LABORATORIUM PODSTAW PROGRAMOWANIA

LAB 1 PODSTAWOWE KONSTRUKCJE

Fukcje biblioteczne

printf

deklaracja

```
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...)

#include <stdarg.h>

int vprintf(const char *format, va_list ap);
int vfprintf(FILE *stream, const char *format, va_list ap);
int vsprintf(char *str, const char *format, va_list ap);
int vsnprintf(char *str, size_t size, const char *format, va_list ap);
```

Opis

Funkcje formatują tekst zgodnie z podanym formatem opisanym poniżej. Funkcje printf i vprintf wypisują tekst na standardowe wyjście (tj. do stdout); fprintf i vfprintf do strumienia podanego jako argument; a sprintf, vsprintf, snprintf i vsnprintf zapisują go w podanej jako argument tablicy znaków.

Funkcje vprintf, vfprintf, vsprintf i vsnprintf różnią się od odpowiadających im funkcjom printf, fprintf, sprintf i snprintf tym, że zamiast zmiennej liczby argumentów przyjmują argument typu va_list.

Funkcje snprintf i vsnprintf różnią się od sprintf i vsprintf tym, że nie zapisuje do tablicy nie więcej niż `size` znaków (wliczając kończący znak '\0'). Oznacza to, że można je używać bez obawy o wystąpienie przepełnienia bufora.

Argumenty

format
: format, w jakim zostaną wypisane następne argumenty

stream
: strumień wyjściowy, do którego mają być zapisane dane

str
: tablica znaków, do której ma być zapisany sformatowany tekst

size
: rozmiar tablicy znaków

ap
: wskaźnik na pierwszy argument z listy zmiennej liczby argumentów

Format

Format składa się ze zwykłych znaków (innych niż znak '%'), które są kopiowane bez zmian na wyjście oraz sekwencji sterujących, zaczynających się od symbolu procenta, po którym następuje:

- dowolna liczba flag,
- opcjonalne określenie minimalnej szerokości pola,
- opcjonalne określenie precyzji,
- opcjonalne określenie rozmiaru argumentu,
- określenie formatu.

Jeżeli po znaku procenta występuje od razu drugi procent to cała sekwencja traktowana jest jak zwykły znak procenta (tzn. jest on wypisywany na wyjście).

Flagi

W sekwencji możliwe są następujące flagi:

- **-** (minus) oznacza, że pole ma być wyrównane do lewej, a nie do prawej.
- **+** (plus) oznacza, że dane liczbowe zawsze poprzedzone są znakiem (plusem dla liczb nieujemnych lub minusem dla ujemnych).
- spacja oznacza, że liczby nieujemne poprzedzone są dodatkową spacją; jeżeli flaga plus i spacja są użyte jednocześnie to spacja jest ignorowana.
- **#** (*hash*) powoduje, że wynik jest przedstawiony w *alternatywnej postaci*:
    - dla formatu **o** powoduje to zwiększenie precyzji, jeżeli jest to konieczne, aby na początku wyniku było zero;
    - dla formatów **x** i **X** niezerowa liczba poprzedzona jest ciągiem **0x** lub **0X**;

    - dla formatów **a**, **A**, **e**, **E**, **f**, **F**, **g** i **G** wynik zawsze zawiera kropkę nawet jeżeli nie ma za nią żadnych cyfr;
    - dla formatów **g** i **G** końcowe zera nie są usuwane.
- **0** (zero) dla formatów **d**, **i**, **o**, **u**, **x**, **X**, **a**, **A**, **e**, **E**, **f**, **F**, **g** i **G** do wyrównania pola wykorzystywane są zera zamiast spacji za wyjątkiem wypisywania wartości nieskończoność i NaN. Jeżeli obie flagi 0 i - są obecne to flaga zero jest ignorowana. Dla formatów **d**, **i**, **o**, **u**, **x** i **X** jeżeli określona jest precyzja flaga ta jest ignorowana.

Szerokość pola i precyzja

Minimalna szerokość pola oznacza ile najmniej znaków ma zająć dane pole. Jeżeli wartość po formatowaniu zajmuje mniej miejsca jest ona wyrównywana spacjami z lewej strony (chyba, że podano flagi, które modyfikują to zachowanie). Domyślna wartość tego pola to 0.

