Budapesti Műszaki és Gazdaságtudományi Egyetem
Közlekedés- és Járműirányítási Tanszék

Programozás C- és Matlab nyelven
C programozás kurzus
BMEKOKAM603
Mutatók, File Kezelés

Dr. Bécsi Tamás

8. Előadás

Dinamikus memóriafoglalás


```
void * malloc ( size_t size );
```

- Lefoglal *size* byte memóriát, visszatér az elejével
- Hiba esetén NULL a visszatérési érték

```
void * calloc ( size_t num, size_t size );
```

- Lefoglal *size*num* byte memóriát, és minden bíte értékét 0-ra állítja, majd visszatér az a memóriaterület elejével
- Hiba esetén NULL a visszatérési érték

```
void * realloc ( void * ptr, size_t size );
```

- Megváltoztatja a lefoglalt memóriaterület méretét
- Amennyiben az adott helyen nem fér el, új helyen foglalja le, és átmásolja a tartalmat
- Hiba esetén NULL a visszatérési érték

```
void free ( void * ptr );
```

- Felszabadítja a lefoglalt memóriát

Dinamikus memóriafoglalás
sizeof

- A fordítás idején hatásos `sizeof` unáris operátor bármilyen C nyelvű objektum méretének meghatározására használható. A

  `sizeof objektum`

  és

  `sizeof (típusnév)`

  kifejezések egy egész számot adnak eredményül, ami a megadott objektum vagy adattípus bájtokban mért mérete.

Példa dinamikus memóriafoglalásra malloc


```
#include<stdio.h>
#include<stdlib.h>
int main(void){
  unsigned m,i; double *t;
  printf("Mekkora tomb kell? "); scanf("%u",&m);
  // Memóriafoglalás
  t=(double*)malloc(m*sizeof(double));
  if(t==NULL)
  { printf("Sikertelen allokalas.\n"); return 1; }
  // Innentől úgy használjuk, mint egy közönséges tömböt.
  for(i=0;i<m;i++) t[i]=i;
  for(i=m-1;i+1!=0;i--)
      printf("%g ",t[i]);
  // felszabadítás, ha már nem kell
  free(t);
  return 0;
}
```

Példa dinamikus memóriafoglalásra 2. realloc


```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int *t,n=5,i,szam;
   t=(int*) malloc(n*sizeof(int));
   for (i=0;i<n;i++)
        *(t+i)=i;
   printf("Irj be egy szamot, es hozzafuzom:");
   scanf("%d",&szam);
   t=(int*) realloc(t,(n+1)*sizeof(int));
   t[n]=szam;
   printf("Ime:\n");
   for (i=0;i<n+1;i++)
        printf("%d ",t[i]);
   free(t);
}
```

```
Irj be egy szamot, es hozzafuzom:123
Ime:
0 1 2 3 4 123
```

6.2. Struktúrák és függvények
Struktúra mutatók

- Ha nagy struktúrát kell átadnunk egy függvénynek, akkor sokkal hatékonyabb, ha a struktúra mutatóját adjuk át és nem pedig a teljes struktúrát másoljuk át. A struktúra mutatójának deklarációja:

  `struct pont *pp;`

  Ez egy struct pont típusú struktúrát kijelölő mutatót hoz létre. Ha pp egy pont struktúrát címez, akkor `*pp` maga a struktúra, és `(*pp).x`, ill. `(*pp).y` pedig a struktúra tagjai. A pp értékét felhasználva pl. azt írhatjuk, hogy

```
struct pont kezdet, *pp;
pp = &kezdet;
printf("kezdet: (%d, %d)\n", (*pp).x, (*pp).y);
```

6.2. Struktúrák és függvények
Struktúra mutatók

- A zárójelre a `(*pp).x` kifejezésben szükség van, mert a `.` struktúratag operátor precedenciája nagyobb, mint a `*` operátoré. A `*pp.x` kifejezés azt jelentené, mint a `*(pp.x)`, ami viszont szintaktikailag hibás, mivel jelen esetben x nem mutató. A struktúrák mutatóit gyakran használjuk rövidített jelölési formában. Ha p egy struktúra mutatója, akkor a `p-> struktúratag` kifejezés közvetlenül a struktúra megadott tagját címzi.

