# Wstęp do programowania

## na bazie języka C

S. Prata, Język C. Szkoła programowania, Wydanie V, Helion 2006

B.W. Kernighan, D.M. Ritchie, Język C, WNT, Warszawa 1988

J. Bielecki, Encyklopedia języka C, WKiŁ, Warszawa 1989

Język C - 1972 - Dennis Ritchie (Bell Labs) wspólnie z Kenem Thompsonem (UNIX)

## C - język dla programistów.

**Wydajność**
- wykorzystanie możliwości komputerów,
- niewielka objętość programów i dużą szybkość działania,
- wysoki stopień precyzji, porównywalna z asemblerem.

**Przenośność**
- programy napisane w C na jednej platformie systemowej mogą być uruchamiane na innych platformach natychmiast lub po minimalnych modyfikacjach.

**Moc i elastyczność**
- większa część systemu operacyjnego UNIX jest napisana w C,
- wiele kompilatorów innych języków zostało napisanych w C.

**Ukierunkowanie na programistę**
- oferuje dostęp do sprzętu i pozwala na operowanie poszczególnymi bitami w pamięci.
- udostępnia bogaty wybór operatorów pozwalający wyrażać się w zwięzły sposób,
- większość implementacji języka C zawiera także duży zbiór przydatnych funkcji

Słabe strony - swoboda wyrażania się, jaką daje język C, wymaga także większej odpowiedzialności. Ceną wolności jest nieustanna czujność.

- zwięzłość C, połączona z bogactwem oferowanych operatorów, pozwala tworzyć kod źródłowy charakteryzujący się niezwykłym wprost zawikłaniem.

C++

Wielu producentów oprogramowania preferuje C++, zwłaszcza przy pracy nad dużymi projektami.

- C++ Jest niemal dokładnym nadzbiorem C - niemal każdy poprawny program w C jest także poprawnym programem w C++.
- W zasadzie każdy, poprawnie napisany, złożony program w C jest programem obiektowym.
- C++ rozszerza C o wiele przydatnych narzędzi, w tym dotyczących programowania obiektowego.

# Siedem kroków pisania programów w C:

## Krok 1: Określenie celów programu

- jakich danych program będzie potrzebował

- jakie informacje powinien przedstawić użytkownikowi.

## Krok 2: Projektowanie programu

- jak powinien wyglądać interfejs użytkownika

- jaki powinien być sposób organizacji programu

- kto będzie użytkownikiem program

- ile czasu pozostało na jego ukończenie

- sposób reprezentacji danych w programie (i ewentualnie w plikach zewnętrznych)

- metody, które zostaną użyte do ich przetwarzania

## Krok 3: Pisanie kodu

- przełożenie projektu na język C

- dokumentacja programu (komentarze w kodzie)

## Krok 4: Kompilacja

- Kompilator jest programem służącym do przetwarzania kodu źródłowego na kod wykonywalny.

- Kompilatory zajmują się także włączaniem kodu z bibliotek do programu docelowego.

- Wynikiem końcowym procesu kompilacji jest plik wykonywalny zawierający kod zrozumiały dla komputera.

- Kompilator sprawdza również, czy program zawiera błędy składniowe polegające na nieprzestrzeganiu zasad gramatyki  języka C.

- W przypadku znalezienia błędów są one sygnalizowane, a plik wykonywalny nie zostaje utworzony.

## Krok 5: Uruchomienie programu

## Krok 6: Testowanie i usuwanie błędów

- Czy program spełnia  wytyczone założenia?  Czy program zawiera błędy semantyczne?

Błąd semantyczny polega na zastosowaniu zasad języka we właściwy sposób, ale w niewłaściwym celu.

Proces szukania i naprawiania błędów nosi nazwę odpluskwiania lub debugowania (ang. debugging).

## Krok 7: Modyfikacja programu

- poprawianie efektywności, ergonomiczności interfejsu itp.

# Mechanika programowania

Program w języku C zapisujemy w pliku tekstowym zwanym **plikiem kodu źródłowego** lub **plikiem źródłowym**. Większość środowisk wymaga, aby nazwa tego pliku miała rozszerzenie **.c**

Kompilacja polega na przetworzeniu **kodu źrodłowego** na **kod maszynowy** i umieszczeniu go w **pliku kodu obiektowego** (ang. objectfile). Nie jest to jeszcze **plik wykonywalny**.

Pierwszym elementem, którego brakuje w pliku obiektowym jest tzw. **kod startowy** (ang. start-up code), który służy za interfejs pomiędzy programem a systemem operacyjnym.

