

Capitolul 10

FIȘIERE

- Modul de realizare a operațiilor de intrare/ieșire
- Modul text și modul binar în lucrul cu fișiere
- Funcții de I/O (intrare/ieșire) pentru lucrul cu fișiere
- Fișiere cu acces direct și funcții de bibliotecă specifice
- Alte funcții de bibliotecă specializate în prelucrarea fișierelor

Operațiile de I/O (*input/output* – engl., adică intrare/ieșire sau citire/scriere de informații):

- Asigură comunicarea între utilizator și sistemul de calcul
- Pentru specificarea acestor operații, limbajul de programare C nu include instrucțiuni, ci numeroase **funcții de bibliotecă**
- O mare parte dintre acestea vor fi prezentate în cele ce urmează.

10.1. Modul de realizare a operațiilor de citire/scriere în limbajul C

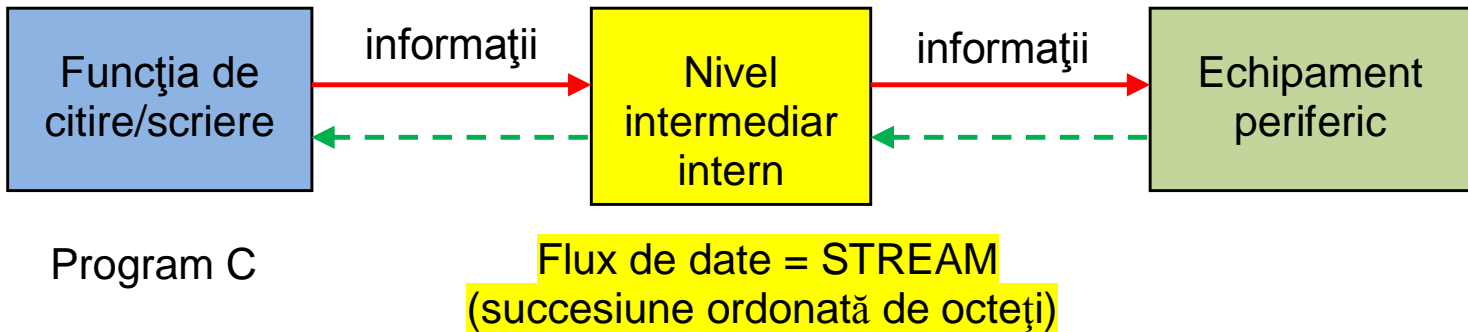
În capitolele anterioare:

- mediu de “intrare” (utilizat ca sursă de citire a informațiilor) - **tastatura**
- mediu de “ieșire” (utilizat ca destinație de scriere a informațiilor) - **ecranul**.

Pot fi utilizate și alte echipamente periferice: imprimantă, disc magnetic etc.

Unul dintre obiectivele limbajului C este de a permite programatorului să folosească **funcții standard de citire/scriere** fără a fi preocupat de echipamentul periferic pe care se va realiza operația respectivă la nivel fizic.

Principiul ce stă la baza asigurării acestei caracteristici poate fi ilustrat astfel:



Nivelul intermediar intern este un **nivel logic** utilizat cu rol de „echipament periferic virtual” în raport cu care se realizează operațiile de citire/scriere efectuate de funcțiile C specializate.

Programatorul nu va mai fi preocupat de particularitățile echipamentului periferic real, ci va utiliza funcții de citire/scriere informații din/în astfel de fluxuri de date interne.

Exemple de fluxuri interne standard (fișiere interne standard):

- **stdin** (*standard input*) – asociat, implicit, cu **tastatura** pentru citirea informațiilor;
- **stdout** (*standard output*) – asociat, implicit, cu **ecranul** monitorului pentru afișarea informațiilor;
- **stderr** (*standard error*) – asociat, implicit, cu **ecranul** monitorului pentru afișarea erorilor de execuție.

Noțiunea de **fișier** poate fi definită ca un concept logic, foarte general, ce poate fi asociat oricăror forme de memorare a informației (disc magnetic, ecran, imprimantă etc.). **Proprietățile fișierelor diferă** însă în raport cu suportul folosit pentru păstrarea informației.

De exemplu, accesarea informațiilor se poate face numai **secvențial** - în cazul imprimantei, dar și **direct** – în cazul discului magnetic.

