C programozás

10 óra
Filekezelés, parancssori argumentumok

File : stream kezelés

`#include <stdio.h>` kell, ebben:

`FILE` :egy típusnév, melyre mutató pointerrel használhatunk file-okat, pl.:

```
FILE * befile, * kifele;
```

File megnyitása:

```
FILE * fopen (char * filenév,  char * mode);
```

`mode` első karaktere lehet:

- `r` :read = olvasásra nyitja meg
- `w` :write = írásra létrehozza
- `a` :append = hozzáírásra nyitja meg, a file eddigi tartalma végére pozicionálva (konkatenáció)

második karaktere lehet:

- `+` :munka-file (írható és olvasható is)

majd ezek után lehet valamelyik:

- `b` :bináris file
- `t` :text, azaz szöveges file

Ha nem sikerül megnyitni, `NULL` -al tér vissza, pl. rossz file-név, mode,

- `r` : a file nem létezik
- `w` : a hordozó írásvédett, vagy tele van

Mindig ellenőrizni kel !!!

Pl. az `adataim.dat` file-t nyitja meg olvasásra, szövegfile-ként:

```
if ((befile = fopen ("adataim.dat", "rt")) == NULL)
    fprintf (stderr, "\nNem tudom megnyitni az"
           " \"adataim.dat\" file-t olvasásra\n");
```

File lezárása:

= buffer kiírása, directory update

```
int fclose (FILE * fp);
```

= `0` ha OK,

= `EOF` ha hiba történt. (`EOF`: makró, spec. értéket ad meg, pl. –1)

Igaz ugyan, hogy a program befejezésekor automatikusan le kell. hogy záródjanak a file-ok, de nem szokás a rendszerre bízni!

Hiba vizsgálatára használhatók:

```
int feof (FILE * stream);
```

`!= 0`, ha end-of-file jött

```
int ferror (FILE *stream);
```

`!= 0` ha hiba történt, hibakód: `extern int errno`

Szabványos stream-ek:

szövegesek, program indításakor már meg vannak nyitva:

`stdin` : szabványos bemenet (átirányítása: `<fnév`)

`stdout` :szabványos kimenet (átirányítása: `>fnév`)

`stderr` : szabványos hiba-kimenet (átirányítása: `2>fnév`)

Pl. program, mely 1. paramétereként megadott szövegfile-t kiírja a 2. paraméterként kapott nevű file-ba:

```
#include <stdio.h>

int main (int    argc,       /* paraméterek száma +1 */
          char * argv[])     /* paraméterstringek */
{
  int c;
  FILE * infile, * outfile;

  if (argc != 3) {
    fprintf (stderr, "\nHasználata: %s  "
                     "bem.file  kim.file\n", argv[0]);
    return 1;
  }

  if ( (infile = fopen (argv[1], "rt")) == NULL ) {
    fprintf(stderr, "\nNem tudom olvasásra megnyitni:"
                    "  %s\n", argv[1]);
    return 2;
  }

  if ( (outfile = fopen(argv[2],"wt")) == NULL) {
    fprintf (stderr, "\nNem tudom írásra megnyitni:"
                     "  %s\n", argv[2]);
    return 3;
  }

  while ( (c=getc(infile)) != EOF)  putc(c,outfile);

  fclose(infile);  fclose(outfile);
} /* main() */
```

Bináris, azaz formázás nélküli beolvasás és kiírás:

```
size_t fwrite (const void * innen,
               size_t       elemhossz,
               size_t       elemszám,
               FILE       * stream);
```

- kiír az `innen` által mutatott területről
- `elemszám` darab
- `elemhossz` méretű [byte] adatot a
- `stream` file-ba,
- visszaadja a sikeresen kiírt elemek számát, (nem byte-számot), ami hiba esetén `elemszám` -nál kevesebb.

```
size_t fread (void   * ide,
              size_t   elemhossz,
              size_t   elemszám,
              FILE   * stream);
```

- beolvas az `ide` által mutatott területre
- `elemszám` darab
- `elemhossz` méretű [byte] adatot a
- `stream` file-ból,
- visszaadja a sikeresen beolvasott elemek számát, (nem byte-számot), ami `elemszám` -nál kevesebb is lehet, ha közben a file végére ért, hiba esetén 0-t ad.

Ezek szekvenciálisan (sorrendben) írnak/olvasnak.

Közvetlen pozícionálás file-ban:

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

`offset` :relatív cím [byte]

`whence` :mihez relatív:

- `SEEK_SET` : file elejéhez képest
- `SEEK_CUR` : jelenlegi pozícióhoz k.
- `SEEK_END` : file végéhez képest

= 0 : O.K.

