Literały

Literały w kodzie źródłowym programu określają literalnie, a więc wprost, pewną wartość znakową, liczbową lub tekstową. Literały dzielimy na:

a) Literały znakowe. Reprezentują jeden znak kodu ASCII. Zapisywany znak ujęty jest w apostrofy.

`'a'`	mała litera **a**
`'8'`	cyfra **8**
`'\n'`	znak sterujący: nowa linia (w kodzie ASCII dwa znaki CR i LF)
`'\f'`	znak sterujący: nowa strona
`'\b'`	znak sterujący: backspace
`'\0'`	NULL, znak o kodzie 0
`'\\'`	znak backslash
`'\''`	znak apostrof
`'\"'`	znak cudzysłów

b) Literały całkowite.

`13`	zapis dziesiętny, wartość 13
`013`	zapis ósemkowy, wartość 11
`0x13`	zapis szesnastkowy, wartość 19

c) Literały zmiennoprzecinkowe.

`13.7`	zapis normalny
`24.9e7`	notacja naukowa liczby 249000000
`7.1e-5`	notacja naukowa liczby 0.000071

d) Literały łańcuchowe. Ciąg znaków ujęty w cudzysłowy. Na końcu podanego ciągu znaków kompilator dodatkowo umieszcza znak NULL.

```
"Pierwsze spotkanie z jezykiem C"
```

Literał łańcuchowy może zawierać znaki sterujące.

```
"Ten string\nzostanie wypisany\nw trzech liniach"
```

```
Ten string
zostanie wypisany
w trzech liniach
```

Stałe i zmienne

Stałe – posiadają stałą wartość, np. –5.122

Stałe można definiować na dwa sposoby:

- za pomocą dyrektywy preprocesora **#define**, np.**:** **`#define pi 3.14159`** (typ stałej wynika ze sposobu zapisu tej stałej)
- z użyciem słowa kluczowego **const**, np. **`const double pi=3.14159265`** (typ stałej wynika z podanej po słowie **const** nazwy typu)

Stała może mieć nadaną wartość tylko raz, nie może ona ulec zmianie w trakcie wykonania programu.

Przykładowe stałe:

100
-257
0
} stałe typu całkowitego (stosowany jest zapis stałopozycyjny)

2.572
-1.
7.1E5
} stałe typu rzeczywistego (stosowany jest zapis zmiennopozycyjny)

Zmienne – mogą przyjmować wartości z określonego zbioru, wynikającego z jej typu; każda zmienna posiada nazwę (identyfikator); w trakcie realizacji programu zmienna może przyjmować różne wartości.

Przykładowe zmienne: **`alfa, beta, j23`**

Stałe i zmienne (a także inne obiekty w języku C) muszą posiadać **identyfikatory** (nazwy), które umożliwiają dostęp do tych obiektów. Identyfikatory te są określane przez użytkownika w deklaracjach (definicjach), jednocześnie z określeniem **typu**.

Identyfikator jest ciągiem liter i cyfr, przy czym pierwszym znakiem musi być litera. Rozróżniane są wielkie i małe litery alfabetu.

Np.: **`alfa1, Alfa1, j23`**

Typy stałych i zmiennych

Każdy obiekt programu musi mieć zdefiniowany **typ** (czyli dopuszczalne wartości obiektu i możliwe do wykonania na nim operacje).

Typy zmiennych i stałych są podawane w **deklaracjach lub definicjach**, które w pliku źródłowym muszą występować przed użyciem tych zmiennych i stałych w instrukcjach.

Typ danej dokładnie charakteryzuje zbiór wartości, do którego należy stała, lub które może przyjmować zmienna.

Typ określa też wewnętrzną reprezentację danej w pamięci komputera (zapis stałopozycyjny lub zmiennopozycyjny).