Precyzja dla formatów:

- **d**, **i**, **o**, **u**, **x** i **X** określa minimalną liczbę cyfr, które mają być wyświetlone i ma domyślną wartość 1;
- **a**, **A**, **e**, **E**, **f** i **F** - liczbę cyfr, które mają być wyświetlone po kropce i ma domyślną wartość 6;
- **g** i **G** określa liczbę cyfr znaczących i ma domyślną wartość 1;
- dla formatu **s** - maksymalną liczbę znaków, które mają być wypisane.

Szerokość pola może być albo dodatnią liczbą zaczynającą się od cyfry różnej od zera albo gwiazdką. Podobnie precyzja z tą różnicą, że jest jeszcze poprzedzona kropką. Gwiazdka oznacza, że brany jest kolejny z argumentów, który musi być typu int. Wartość ujemna przy określeniu szerokości jest traktowana tak jakby podano flagę **-** (minus).

Rozmiar argumentu

Dla formatów **d** i **i** można użyć jednego ze modyfikator rozmiaru:

- **hh** - oznacza, że format odnosi się do argumentu typu signed char,
- **h** - oznacza, że format odnosi się do argumentu typu short,
- **l** (el) - oznacza, że format odnosi się do argumentu typu long,
- **ll** (el el) - oznacza, że format odnosi się do argumentu typu long long,
- **j** - oznacza, że format odnosi się do argumentu typu intmax_t,
- **z** - oznacza, że że format odnosi się do argumentu typu będącego odpowiednikiem typu size_t ze znakiem,
- **t** - oznacza, że że format odnosi się do argumentu typu ptrdiff_t.

Dla formatów **o**, **u**, **x** i **X** można użyć takich samych modyfikatorów rozmiaru jak dla formatu **d** i oznaczają one, że format odnosi się do argumentu odpowiedniego typu bez znaku.

Dla formatu **n** można użyć takich samych modyfikatorów rozmiaru jak dla formatu **d** i oznaczają one, że format odnosi się do argumentu będącego wskaźnikiem na dany typ.

Dla formatów **a**, **A**, **e**, **E**, **f**, **F**, **g** i **G** można użyć modyfikatorów rozmiaru **L**, który oznacza, że format odnosi się do argumentu typu long double.

Dodatkowo, modyfikator **l** (el) dla formatu **c** oznacza, że odnosi się on do argumentu typu wint_t, a dla formatu **s**, że odnosi się on do argumenty typu wskaźnik na wchar_t.

Format

Funkcje z rodziny printf obsługują następujące formaty:

- **d**, **i** - argument typu int jest przedstawiany jako liczba całkowita ze znakiem w postaci **[-]ddd**.
- **o**, **u**, **x**, **X** - argument typu unsigned int jest przedstawiany jako nieujemna liczba całkowita zapisana w systemie oktalnym (**o**), dziesiętnym (**u**) lub heksadecymalnym (**x** i **X**).
- **f**, **F** - argument typu double jest przedstawiany w postaci **[-]ddd.ddd**.
- **e**, **E** - argument typu double jest reprezentowany w postaci **[i]d.ddde+dd**, gdzie liczba przed kropką dziesiętną jest różna od zera, jeżeli liczba jest różna od zera, a + oznacza znak wykładnika. Format **E** używa wielkiej litery E zamiast małej.
- **g**, **G** - argument typu double jest reprezentowany w formacie takim jak **f** lub **e** (odpowiednio **F** lub **E**) zależnie od liczby znaczących cyfr w liczbie oraz określonej precyzji.
- **a**, **A** - argument typu double przedstawiany jest w formacie **[-]0xh.hhhp+d** czyli analogicznie jak dla **e** i **E**, tyle że liczba zapisana jest w systemie heksadecymalnym.
- **c** - argument typu int jest konwertowany do unsigned char i wynikowy znak jest wypisywany. Jeżeli podano modyfikator rozmiaru **l** argument typu wint_t konwertowany jest do wielobajtowej sekwencji i wypisywany.