!= 0 : hiba, pl. file nincs megnyitva

A további `fread` ill. `fwrite` műveletek az így beállított ponttól dolgoznak.

Ha `fseek`-el a file vége utánra állunk, majd írunk, a kimaradt részre bináris nullák kerülnek.

Pillanatnyi pozíció, azaz offset leolvasása file elejéhez képest, ill. hiba setén `-1L` :

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

Megjegyzés: egyes rendszerekben `long` nem elég hosszú a file méretének megadásához, ekkor `ofs_t` az erre definiált típus.

Formázott kiírás szöveg (text) file-ba

Az eddigi műveletek egy memóriaterületet mozgattak bináris alakban, azaz ember számára nem olvashatóan.

```
 printf (           char * control, ...);     :stdout-ra
fprintf (FILE * fp, char * control, ...);     :fp file-ba
sprintf (char * sp, char * control, ...);     :sp bufferbe
```

A control string karaktereit változtatás nélkül kiírja, kivéve a `%` jellel kezdődő nyomtatásvezérlő mezőket, ezeknek rendre egy-egy (esetleg 2, 3) további paraméter kell a `...` helyen.

Ezek meglétét és helyességét a compiler ill. a függvény nem tudja ellenőrizni!

A nyomtatásvezérlő mező alakja:

`% [flag-ek] [szélesség] [.pontosság] [h|l|L] típus`

Megadja, hogy milyen típusú a megfelelő argumentum, illetve, hogy azt milyen alakban akarjuk kiírni. (A [..] itt most elhagyható részt jelöl.)

Ha a `%` -ot követő karakter nem ennek megfelelő, akkor az illető karakter nyomtatódik ki, pl. `%%` a `%` jelet nyomtatja.

`flag-ek`:

- **`-`** : az argumentum balra igazítását végzi; ha nincs, jobbra igazít
- **`+`** : + előjelet is mindig kiírja, egyébként csak - -t.

`szélesség` : minimális mezőszélesség: decimális szám, vagy `*`, ekkor a mezőszélességet a következő `int` argumentum adja meg

`.` : mezőszélesseg és a pontosság elválasztása

pontosság : **float és double esetében a tizedes jegyek száma, karakterlánc esetén a max. karakterszám;**
ha *, akkor a következő int argumentum adja

h|l|L : **hossz módosító; a kiírandó érték:**
- **h : short int**
- **l : long int**
- **L : long double**

típus : **érték típusa, a kiírás formája**

d : **int**		**:decimális**
i : **int**		**:decimális**
o : **int**		**:előjel nélküli oktális**
x : **int**		**:előjel nélküli hexadecimális**
u : **unsigned**		**:előjel nélküli decimális**
c : **char**		**:karakter**
s : **char ***		**:karakterlánc**
e : **double**		**:kitevős alak: m.nnnnnne±kk**
E : **double**		**:kitevős alak: m.nnnnnnE±kk**
f : **double**		**:fixpontos alak: -mmm.nnnnn**
g : **double**		**:e és f közül a pontosabb**
G : **double**		**:E és f közül a pontosabb**

Pl.:

```
  int x=123, h=7;
  long lint=123456;
  double d=1e-8;

 printf (
   "\n First =%d  2nd:%3cc  >%*li<    (%10G)  \"%s\" ",
              x,      'q',   h,lint,     d,    "This");
```

Eredmény:

```
First =123  2nd:  qc  > 123456<    (     1E-08)  "This"
```

Formázott beolvasás szöveg (text) file-ból

```
int  scanf (           char * control, ... );   :stdin-ről
int fscanf (FILE *fp,  char * control, ... );   :file-ból
int sscanf (char *str, char * control, ... );   :string-ből
```

A control, azaz formátumvezérlő stringgel megegyező szöveget kell találnia, egyébként leáll, kivéve ha a vezérlő stringben van:

közök (szóköz, tabulátor, újsor): bemenetben átugrik minden közt;

`%` -kal kezdődő vezérlőmezők: adott típusú értékeket olvas és tárol, ezeknek rendre egy-egy további memóriacím kell a `...` helyen, kivéve tárolás elnyomásakor (pl. `%*c`) !

!!! AZ AKTUÁLIS ... ARGUMENTUMOKNAK CÍMEKNEK (MUTATÓKNAK) KELL LENNIÜK !!!