Zawarta w deklaracji (definicji) obiektu informacja o **typie** jest wykorzystywana przez kompilator, który rezerwuje odpowiednią wielkość pamięci dla danej zmiennej, stosownie do zakresu wartości, które może ona przyjmować i wyznacza odpowiednią maszynową reprezentację tej zmiennej (w postaci binarnej). Przykłady:

`int alfa, beta;` (zmienne typu całkowitego – zapis stałopozycyjny)

`float j23;` (zmienna typu rzeczywistego – zapis zmiennopozycyjny)

Informacja o typach danych pozwala także kompilatorowi na sprawdzenie poprawności wielu różnych konstrukcji użytych w zapisie algorytmu w języku programowania i na wykrycie wielu błędów w programie już na etapie kompilacji. Np. wykryta może być próba przypisania wartości logicznej zmiennej typu rzeczywistego. Stąd też określanie typów dla wszystkich danych uważane jest za zaletę języków programowania wysokiego poziomu.

Typy danych języka C

Typy proste (arytmetyczne). Arytmetyczne typy, w które wyposażony jest język:

char,

int,

float,

double.

Typy pochodne. Tworzone przy pomocy operatorów, np.:

[ ] tablica obiektów

***** wskaźnik na obiekt

Typy arytmetyczne

char	(1 bajt)	małe liczby całkowite, znaki ASCII.
int	(2 bajty)	liczby całkowite.
long int	(4 bajty)	długie liczby całkowite.
float	(4 bajty)	liczby rzeczywiste.
double	(8 bajtów)	liczby rzeczywiste zwiększonej precyzji.
long double	(10 bajtów)	liczby rzeczywiste zwiększonej precyzji.

Typ całkowity może być dodatkowo poprzedzony modyfikatorem **signed** lub **unsigned**.

Zakresy wartości danych różnych typów arytmetycznych:

char	**-128 ... 127**
signed char	**-128 ... 127**
unsigned char	**0 ... 255**
int	**-32768 ... 32767**
signed int	**-32768 ... 32767**
unsigned	**0 ... 65535**
unsigned int	**0 ... 65535**
long	**-2147483648 ... 2147483647**
long int	**-2147483648 ... 2147483647**
signed long int	**-2147483648 ... 2147483647**
unsigned long	**0 ... 4294967295**
unsigned long int	**0 ... 4294967295**
float	**+/- ($3.4*10^{-38}$ ... $3.4*10^{38}$)**
double	**+/- ($1.7*10^{-308}$ ... $1.7*10^{308}$)**
long double	**+/- ($1.7*10^{-4932}$ ... $1.7*10^{4932}$)**

Typy pochodne *zostaną omówione w dalszych częściach kursu.*

Elementy pliku źródłowego w języku C

Plik źródłowy ogólnie składa się z:

dyrektyw preprocesora,

deklaracji (definicji),

instrukcji,

komentarzy.

1. **Dyrektywy preprocesora.** Pierwszym różnym od spacji znakiem linii dyrektywy preprocesora jest znak #. Dyrektywy te służą wstępnej modyfikacji programu źródłowego przed jego kompilacją. Dyrektywa

   `#include <plik_nagłówkowy>`

   włącza wskazany plik w miejsce jej wystąpienia. Plik poszukiwany jest w standardowym katalogu włączanych plików.

Zawartość pliku kot.c:	Kolejność w kompilacji:
linia1;	linia1;
linia2;	linia2;
#include <pies.c>	linia5;
linia3;	linia8;
linia4;	linia9;
	linia6;
Zawartość pliku pies.c	linia7;
linia5;	linia3;
#include <owca.h>	linia4;
linia6;	
linia7;	
Zawartość pliku owca.h:	
linia8;	
linia9;	

2. **Deklaracje (definicje)**

Każdy obiekt (zmienna) występujący w programie musi być zdefiniowany. Definicja jest równocześnie deklaracją obiektu. Program może (nie musi) zawierać dodatkowe deklaracje obiektu).

Uwaga: Obiekt w programie może być wielokrotnie deklarowany oraz musi być tylko jednokrotnie zdefiniowany. Na końcu deklaracji i definicji obiektu występuje średnik.