- **s** - argument powinien być typu wskaźnik na char (lub wchar_t). Wszystkie znaki z podanej tablicy, kończące się na null, są wypisywane.
- **p** - argument powinien być typu wskaźnik na void. Jest on konwertowany na serię drukowalnych znaków w sposób zależny od implementacji.
- **n** - argument powinien być wskaźnikiem na liczbę całkowitą ze znakiem, do którego zwracana jest liczba zapisanych znaków.

W przypadku formatów **f**, **F**, **e**, **E**, **g**, **G**, **a** i **A** wartość nieskończoność jest przedstawiana w formacie **[-]inf** lub **[-]infinity** zależnie od implementacji. Wartość NaN jest przedstawiana w postaci **[-]nan** lub **[i]nan(*sekwencja*)**, gdzie **sekwencja** jest zależna od implementacji. W przypadku formatów określonych wielką literą również wynikowy ciąg znaków jest wypisywany wielką literą.

Wartość zwracana

Jeżeli funkcje zakończą się sukcesem zwracają liczbę znaków w tekście (wypisanym na standardowe wyjście, do podanego strumienia lub tablicy znaków) nie wliczając kończącego '\0'. W przeciwnym wypadku zwracana jest liczba ujemna.

Wyjątkami są funkcje snprintf i vsnprintf, które zwracają liczbę znaków, które zostałyby zapisane do tablicy znaków, gdyby była wystarczająco duża.

Przykłady użycia

```
#include <stdio.h>

int main()
{
    int i = 4;
    float f = 3.1415;
    const char *s = "Monty Python";
    printf("i = %d\nf = %.1f\nWskaznik s wskazuje na napis: %s\n", i, f, s);
    return 0;
}
i = 4
f = 3.1
Wskaznik s wskazuje na napis: Monty Python
```

```
#include <stdio.h>
int main( )
{
int x = 10;
long y = 20;
double s;

s = x + y;
printf (”%s obliczen %d + %ld = %f” , ”Wynik” , x , y , s );
}

efekt na ekranie
→
Wynik obliczen 10 + 20 = 30.00000
```

scanf

deklaracja

W pliku nagłówkowym stdio.h:

```
int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
int sscanf(const char *str, const char *format, ...);
```

W pliku nagłówkowym stdarg.h:

```
int vscanf(const char *format, va_list ap);
int vsscanf(const char *str, const char *format, va_list ap);
int vfscanf(FILE *stream, const char *format, va_list ap);
```

Opis

Funkcje odczytują dane zgodnie z podanym formatem opisanym niżej. Funkcje scanf i vscanf odczytują dane ze standardowego wejścia (tj. stdin); fscanf i vfscanf ze strumienia podanego jako argument; a sscanf i vsscanf z podanego ciągu znaków.

Funkcje vscanf, vfscanf i vsscanf różnią się od odpowiadających im funkcjom scanf, fscanf i sscanf tym, że zamiast zmiennej liczby argumentów przyjmują argument typu va_list.

Argumenty

format
: format odczytu danych

stream
: strumień wejściowy, z którego mają być odczytane dane

str
: tablica znaków, z której mają być odczytane dane

ap
: wskaźnik na pierwszy argument z listy zmiennej liczby argumentów

Format

Format składa się ze zwykłych znaków (innych niż znak '%') oraz sekwencji sterujących, zaczynających się od symbolu procenta, po którym następuje:

- opcjonalna gwiazdka,
- opcjonalne maksymalna szerokość pola,
- opcjonalne określenie rozmiaru argumentu,
- określenie formatu.

Jeżeli po znaku procenta występuje od razu drugi procent to cała sekwencja traktowana jest jak zwykły znak procenta (tzn. jest on wypisywany na wyjście).
Wystąpienie w formacie białego znaku powoduje, że funkcje z rodziny scanf będą odczytywać i odrzucać znaki, aż do napotkania pierwszego znaku nie będącego białym znakiem.
Wszystkie inne znaki (tj. nie białe znaki oraz nie sekwencje sterujące) muszą dokładnie pasować do danych wejściowych.
Wszystkie białe znaki z wejścia są ignorowane, chyba że sekwencja sterująca określa format **[**, **c** lub **n**.
Jeżeli w sekwencji sterującej występuje gwiazdka to dane z wejścia zostaną pobrane zgodnie z formatem, ale wynik konwersji nie zostanie nigdzie zapisany. W ten sposób można pomijać część danych.
Maksymalna szerokość pola przyjmuje postać dodatniej liczby całkowitej zaczynającej się od cyfry różnej od zera. Określa ona ile maksymalnie znaków dany format może odczytać. Jest to szczególnie przydatne przy odczytywaniu ciągu znaków, gdyż dzięki temu można podać wielkość tablicy (minus jeden) i tym samym uniknąć błędów przepełnienia bufora.