6.2. Struktúrák és függvények
Struktúra mutatók

A `.` és `->` struktúraoperátorok a függvény argumentumát tartalmazó `()` kerek és az indexet tartalmazó `[]` szögletes zárójelekkel együtt a legmagasabb precedenciájú operátorok, így rendkívül szorosan kötnek. Például, ha adott a

```
struct { int hossz; char *str; } *p;
```

deklaráció, akkor a `++p->hossz` kifejezés a hossz változót inkrementálja és nem a p-t, mivel a precedencia-szabályoknak és a végrehajtási sorrendnek megfelelő alapértelmezés `++(p->hossz)`. A kötés zárójelezéssel változtatható meg, pl. a

`(++p)->hossz` a hossz változóhoz való hozzáférés előtt inkrementálja a p értékét, a `(p++)->hossz` pedig a hozzáférés után inkrementál.

File Kezelés
Alapfogalmak

- Cél: Adatok tárolása számítógépen
- Kétféle file típust különböztetünk meg:
  - szöveges (text)
    - „Olvasható” file, (Jegyzettömb/Notepad stb.)
    - Jellemzően szövegeket tartalmaz
  - bináris (binary)
    - Adatok kezelése olyan formátumban, „ahogy a memóriában van”.
- Alapvetően a kettő között nincs nagy különbség, csak néhány esetben, pld. a sorlezárásoknál

File Kezelés
Lépések

1. File Mutató

```
FILE *fp;
```

2. Megnyitás

```
FILE *fopen(const char *filename, const char
*mode);
```

3. Műveletek

Pozícionálás

Írás/olvasás

4. Bezárás

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

File Megnyitása


```
FILE *fopen(const char *filename, const char *mode);
FILE *fopen(filenév, megnyitási mód);
```

- A filenév elérési úttal, vagy a nélkül tartalmazza a file nevét
- Sikertelen megnyitás esetén a file mutató NULL értékkel tér vissza
- Például:

```
FILE *f; //File pointer
f = fopen("C:\\proba\\teszt.txt","w");//Megnyitás
if (f==NULL)
   { printf("Hiba!");
   return 2; }
```

File kezelés
Megnyitási módok

- `r,w,a (read, write, append)`
- `b (binary) – (hiánya esetén text mode)`
- `+ (update)`

	r rb 	w wb 	a ab 	r+ r+b rb+	w+ w+b wb+	a+ a+b ab+
Léteznie kell	x			x		
Létrehozás, ha létezik, törlődik		x			x	
Olvasható tetszőleges pozícióban	x			x	x	x
Írható tetszőleges pozícióban		x		x	x	
Csak a végére írhatunk (hozzáfűzés)			x			x

File Kezelés
Írás text módban


```
int fputs ( const char * str, FILE * stream );
```

- Az *str* által átadott stringet írja a *stream* (file)-ba.
- '\0' karakterig ír, az már nem kerül be
- Siker esetén egy nemnegatív számmal tér vissza

```
int fputc ( int character, FILE * stream );
```

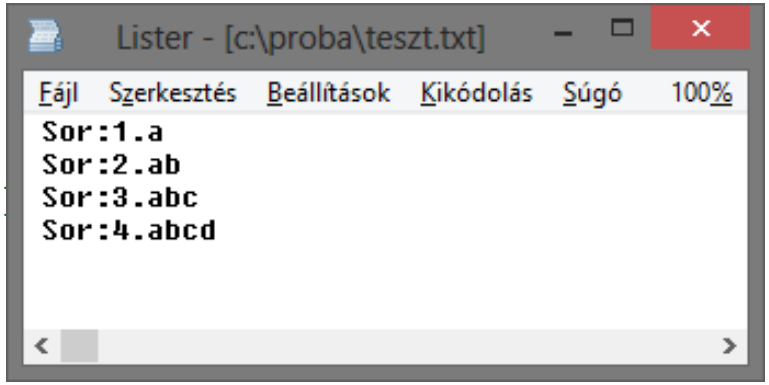
- Az átadott *character* –t írja a *stream* (file)-ba
- Siker esetén visszaadja a karaktert

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

- Hasonlóan működik, mint a printf, csak file-ba ír.

File Kezelés
Írás text módban, példa


```
int filecreate(char filename[]){
 FILE *f; //File mutató
 char c; int i;
 f = fopen(filename,"w"); // Megn
 if (f==NULL)
  { printf("Hiba!"); return 1; }
 for (i = 1; i <=4; i++)
 {
   fputs("Sor:",f);          //Sor beírása fputs segítségével
   fprintf(f,"%d.", i);      //fprintf, mint a printf
   for (c='a';c<'a'+i;c++)
     fputc(c,f);             //Egy karakter beírása
   fprintf(f,"\n");          //Sor lezárása \n-el
 }
 fclose(f); // file bezárása
 return 0;
}
```



File Kezelés
Olvasás text módban


```
char * fgets ( char * str, int num, FILE * stream );
```

- *str*-be karaktereket olvas be a *stream* (file)-ból
- Az olvasás (num-1) karakterszám elérésekor, új sor, vagy file vége esetén ér véget.
- Hiba esetén, illetve ha már nincs mit beolvasni a file végéről, NULL mutatóval tér vissza

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

- Visszaadja a karaktert, az aktuális file pozícióból, a pozíció egyet lép
- Hiba, vagy file vége esetén EOF (-1) a visszatérési érték

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

- Hasonlóan működik, mint a scanf, csak file-ból olvas.