Drugim brakującym elementem jest **kod procedur bibliotecznych**. Niemal wszystkie programy w C wykorzystują procedury (zwane funkcjami) będące częścią standardowej biblioteki języka.

Rola **linkera** polega na scaleniu - kodu obiektowego, standardowego kodu startowego dla danego systemu oraz kodu pochodzącego z bibliotek  i ostatecznie utworzeniu z nich jednego pliku **wykonywalnego**.

## Prosty przykład

```
#include <stdio.h>          // dołączenie innego pliku
int main(void)              // nazwa funkcji
      {                     // początek treści funkcji
      int num;              // instrukcja deklaracji
      num = 1;              // instrukcja przypisania
      printf("Jestem prostym ");     // instrukcja pisania
      printf("komputerem.\n");       // instrukcja pisania
      printf("Moja ulubiona liczba jest %d, bo jest pierwsza.\n",num);   // instrukcja pisania
      return 0;                      // instrukcja zwrotu
      }                              // koniec treści funkcji
```

wynik działania programu

```
Jestem prostym komputerem.
Moja ulubiona liczba jest 1, bo jest pierwsza.
```

## Nazwy

Znaki do dyspozycji to litery (małe i wielkie), cyfry oraz znak podkreślenia _.

Pierwszy znak każdej nazwy nie może być cyfrą.

Przykład

```
/* st_cale.c — zamienia 2 stopy na cale */
#include <stdio,h>
int main(void)
        {
        int stopy,   cale;
        stopy = 2;
        cale = 12  *  stopy;
        printf("%d stopy równają sie %d calom! \n",  stopy, cale);
        return 0;
        }
```

## Słowa kluczowe

Słowa kluczowe stanowią słownictwo języka C.

Ponieważ mają one specjalne znaczenie, nie można wykorzystywać ich jako nazw zmiennych.

| | | | |
|---|---|---|---|
| auto | double | inline | static |
| break | else | int | struct |
| case | enum | long | switch |
| char | extern | register | typedef |
| complex | float | restrict | union |
| const | for | return | unsigned |
| continue | goto | short | void |
| default | if | signed | volatile |
| do | imaginary | sizeof | while |

# Zmienne i stałe

Niektóre dane są ustalone przed uruchomieniem programu i pozostają niezmienione przez cały czas jego działania - są to **stałe** (ang. *constants*).

Dane, które mogą się zmieniać w trakcie działania programu, nazywamy **zmiennymi** (ang. *variables*).

W naszym przykładowym programie **cale** jest zmienną, a **12** - stałą.

*Słowa kluczowe typów danych*

| **Pierwotne słowa kluczowe** | **Dodatkowe słowa kluczowe** |
|---|---|
| int | signed |
| long | void |
| unsigned | volatile |
| const | |
| char | |
| float | |
| double | |

- nie mogą być użyte jako nazwy

# Typy danych w C

## Typ int

Typ **int** jest liczbą całkowitą ze znakiem (ang. signed integer).

*Deklaracja zmiennej typu* **int**

```
int eligent;
int lwy, tygrysy, lamparty;
```

*Inicjalizacja zmiennej*

Inicjalizacja zmiennej oznacza przypisanie jej pierwotnej (początkowej) wartości.

```
int lwy = 21;
int tygrysy = 32, lamparty = 14;
int konie, krowy = 94;        // prawidłowy, ale kiepski styl
```

*Stałe typu* **int**

21, -32, 14 , 94

## Wyświetlanie wartości typu int

```
/* printf.c  - przedstawia niektóre właściwości funkcji printf()  */
#include <stdio.h>
int main(void)
        {
        int dziesięć = 10;
        printf("Właściwy sposób:  ");
        printf("%d minus %d równa się %d\n", dziesięć, 2, dziesięć - 2 );
        printf("Błędny sposób:  ");
        printf("%d minus %d równa się %d\n", dziesięć );    // brakuje 2 argumentów
        return 0;
        }
```

wynik:

```
Właściwy sposób:  10 minus 2 równa się 8
Błędny sposób:  10 minus 342 równa się 6618680
```

## Wartości ósemkowe i szesnastkowe

Przedrostek **0x** lub **0X** oznacza *wartość szesnastkową*,
- stąd, chcąc zapisać liczbę **16** w systemie szesnastkowym, piszemy **0x10** lub **0X10**.

