

---

# **A C programozási nyelv VI. Parancssori argumentumok File kezelés**

Ficsor Lajos

Miskolci Egyetem

Általános Informatikai Tanszék

# Parancssori argumentumok 1.

- A programot parancssorból indítva adatokat adhatunk meg számára.
- A *main* függvény prototípusának teljes alakja  
`int main(int argc, char* argv[], char* env[])`
- A `main` függvény visszatérési értékét az aktivizáló program kapja meg.
  - 0 visszaadott érték: sikeres végrehajtás
  - A visszatérési értéket a **main** függvény **return** utasítása, vagy bárhol az **exit** függvény állíthatja be.

# Parancssori argumentumok 2.

- Az **exit** függvényhez az **stdlib.h** vagy **process.h** fejlécfile szükséges. Prototípusa:

**void exit(int status)**

- A **main** függvény első két paramétere a program aktivizálásához használt parancssorban található paramétereket adja vissza

# Parancssori argumentumok 3.

- **argc** a paraméterek száma + 1
- **argv[0]** a parancssorban a prompt után gépelt első stringre (a program neve, esetleg elérési úttal) mutató pointer
- **argv[1]** az első paraméterre mutató pointer
- .
- .
- **argv[argc-1]** az utolsó paraméterre mutató pointer
- **argv[argc]** a **NULL** mutató

# Parancssori argumentumok 4.

- Hasonlóan az `env[0]`, `env[1]`, ..., `env[n-1]` a program hívásakor érvényes környezeti változókat tartalmazzák, (ahol `n` a környezeti változók száma), `env[n]=NULL`.
- A környezeti változók operációs rendszer függők!
- A parancssor paramétereit helyköz választja el. Ha egy paraméterben helyköz lenne, azt idézőjelek közé kell írni. (A program az idézőjelet nem kapja meg!)

# File a C nyelvben

- A C byte-ok sorozataként tekinti a file-okat.
- A programozó feladata a file tartalmát értelmezni.
- Kivétel: ha a file-t *text* típusúnak deklaráljuk
  - ' \n ' ("sorvég") karakter kiírása az operációs rendszertől függő sor vége jelet generál
  - sor vége jel beolvasása a ' \n ' karaktert eredményezi

# File megnyitása

- Minden file-t egy **FILE** (előredefiniált) típusú struktúra azonosít.
- Van három előre definiált **FILE\*** változó:  
**stdin** : standard input  
**stdout**: standard output  
**stderr**: standard hibacsatorna
- Minden file-t a használata előtt meg kell nyitni. Ezzel kapunk egy egyedi azonosítót, amellyel a későbbiekben hivatkozunk rá. (**fopen** függvény)

# Az fopen függvény 1.

Prototípusa:

**FILE\* fopen(char\* filenev, char\* mod)**

- **filenév** paraméter a file neve (az adott operációs rendszer szabályai szerint)
- **mod** egy, két vagy három karakteres string, amely meghatározza a file használatának a módját, az alábbiak szerint:



# Az fopen függvény 2.

Mód jele	Használat módja	A megnevezett file-lal való kapcsolat
r	Csak olvasásra	Léteznie kell!
w	Csak írásra	Ha létezik, előző tartalma elvész, ha nem létezik, létrejön egy új (üres) file.
a	Hozzáírás	Ha létezik, az írás az előző tartalom végétől kezdődik. Ha nem létezik, létrejön egy új file.
r+	Olvasás és írás	Léteznie kell!
w+	Olvasás és írás	Ha létezik, előző tartalma elvész, ha nem létezik, létrejön egy új (üres) file.
a+	Olvasás és hozzáírás	Ha létezik, az írás az előző tartalom végétől kezdődik. Ha nem létezik, létrejön egy új file.