Akkor áll le, ha:

- **a bemeneti szöveg nem egyezik a formátum szöveggel,**
- **a teljes control stringet feldolgozta,**
- **a bemeneti adatok végére ért (EOF, v. string vége),**
- **valamelyiket nem tudta beolvasni, mert hibás volt a beolvasandó adat formátuma.**

Visszatérési értéke: hány értéket olvasott be és tárolt el az argumentumokban sikeresen.

Érték előtti közöket általában átlépi, kivéve `%c` és `%[....]`

A control mező felépítése (a [..] közötti rész elhagyható, | választható):

% [*] [szélesség] [h | l | L] típus

% **:vezérlőmező kezdete**

***** **:tárolás elnyomása: ezen formátum-előírás szerinti szöveget átolvassa a file-ból, de az értékét nem tárolja el, így ...scanf visszatérési értékébe sem számlálja bele**

szélesség: decimális szám: max. mezőszélesség

l **:adathossz módosító, a megfelelő argumentum:**

int* helyett long*

float* helyett double*

L **:adathossz módosító: float* helyett long double***

típus: **konverzió típusa:**

(Elöl (l) ill. (L) azt jelenti, hogy lehet ilyen hossz-módosító)

(l) d : int* (long*) :decimális egész számot vár

(l) o : int* (long*) : oktális egész számot vár

(l) x : int* (long*) : hexadecimális egész számot vár

h : short* : short egész számot vár

(l|L) f : float* (double* vagy long double)

: lebegőpontos számot vár

c : char : egyetlen karaktert vár, az üres karaktert (szóköz, újsor, tabulátor) is beolvassa. Ha át akarjuk ugorni az első értékes karakter előtti üres karaktereket, akkor " %c" kell, azaz a %c előtt kell legalább egy szóköz v. tabulátor.

s : `char*` : egyetlen szót vár, előtte az üres karaktereket átlépi, első üres karakterig olvas, lezáró `'\0'` karaktert is eltárolja

`[`karakterek`]` : `char*` :stringet olvas be, amíg ilyen karakterek jönnek be, a `[` és `]` között a karakterek megengedett halmaza adandó meg. Pl.:

`[-+0-9]` :előjelek, dec. számjegyek, bármely sorrendben, pl.

`12-43+0.1`

esetén az aláhúzott részt olvassa be

`[a-zA-Z]` :betűk

`[^ \n\t]` :minden, csak üres karakter nem, mert `^` a komplemens halmazt jelenti

Pl. `%*[^:]%99[^\n]` :átugrik mindent ':'-ig, majd sor végéig beolvas mindent, de legfeljebb 99 karaktert

Program paraméterek

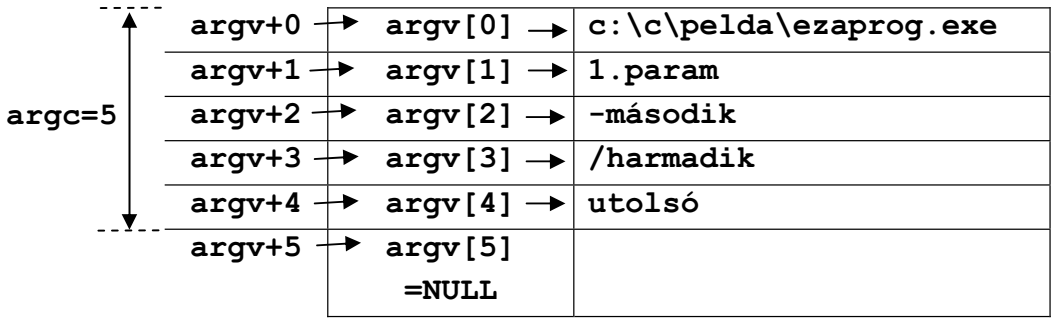
A program, azaz a **main** függvény argumentumaiként a program neve, paraméterei és azok száma megkaphatóak a következő formában.
Pl. ha egy programot így indítunk:

```
c:\c\pelda> ezaprog 1.param -második /harmadik utolsó <bemf.dat
```

akkor a paraméterekhez így férhetünk hozzá:

```
int main ( int    argc,        : program-argumentumok száma
           char * argv [] )    : argumentum-stringekre mutató
                                 pointerek tömbje
{    ....
```

Itt használhatóak a program argumentumok:



A szabványos input és a szabványos output átirányítása:

Pl.:

```
prog   <bemfile.dat   >kimf.ere
```

Ez NEM program-paraméter.

Lásd még pl. a **ProgArgs.c** mintaprogramot!