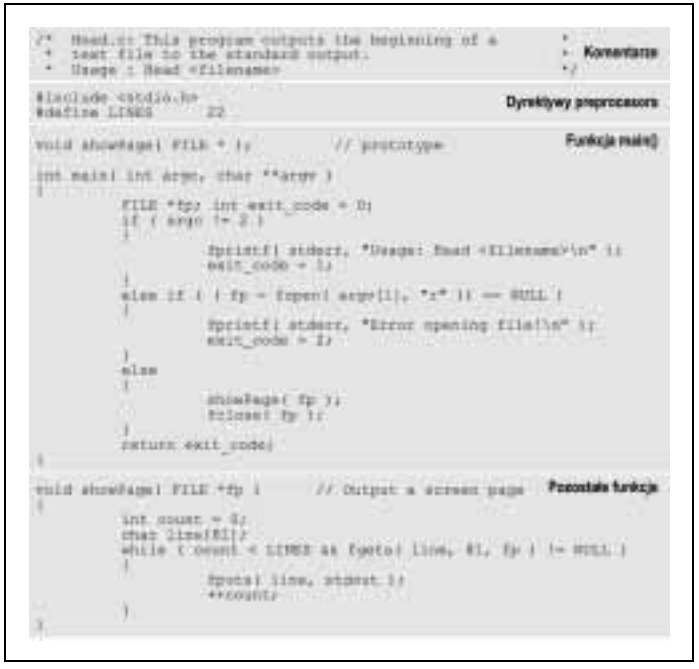
Program w języku C składa się z odrębnych bloków składowych, nazywanych *funkcjami*, mogących się wzajemnie wywoływać. Każda funkcja realizuje określone zadanie. Gotowe funkcje są dostępne w standardowej bibliotece języka; pozostałe funkcje są w miarę potrzeby tworzone przez programistę. Szczególną nazwą funkcji jest *main()* — oznacza ona pierwszą funkcję, która jest wywoływana podczas uruchamiania programu. Wszystkie pozostałe funkcje są podprogramami.

Struktura programu w języku C

Rysunek 1. prezentuje strukturę programu w języku C. Przedstawiony program składa się z funkcji: `main()` oraz `showPage()`. Program drukuje początek pliku tekstowego, którego nazwa została podana w wierszu poleceń podczas jego uruchomienia.

Program źródłowy w języku C składa się z *instrukcji* (ang. *statements*) tworzących funkcje, a także niezbędnych deklaracji oraz dyrektyw preprocesora. W przypadku niewielkich programów, kod źródłowy jest zapisywany w pojedynczym *pliku źródłowym* (ang. *source file*). Większe programy napisane w języku C składają się z wielu plików źródłowych, które mogą być oddzielnie redagowane i kompilowane. Każdy z takich plików źródłowych zawiera funkcje tworzące pewną logiczną całość — np. funkcje wyprowadzające informacje do terminala. Informacje, które są potrzebne w wielu plikach źródłowych — takie jak deklaracje — umieszczane są w plikach nagłówkowych. Mogą one następnie zostać włączone do każdego pliku źródłowego za pomocą dyrektywy `#include`.

Nazwy plików źródłowych posiadają zakończenie *.c*, natomiast nazwy plików nagłówkowych kończą się na *.h*. Plik źródłowy wraz z włączonymi do niego plikami nagłówkowymi nazywany jest *jednostką translacji*.



Rysunek 1. Program w języku C

Nie ma określonej z góry kolejności, w której muszą być definiowane funkcje. Funkcja `showPage()`, przedstawiona na rysunku 1., mogłaby znajdować się również przed funkcją `main()`. Funkcje nie mogą być jednak definiowane wewnątrz innych funkcji.

Kompilator, przetwarzając kolejno każdy z plików źródłowych, rozkłada jego zawartość na *leksemy* (elementy leksykalne — ang. *tokens*), którymi są m.in. nazwy funkcji i operatory. Leksemy mogą być oddzielone od siebie jednym lub większą liczbą niewidocznych znaków, takich jak spacja, tabulacja lub znak nowego wiersza. Tak więc, znaczenie ma jedynie kolejność leksemów

w pliku. Układ kodu źródłowego — na przykład miejsca, w których łamane są wiersze, czy też stosowanie wcięć — jest nieistotny. Wyjątek od tej reguły stanowią jednak *dyrektywy preprocesora*. Są one poleceniami przeznaczonymi do wykonania przez preprocesor danego programu przed kompilacją, a każde z nich zajmuje jeden wiersz, rozpoczynający się znakiem **#**.

Komentarze są dowolnymi ciągami znaków, zawartymi pomiędzy parami znaków /* i */ lub znajdującymi się pomiędzy znakami // a końcem bieżącego wiersza. Na wstępnych etapach translacji, zanim jeszcze zostanie utworzony jakikolwiek kod wynikowy, każdy komentarz jest zastępowany *pojedynczą* spacją. Następnie wykonywane są dyrektywy preprocesora.

