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

Fișiere

- În programele C, fișierele sunt văzute ca fiind un șir de octeți
- Acest șir de octeți este creat în momentul deschiderii unui fișier în program
  - Se creează un canal de comunicare între fișier și program
  - Prin deschiderea unui fișier se returnează un pointer la o structură de tip **FILE**
- Există câteva fișiere standard care sunt deschise automat la pornirea unui program
  - Pointerii de tip **FILE** care sunt automat creați sunt
    - **`stdin`** – intrarea standard (tastatură)
    - **`stdout`** – ieșirea standard (ecran)
    - **`stderr`** – ieșirea standard erori (ecran)
- Funcțiile de prelucrare sunt declarate în bilbioteca **stdio.h**

Fișiere

- Tipuri de fișiere
  - Text – conținutul lui este reprezentarea text (scrisă) a datelor
  - Binare – conținutul lui este reprezentarea din memorie a datelor
- Procesarea conținutului unui fișier presupune
  - Deschiderea fișierului
  - Prelucrarea conținutului acestuia (citiri, scrieri)
  - Închiderea fișierului
- La cererea de deschidere a unui fișier
  - Fișierul a putut fi deschis
    - Pointer-ul la structura **FILE** nu este NULL
    - Se poate prelucra conținutul acestuia iar în final se poate închide fișierul
  - Fișierul nu a putut fi deschis
    - Pointer-ul la structura **FILE** este NULL
    - Nu se poate continua cu prelucrarea conținutului acestuia
    - Nu este necesară închiderea fișierului – acesta neputând fi deschis

Crearea/deschiderea unui fișier

- Se realizează cu funcția **fopen**

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

  - **nume** reprezintă numele fișierului (poate conține cale relativă față de directorul curent sau cale absolută pe disc)
  - **mod** reprezintă modul de prelucrare a fișierului deschis
    - Pentru fișiere text:

      ”r”, **”w”**, **”a”**, **”r+”**, **”w+”**, **”a+”**
    - Pentru fișiere binare – se adaugă caracterul **b** la modurile corespunzătoare pentru fișierele text:

      ”rb”, **”wb”**, **”ab”**, **”r+b”**, **”w+b”**, **”a+b”**
- Orice fișier are un indicator (cursor) care indică octetul curent
  - În modurile **”r”**, **”w”**, **”r+”**, **”w+”**, **”rb”**, **”wb”**, **”r+b”**, **”w+b”** indicatorul este poziționat inițial la începutul fișierului
  - În modurile **”a”**, **”a+”**, **”ab”**, **”a+b”**, indicatorul este poziționat inițial la sfârșitul fișierului
  - După fiecare citire/scriere indicatorul este mutat mai departe cu numărul de octeți citiți/scriși

Crearea/deschiderea unui fișier

- 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	Deshiderea 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! */
}
```

Citirea/scrierea în fișiere text

- Citirea unui caracter din fișier

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

  - Funcția returnează codul ASCII al caracterului citit în caz de succes și **`EOF`** (constantă cu valoarea -1) în caz de eșec
  - **`fgetc(stdin)`** este echivalent cu **`getchar()`**

- Scrierea unui caracter în fișier

```
int fputc ( int caracter, 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')`**

Citirea/scrierea în fișiere text

- Citirea unui șir de caractere (*string*) din fișier

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

  - Funcția citește maximum **num-1** caractere sau până la întâlnirea sfârșitului liniei curente
  - Funcția adaugă automat caracterul terminal '\0' la sfârșitul șirului de caractere citit rezultând un *string*
  - Funcția returnează *string*-ul citit **str** în caz de succes și NULL în caz de eșec

- 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

Citirea/scrierea în fișiere text

- Citirea cu format din fișier

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

  - Funcția se comportă similar cu funcția `scanf` care citește de la intrarea standard (tastatură)
  - Funcția returnează numărul de argumente citite corect în caz de succes și `EOF` în caz de eșec

- Scrierea cu format în fișier

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

  - Funcția se comportă similar cu funcția `printf` care scrie la ieșirea standard (pe ecran)
  - Funcția returnează numărul de caractere scrise în caz de succes și o valoare negativă în caz de eșec