Rozmiar argumentu

Dla formatów **d**, **i**, **o**, **u**, **x** i **n** można użyć jednego ze modyfikator rozmiaru:

- **hh** - oznacza, że format odnosi się do argumentu typu wskaźnik na signed char lub unsigned char,
- **h** - oznacza, że format odnosi się do argumentu typu wskaźnik na short lub wskaźnik na unsigned short,
- **l** (el) - oznacza, że format odnosi się do argumentu typu wskaźnik na long lub wskaźnik na unsigned long,
- **ll** (el el) - oznacza, że format odnosi się do argumentu typu wskaźnik na long long lub wskaźnik na unsigned long long,
- **j** - oznacza, że format odnosi się do argumentu typu wskaźnik na intmax_t lub wskaźnik na uintmax_t,
- **z** - oznacza, że format odnosi się do argumentu typu wskaźnik na size_t lub odpowiedni typ ze znakiem,
- **t** - oznacza, że format odnosi się do argumentu typu wskaźnik na ptrdiff_t lub odpowiedni typ bez znaku.

Dla formatów **a**, **e**, **f** i **g** można użyć modyfikatorów rozmiaru

- **l**, który oznacza, że format odnosi się do argumentu typu wskaźnik na double lub
- **L**, który oznacza, że format odnosi się do argumentu typu wskaźnik na long double.

Dla formatów **c**, **s** i **[** modyfikator **l** oznacza, że format odnosi się do argumentu typu wskaźnik na wchar_t.

Format

Funkcje z rodziny scanf obsługują następujące formaty:

- **d**, **i** odczytuje liczbę całkowitą, której format jest taki sam jak oczekiwany format przy wywołaniu funkcji strtol z argumentem base równym odpowiednio 10 dla **d** lub 0 dla **i**, argument powinien być wskaźnikiem na int;
- **o**, **u**, **x** odczytuje liczbę całkowitą, której format jest taki sam jak oczekiwany format przy wywołaniu funkcji strtoul z argumentem base równym odpowiednio 8 dla **o**, 10 dla **u** lub 16 dla **x**, argument powinien być wskaźnikiem na unsigned int;
- **a**, **e**, **f**, **g** odczytuje liczbę rzeczywistą, nieskończoność lub NaN, których format jest taki sam jak oczekiwany przy wywołaniu funkcji strtod, argument powinien być wskaźnikiem na float;
- **c** odczytuje dokładnie tyle znaków ile określono w maksymalnym rozmiarze pola (domyślnie 1), argument powinien być wskaźnikiem na char;
- **s** odczytuje sekwencje znaków nie będących białymi znakami, argument powinien być wskaźnikiem na char;
- **[** odczytuje niepusty ciąg znaków, z których każdy musi należeć do określonego zbioru, argument powinien być wskaźnikiem na char;
- **p** odczytuje sekwencje znaków zależną od implementacji odpowiadającą ciągowi wypisywanemu przez funkcję printf, gdy podano sekwencję `%p`, argument powinien być typu wskaźnik na wskaźnik na void;
- **n** nie odczytuje żadnych znaków, ale zamiast tego zapisuje do podanej zmiennej liczbę odczytanych do tej pory znaków, argument powinien być typu wskaźnik na int.

Słówko więcej o formacie **[**. Po otwierającym nawiasie następuje ciąg określający znaki jakie mogą występować w odczytanym napisie i kończy się on nawiasem zamykającym tj. **]**. Znaki pomiędzy nawiasami (tzw. *scanlist*) określają możliwe znaki, chyba że pierwszym znakiem jest ^ - wówczas w odczytanym ciągu znaków mogą występować znaki nie występujące w *scanlist*. Jeżeli sekwencja zaczyna się od **[]** lub **[^]** to ten pierwszy nawias zamykający nie jest traktowany jako koniec sekwencji tylko jak zwykły znak. Jeżeli wewnątrz sekwencji występuje znak **-** (minus), który nie jest pierwszym lub drugim jeżeli pierwszym jest ^ ani ostatnim znakiem zachowanie jest zależne od implementacji.