Deklaracja obiektu. Ustala jedynie typ obiektu. Nie przydziela pamięci.

```
extern int x;  // Obiekt x typu całkowitego już gdzieś
               // istnieje poza plikiem źródłowym
```

Definicja obiektu. Ustala identyfikator i typ obiektu (jak przy deklaracji), oraz dodatkowo przydziela dla niego pamięć (a więc fizycznie tworzy obiekt).

```
int a, b;          //ustalenie typu całkowitego dla dwóch
                   //obiektów a, b oraz przydzielenie im
                   //pamięci
```

3. **Instrukcje** programu przetwarzają dane. Dostęp w instrukcjach do obiektów danych jest realizowany przez:

   nazwę obiektu (nazwy stałych i zmiennych),

   wskaźnik na obiekt (wskaźniki na stałe i zmienne),

   literał (bezpośrednie, dosłowne podanie wartości).

Moduł źródłowy zawiera **instrukcje** pogrupowane w bloki funkcyjne. W bloku funkcyjnym instrukcje wykonywane są sekwencyjnie w kolejności ich zapisania w module źródłowym.

4. **Komentarz.** Jest to dowolny fragment tekstu zawarty pomiędzy parami znaków `/*` oraz `*/` lub występujący po parze znaków `//` i rozciągający się do końca danej linii. Komentarze nie są analizowane przez kompilator. Główną funkcją komentarzy jest wyjaśnianie i uzupełnianie tekstu programu uwagami pomocnymi przy poźniejszej modyfikacji tego programu. Komentarz powinien być sensowny, zwarty i aktualny.

```
#include <stdio.h>

int main()

{

   int x/*To jest komentarz*/=17,y=19;

   printf("%d %d\n",x,y);      //To też jest komentarz

}
```

Inicjalizacja obiektów

Przy definiowaniu obiektu można nadać mu wartość początkową. Inicjalizator ma formę wyrażenia. Ogólna postać definicji wraz z inicjalizacją jest następująca:

typ deklarator = inicjalizator;

```
     int a=3,b=7,c=a+b;        // c=10

     int x=2*a+b,y=c;          // x=13, y=10
```

Stałe. Obiekt zadeklarowany z modyfikatorem **const** (stała) ma tę własność, że wartość tego obiektu **nie może być później modyfikowana**. Wartość obiektu stałego można ustalić **jedynie w momencie definiowania stosując inicjalizację**.

```
     const int a=18;

     const float x=3.7;

     a++;                 // Błąd, stałej nie można modyfikować

     x=14.65;             // Błąd, stałej nie można modyfikować
```

Wejście - wyjście

Standardowe wprowadzanie i wyprowadzanie danych realizowane jest w języku C poprzez strumienie. **Strumień** jest ciągiem bajtów skojarzonym z pewnym urządzeniem zewnętrznym komputera (np. klawiaturą, monitorem, plikiem dyskowym). Dla każdego strumienia określona jest jego **bieżąca pozycja** wskazująca, który znak tego strumienia będzie w najbliższej kolejności przetwarzany.

W bibliotece kompilatora są zdefiniowane **strumienie standardowe** obsługiwane przez ustalone funkcje biblioteczne. Tradycyjnie wykorzystywane strumienie standardowe to:

`stdin` - standardowe wejście (zwykle klawiatura)

`stdout` - standardowe wyjście (zwykle monitor)

`stderr` - standardowe wyjście dla komunikatów o błędach (zwykle monitor)

Definicje strumieni *stdin, stdout, stderr* umieszczone są w pliku nagłówkowym **`stdio.h`**.

Funkcje biblioteczne obsługujące strumienie standardowe

`getchar()` definicja w pliku `stdio.h`

Funkcja wprowadza kolejny bajt ze strumienia **`stdin`** i zwraca jako wynik typu int kod ASCII wprowadzonego znaku (od 0 do 255) lub wartość **EOF** (zdefiniowaną zwykle jako -1) w przypadku rozpoznania końca strumienia. Wprowadzane z klawiatury znaki **są kopiowane na ekranie** (tzw. echo) i kolejno umieszczane w buforze aż do naciśnięcia **Enter**. Dopiero potem funkcja **`getchar`** pobiera kolejny znak z bufora. Dodatkowo następuje zamiana znaków **CR** i **LF** (w języku C znak **`'\n'`**) generowanych przez klawisz Enter na znak **LF**.