Przedrostek **0** (zero) oznacza *wartość ósemkową* - liczbę **16** w systemie ósemkowym zapisujemy więc **020**

*Wyświetlanie wartości ósemkowych i szesnastkowych*

```
/* systemy.c — wyświetla 100 w zapisie dziesiętnym, ósemkowym i szesnastkowym    */
#include <stdio.h>
int main(void)
{
   int  x  = 100;
   printf("dziesiętny = %d; ósemkowy = %o; szesnastkowy = %x\n", x, x, x);
   return 0;
}
```

wynik:

dziesiętny = 100; ósemkowy = 144;  szesnastkowy = 64

## Inne typy całkowite

- Typ **short int** lub krócej **short** może (lecz nie musi) zajmować mniej pamięci niż int.
- Typ **long int** lub **long** może zajmować więcej pamięci niż **int**, co pozwala na wyrażanie większych wartości całkowitych.
- Typ **unsigned int** lub po prostu **unsigned** służy do deklarowania zmiennych, które przyjmują tylko wartości nieujemne.
- Kompilatory uznają także typy **unsigned long int** (lub **unsigned long**) oraz **unsigned short int** (lub **unsigned short**).

*Deklarowanie innych typów całkowitych*

**long int** estine;
**long** johns;
**short int** erns;
**short** ribs;
**unsigned int** s_count;
**unsigned** players;
**unsigned long** headcount;
**unsigned short** yesvotes;

*Stałe typu long*

Aby spowodować traktowanie stałej jako wartości typu **long**, dodajemy do liczby przyrostek **L**.

Komputer z 16-bitowym typem **int** i 32-bitowym typem **long** zapisze liczbę **7** w 16 bitach, a liczbę **7L** w 32 bitach.

Przyrostek **L** można również stosować do ósemkowych i szesnastkowych liczb całkowitych, np. **020L**, **0x10L**.

*Wyświetlanie wartości typów long, short i unsigned*

```
/* print2.c — dalsze własności funkcji printf () */
#include <stdio.h>
int main(void)
{
   // system z 32-bitowym typem int
   //     i 16-bitowym typem short
   unsigned int un = 3000000000;  short sn = 200;
   long ln = 65537;
   printf("un = %u,  a  nie  %d\n",  un,  un);
   printf("sn = %hd  i  %d\n",  sn,  sn);
   printf("ln = %ld, a nie  %hd\n",  ln,  ln);
   return 0;
}
```

wynik:

```
un = 3000000000,   a nie -1294967296
sn = 200 i 200
ln = 65537,   a nie 1
```

## Typ char

Typ **char** służy do przechowywania znaków, takich jak litery czy znaki przestankowe.

Z technicznego punktu widzenia jest on typem całkowitym.
Typ **char** w rzeczywistości przechowuje liczby całkowite - kod numeryczny (ASCII), w którym określone liczby całkowite reprezentują określone znaki.

### *Deklarowanie zmiennych typu char*

```
char t;
char akter, yzraa;
```

### *Stałe znakowe i inicjalizacja zmiennych*

```
char klasa = 'A';
char klasa = 65;      // kiepski styl
```

*Znaki niedrukowane*

**char** dzwiek = 7; // sygnał dźwiękowy

| **Sekwencja** | |
|---|---|
| \a | *Alarm* |
| \b | *Znak cofania (back space)* |
| \f | *Wysuw strony (fomfeed)* |
| \n | *Nowa linia (newline)* |
| \r | *Powrót karetki (carriage retum)* |
| \t | *Tabulator poziomy* |
| \v | *Tabulator pionowy* |
| \\ | *Lewy ukośnik„\" (backslash)* |
| \' | *Apostrof* |
| \" | *Cudzysłów* |
| \0oo | *Wartość ósemkom (oo oznacza cyfry ósemkowe)* |
| \xhh | *Wartość szesnastkowa (hh oznacza cyfry szesnastkowe)* |

*Wyświetlanie znaków*

```
/* znakkod.c — wyświetla znak i odpowiadający mu kod */
#include <stdio.h>
int main(void)
 {
    char ch;
    printf("Wpisz jakiś znak\n");
    scanf ("%c",   &ch);      // użytkownik podaje znak
    printf("Kod znaku %c to %d.\n",   ch,  ch);
    return 0;
 }
```

przykładowy wynik:

```
Wpisz jakiś znak.
C
Kod znaku C to 67.
```

## Typy float i double

| *Liczba* | *Notacja naukowa* | *Notacja wykładnicza* |
|---|---|---|
| 1 000 000 000 | = $1.0* 10^{9}$ | = 1.0e9 |
| 123 000 | = $1.23*10^{5}$ | = 1.23e5 |
| 322.56 | = $3.2256*10^{2}$ | = 3.2256e2 |
| 0.000056 | = $5.6* 10^{-5}$ | = 5.6e-5 |

*Deklarowanie zmiennych zmiennoprzecinkowych*

**float** paweł, gawel;
**double** trouble;
**float** pianek = 6.63e-34;
**long double** pkb;

*Stałe zmiennoprzecinkowe*

-1.56E+12  2.87e-3  3.14159  .2  4el6  .8E-5  100.

