



Programarea Calculatoarelor

Cursul 11: Fișiere text. Fișiere binare. Funcții de prelucrare a fișierelor. Argumente la execuția programelor

Ion Giosan

Universitatea Tehnică din Cluj-Napoca
Departamentul Calculatoare



- 2



- Tipuri de fișiere

- Procesarea conținutului unui fișier presupune

- La cererea de deschidere a unui fișier

- 3



- ```
FILE* fopen(const char *nume, const char *mod);
```

- Pentru fișiere text:

**"r", "w", "a", "r+", "w+", "a+"**

- "rb", "wb", "ab", "r+b", "w+b", "a+b"**

- 4



- Moduri de prelucrare a fișierelor text

| Mod | Descriere                                                                                                                                             |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| r   | Deschiderea fișierului pentru citire. Fișierul trebuie să existe!                                                                                     |
| w   | Crearea unui fișier pentru scriere. Dacă fișierul există, conținutul acestuia este șters în totalitate!                                               |
| a   | Deschiderea sau crearea (dacă nu există) unui fișier pentru adăugarea de conținut numai la sfârșitul acestuia                                         |
| r+  | Deschiderea unui fișier pentru actualizarea conținutului (citire și scriere). Fișierul trebuie să existe!                                             |
| w+  | Deschiderea unui fișier pentru actualizarea conținutului (citire și scriere). Dacă fișierul există, conținutul acestuia este șters în totalitate!     |
| a+  | Deschiderea unui fișier pentru citirea conținutului și adăugarea de conținut numai la sfârșitul acestuia. Dacă fișierul nu există, acesta este creat. |



# Crearea/deschiderea unui fișier

---

- Exemple

- Deschiderea unui fișier pentru citire în mod text

```
FILE *pf;
pf = fopen("studenti.txt", "r");
if (pf == NULL) {
 /* nu s-a putut deschide fisierul, nu se pot continua
 operatiile cu acesta! */
}
```

- Crearea unui fișier pentru scriere în mod binar

```
FILE *pf;
pf = fopen("numere_complexe.dat", "wb");
if (pf == NULL) {
 /* nu s-a putut deschide fisierul, nu se pot continua
 operatiile cu acesta! */
}
```



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

- Scrierea unui caracter în fișier

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

- Funcția returnează codul ASCII al caracterului scris în caz de succes și **EOF** în caz de eșec
- `fputc('x', stdout)` este echivalent cu `putchar('x')`



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

- Scrierea unui șir de caractere (*string*) în fișier

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

- Funcția scrie în fișier *string*-ul **str**
- Funcția returnează o valoare ne-negativă în caz de succes și **EOF** în caz de eșec



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

- Scrierea cu format în fișier

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

- 9



```
int fflush ( FILE * pf );
```

- Dacă fișierul este deschis în scriere, conținutul zonei tampon se scrie efectiv în fișierul respectiv
 - Asigură, după apel, că fișierul stocat pe disc conține efectiv ceea ce a fost scris în el cu ajutorul funcțiilor prezentate anterior
- Dacă fișierul e deschis în citire, caracterele necitite încă din zona tampon (*buffer*) se pierd
 - Pentru golirea *buffer*-ului intrării standard (tastatură) se poate face apelul `fflush(stdin)`
- Funcția returnează zero în caz de succes și **EOF** în caz de eroare
- Observație: la închiderea oricărui fișier, zona tampon este golită automat



- ```
long ftell(FILE *pf);
```

- ```
int fseek (FILE *pf, long deplasament, int origine);
```

- Funcția returnează zero în caz de succes și altă valoare în caz de eșec



- ```
int feof(FILE *pf);
```

- Închiderea unui fișier

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

- Funcția returnează zero dacă închiderea fișierului s-a realizat cu succes
- Funcția returnează **EOF** dacă închiderea fișierului a eșuat



# Ataşarea unui alt fişier la un fişier deja deschis (pointer la FILE)

---

```
FILE * freopen (const char * nume, const char * mode,
 FILE * pf);
```

- De obicei se foloseşte la ataşarea unui fişier la unul dintre fişierele standard `stdin`, `stdout`, `stderr`
- Funcţia returnează `pf` în caz de succes şi `NULL` în caz de eşec
- Exemplu

```
freopen("out.txt", "w", stdout);
printf("Aceasta fraza se va scrie in fisier!");
fclose(stdout);
```

- Închide orice alt fişier ataşat la `stdout` şi deschide `out.txt` care este ataşat la `stdout`
- Tot ce va fi scris la ieşirea standard va fi redirectat şi scris în fişierul `out.txt`



# Exemplu – fișiere text

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
typedef struct {
 char nume[20];
 char UM[10];
 float cantitate;
 float pret;
} produs;
int main()
{
 FILE *pf;
 pf = fopen("produse.txt", "w");
 if (pf == NULL) {
 printf("Nu se poate crea fisierul!"); exit(1);
 }
 produs p[] = {{"paine taraneasca", "buc", 35, 4.58796},
 {"lapte dietetic", "litru", 85.58941, 3.4756},
 {"oua de casa", "buc", 10865, 0.568974}
 };
}
```



15



# Exemplu – fișiere text (continuare)

```
fseek(pf, 6, SEEK_CUR);
fprintf(pf, "sin");
fflush(pf);

fseek(pf, 12, SEEK_CUR);
float cant, pret;
fscanf(pf, "%f| %f", &cant, &pret);
printf("%f*%f*\n", cant, pret);

fseek(pf, 0, SEEK_END);
fputs("====", pf);
fflush(pf);

printf("Este sfarsitul fisierului? %d\n", feof(pf));
c=fgetc(pf);
printf("Caracter citit %d \nEste sfarsitul fisierului? %d\n",
 c, feof(pf));
printf("Fisierul contine %ld octeti", ftell(pf));
fclose(pf);
return 0;
}
```