Principalele caracteristici ale celor două tipuri de acces vor fi prezentate în detaliu în cadrul acestui capitol.

Transferul informațiilor către/dinspre dispozitivele periferice se realizează cu o viteză incomparabil mai mică decât cea ce care lucrează UC (unitatea centrală), sau chiar memoria internă a sistemului de calcul.

Creșterea performanței implică reducerea numărului de operații fizice de citire/scriere efectuate cu echipamentele periferice. Acest deziderat poate fi realizat prin folosirea unor **zone de memorie ajutătoare (buffer – engl.)** ce păstrează temporar informațiile ce urmează să fie transferate la/de la echipamentul periferic în cauză.

Sistemele de operare lucrează într-un mod similar. De exemplu, citirea a 5000 de caractere se poate face cu ajutorul unei singure operații sau cu ajutorul a 500 de operații. Dimensiunea „zonei” transferate depinde de dispozitiv – o linie de caractere de la tastatură sau un bloc de 512 sau 4096 octeți de pe disc magnetic etc.

Limbajul C include instrumente ce permit scrierea unor programe care să lucreze asemănător sistemului de operare, adică prin:

- **gruparea datelor în linii** – toți octeții ce trebuie transferați sunt memorați în *buffer* până se transmite caracterul de sfârșit de linie sau până se umple zona buffer. Abia apoi se transferă efectiv datele;
- **gruparea datelor în blocuri de octeți**
- **lucrul la nivel de caracter**

Ilustrarea a două moduri de lucru (grupat, respectiv direct -caracter cu caracter) este prezentată în următoarele exemple de program.

Exemplu de program C pentru modul de lucru „grupat”:

/* Programul citeste o fraza introdusa de la tastatura si o afiseaza, scrisa cu majuscule pe ecran. Se considera ca fraza se incheie cu caracterul punct, motiv pentru care orice caracter tastat dupa acesta va fi ignorat */

```
#include <stdio.h>
#include <ctype.h>

int main (void)
{
    char c;
    puts("Tastati o fraza ce se incheie cu punct \n");
    do
    {
        c=getchar();
        putchar(toupper(c));
    }while (c!='.');
    return 0;
}
```

De la tastatură, scriem:

Test; <ENTER>
Randul doi. Gata <ENTER>

Pe ecran apare:

Test;
TEST;
Randul doi. Gata
RANDUL DOI.

Observație: pe ecran apar în ecou și caracterele introduse de la tastatură. Rezultatele produse prin program (fraza scrisă cu majuscule) sunt cele evidențiate la redactarea acestui material prin font „bold”.

Demonstratie: FIS_1.C

Exemplu de program C pentru modul de lucru „caracter cu caracter” („direct”):

