WEJŚCIE I WYJŚCIE

Uwaga Funkje wejścia i wyjścia nie są częścią samego języka.

Standardowe wejście i wyjście

W bibliotece standardowej zaimplementowano prosty model znakowego wejścia i wyjścia.
Strumień znaków składa się z ciągu wierszy; każdy wiersz jest kończony nzakiem nowego wiersza.

Czytanie po jednym znaku ze standardowego wejścia:

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

Funkcja ta przy każdym wywołaniu podaje następny znak z wejścia lub EOF, gdy napotkała koniec pliku. Stała EOF jest zdefiniowana w nagłówku ¡stdio.h¿.

Wypisywanie danych po jednym znaku

```
int putchar(int );
```

Funkcja ta zwraca wartość wypisanego znaku lub EOF (jako sygnał wystąpienia błędu).

W każdym pliku źródłowym programu, który korzysta z funkcji bobliotecznych realizujących operacje wejścia-wyjścia, przed pierwszym wywołaniem musi wystąpić:

```
#include <stdio.h>
```

Dla programów czytających tylko z jednego strumienia danych wejściowych i zapisujących do jednego strumienia danych wyjściowych użycie **getchar, putchar** i **printf** wystarczy.

Przykład:

```
#include <stdio.h>
#include <ctype.h>

main()   /* lower: zamien wielkie litery na male */
{
   int c;
   while ((c=getchar()) != EOF)
      putchar(tolower(c));
   return 0;
}
```

Formatowane wyjście - funkcja *printf*

```
int printf(char *format,arg1, arg2,...);
```

Wartością zwracaną jest liczba wypisanych zmiennych.

Format zawiera:

- zwykłe znaki kopiowane bezposrednio do strumienia wyjsciowego
- specyfikacje przekształceń kolejnych argumentów (rozpoczyna je znak `%`

Między znakiem `%` i znakiem przekształcenia mogą wystąpić w poniższej kolejności:

- minus, zlecający dosunięcie przekształconego argumentu do lewego krańca pola
- liczba określająca minimalny rozmiar pola
- kropka oddzielająca rozmiare pola od precyzji
- liczba określająca precyzję (dla tekstu - maksymalną liczbę znaków, liczbę cyfr po kropce dla wartości zmiennopozycyjnej lub minimalną liczbę cyfr dla wartości całkowitej
- jedna z liter: h - jeśli argument całkowity trzeba wypisać jako *short*, lub l (litera l) - jeśli jako *long*

Podstawowe przekształcenia funkcji printf		
Znak	Typ argumentu	Dana wyjściowa
d, i	int	liczba dziesiętna
o	int	liczba ósemkowa bez znaku (bez cyfry 0)
x, X	int	liczba szesnastkowa bez znaku 0x lub 0X z literami abcdef lub ABCDEF dla 10,11,12,13,14,15
u	int	liczba dziesiętna bez znaku
c	char	pojedyńczy znak
s	char *	ciąg znaków wypisywany do napotkania znaku końca łańcucha lub wyczerpania liczby znaków określonej przez precyzję
f	double	[-]m.dddddd, gdzie liczbę cyfr d określa precyzja (domyślnie 6)
e, E	double	[-]m.ddddddde+-xx lub [-]m.dddddE+-xx, gdzie liczbę cyfr d określa precyzja (domyślnie 6)
g, G	double	w formacie %e (%E), gdy wykładnik jest mniejszy od -4, albo większy lub równy precyzji; w przeciwnym przypadku %f; nie wypisuje się nie znaczących zer i kończącej kropki dziesiętnej
p	void *	wskaźnik; postać zależna od implementacji
%		nie ma przekształcenia argumentu; wypisany znak %

Szerokość pola lub precyzję można w specyfikacji zastąpić znakiem *. Wówczas kolejny argument funkcji określa zastępowaną przez * wielkość, np.

```
printf("%.*s", max,s);
```

Zestawienie działania różnych specyfikacji podczas wypisywania tekstu "ahoj, przygodo" (14 znaków):

```
%s              :ahoj, przygodo:
%10s            :ahoj, przygodo:
%.10s           :ahoj, przy:
%-10s           :ahoj, przygodo:
%.20s           :ahoj, przygodo:
%-20s           :ahoj, przygodo      :
%20.10s         :          ahoj, przy:
%-20.10s        :ahoj, przy          :
```

Uwaga: Funkcja printf ma zmienną listę argumentów, których liczba jest określana na podstawie pierwszego z nich. Zatem:

```
printf(s); /* Niebezpieczne; zle,
           gdy w s jest znak % */
printf("%s",s);  /* Bezpieczne  */
```

Funkcja sprintf:

```
int sprintf
  (char *string, char *format, arg1, arg2,...);
```

Formatowane wejście - funkcja scanf

```
int scanf(char *format, ... );

int sscanf
  (char *string, char *format, arg1, arg2, ... );
```

Argument *format* specyfikuje format wejścia; *arg1, arg2, itd.* muszą być wskaźnikami (wskazują miejsce przekazania danych wejściowych. Funkcja *scanf* kończy działanie po zinterpretowaniu całego formatu, albo gdy typ danej jest niezgodny ze specyfikacją.