Zadanie: uruchom następujący program:

```
#include <stdio.h>
int main()
{
   int x;
   x=getchar();
   printf("%d\n",x);
   system(“PAUSE”);
   return 0;
}
```

```
A                    KLAWIATURA A↵

65
```

getch() **definicja w pliku conio.h**

Funkcja ta podobnie jak getchar wprowadza kolejny bajt ze strumienia **stdio** i zwraca jako wynik typu int kod ASCII naciśniętego znaku lub **EOF**. Znaki wprowadzane z klawiatury **nie są kopiowane na ekranie** (brak echa). Funkcja zwraca kod ASCII znaku natychmiast po jego naciśnięciu (bez umieszczania tego znaku w buforze).

Zadanie: uruchom następujący program:

```
#include <stdio.h>
#include <conio.h>
main()
{
   int x;
   x=getch();
   printf("%d\n",x);
   system(“PAUSE”);
   return 0;
}
```

```
                    KLAWIATURA A

65
```

`putchar(`*znak*`)`** **definicja w pliku `stdio.h`**

Funkcja wyprowadza podany znak do standardowego strumienia **`stdout`** i zwraca jako wynik typu int kod ASCII wyprowadzanego znaku.

Zadanie: uruchom następujący program:

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int c;
   c=getch();
   putchar(c+1);
   getch();
   return 0;
}
```

KLAWIATURA a
b

`scanf(`*format wejściowy, lista wejściowa*`)`** **definicja w pliku `stdio.h`**

Funkcja wprowadza kolejne znaki ze standardowego strumienia **`stdin`**, poddaje je interpretacji zgodnie z *formatem wejściowym* i uzyskane wartości umieszcza w obszarach pamięci wskazanych na *liście wejściowej*. Format wejściowy jest łańcuchem znakowym, który może zawierać:

odstępy (spacje, znaki tabulacji, nowe linie),

specyfikatory formatu wejściowego,

zwykłe znaki (różne od odstępów i znaku %).

Zgodnie z podanym formatem znaki strumienia **`stdin`** tworzą kolejne *pola wejściowe.* Pole wejściowe stanowi:

- ciąg znaków do następnego odstępu (bez wliczania go),
- ciąg znaków, którego zakończenie spowodowane zostało wystąpieniem w strumieniu znaku niedopuszczalnego przez typ konwersji podany w specyfikatorze formatu,
- ciąg znaków o długości równej szerokości pola podanej w specyfikatorze formatu

Składniki formatu wejściowego sterują przetwarzaniem pól wejściowych:

- Odstęp powoduje pominięcie na wejściu sekwencji sąsiadujących znaków odstępu.
- Znak % rozpoczyna specyfikator formatu określający sposób przekształcenia odpowiadającego mu pola wejściowego na wartość, która zostaje umieszczona w obszarze pamięci, którego adres podaje kolejny argument funkcji **`scanf`**. **Lista wejściowa** tej funkcji zawiera oddzielone przecinkami **wskaźniki** na zmienne odpowiednich typów.
- Zwykły znak (różny od odstępu i nie należący do specyfikatora) powoduje, że taki sam znak jest odczytywany ze strumienia **`stdin`**, ale nie jest umieszczany w pamięci. Jeżeli znak taki nie występuje jako kolejny znak strumienia, to funkcja scanf kończy pracę. Dwa znaki %% obok siebie reprezentują jeden znak % traktowany jako zwykły znak.

Każdy *specyfikator formatu wejściowego* jest ciągiem rozpoczynającym się od znaku % i ma postać (nawiasy [ ] oznaczają opcje):

% **`[*] [szerokość] [modyfikator] typ`**

Gwiazdka ***** po znaku % w specyfikatorze formatu powoduje, że odpowiednie pole wejściowe jest analizowane w normalny sposób zgodnie z tym specyfikatorem jednak bez zapisywania jego wartości w pamięci.