File Kezelés
Olvasás text módban, fgetc

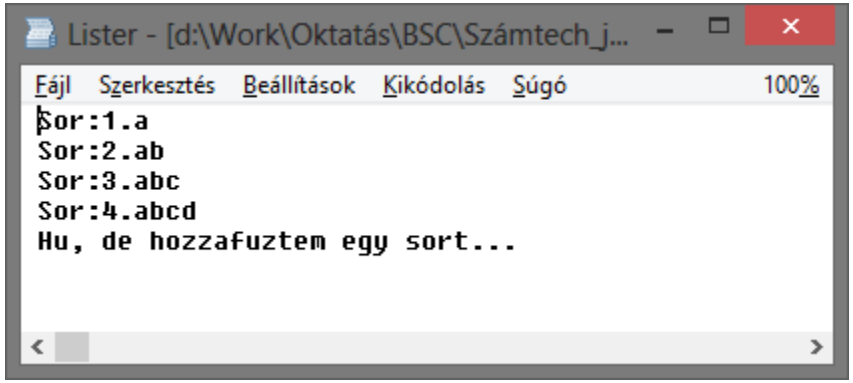

```
int listfile(char filename[])
{
  FILE *f; //File pointer
  char c;
   // Megnyitás olvasásra
   if ((f = fopen(filename,"r"))==NULL)
     return 1;
   c=fgetc(f);
   while(c!=EOF){
       printf("%c",c);
       c=fgetc(f);
   }
   fclose(f); // File bezárása
   return 0;
}
```

File Kezelés
Olvasás text módban, fgets


```
int readline_file(char filename[])
{
    FILE *f; //File pointer
    char s[200];
    f = fopen(filename,"r"); // megnyitás olvasásra
    while(fgets(s,200,f)!=0)
        printf("%s",s);
    fclose(f); //Bezárás
    return 0;
}
```

File Kezelés
Hozzáfűzés


```
int append_file(char filename[])
{
    FILE *f; //File pointer
    char s[200];
    f = fopen(filename,"a"); // Megnyitás hozzáfűzésre
    fprintf(f,"Hu, de hozzafuztem egy sort...");
    fclose(f);
}
```



File kezelés
Pozícionálás

- `int fseek ( FILE * stream, long int offset, int origin );`
- A File olvasás/írás aktuális pozícióját állítja be az *origin*-ben megadott helyhez viszonyítva *offset* távolságra
- Írás és Olvasásra megnyitott file-ok esetén az fseek meghívása szükséges a kétféle operáció közötti váltáshoz.
- Siker esetén 0-val tér vissza.

Konstans kifejezés az origin mezőben	referencia pozíció
SEEK_SET	A file Eleje
SEEK_CUR	Az aktuális pozíció
SEEK_END	A file vége

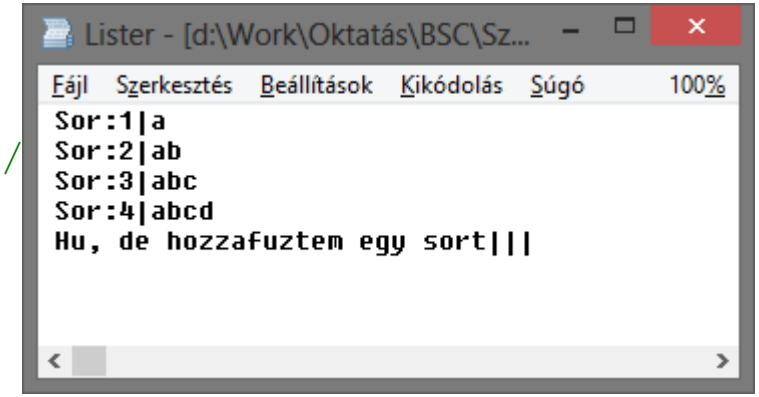
File kezelés
Pozíció lekérés


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

- Visszaadja az aktuális pozíciót a File-ban
- Hiba esetén a visszatérési érték -1