## Az fopen függvény 3.

- A függvény visszatérési értéke a file leíró struktúrára mutató pointer, ha a megnyitás sikeres, a **NULL** pointer egyébként. Szokásos használata:

```
FILE* be;
```

```
    .    .    .  
if ( (be=fopen( "bemeno.txt", "rt" ) ) == NULL )  
{  
    fprintf (stderr, "Nyitási hiba!\n");  
    exit(1); /* A program befejezése */  
}  
else  
{ /* A file feldolgozása */ }
```

# A file lezárása 1.

- Minden file-t a használata után le kell zárni. Erre szolgál az

**int fclose(FILE\* f)**

- paramétere a lezárandó file leírója
- visszatérési értéke **0** sikeres lezárásnál, **EOF** egyébként



# A file lezárása 2.

## Megjegyzések:

- A file lezárása után az azonosítására használt változó újra felhasználható.
- A főprogram (main függvény) végén minden nyitott file automatikusan lezáródik, mielőtt a program befejezi a futását.
- A file lezárása előtt a nem üres buffer is kiíródik a file-ba. Ha a program futása rendellenesen áll le, adatok veszhetnek el.

# File írása és olvasása 1.

```
int fprintf(FILE* f, char*s, . . . )
```

- Mint a **printf**, csak az **f** file-ba ír ki.

```
int fscanf(FILE* f, char*s, . . . )
```

- Mint a **scanf**, csak az **f** file-ból olvas.

```
int getc(FILE* f)
```

- Mint a **getchar**, csak az **f** file-ból olvas.

```
int putc(char c, FILE* f)
```

- Kiírja a **c** karaktert az **f** file-ba. Visszatérési értéke a kiírt karakter, hiba esetén **EOF**.

## File írása és olvasása 2.

`int fputs(char*s, FILE* f)`

- Kiírja az `s` stringet, a záró nulla nélkül.  
Visszatérés: az utoljára kiírt karakter, vagy **EOF**.

`char* fgets(char*s, int n, FILE* f)`

- Beolvas az `s` stringbe a sorvége karakterig, de maximum `n-1` karaktert. A string végére teszi a záró nulla karaktert. Visszatérési értéke a beolvasott stringre mutató pointer, vagy **NULL**.

## File írása és olvasása 3.

`int fread(void* p, int meret, int n, FILE* f)`

- A függvény **n** db, **meret** méretű adatot olvas be, és elhelyezi azokat a **p** címtől kezdődően.
- A **p** tetszőleges objektumot megcímezhet. A programozó felelőssége, hogy
  - a memóriaterület elegendő hosszúságú-e
  - a feltöltött memóriaterület értelmezhető-e.
- Visszatérési értéke a beolvasott adatok száma. Hiba vagy file vége esetén ez **n**-től kisebb.

# File írása és olvasása 4.

```
int fwrite(void*p, int meret, int n,  
FILE* f)
```

- Mint az **fread**, de kiír.



# Pozicionálás a file-ban 1.

- Minden file-hoz tartozik egy aktuális pozíció, amit byte-ban számítunk.
- Egy file megnyitása után az aktuális pozíció 0, azaz a file elejére mutat.
- A végrehajtott írási és olvasási műveletek az aktuális pozíciót is állítják, az átvitt byte-ok számának megfelelően.
- Az aktuális pozíciót a programból is állíthatjuk (a file tetszőleges sorrendben feldolgozható)

## Pozicionálás a file-ban 2.

`int fseek(FILE* f, long offset, int bazis)`

- Hozzáadja (**előjelesen!**) az **offset** értékét a **bazis** által meghatározott értékhez, és ez lesz az új aktuális pozíció.
- **bazis** lehetséges értékei (előredef. konstansok):
  - **SEEK\_SET** a file eleje
  - **SEEK\_CUR** az aktuális file pozíció
  - **SEEK\_END** a file vége
- Visszatérési értéke **0**, ha sikeres, nem **0** egyébként.

# Pozicionálás a file-ban 3.

`long ftell(FILE* f)`

- Visszaadja az aktuális file pozíció értékét, a file elejétől számolva. Sikertelenség esetén a visszatérési érték **-1**.

`void rewind(FILE* f)`

- A file elejére állítja a file pozíciót.