Formaty **A**, **E**, **F**, **G** i **X** są również dopuszczalne i mają takie same działanie jak **a**, **e**, **f**, **g** i **x**.

Wartość zwracana

Funkcja zwraca EOF jeżeli nastąpi koniec danych lub błąd odczytu zanim jakiekolwiek konwersje zostaną dokonane lub liczbę poprawnie wczytanych pól (która może być równa zero).

Przykładowe wywołanie służące do pobrania tylko i wyłącznie znaków: - (myślnik), ' (apostrof), (spacja) oraz małych i wielkich liter:

```
scanf(" %[-' A-Za-z]s",&zmienna);
```

```
#include <stdio.h>

int main( )
{
int x;
double y;
char znak;
printf( ”Podaj jedna liczbe calkowita:” );
scanf ( ” %d” , &x );
printf( ”Podaj jedna liczbe rzeczywista i jeden znak:”);
scanf ( ”%lf %c” , &y , &znak );
}

Wydruk
→
Podaj jedna liczbe calkowita:
Odczyt
←123 ↲
Wydruk
→
Podaj jedna liczbe rzeczywista i jeden znak:
Odczyt
←456.789 a
↲
Wynik wczytywania: x == 123, y == 456.789, znak == ’a’
```

getchar

```
int getchar(void);
```

Plik nagłówkowy

stdio.h

Opis

Funkcja `getchar()` czyta znak ze standardowego wejścia i go stamtąd usuwa. Wywołanie `getchar()` jest równoważne wywołaniu `getc(stdin)`.

Wartość zwracana

Funkcja `getchar()` zwraca kod pierwszego znaku ze standardowego wejścia traktowany jako `unsigned char` przekształcony do typu `int`. W przypadku końca pliku lub błędu funkcja zwraca wartość `EOF`.

Przykład użycia

```
#include<stdio.h>

int main()
{
    int i;
    i = getchar();
    printf("Przeczytano znak o numerze %d.", i);
    fflush(stdin);
    return 0;
}
```

fflush

Składnia:

```
#include "stdio.h"
int fflush(FILE * stream);
```

Opis:

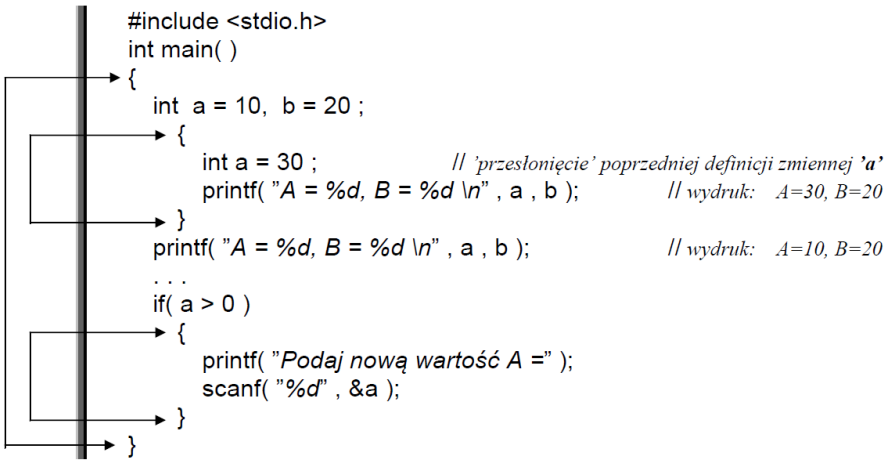
Funkcja fflush wymusza zapisanie danych znajdujących się w buforach obsługi podanego pliku. Plik pozostaje nadal otwarty. Funkcja fflush zwraca wartość `0` w przypadku wywołania zakończonego sukcesem lub wartość `EOF`, jeśli wystąpił błąd.