Jeśli liczba zmiennoprzecinkowa nie posiada przyrostka, przechowywana jest jako typ **double**.

za pomocą przyrostka **f** lub **F** możliwa jest zmiana tego standardowego zachowania

przykład:  2.3f  9.11E9F

*Wyświetlanie wartości zmiennoprzecinkowych*

```
/* pokaz_fl.c — wyświetla wartość typu float na dwa sposoby */
#include <stdio.h>
int main(void)
  {
  float wartość = 32000.0;
  printf("%f można także zapisać %e\n", wartość, wartość};
  return 0;
  }
```

wynik: 32000.000000 można także zapisać 3.200000e+004

## Podsumowanie: podstawowe typy danych

*Słowa kluczowe:*

Podstawowe typy danych deklarujemy za pomocą ośmiu słów kluczowych **- int, long, short, unsigned, char, float, double** i **signed**.

*Typy całkowite ze znakiem:*

Typy te mogą osiągać wartości ujemne lub dodatnie.

**int**: Podstawowy typ całkowity. Jego rozmiar jest dostosowany do komputera, ale zgodnie ze standardem ANSI - nie może być mniejszy niż 16 bitów.

**short** lub **short int**: Jego rozmiar jest nie większy niż rozmiar typu int (może być mniejszy). Standard ANSI gwarantuje dla typu short rozmiar co najmniej 16 bitów.

**long** lub **long int**: Jego zakres jest co najmniej tak duży, jak typu int (może być większy). Standard ANSI gwarantuje dla typu long rozmiar co najmniej 32 bitów.

*Typy całkowite bez znaku:*

Typy te mogą osiągać tylko wartości nieujemne, co zwiększa maksymalną wartość, jaką można w nich przechować. Aby zadeklarować zmienną całkowitą bez znaku, przed słowem określającym żądany typ dodajemy słowo kluczowe **unsigned**, np. **unsigned int**, **unsigned long**, **unsigned short**. Samo słowo **unsigned** oznacza typ **unsigned int**.

*Typy znakowe:*

Służą one do przechowywania symboli typograficznych, takich jak **A**, **&** czy **+**.
Typ **char** z definicji zajmuje 1 bajt pamięci.
**char**: jest to słowo kluczowe typu znakowego.
To, czy typ **char** posiada znak, zależy od implementacji języka C. Standard ANSI C pozwala określić żądaną postać typu przy pomocy słów **signed** i **unsigned**.

*Typy zmiennoprzecinkowe:*

Typy te mogą osiągać wartości ujemne lub dodatnie.
**float**: podstawowy typ zmiennoprzecinkowy; potrafi dokładnie wyrazić przynajmniej sześć cyfr znaczących.
**double**: większa jednostka do przechowywania wartości zmiennoprzecinkowych; typ ten może zmieścić więcej cyfr znaczących i większe wykładniki niż float.
**long double**: jeszcze większa jednostka do przechowywania wartości zmiennoprzecinkowych; typ ten może zmieścić więcej cyfr znaczących i większe wykładniki niż double.

```c
/* typy.c — wyświetla rozmiary typów */
#include <stdio.h>
int main(void)
{
    printf("Typ int ma rozmiar %d bajtów.\n",          sizeof(int));
    printf("Typ char ma rozmiar %d bajtów.\n",       sizeof(char));
    printf("Typ long ma rozmiar %d bajtów.\n",       sizeof(long));
    printf("Typ float ma rozmiar %d bajtów.\n",      sizeof(float));
    printf("Typ double ma rozmiar %d bajtów.\n", sizeof(double));
    return. 0;
}
```

# Łańcuchy znakowe

***Łańcuch znakowy*** jest ciągiem składającym się z jednego lub więcej znaków.

przykład łańcucha:

"Czułem sie jak pies na łańcuchu."

Znaki cudzysłowu nie są częścią łańcucha - informują one jedynie kompilator, ze to, co znajduje się pomiędzy nimi, jest łańcuchem.

Łańcuchy są przechowywane w tablicach zbudowanych z elementów typu char.

| C | z | u | ł | e | m |  | s | i | ę |  | j | a | k |  | p | i | e | s |  | n | a |  | ł | a | ń | c | u | c | h | u | . | \0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

na ostatniej pozycji w tablicy znajduje się znak **\0** - jest to tzw. ***znak zerowy*** (ang. *null character*)

Łańcuchy w języku C są zawsze przechowywane razem z tym znakiem - tablica musi mieć długość przynajmniej o jedną komórkę większą niż długość zapisanego w niej łańcucha.