Vidarea zonei tampon (*buffer*) a unui fișier

```
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

Poziția indicatorului în fișier

- Determinarea poziţiei curente a indicatorului într-un fişier

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

  - Funcția returnează octetul corespunzător poziției curente a indicatorului în fișier (relativă față de începutul fișierului)

- Poziţionarea indicatorului într-un fişier pe un anumit octet

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

  - **deplasament** definește numărul de octeți cu care se va face deplasarea indicatorului față de origine
  - **origine** poate fi una din constantele
    - **SEEK_SET** – începutul fișierului
    - **SEEK_CUR** – poziția curentă a indicatorului în fișier
    - **SEEK_END** – sfârșitul fișierului
  - Funcția returnează zero în caz de succes și altă valoare în caz de eșec

Poziția indicatorului în fișier. Închiderea fișierelor

- Verficarea dacă nu mai există date de procesat – indicatorul din fișier este după ultimul octet conținut de fișier

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

  - Funcția returnează **true** dacă indicatorul este dincolo de sfârșitul fișierului
    - Se ajunge de obicei în această situație după o operație de citire care nu mai poate fi efectuată datorită faptului că indicatorul în fișier este deja la sfârșitul fișierului sau se încearcă citirea dincolo de sfârșitul fișierului
- 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

Atașarea unui alt fișier la un fișier deja deschis (pointer la FILE)

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

- De obicei se folosește la atașarea unui fișier la unul dintre fișierele standard `stdin`, `stdout`, `stderr`
- Funcția returnează `pf` în caz de succes și NULL în caz de eșec
- Exemplu

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

  - Închide orice alt fișier atașat la `stdout` și deschide `out.txt` care este atașat la `stdout`
  - Tot ce va fi scris la ieșirea standard va fi redirectat și scris în fișierul `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}
                 };
```

Exemplu – fișiere text (continuare)

```
int np=sizeof(p)/sizeof(produs);
for (int i=0;i<np;i++)
    fprintf(pf,"%s|%s|%f|%f\n",
                p[i].nume,p[i].UM,
                p[i].cantitate,p[i].pret);
fclose(pf);

pf = fopen("produse.txt", "r+");
char c=fgetc(pf);
c= toupper(c);

fseek(pf, 0, SEEK_SET);
fputc(c, pf);
fflush(pf);

char linie[200];
fseek(pf, -1, SEEK_CUR);
fgets(linie,sizeof(linie),pf);
printf("%s",linie);
```

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;
}
```

Exemplu – fișiere text (continuare)

Conținutul fișierului **produse.txt** la terminarea programului

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

Rezultate afișate pe ecran

```
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
```

Fișiere binare

- Fișierele binare pot fi considerate ca având o succesiune de înregistrări, fiecare înregistrare conținând un set de articole
  - Un articol este o dată de un tip predefinit sau definit de utilizator

- Permite accesul direct la oricare înregistrare/articol
- Datele pot fi inserate și actualizate foarte ușor
- Informația scrisă (reprezentarea internă din memorie)
  - Ocupă de obicei mai puțin spațiu decât dacă ar fi scrisă în mod text (în cazul numerelor)
  - Nu poate fi citită direct – prin vizualizare cu un editor de texte – de către un operator uman
  - Toate datele de același tip ocupă același spațiu de memorie

Citirea/scrierea în fișiere binare

- Citirea dintr-un fișier binar

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

- Scrierea într-un fișier binar

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

- Pentru ambele funcții
  - **`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
- La citirea/scrierea elementelor unor tablouri
  - 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);
```

Exemplu – fișiere binare (continuare)

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

    printf("Fisierul contine %ld octeti",ftell(pf));
    fclose(pf);
    return 0;
}
```

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

```
paine taraneasca    buc              B‘Ð’@lapte
sintetic      litru       Ç--B;p^@oua de casa
buc          Ä)FH¨?
```

Exemplu – fișiere binare (continuare)

Rezultate afișate pe ecran

```
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
```

Ștergerea unui fișier

- Poate fi realizată cu funcția **remove**

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

  - **nume** este numele fișierului (împreună cu calea absolută/relativă)
  - Funcția returnează zero în caz de succes și altă valoare în caz de eșec

- Exemplu

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

Trimiterea argumentelor la execuția programului

- Pentru a putea trimite argumente la execuția programului, funcția **`main`** trebuie să respecte următorul prototip

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

- Unde
  - **`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
- Exemplu

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

  - În urma acestui apel valorile pentru **`argc`** și **`argv`** sunt
    - **`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;
}
```

Trimiterea argumentelor la execuția programului - exemplu

```
int main(int argc, char* argv[])
{
    if (argc<3) {
        printf("Eroare: numar insuficient de argumente!");
        exit(1);
    }
    if ((strcmp(argv[1],"aduna")!=0)&&
        (strcmp(argv[1],"inmulteste")!=0)) {
        printf("Eroare: operatia \"%s\" este
                necunoscuta!",argv[1]);
        exit(2);
    }
    if (strcmp(argv[1],"aduna")==0)
        printf("Suma argumentelor este %d",
                calcul("aduna",argc,argv));
    if (strcmp(argv[1],"inmulteste")==0)
        printf("Produsul argumentelor este %d",
                calcul("inmulteste",argc,argv));
    return 0;
}
```

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
```