Jedynym wymaganym elementem specyfikatora formatu jest **znak** określający typ wprowadzanych danych:

Dane numeryczne

Znak	Zawartość pola wejściowego	Typ argumentu funkcji **scanf**
d	Liczba całkowita dziesiętna	**int*,** wskaźnik na int
D	Liczba całkowita dziesiętna	**long*,** wskaźnik na long
u	Liczba całkowita dziesiętna bez znaku	**unsigned int*,** wskaźnik na unsigned int
U	Liczba całkowita dziesiętna bez znaku	**unsigned long*,** wskaźnik na unsigned long
o	Liczba całkowita ósemkowa	**int*,** wskaźnik na int
O	Liczba całkowita ósemkowa	**long*,** wskaźnik na long
x	Liczba całkowita szesnastkowa	**int*,** wskaźnik na int
X	Liczba całkowita szesnastkowa	**long*,** wskaźnik na long
i	Liczba całkowita dziesiętna, ósemkowa lub szesnastkowa	**int*,** wskaźnik na int
I	Liczba całkowita dziesiętna, ósemkowa lub szesnastkowa	**long*,** wskaźnik na long
e,E	Liczba rzeczywista	**float*,** wskaźnik na float
f	Liczba rzeczywista	**float*,** wskaźnik na float
g,G	Liczba rzeczywista	**float*,** wskaźnik na float

Dane znakowe

Znak	Zawartość pola wejściowego	Typ argumentu funkcji **`scanf`**
c	Znak ASCII	**`char*,`** wskaźnik na char
s	Ciąg znaków ASCII	**`char[],`** wskaźnik na tablicę char

Wskaźniki

Znak	Zawartość pola wejściowego	Typ argumentu funkcji **`scanf`**
p	Szesnastkowy adres (**offset** lub **segment:offset**)	Wskaźnik

Przed znakiem określającym *typ* można umieścić **modyfikator** zmieniający działanie specyfikatora:

- **`l`** - wprowadza liczby całkowite w ich długich wersjach **`long`**, a liczby rzeczywiste jako **`double`**,
- **`L`** - wprowadza liczby rzeczywiste jako **`long double`**.

Maksymalną szerokość każdego pola wejściowego można określić podając w specyfikatorze formatu liczbę całkowitą **`szerokość`**.

Kolejne pole wejściowe dla specyfikatora z typem **`c`** rozpoczyna się w bieżącej pozycji strumienia, a dla specyfikatorów z innymi typami początek pola ustala się pomijając dodatkowo w strumieniu wejściowym sąsiadujące ze sobą odstępy.

Zakończenie przetwarzania danego pola wejściowego i przejście do kolejnego pola następuje w przypadku, gdy:

- kolejnym znakiem strumienia jest odstęp (co kończy pole wejściowe),
- przetworzonych zostało tyle znaków strumienia, ile wynosi podana **szerokość**,
- znak w bieżącej pozycji strumienia nie może zostać przetworzony zgodnie z podanym specyfikatorem formatu,

Znak, który spowodował zakończenie przetwarzania danego pola wejściowego pozostawiany jest w strumieniu i będzie przetwarzany podczas analizowania następnego pola wejściowego lub podczas wywołania innej funkcji wprowadzania danych.

Zakończenie działania funkcji **scanf** następuje, gdy:

- napotkano koniec strumienia **EOF**,
- wyczerpany został format wejściowy,
- w bieżącej pozycji strumienia nie występuje wymagany przez format wejściowy zwykły znak (nie należący do specyfikatora).

Funkcja **scanf** zwraca jako wynik swojego działania liczbę przetworzonych pól wejściowych.

Zadanie: uruchom następujący program:

```
#include <stdio.h>
int main()
{
   char k;
   scanf("%c",&k);
   printf("Wczytano znak o kodzie = %d\n",k);
   scanf("%c",&k);
   printf("Wczytano znak o kodzie = %d\n",k);
   system("PAUSE");
   return 0;
}
```

```
A                          KLAWIATURA A↵
Wczytano znak o kodzie = 65
Wczytano znak o kodzie = 10
```

Zadanie: dokonaj analizy następującego programu:

```
#include <stdio.h>
int main()
{
      long int k,i;
      k=999;
      printf("\nPrzed wczytaniem k = %d\n",k);
      scanf("kot%d",&k);
      printf("    Po wczytaniu k = %d\n",k);
      system("PAUSE");
      return 0;
}
```

```
Przed wczytaniem k = 999
kot47                          KLAWIATURA kot47↵
    Po wczytaniu k = 47
```

```
Przed wczytaniem k = 999
kot  45kot  17                 KLAWIATURA kot  45kot  17↵
    Po wczytaniu k = 45
```

```
Przed wczytaniem k = 999
kokot 33 kot 88                KLAWIATURA kokot 33 kot 88↵
    Po wczytaniu k = 999
```

`printf`*(format wyjściowy, lista wyjściowa)* **definicja w pliku `stdio.h`**

Funkcja wyprowadza do standardowego strumienia **`stdout` wartości wyrażeń** będących kolejnymi argumentami funkcji umieszczonymi na *liście wyjściowej*. Format wyjściowy steruje przekształcaniem wartości wyrażeń do wyprowadzanej postaci znakowej. Do strumienia **`stdout`** przekazywane są również wszystkie znaki formatu wyjściowego, które nie należą do **specyfikatorów formatu**. Liczba wyprowadzanych argumentów musi być zgodna z liczbą specyfikatorów umieszczonych w formacie wyjściowym.

Każdy *specyfikator formatu wyjściowego* jest ciągiem rozpoczynającym się od znaku % i ma postać (nawiasy [ ] oznaczają opcje):

% **`[znacznik] [szerokość] [.precyzja] [modyfikator] typ`**

Parametr **`szerokość`** określa minimalną liczbę znaków wyprowadzanych przy udziale danego specyfikatora formatu:

- W przypadku konieczności wyprowadzenia mniejszej liczby znaków niż podana **`szerokość`**, wyprowadzane pole **uzupełniane jest z lewej strony spacjami** do podanego rozmiaru.
- W przypadku wyprowadzania liczby, której zapis wymaga większej liczby znaków niż podana **`szerokość`**, **wyprowadzane są wszystkie niezbędne znaki** bez względu na wartość parametru **`szerokość`**.

Parametr **`precyzja`** ustala:

- Dla specyfikatorów typu **f, e, E** liczbę cyfr części ułamkowej. Ostatnia wyprowadzana cyfra jest zaokrąglana.
- Dla specyfikatorów typu **g, G** maksymalną liczbę cyfr znaczących.
- Dla specyfikatorów typu **d, i, u, o, x, X** minimalną liczbę wyprowadzanych cyfr z ewentualnym uzupełnieniem zerami z lewej strony.

Parametry specyfikatora formatu **`szerokość`** oraz **`precyzja`** mogą mieć postać liczby całkowitej lub znaku gwiazdki **`*`.** W przypadku gwiazdki

wartością odpowiedniego parametru jest wartość elementu *listy wyjściowej* poprzedzającego właściwy wyprowadzany element.

Jedynym wymaganym elementem specyfikatora formatu jest **znak** określający typ wprowadzanych danych:

Dane numeryczne

Znak	Element listy wyjściowej	Postać danych wyjściowych
d	Wartość całkowita	Dziesiętna liczba całkowita
i	Wartość całkowita	Dziesiętna liczba całkowita
u	Wartość całkowita	Dziesiętna liczba całkowita bez znaku
o	Wartość całkowita	Ósemkowa liczba całkowita bez znaku
x	Wartość całkowita	Szesnastkowa liczba całkowita bez znaku; cyfry szesnastkowe zawierają: a, b, c, d, e, f
X	Wartość całkowita	Szesnastkowa liczba całkowita bez znaku; cyfry szesnastkowe zawierają: A, B, C, D, E, F
f	Wartość rzeczywista	Dziesiętna liczba rzeczywista w zapisie normalnym
e,E	Wartość rzeczywista	Dziesiętna liczba rzeczywista w notacji naukowej, odpowiednio z literą **e** lub **E**
g,G	Wartość rzeczywista	Dziesiętna liczba rzeczywista w zapisie normalnym lub w notacji naukowej z literą **e** dla g lub **E** dla G

Dane znakowe

Znak	Element listy wyjściowej	Postać danych wyjściowych
c	Wartość całkowita, kod znaku ASCII	Jeden znak ASCII
s	Wskaźnik na łańcuch znakowy	Ciąg znaków o długości **`szerokość`** lub do bajtu NULL

Wskaźniki

Znak	Element listy wyjściowej	Postać danych wyjściowych
p	Wskaźnik	Szesnastkowy adres (**offset** lub **segment:offset**)

Element formatu wyjściowego postaci %% powoduje wyprowadzenie jednego znaku %.

Przed znakiem określającym *typ* można umieścić **modyfikator** zmieniający działanie specyfikatora:

- **`l`** - wyprowadza liczby całkowite w ich długich wersjach **`long`**; **uwaga:** do wyprowadzenia liczb rzeczywistych **`double`** nie zachodzi konieczność stosowania modyfikatora,
- **`L`** - wyprowadza liczby rzeczywiste typu **`long double`**.

Parametr **`znacznik`** dodatkowo precyzuje sposób wyprowadzania danych i przyjmuje wartości:

- Wyrównanie wyprowadzanych w danym polu znaków do jego lewej krawędzi.

+ Wyprowadzenie dodatkowo znaku + dla liczby dadatniej (standardowo wyprowadzany jest jedynie znak - dla liczb ujemnych).

Wyprowadzenie liczby ósemkowej z zerem z lewej strony liczby oraz liczby szesnastkowej z sekwencją 0x lub 0X z lewej strony tej liczby.

Funkcja **`printf`** zwraca jako wynik liczbę wyprowadzonych znaków do standardowego strumienia **`stdout`**.

Zadanie: uruchom następujący program:

```
#include <stdio.h>

int main()

{

   int k;

   printf("Podaj liczbe calkowita: ");

   scanf("%d",&k);

   printf("\n  Notacja dziesietna = %d",k);

   printf("\n    Notacja osemkowa = %o",k);

   printf("\nNotacja szesnastkowa = %x",k);

   printf("\nNotacja szesnastkowa = %X\n",k);

   system("PAUSE");

   return 0;

}
```

Zadanie dodatkowe: zastąp funkcję **system** funkcją **getch**

Wyrażenia

Wyrażenie jest ciągiem operatorów i argumentów (np. stałych lub zmiennych), który określa sekwencję obliczeń prowadzącą do uzyskania wartości wynikowej. W wyrażeniu mogą występować pary nawiasów ( ) wskazujące kolejność wykonywania działań przy obliczaniu wartości tego wyrażenia.

Stałe wyrażenie jest wyrażeniem, którego wartość jest znana na etapie kompilacji.

Wartość wyrażenia wyliczana jest poprzez wykonanie operacji określonych przez występujące w wyrażeniu operatory.

Operatory arytmetyczne:

+ dodawanie i jednoargumentowy plus

- odejmowanie i jednoargumentowa zmiana znaku

***** mnożenie

/ dzielenie (dla typów całkowitych obcięcie części ułamkowej)

% modulo

Zadanie: uruchom następujący program:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    float x,y;
    printf("podaj dwie liczby rzeczywiste x,y:\n?");
    scanf("%f",&x);
    putch('?');
    scanf("%f",&y);
    printf("x+y=%.2f, x-y=%.3f, x*y=%.4f, x/y=%.6f",
            x+y,x-y,x*y,x/y);
    getch();
    return 0;
}
```

Zadanie C02: uruchom następujący program:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int x,y;
    printf("podaj dwie liczby calkowite x,y:\n?");
    scanf("%d",&x);
    putch('?');
    scanf("%d",&y);
    printf("x+y=%d, x-y=%d, x*y=%d, x/y=%d, x%%y=%d",
            x+y, x-y, x*y, x/y, x%y);
    getch();
    return 0;
}
```

Prześlij plik w formacie .txt z zadaniem C02 przez platformę Moodle.