*Deklarowanie łańcuchów*

**char** imię[40];

Nawiasy kwadratowe **[ ]** po nazwie imię sygnalizują, że deklarowana zmienna jest tablicą,

Liczba **40** w nawiasie określa ilość elementów tablicy, zaś słowo **char** określa typ elementów.

```
/* chwali.c — wykorzystuje łańcuchy */
#include <stdio.h>
#define POCHWALA "Ach, jakie wspaniale imię!"
int main(void)
{
  char imię[40];
  printf("Jak masz na imie?\n");
  scanf("%s", imię);
  printf("Witaj, %s. %s\n", imię, POCHWALA);
  return 0;
}
```

Przebieg działania programu chwali .c:

```
Jak masz na imię? Janek Scalak
Witaj, Janek. Ach,  jakie wspaniale imię!
```

*Łańcuchy a znaki*

Stała łańcuchowa **"x"** nie jest tym samym, co stała znakowa **' x'**.

**"x"** składa się z dwóch znaków, **' x'** oraz **' \0'** !

*Funkcje operujące na łańcuchach:*

```
/* chwal2.c */
#include <stdio.h>
#include <string.h>          // dostarcza prototypu dla strlen()
#define POCHWALA  "Ach, jakie wspaniale imię!"
int main(void)
{
   char imię[40];
   printf("Jak masz na imie?\n");
   scanf("%s", imię);
   printf("Witaj,  %s.  %s\n",  imię,  POCHWALA);
   printf("Twoje imię, składające sie z %d liter, zajmuje %d komórek pamięci.\n",  strlen(imię),  sizeof imię);
   printf(" POCHWALA składa sie z %d liter ",  strlen(POCHWALA));
   printf("i zajmuje %d komórek pamięci.\n",  sizeof(POCHWALA));
   return 0;
}
```

# Stałe i preprocesor C

*Składnia:*

```
#define   NAZWA   wartość
```

```
/* pizza.c — korzysta z definiowanych stałych w kontekście pizzy */

#include <stdio.h>
#define   PI   3.14159
int  main(void)
{
   float   powierzchnia,   obwód,   promień;  // deklaracja zmiennych
   printf("Ile wynosi promień Twojej pizzy?\n");
   scanf("%f",   &promien);
   powierzchnia  =  PI  *  promień  *  promień;
   obwód =2.0  *  PI  *  promień;
   printf("Podstawowe parametry Twojej pizzy to:\n");
   printf("obwód  =  %1.2f,  powierzchnia  =  %1.2f\n",   obwód,   powierzchnia);
   return 0;
}
```

# Modyfikator const

*Składnia:*

```
const int MIESIĄCE = 12;        // MIESIĄCE jest stalą symboliczna równą 12
```

Taka deklaracja sprawia, że MIESIĄCE staje się wartością przeznaczoną tylko do odczytu.

# Łączenie #define i #include

Załóżmy, że opracowujesz cały pakiet programów, z których każdy wykorzystuje ten sam zestaw stałych. Aby ułatwić sobie pracę możesz postąpić następująco:

► Zbierz wszystkie dyrektywy **#define** w jednym pliku; nazwij go np. ***const.h*** .

► U góry każdego pliku źródłowego programu dodaj następujący wiersz:

**#include "const.h"** // UWAGA: NIE <const.h> !

```
/* alias.h */
#define   program       int main(void)
#define   begin         {
#define   end           return 0;}
#define   następnie     ;
#define   pobierz       scanf
#define   powiedz       printf
#define   DWA           2
#define   razy          *
#define   całkowita      int
#include <stdio.h>
```

```
/* alias.c — obficie wykorzystuje preprocesor */
#include "alias.h"    // włączenie alias.h
program
begin
  całkowita twoja, moja        następnie
  powiedz("Podaj proszę jakaś liczbę całkowita.\n")
                                następnie
  pobierz("%d",  &twoja)        następnie
  moja  =  twoja razy DWA      następnie
  powiedz("%d jest dwa razy większa niz Twoja liczba!\n",
            moja)               następnie
end
```

# Funkcja printf()

Specyfikatory konwersji (formatu) i odpowiadające im dane wyjściowe.