Zbiory znaków

Język ANSI C określa dwa zbiory znaków. Pierwszym z nich jest *źródłowy zbiór znaków* (ang. *source character set*), będący zbiorem znaków, które mogą zostać użyte w pliku źródłowym. Drugim jest natomiast *zbiór znaków wykonania programu* (ang. *execution character set*), składający się z wszystkich znaków interpretowanych w trakcie wykonywania programu — będących na przykład znakami zawartymi w stałych tekstowych.

Każdy z tych zbiorów znaków zawiera w sobie *podstawowy zbiór znaków* (ang. *basic character set*), obejmujący:

- 52 wielkie i małe litery alfabetu łacińskiego:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

- Dziesięć cyfr dziesiętnych (w przypadku których wartość każdego znaku po 0 jest o jeden większa od wartości poprzedniej cyfry):

```
0 1 2 3 4 5 6 7 8 9
```

- 29 wymienionych poniżej znaków graficznych:

```
! " # % & ' ( ) * + , - . / : ;
< = > ? [ \ ] ^ _ { | } ~
```

- Pięć niewidocznych znaków:

  odstęp, tabulacja pozioma, tabulacja pionowa, znak nowego wiersza, znak wysuwu strony.

Dodatkowo, podstawowy zbiór znaków wykonania programu zawiera:

- Pusty znak `\0`, kończący ciąg znaków.
- Znaki sterujące, reprezentowane za pomocą prostych *sekwencji sterujących* (ang. *escape sequences*), przedstawionych w tabeli 1., umożliwiających sterowanie urządzeniami wyjściowymi, takimi jak terminale lub drukarki.

Tabela 1. Standardowe sekwencje sterujące

Sekwencja sterująca	Działanie w przypadku terminalu	Sekwencja sterująca	Działanie w przypadku terminalu
`\a`	Alarm (sygnał dźwiękowy)	`\'`	Znak '
`\b`	Znak cofania	`\"`	Znak "
`\f`	Wysuw strony	`\?`	Znak ?
`\n`	Znak nowego wiersza	`\\`	Znak \
`\r`	Znak powrotu karetki	`\o` `\oo` `\ooo` (o — cyfra ósemkowa)	Znak o podanym kodzie ósemkowym
`\t`	Tabulacja pozioma	`\xh..` (h.. — ciąg cyfr szesnastkowych)	Znak o podanym kodzie szesnastkowym
`\v`	Tabulacja pionowa		

Wszelkie inne znaki — w zależności od konkretnego kompilatora — mogą być używane w komentarzach, ciągach znaków i stałych napisowych. Mogą one na przykład zawierać symbol dolara lub znaki diakrytyczne. Użycie takich znaków może mieć jednak wpływ na przenośność programu.

Zbiór wszystkich możliwych do wykorzystania znaków jest nazywany *powiększonym zbiorem znaków* (ang. *extended character set*) i stanowi on zawsze nadzbiór podstawowego zbioru znaków.

Niektóre języki wykorzystują znaki, do zapamiętania których potrzeba więcej niż jednego bajtu. Takie *znaki wielobajtowe* (ang. *multibyte characters*) mogą znajdować się w powiększonym zbiorze znaków. Ponadto, standard ANSI C99 udostępnia typ całkowity `wchar_t` (ang. *wide character type — rozszerzony typ znakowy*), o wielkości wystarczającej do reprezentowania znaków zawartych w powiększonym zbiorze znaków. Często stosowany jest nowoczesny system kodowania znaków *Unicode*, poszerzający standardowy kod ASCII w sposób umożliwiający reprezentację około 35 000 znaków pochodzących z 24 krajów.

C99 wprowadza również *sekwencje trójznaków* (ang. *trigraph sequences*). Sekwencje te, wymienione w tabeli 2., mogą być używane do wprowadzania znaków graficznych, które nie są dostępne na wszystkich klawiaturach. Na przykład, do wprowadzenia symbolu „potoku" `|` można użyć sekwencji `??!`.

Tabela 2. Sekwencje trójznaków

Trójznak	`??=`	`??(`	`??/`	`??)`	`??'`	`??<`	`??!`	`??>`	`??-`
Znaczenie	`#`	`[`	`\`	`]`	`^`	`{`	`|`	`}`	`~`

Identyfikatory

Identyfikatorami są nazwy zmiennych, funkcji, makroinstrukcji, typów itd. Ich tworzenie podlega następującym regułom:

- Identyfikator składa się z ciągu liter (od `A` do `Z` i od `a` do `z`), cyfr (od 0 do 9) oraz znaków podkreślenia (_).
- Pierwszym znakiem identyfikatora nie może być cyfra.
- W identyfikatorach rozróżniane są małe i wielkie litery.
- Długość identyfikatora nie jest ograniczona. Jednakże na ogół znaczących jest tylko jego pierwszych 31 znaków.