File kezelés
Pozícionálás, példa


```
int readwrite_char_file(char filename[])
{
  FILE *f; //File pointer
  char c;
  f = fopen(filename,"r+"); // olvasás/
  c=fgetc(f);
  while(c!=EOF)
  {
    if (c=='.')
     {
       fseek(f,-1,SEEK_CUR); //Pozícionálás
       fputc('|',f);
       fseek(f,0,SEEK_CUR); //Csak a módok közötti váltás miatt!
     }
    c=fgetc(f);
  }
  fclose(f); // File bezárása
  return 0;
}
```



File Kezelés
Bináris mód


```
size_t fread ( void * ptr, size_t size,
size_t count, FILE * stream );
```

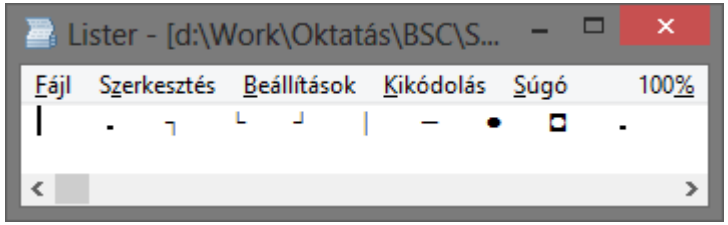
- *ptr*-be olvas *count* darab *size* byte méretű adatot a *stream* file-ból, az aktuális file pozícióból.
- Visszatérési értéke a sikeresen beolvasott adatok száma.

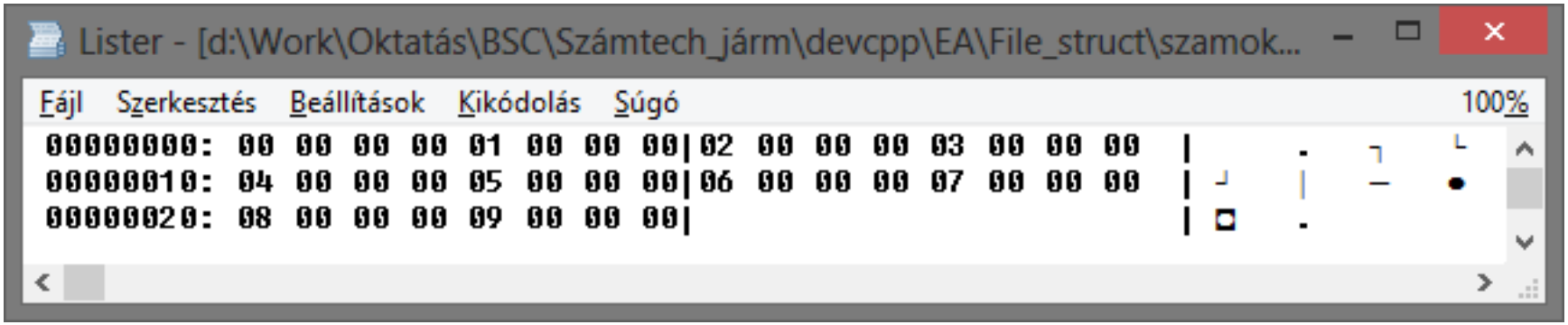
```
size_t fwrite ( const void * ptr, size_t
size, size_t count, FILE * stream );
```

- *count* darab *size* byte méretű adatot ír a *stream* file-ba, az aktuális file pozícióban.
- Az aktuális file pozíció értelemszerűen az írás végére változik
- visszatérési értéke a beírt elemek száma (count siker esetén)

Bináris file kezelés
Példa 1


```
void decimalfile()
{
   int i;
   FILE *f=fopen("szamok.dat", "wb");
   for (i=0;i<10;i++)
       fwrite(&i,sizeof (int),1,f);
   fclose(f);
}
```





Bináris file kezelés
Példa 2 struktúra


```
1:    mbc 1
2:    bbc 7
3:    bbc 38
4:    zbc 30
5:    cbc 24
```

```
struct tember {int kor;char nev[11];};
.....
void createfile(char fn[])
{
    struct tember e={65,"abc"};
    int i;
    FILE *f=fopen(fn, "wb");
    srand(time(0));
    for (i=0;i<5;i++)
    {
       e.nev[0]='a'+rand()%26;
       e.kor=rand()%40;
       fwrite(&e,sizeof (struct tember),1,f);
    }
    fclose(f);
}
```

Bináris file kezelés
Példa 3 listázás


```
1:    mbc 1
2:    bbc 7
3:    bbc 38
4:    zbc 30
5:    cbc 24
```

```
void listfile(char fn[])
{
    struct tember e={65,"abc"};
    int i=1;
    FILE *f=fopen(fn, "rb");
    while (fread(&e,sizeof (struct tember),1,f))
    {
      printf("%2d: %5s %d \n",i++,e.nev,e.kor);
    }
    fclose(f);
}
```