| **Specyfikator konwersji** | **Dana wyjściowa** |
|---|---|
| **%c** | Pojedynczy znak |
| **%d** | Dziesiętna liczba całkowita (ze znakiem) |
| **%e** | Liczba zmiennoprzecinkowa w notacji z literą **e** |
| **%E** | liczba zmiennoprzecinkowa w notacji z literą **E** |
| **%f** | Liczba zmiennoprzecinkowa w zapisie dziesiętnym |
| **%g** | Równoważny **%e**, jeśli wykładnik jest mniejszy niż -4 lub większy bądź równy dokładności. W przeciwnym wypadku równoważny **%f**. |
| **%G** | Równoważny **%E**, jeśli wykładnik jest mniejszy niż-4 lub większy bądź równy dokładności. W przeciwnym wypadku równoważny **%f**. |
| **%i** | Dziesiętna liczba całkowita (ze znakiem) |
| **%o** | Ósemkowa liczba całkowita (bez znaku) |
| **%p** | Wskaźnik |
| **%s** | Łańcuch znakowy |
| **%u** | Dziesiętna liczba całkowita (bez znaku) |
| **%x** | Szesnastkowa liczba całkowita, cyfry szesnastkowe pisane małą literą (0f) |
| **%X** | Szesnastkowa liczba całkowita, cyfry szesnastkowe pisane wielką literą (OF) |
| **%%** | Wyświetla znak procentu |

*Modyfikatory funkcji printf ().*

| ***Modyfikato*** | ***Znaczenie*** |
|---|---|
| **.liczba** | Dokładność.<br>W przypadku konwersji typów **%e, %E** jest to liczba cyfr po kropce.<br>Dla konwersji **%g** i **%G** - maksymalna liczba cyfr znaczących.<br>Dla konwersji **%s** - maksymalna liczba wyświetlanych znaków.<br>Dla konwersji całkowitych - minimalna liczba wyświecanych cyfr, w razie potrzeby na początku liczby dodawane są zera.<br>Sam modyfikator **.** jest równoważny **.0**, a więc **%.f** to tyte, co **%.0f**<br><br>Przykład: **%5. 2f** wyświetla wartość **float** w polu o szerokości pięciu znaków z dwoma cyframi po przecinku. |
| **h** | W połączeniu ze specyfikatorem całkowitym oznacza wartość **short int** lub **unsigned short int**.<br><br>Przykłady: **%hu**, **%hx**, **%6. 4hd** |
| **l** | W połączeniu ze specyfikatorem całkowitym oznacza wartość **long int** lub **unsigned long int**.<br><br>Przykłady: **%ld**, **%8lu** |
| **L** | W połączeniu ze specyfikatorem zmiennoprzecinkowym oznacza wartość **long double**.<br><br>Przykłady: **%Lf**, **%10.4Le** |

| ***Modyfikato*** | ***Znaczenie*** |
|---|---|
| znacznik | Pięć dostępnych znaczników (**-, +, odstęp, #** i **0**) – tabela poniżej. Można stosować kilka znaczników na raz (lub też nie stosować żadnego). |
| liczba | Minimalna szerokość pola. Jeśli wyświetlana wartość lub łańcuch nie zmieści się w polu o podanej szerokości, zostanie użyte większe pole. |

*Znaczniki (ang.flags) funkcji printf ().*

| ***Znacznik*** | ***Znaczenie*** |
|---|---|
| **-** | Wyrównuje wartość do lewej krawędzi pola.<br>Przykład: **%-20s** |
| **+** | Wyświetla wartość ze znakiem plus lub minus, w zależności od tego, czy jest dodatnia czy ujemna.<br>Przykład: **%+6.2f** |
| **odstęp** | Wyświetla wartość z odstępem na początku (ale bez znaku plus), jeśli jest dodatnia, a ze znakiem minus, jeśli jest ujemna. |
| **#** | Wyświetla wartość w postaci alternatywnej, zależnej od specyfikatora.<br>Poprzedza wartości typu **%o** zerem, a wartości typu **%x** i **%X** - odpowiednio symbolem **0x** lub **0X**.<br>W przypadku wszystkich formatów zmiennoprzecinkowych, użycie znacznika **#** zapewnia obecność kropki dziesiętnej, nawet jeśli nie następują po niej żadne cyfry.<br>W przypadku formatów **%g** i **%G** znacznik zapobiega usunięciu końcowych zer.<br>Przykłady: **%#o**, **%#8.1f**, **%+#10.3E** |
| **0** | W przypadku wartości liczbowych, wypełnia pole początkowymi zerami zamiast odstępami.<br>Znacznik jest ignorowany, jeśli występuje razem ze znacznikiem **-** lub jeśli dla wartości całkowitej określona została dokładność.<br>Przykłady: **%010d**, **%08.3f** |