Zwraca liczbę poprawnie wczytanych i zapisanych danych, albo po napotkaniu końca pliku znak EOF.

W argumencie *format* mogą wystąpić:

- odstępy oraz znaki tabulacji - są ignorowane
- zwykłe czarne znaki (ale nie %), które spodziewamy się zastać w strumieniu wejściowym.
- specyfikacje przekształceń złożone ze znaku % , opcjonalnego znaku * wstrzymującego przypisanie, opcjonalnej liczby określającej maksymalny rozmiar pola, jednego z opcjonalnych znaków h, l lub L ustalających rozmiar wyniku oraz ze znaku przekształcenia.

Znak * wskazuje, że kolejne pole wejściowe ma być pominięte (nie będzie przypisania).

Podstawowe przekształcenia funkcji scanf		
Znak	Dana wejściowa	Typ argumentu
d	liczba całkowita dziesiętna	int *
i	liczba całkowita; może wystąpić w postaci ósemkowej (z wiodącym 0) lub szesnastkowej (z wiodącymi 0x lub 0X)	int *
o	liczba całkowita w postaci ósemkowej (razem z wiodącym 0 lub bez)	int *
u	liczba całkowita dziesiętna bez znaku	unsigned int *
x	liczba całkowita w postaci szesnastkowej (z wiodącymi 0x lub 0X, albo bez)	int *
c	znaki; następne znaki z wejścia (domyślnie 1) umieszcza się we wskazanej tablicy; nie obowiązuje zwykła zasada pomijania białych plam; aby przeczytać najbliższy czarny znak, należy użyć % 1s	char *
s	tekst (ale nie napis, tj. ciąg znaków występujący bez znaków cudzysłowu); argument powinien wskazywać na tablicę znakową o rozmiarze wystarczającym do przyjęcia tekstu wraz z dodanym na końcu znakiem końca łancucha	char *
e, f, g	liczba zmiennopozycyjna z opcjonalnym znakiem, opcjonalną kropką dziesiętną i opcjonalnym wykładnikiem	float *
%	literalnie znak % ; nie będzie żadnego przypisania	

Znaki precyzujące:

- h - w odniesieniu do znaków przekształceń d,i,o,u,x informuje, że argument jest wskaźnikiem do obiektu typu *short*
- l - argument powinien być wskaźnikiem do obiektu typu *long* danych całkowitych; oraz do obiektu typu double dla danych zmiennoprzecinkowych.

Obsługa plików

Otwarcie pliku przy pomocy bibliotecznej funkcji *fopen.* Oto niezbędne deklaracje:

```
FILE *fp;
FILE *fopen(char *name, char *mode);
```

Funkcja fopen zwraca wskaźnik na pewną zadeklarowaną w pliku nagłówkowym ¡stdio.h¿ strukturę o nazwie FILE, która zawiera następujące informacje o pliku:

- położenie bufora,
- bieżąca pozycja znaku w buforze, rodzaj dostępu do pliku (czytanie, pisanie, itd.)
- sygnały o wystąpieniu błędów lub o napotkaniu końca pliku

Wywołanie funkcji *fopen* w programie ma postać:

```
fp=fopen(name, mode);
```

Znaczenie argumentów:

```
name -  nazwa pliku
mode -  typ dostepu:
            czytanie    "r"
            pisanie     "w"
            dopisywanie "a"
```

Kilka uwag o pracy z plikami:

- otwarcie pliku nieistniejącego powoduje jego utworzenie
- otwarcie do zapisu pliku istniejącego powoduje zamazanie jego zawartości
- otwarcie pliku istniejącego do dopisywania chroni poprzednią zawartość
- próba czytania z pliku, który nie istnieje jest błędem
- w przypadku funkcja *fopen zwraca wartość NULL*

Czytanie z pliku i zapis do pliku:

```
int getc(FILE *fp); /* do czytania po znaku */
/* zwraca znak albo EOF, gdy blad */

int putc(FILE *fp); /* do zapisu znaku */
/* zwraca wartosc znaku, albo EOF przy bledzie */
```

albo przy czytaniu i zapisie formatowanym:

```
    int fscanf(FILE * fp, char *format, ... );

    int fprintf(FILE *fp, char *format, ... );
```

Przy uruchamianiu programu operacyjnego środowisko systemu operacyjnego jest odpowiedzialne za otwarcie trzech plików i udostępnienie programowi ich wskaźników:

```
stdin   - standardowe wejscie
          (normalnie zwiazane z klawiatura)
stdout  - standardowe wyjscie
          (normalnie zwiazane z ekranem)
stderr  - standardowe wyjscie bledow
          (rowniez zwiazane z ekranem)
```