Słowa kluczowe (ang. *keywords*) są słowami zastrzeżonymi i nie mogą być używane w charakterze identyfikatorów. Poniżej zamieszczono listę słów kluczowych języka C:

`auto`	`enum`	`restrict`(*)	`unsigned`
`break`	`extern`	`return`	`void`
`case`	`float`	`short`	`volatile`
`char`	`for`	`signed`	`while`
`const`	`goto`	`sizeof`	`_Bool`(*)
`continue`	`if`	`static`	`_Complex`(*)
`default`	`inline`(*)	`struct`	`_Imaginary`(*)
`do`	`int`	`switch`	
`double`	`long`	`typedef`	
`else`	`register`	`union`	

Nazwy zewnętrzne, czyli identyfikatory funkcji i zmiennych łączonych zewnętrznie, mogą podlegać dodatkowym ograniczeniom, w zależności od używanego programu łączącego — w przenośnych programach, napisanych w języku C, nazwy zewnętrzne powinny być dobierane w taki sposób, by znaczenie miało tylko ich pierwszych osiem znaków, również w przypadku gdy program łączący *nie rozróżnia* wielkich i małych liter.

Poniżej podano przykłady identyfikatorów:

Poprawne: `a, DM, dm, FLOAT, _var1, topOfWindow`

Niepoprawne: `do, 586_cpu, zähler, nl-flag, US_$`

Rodzaje i zasięg identyfikatorów

Każdy identyfikator należy do jednej z podanych poniżej czterech kategorii:

- *Nazwy etykiet*.
- *Znaczniki* (ang. *tags*) struktur, unii i wyliczeń. Są to identyfikatory występujące bezpośrednio po słowach kluczowych `struct`, `union` lub `enum` (patrz rozdział „Typy pochodne").
- Nazwy *składowych* struktur lub unii. Każda struktura i unia posiada odrębną przestrzeń nazw, zawierającą jej składowe.
- Wszystkie pozostałe identyfikatory, nazywane *zwykłymi identyfikatorami*.

Identyfikatory należące do różnych kategorii mogą być takie same. Na przykład, nazwa etykiety może zostać również użyta w charakterze nazwy funkcji. Ponowne użycie identyfikatorów zdarza się najczęściej w przypadku struktur — ten sam ciąg znaków może zostać użyty do oznaczenia typu struktury, jednej z jej składowych, oraz zmiennej — jak w poniższym przykładzie:

```
struct person {char *person; /*...*/} person;
```

Te same nazwy mogą być również użyte jako nazwy składowych różnych struktur.

Każdy identyfikator znajdujący się w kodzie źródłowym posiada swój *zasięg* (ang. *scope*). Zasięgiem nazywamy tę część programu, w której można użyć danego identyfikatora. Poniżej wymieniono cztery możliwe zasięgi identyfikatorów:

Prototyp funkcji

Identyfikatory znajdujące się na liście deklaracji argumentów prototypu funkcji (nie definicji funkcji) posiadają *zasięg prototypu funkcji*. Ponieważ identyfikatory te nie mają żadnego znaczenia poza samym prototypem, są tylko czymś więcej niż komentarzami.

Funkcja

Zasięg funkcji posiadają jedynie *nazwy etykiet*. Ich użycie jest ograniczone do bloku funkcji, w którym etykiety te zostały zdefiniowane. Nazwy etykiet muszą być unikatowe w obrębie funkcji. Instrukcja `goto` powoduje skok do instrukcji oznaczonej etykietą, znajdującej się w obrębie tej samej funkcji.

Blok

Identyfikatory nie będące etykietami, zadeklarowane w obrębie bloku, posiadają *zasięg bloku*. Argumenty definicji funkcji mają również zasięg bloku. Zasięg bloku rozpoczyna się deklaracją identyfikatora, natomiast kończy się zamykającym nawiasem klamrowym (`}`).

Plik

Identyfikatory zadeklarowane na zewnątrz wszystkich bloków i list argumentów posiadają *zasięg pliku*. Zasięg pliku rozpoczyna się w miejscu deklaracji identyfikatora i rozciąga się do końca pliku źródłowego.

Identyfikator nie będący nazwą etykiety nie musi być *widoczny* w całym swoim zasięgu. Jeżeli, na przykład, identyfikator tego samego rodzaju i posiadający taką samą nazwę, jak istniejący już identyfikator, zostanie zadeklarowany w zagnieżdżonym bloku, to zewnętrzna deklaracja identyfikatora zostanie tymczasowo zasłonięta. Staje się ona ponownie widoczna w miejscu, gdzie kończy się zasięg wewnętrznej deklaracji.