Przykłady:

```
/* szerok.c — szerokość pola */
#include <stdio.h>
#define STRONY 732
int main(void)
{
   printf("*%d*\n", STRONY);
   printf("*%2d*\n", STRONY);
   printf("*%10d*\n", STRONY);
   printf("*%-10d*\n", STRONY);
   return 0;
}
```

*Wynik:*

```
*732*
*732*
*       732*
*732        *
```

```c
/*
 float.c
- modyfikatory dla wartości zmiennoprzecinkowych
*/
#include <stdio.h>
int main(void)
{
  // stała  zdefiniowana  za  pomocą  const
  const double CZYNSZ = 2345.67;
  printf("*%f*\n", CZYNSZ);
  printf("*%e*\n", CZYNSZ);
  printf("*%4.2f*\n", CZYNSZ);
  printf("*%3.1f*\n, CZYNSZ);
  printf("*%10.3f*\n", CZYNSZ);
  printf("*%10.3e*\n", CZYNSZ);
  printf("*%+4.2f*\n", CZYNSZ);
  printf("*%010.2f*\n", CZYNSZ);
  return 0;
}
```

wynik:

```
*2345.670000*
*2.345670e+003*
*2345.67*
*2345.7*
* 2345.670*
*2.346e+003*
*+2345.67*
*0002345.67*
```

```c
/* łańcuchy.c — formatowanie łańcuchów  */
#include <stdio.h>
#define  NOTATKA   "Doskonała gra aktorów!"
int main(void)
{
    printf("*%2s*\n",   NOTATKA);
    printf("*%25s*\n",  NOTATKA);
    printf("*%25.5s*\n",  NOTATKA);
    printf("*%-25.5s*\n",  NOTATKA);
    return 0;
}
```

*dane wyjściowe programu:*

```
*Doskonała gra aktorów!*
*    Doskonała gra aktorów!*
*                         Dosko*
*Dosko                         *
```

## Funkcja scanf()

**dwie proste zasady:**

1. Jeśli używasz **scanf ()**, aby wczytać wartość do zmiennej należącej do jednego z omówionych wcześniej typów podstawowych, przed nazwą zmiennej dodaj **&**.
2. Jeśli używasz **scanf ()**, aby wczytać łańcuch do tablicy znaków, nie dodawaj przedrostka **&**.

```
/* wejscie.c — kiedy używać  & */
#include <stdio.h>
int main(void)
{
    int  wiek;                  // zmienna
    float  majątek;             // zmienna
    char  zwierzątko[30];       // łańcuch
    printf("Podaj swój wiek, majątek i ulubione zwierzątko.\n");
    scanf("%d  %f",  &wiek,  &majatek);   // tu używamy &
    scanf("%s",   zwierzątko);              // przy tablicy znakowej nie ma &
    printf("%d,  %.0f zl,  %s\n",   wiek,   majątek,   zwierzątko);
    return 0;
}
```

## Inne znaki w formacie funkcji scanf()

```
…
scanf("%d/%d", &a, &b);
…
```

oznacza, że funkcja spodziewa się danych w postaci liczba**/**liczba, np. 2/7 (a nie 2 7).

```
…
scanf("->%d ", &x);
…
```

oznacza, że funkcja spodziewa się liczby poprzedzonej łańcuchem "**->**".

Uwaga:

```
…
scanf("-> %d ", &x);  // odstep!
…
```

wymaga, aby w danych pomiędzy łańcuchem "->" i liczbą wystąpiła (co najmniej jedna) spacja.

*Specyfikatory konwersji dla funkcji scanf ()*

| ***Specyfikator*** | ***Znaczenie*** |
|---|---|
| **%c** | Interpretuje daną wejściową jako znak |
| **%d** | Interpretuje daną wejściową jako dziesiętną liczbę całkowitą ze znakiem. |
| **%e, %f, %g** | Interpretuje daną wejściową jako liczbę zmiennoprzecinkową. |
| **%E, %G** | Interpretuje daną wejściową jako liczbę zmiennoprzecinkową. |
| **%i** | Interpretuje daną wejściową jako dziesiętną liczbę całkowitą ze znakiem. |
| **%o** | Interpretuje daną wejściową jako ósemkową liczbę całkowitą ze znakiem. |
| **%p** | Interpretuje daną wejściową jako wskaźnik (adres). |
| **%s** | Interpretuje daną wejściową jako łańcuch, rozpoczynający się pierwszym znakiem drukowanym i kończący się ostatnim takim znakiem. |
| **%u** | Interpretuje daną wejściową jako dziesiętną liczbę całkowitą bez znaku. |
| **%x, %X** | Interpretuje daną wejściową jako szesnastkową liczbę całkowitą ze znakiem. |

Modyfikatory funkcji scanf ().