```
Paine taraneasca|buc|35.000000|4.587960
lapte sintetic|litru|85.589409|3.475600
oua de casa|buc|10865.000000|0.568974
=====
```

```
Paine taraneasca|buc|35.000000|4.587960
*85.589409*3.475600*
Este sfarsitul fisierului? 0
Caracter citit -1
Este sfarsitul fisierului? 16
Fisierul contine 126 octeti
```



- Diagram illustrating the structure of a record in a file:
- |         |         |         |     |         |         |         |     |
|---------|---------|---------|-----|---------|---------|---------|-----|
| ARTICOL | ARTICOL | ARTICOL | ... | ARTICOL | ARTICOL | ARTICOL | EOF |
|---------|---------|---------|-----|---------|---------|---------|-----|
- The record is composed of multiple fields, each labeled "ARTICOL", followed by an "EOF" (End of File) marker. The fields are grouped by a bracket labeled "înregistrare" (record).

- 18



- ```
unsigned fread ( void * ptr, unsigned dim,
                unsigned nr, FILE * pf );
```

- ```
unsigned fwrite (void * ptr, unsigned dim,
 unsigned nr, FILE * pf);
```

- **ptr** este pointer la zona de memorie unde se citește/scrie
- **dim** este dimensiunea unei înregistrări exprimată în octeți
- **nr** este numărul de înregistrări care se vor citi/scrie

- Se returnează numărul de înregistrări citite/scrise corect

- Primul argument este pointer la elementul de unde începe citirea/scrierea
- Al doilea argument este dimensiunea unui element în octeți
- Al treilea argument este numărul de elemente care se vor citi/scrie



# Exemplu – fișiere binare

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct {
 char nume[20];
 char UM[10];
 float cantitate;
 float pret;
} produs;
int main() {
 FILE *pf = fopen("produse.dat", "wb");
 if (pf == NULL) {
 printf("Nu se poate crea fisierul!"); exit(1);
 }
 produs p[] = {{"paine taraneasca", "buc", 35, 4.58796},
 {"lapte dietetic", "litru", 85.58941, 3.4756},
 {"oua de casa", "buc", 10865, 0.568974}};
 int np=sizeof(p)/sizeof(produs);
 fwrite(p, sizeof(produs), np, pf);
 fclose(pf);
}
```



# Exemplu – fișiere binare (continuare)

---

```
pf = fopen("produse.dat", "rb");
produs r[np];
fread(r, sizeof(produs), np, pf);
for (int i=0; i<np; i++)
 printf("%s| %s| %f| %f\n",
 r[i].nume, r[i].UM,
 r[i].cantitate, r[i].pret);
printf("\n");
fclose(pf);
```

```
pf = fopen("produse.dat", "r+b");
strcpy(r[1].nume, "lapte sintetic");
r[1].cantitate-=10;
fseek(pf, sizeof(produs), SEEK_SET);
fwrite(r+1, sizeof(produs), 1, pf);
fflush(pf);
```



Conținutul fișierului **produse.dat** (vizualizat cu un editor de texte) la terminarea programului

22



```
paine taraneasca|buc|35.000000|4.587960
lapte dietetic|litru|85.589409|3.475600
oua de casa|buc|10865.000000|0.568974
```

```
paine taraneasca|buc|35.000000|4.587960
lapte sintetic|litru|75.589409|3.475600
oua de casa|buc|10865.000000|0.568974
```

**Fisierul contine 120 octeti**



- ```
int remove ( const char * filename );
```

- Exempu

```
if( remove( "fisier.txt" ) != 0 )
    perror( "Nu s-a putut sterge!" ); //scrie in stderr
else
    puts( "Sters cu succes!" );
```




- ```
int main(int argc, char *argv[])
```

- `int argc` este numărul de argumente trimise
- `char *argv[]` este un șir de *string*-uri constante, care conține numele argumentelor în ordine (`argv[0]` este primul argument – și care este întotdeauna numele programului)
- Argumentele trebuie să fie separate prin unul sau mai multe spații

```
...> utilitar.exe aduna 20 45 705
```

- argc : 5
- argv[0] : "utilitar.exe"
- argv[1] : "aduna"
- argv[2] : "20"
- argv[3] : "45"
- argv[4] : "705"



# Trimiterea argumentelor la execuția programului - exemplu

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define N 200

int calcul(char* operatie, int argc, char * argv[])
{
 int r=(strcmp(operatie,"aduna")==0)?0:1;
 for (int i=2; i<argc; i++) {
 int x;
 char linie[N];
 int n=sscanf(argv[i],"%d%s",&x,linie);
 if (n!=1) {
 printf("Eroare: argumentul \"%s\" nu este numar
 intreg!",argv[i]);
 exit(3);
 }
 r=(strcmp(operatie,"aduna")==0)?r+x:r*x;
 }
 return r;
}
```



27



# Trimiterea argumentelor la execuția programului - exemplu

---

Rezultate obținute în urma execuției programului în diverse situații:

...>utilitar.exe aduna

Eroare: numar insuficient de argumente!

...>utilitar.exe anuleaza 3 4

Eroare: operatia "anuleaza" este necunoscuta!

...>utilitar.exe aduna 34 sw32 56

Eroare: argumentul "sw32" nu este numar intreg!

...>utilitar.exe aduna 34 32.35 56

Eroare: argumentul "32.35" nu este numar intreg!

...>utilitar.exe aduna 45 2 100

Suma argumentelor este 147

...>utilitar.exe inmulteste 45 2 100 30

Produsul argumentelor este 270000