Przykład programu sklejającego zawartość dwóch plików:

```
#include <stdio.h>
/* cat: sklej zawartosc plikow; wersja 1 */
main (int argc, char *argv[])
{
   FILE *fpstr;
   void filecopy(FILE *, FILE *);
   if (argc == 1) /* bez argumentow; kopiuj z stdin */
     filecopy(stdin, stdout);
   else
     while (--argc > 0)
```

```
      if ((fpstr = fopen(*++argv,"r")) == NULL) {
        printf("cat: nie moge otworzyc %s\n",*argv);
        return 1;
      } else {
        filecopy(fpstr, stdout);
        fclose(fpstr);
      }
   return 0;
}
/* filecopy: kopiuj zawartosc pliku ifp
  do pliku ofp  */
void filecopy(FILE *ifp, FILE *ofp)
{
   int c;
   while ((c=getc(ifp)) != EOF)
     putc(c,ofp);
}
```

Obsługa błędów - plik *stderr* i funkcja *exit*

Poprawiony program cat (tak, by wypisywa l komunikaty o błędach na ekran:

```
#include <stdio.h>
#include <stdio.h>
/* cat: sklej zawartosc plikow; wersja 2 */
main (int argc, char *argv[])
{
   FILE *fpstr;
   void filecopy(FILE *, FILE *);
   char *prog = argv[0];
   /* nazwa programu do komunikatow */

   if (argc == 1) /* bez argumentow; kopiuj z stdin */
     filecopy(stdin, stdout);
   else
     while (--argc > 0)
       if ((fpstr = fopen(*++argv,"r")) == NULL) {
         fprintf(stderr,
            "%s: nie moge otworzyc %s\n",prog,*argv);
         exit(1);
       } else {
         filecopy(fpstr, stdout);
         fclose(fpstr);
       }
   if (ferror(stdout)) {
      fprintf(stderr,
         "%s: blad pisania do stdout\n", prog);
      exit(2);
   }
   exit(0);
}
```

Wprowadzanie i wyprowadzanie tekstu

```
char *fgets(char *line, int maxline, FILE *fp);
char *fputs(char *line, FILE *fp);
```

Funkcja **fgets** czyta kolejny wiersz z pliku wskazywanego przez **fp** (włącznie ze znakiem nowego wiersza) i wstawia do tablicy znakowej **line**.
Zwraca: a) normalnie - wskaźnik do **line**; po wykryciu błędu, albo napotkaniu końca pliku - **NULL**.

Funkcja **fputs** wypisuje do wskazanego pliku tekst (nie musi zawierać znaku nowego wiersza).
Zwraca: 0, a w przypadku błędu - **EOF**.

Realizacja **fgets** i **fputs** w bibliotece standardowej:

```
/* fgets: wez co najwyzej n znakow z pliku iop */
char *fgets(char *s, int n, FILE *iop)
{
   register int c;
   register char *cs;
   cs=s;
   while (--n > 0 && (c=getc(iop)) != EOF)
      if ((*cs++=c) == '\n')
        break;
      *cs='\0';
      return (c==EOF && cs ==s) ? NULL:s;
}
/* fputs: wypisz s do pliku iop */
int fputs(char *s, FILE *iop)
{
   int c;
   while (c=*s++)
      putc(c,iop);
   return ferror(iop) ? EOF:0;
}
```

Funkcje wyznaczające pozycja w pliku

```
int fseek(FILE *stream, long offset, int origin);
```

Funkcja **fseek** wyznacza pozycję w strumieniu **stream**; następne czytanie lub pisanie będzie odnosić się do danych zaczynających się od nowej pozycji. Dla plików binarnych nowa pozycja wypada w miejscu oddalonym o **offset** znaków od punktu odniesienia **origin** (który może mieć wartości: bf `SEEK_SET` (początek pliku), `SEEK_CUR}` (bieżąca pozycja) lub `SEEK_END` (koniec pliku). Dla plików tekstowych wartość **offset** musi być równa zeru lub wartości zwróconej przez funkcję **ftell** (w tym przypadku **origin** musi się równać `SEEK_SET`). Funkcja **fseek** zwraca wartość różną od zera w przypadku wystąpienia błędu.

```
long ftell(FILE *stream);
```

Funkcja **ftell** zwraca wartość bieżącej pozycji dla strumienia **stream** lub **-1L** w przypadku błędu.

```
void rewind(FILE *stream);
```

Wywołanie **rewind(fp)** jest równoważne ciągowi wywołań

```
    fseek(fp,0L,SEEK_SET);    clearerr(fp);
```

```
int fgetpos(FILE *stream, fpos_t *ptr);
```

Funkcja **fgetpos** zapamiętuje bieżąą pozycję strumienia **stream** w miejscu wskazywanym przez ***ptr**. Z tej wartości można później skorzystać w funkcji **fsetpos**. Typ `fpos_t` jest odpowiednim typem obiektu do przechowywania takiej wartości. W przypadku błędu funkcja **fgetpos** zwraca wartość różną od zera.

```
int fsetpos(FILE *stream, const fpos_t *ptr);
```

Funkcja **fsetpos**ustawia bieżącą pozycję strumienia **stream** według wartości zapamiętanej przez funkcję **fgetpos** w miejscu wskazanym przez ***ptr**. W przypadku błędu funkcja **fsetpos** zwraca wartość różną od zera.