| ***Modyfikator*** | ***Znaczenie*** |
|---|---|
| ***** | Powstrzymuje odczytanie wartości.<br>Przykład: **%*d** |
| **liczba** | Maksymalna szerokość pola.<br>Odczytywanie danych kończy się, gdy osiągnięta zostanie maksymalna szerokość pola lub w momencie wystąpienia pierwszego znaku niedrukowanego.<br>Przykład: **%10s**. |
| **h** | **%hd** i **%hi** powodują zapisanie wartości w zmiennej typu **short int**.<br>**%ho**, **%hx** i **%hu** oznaczają typ **unsigned short int**.<br>Specyfikatory **d**, **i**, **o** i **x** bez modyfikatora h oznaczają typ **int**. |
| **l** lub **L** | **%ld** i **%li** powodują zapisanie wartości w zmiennej typu **long**.<br>**%lo**, **%lx** i **%lu** oznaczają typ **unsigned long**.<br>**%le**, **%lf** i **%lg** oznaczają typ **double**.<br>Dodanie **L** zamiast **l** do specyfikatorów **e**, **f** i **g** powoduje, że wartość zostanie zapisana w zmiennej typu **long double**.<br>Specyfikatory **e**, **f** i **g** bez modyfikatorów **l** lub **L** oznaczają typ **float**. |

## Modyfikator * w funkcjach printf() i scanf()

```
/*  zmszer.c
 - wykorzystuje pole o zmiennej  szerokości  */
#include <stdio.h>
int main(void)
{
   unsigned  szerokość,  dokładność;
   int liczba  = 256;
   double waga  =  242.5;
   printf("Jaka szerokość pola?\n");
   scanf("%d",   Szerokosc);
   printf("Liczba jest równa :%*d:\n",
                                        szerokość,   liczba);
   printf("Teraz podaj szerokość i dokładność:\n");
   scanf("%d %d",   &szerokosc,   &dokladnosc) ;
   printf("Waga  =  %*.*f\n",   szerokość,
                                    dokładność,   waga);
   return 0;
}
```

przykładowy przebieg działania programu:

```
Jaka szerokość pola?
6
Liczba jest równa :  256:
Teraz podaj szerokość i dokładność:
8  3
Waga = 242.500
```

W przypadku funkcji **scanf ()** modyfikator ***** powoduje pominięcie wartości odpowiadającej specyfikatorowi, przy którym został umieszczony.

Możliwość pominięcia niektórych pozycji jest przydatna na przykład wówczas, gdy należy odczytać określoną kolumnę z pliku tekstowego, w którym dane zapisane są w postaci wielu jednolitych kolumn.

```
/* pomin2.c — pomija pierwsze dwie z podanych liczb całkowitych */
#include <stdio.h>
int main(void)
{
    int   n;
    printf("Podaj proszę trzy liczby całkowite:\n");
    scanf("%*d  %*d  %d",    &n);
    printf("Ostatnia podana liczba była  %d\n",   n);
    return 0; .
}
```

przebieg działania programu:

```
Podaj proszę trzy liczby całkowite:
1976 1992 1996
Ostatnia podana liczba była  1996
```

□

## Zapis tekstowy do pliku i do bufora pamięci

```
printf("Liczby do analizy: %d, %f", x, y);        // wypisanie w oknie konsoli
```

---

```
FILE *fp;                                 // deklaracja wskaźnika do pliku
...
fp = fopen("plik.txt", "w");     // otwarcie pliku  plik.txt  do zapisu w trybie tekstowym (!!!)
...
fprintf(fp, "Liczby do analizy: %d, %f", x, y);         // zapis do pliku
...
fclose(fp);                               // zamknięcie pliku
```

---

```
char bufor[1024];                                    // deklaracja bufora
...
sprintf(bufor, "Liczby do analizy: %d, %f", x, y);    // zapis do bufora
```

## Odczyt tekstowy z pliku i z bufora pamięci

```
scanf("Liczby do analizy: %d, %f", &x, &y);   // odczyt z klawiatury
```

---

```
FILE *fp;                          // deklaracja wskaźnika do pliku
...
fp = fopen("plik.txt", "r");       // otwarcie pliku  plik.txt  do czytania w trybie tekstowym (!!!)
...
fscanf(fp, "Liczby do analizy: %d, %f", &x, &y);    // odczyt z pliku
...
fclose(fp);                        // zamknięcie pliku
```

---

```
char bufor [1024] = "Liczby do analizy: 2, 2.71828";    // deklaracja bufora
...
sscanf(bufor, "Liczby do analizy: %d, %f", &x, &y);      // odczyt z bufora
```

□