KOMPILACJA PROGRAMU

gcc	kompilator języka C
gcc plik.c	kompilacja pliku *plik.c*. Plik ze skompilowanym programem ma nazwę *a.out*
gcc plik.c -o program.exe	kompilacja pliku *plik.c*. Plik ze skompilowanym programem ma nazwę *program.exe*
gcc plik1.c plik2.c folder/kod.c ../projekt2/main.c	kompilacja kilku plików do jednego programu
./a.out	uruchomienie programu *a.out* (plik znajduje się w aktualnie odwiedzanym folderze)

PODSTAWOWE KONSTRUKCJE JĘZYKA C

• „instrukcja” grupująca - nawiasy klamrowe **{ }** są używane do grupowania wielu deklaracji i instrukcji w jedną instrukcję złożoną (jeden blok).



```
#include <stdio.h>
int main( )
{
   int  a = 10,  b = 20 ;
   {
      int a = 30 ;                  // 'przesłonięcie' poprzedniej definicji zmiennej 'a'
      printf( "A = %d, B = %d \n" , a , b );          // wydruk:   A=30, B=20
   }
   printf( "A = %d, B = %d \n" , a , b );             // wydruk:   A=10, B=20
   . . .
   if( a > 0 )
   {
      printf( "Podaj nową wartość A =" );
      scanf( "%d" , &a );
   }
}
```

• Instrukcja warunkowa (może mieć jedną z dwu postaci) prosta:

```
if ( wyrażenie ) instrukcja_wewnętrzna ;
```

instrukcja ta sprawdza czy *wyrażenie* jest prawdziwe (ma wartość różną od zera)

tzn. **if** (*wyrażenie*) jest równoważne **if** (*wyrażenie* != 0)

przykład:

```
#include <stdio.h>
int main( )
{
int liczba;
printf( "Podaj dowolna liczbe calkowita A = " ); scanf( "%d" , &liczba );
if( liczba % 2 == 0) // jeżeli reszta z dzielenia przez 2 jest 0
printf( "Podana liczba jest parzysta" );
}
```

lub instrukcja warunkowa złożona:

```
if ( wyrażenie ) instrukcja_1 ;
     else instrukcja_2 ;
```

przykład:

```
#include <stdio.h>
int main( )
{
char z;
printf( ”Podaj dowolna duza litere Z = ” );
scanf( ”%c” , &z );
if( z >= ’A’ && z<=’Z’ )
       printf( ”\n\n Dobrze! To jest duza litera” );
else
       printf( ”\n\n Zle! To NIE jest duza litera” );

printf( ”\n\n Nacisnij ENTER, aby zakonczyc program” );
fflush(stdin); // wyczyszczenie bufora strumienia <stdin> tzn. klawiatury
getchar( );

}
```

- Instrukcja wyboru:

```
switch ( wyrażenie_całkowite )
{
case wartość_1   : instrukcja_1;
                  break;
case wartość_2   :
case wartość_3   :
case wartość_4   : instrukcja_234;
                  break;
default          : instrukcja_domyslna;
                  break;
```

przykład:

```
#include <stdio.h>
int main( )
{
int liczba;
printf( "Podaj wartość liczby całkowitej A =" );
scanf( "%d" , &liczba );
switch( liczba )
{
case 0 : printf( ”Podałeś liczbę zerową” ); break;
case -5 : printf( ”Podałeś liczbę minus pięć” ); break;
case 7 : printf( ”Podałeś liczbę siedem” );
break;
case 9 : printf( ”Podałeś liczbę dziewięć” ); break;
default: printf( "Podałeś inną liczbę niż: 0, -5, 7, 9 " );
break;
}
}}
```

Zadania:

1. Program pobierający liczbę całkowitą i sprawdzający czy bit 5 i 11 ma wartość 1
2. Program wypisujący funkcją printf tą samą zmienną typu int w formacie zmiennoprzecinkowym i napisowym
3. Program kalkulator – interfejs : Pobranie liczb poddawanych obliczeniom (zmienny przecinek), pobranie operatora (+-/*), wyświetlenie całej operacji z wynikiem
4. Program obliczający wartość maksymalną i minimalną z 3 liczb- interfejs : pobranie 3 liczb, wyświetlenie wyniku
5. Program sprawdzający czy podany z klawiatury znak to duża litera.
6. Program, który wczyta 4 liczby rzeczywiste z klawiatury a następnie sprawdzi czy wśród nich jest więcej liczb dodatnich (>=0) czy ujemnych (<0).