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
int main (void)
{
    char c;
    puts("Tastati o fraza ce se incheie cu punct \n");
    do
    {
        c=getche();
        putchar(toupper(c));
    }while (c!='.');
    return 0;
}
```

De la tastatură, scriem:

Test.

Pe ecran apare, pe măsură ce scriem informațiile de la tastatură:

TTeEsStT..

Observație: se reamintește faptul că funcția **getche()** preia caracterul introdus de la tastatură imediat ce apăsăm tasta (mod de lucru direct)!. Caracterul respectiv se afișează și în ecou, pe ecran.

Demonstratie: FIS_2.C

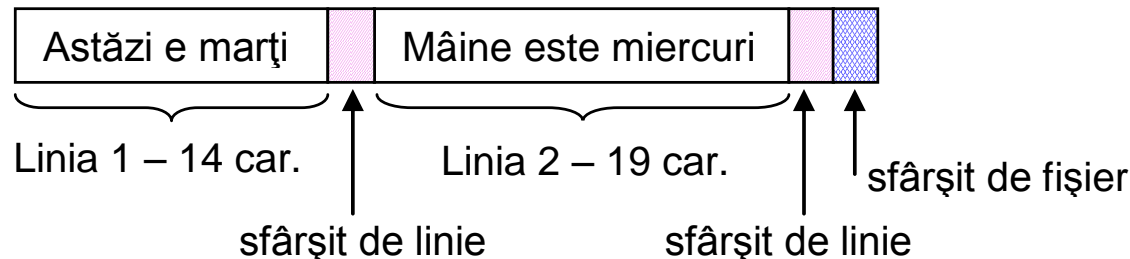
OBSERVATII IMPORTANTE

- Fiecare informație conținută într-un fișier are asociată **o poziție în cadrul fișierului**
- Operațiile de citire/scriere folosesc aceste informații prin **„deplasări”** în cadrul fișierului, adică prin **modificarea poziției curente de citire/scriere**.
- Se spune că prin citirea/scrierea unei informații **se avansează în fișier** imediat după aceasta, la informația următoare.

10.2 Modul text și modul binar în lucrul cu fișiere

Modul text, respectiv binar sunt **moduri de reprezentare a informației** în fluxurile de date interne. Deoarece crearea fișierelor cu ajutorul unui program C se face cu specificarea modului de reprezentare-interpretare a informațiilor dorit (text sau binar), este necesar ca programatorul să cunoască principalele caracteristici ale acestora.

Flux de informație de tip text: o succesiune de **caractere** organizate în **linii** de lungimi diferite (număr diferit de caractere), liniile fiind separate prin delimitatori de **sfârșit de linie** (grup de caractere ce au acest rol specific).

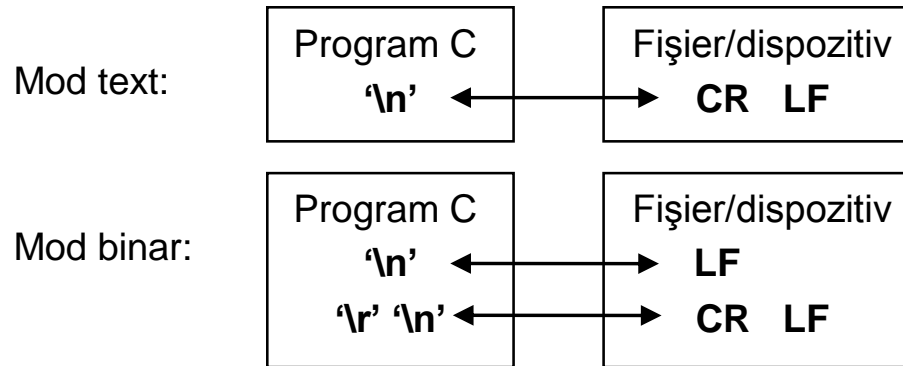


Flux de informație de tip binar: o succesiune de octeți.

Principalele deosebiri dintre cele două moduri de lucru cu fișiere se referă la două aspecte:

- tratarea sfârșitului de linie
- tratarea sfârșitului de fișier

În programele C, constanta **'\n'** (cod ASCII: 0xA sau 10₁₀) reprezintă caracterul LF (Line Feed) iar constanta **'\r'** (cod ASCII: 0xD sau 13₁₀) reprezintă caracterul CR (Carriage Return). Folosind aceste informații, figura următoare ilustrează principalele deosebiri legate de tratarea **sfârșitului de linie** în mod text, respectiv binar, mai ales în anumite implementări destinate unor versiuni de sisteme de operare de tip Windows (în sistemele de tip UNIX nu este cazul):



Scrierea informațiilor în fișier se poate face:

- în format intern (binar)
- în format extern (ca text, adică folosind reprezentarea ASCII)

Demonstratie: [fișier .exe deschis cu Notepad](#)

Dacă, de exemplu, informațiile vor fi scrise în fișier în mod binar și apoi se va încerca să fie citite în mod text, ele pot fi interpretate, ca semnificație, diferit. De exemplu, pentru un compilator care reprezintă tipul **int** pe 2 octeți, o variabilă cu valoarea **3338** se reprezintă astfel:

- **intern (binar)** – 2 octeți cu codurile **0xA** și **0xD**, adică **0000101000001101**
- **extern (text)** - 2 caractere, **LF** și **CR**, adică aceeași combinație de biți, dar interpretată altfel!

Pentru marcarea fizică a **sfârșitului de fișier**, în **mod text** se folosește **un caracter special**, cu cod ASCII **26₁₀**, adică **0x1A**. El este scris ca ultim caracter în fișier („terminator de fișier”). În limbajul C se folosește o constantă predefinită, descrisă în *stdio.h*, numită **EOF**, care este folosită de funcțiile de I/O pentru a semnala „sfârșit de fișier”. Valoarea numerică asociată acestei constante este **-1**.

În **mod binar** nu există un astfel de „terminator de fișier”! Se ține doar, implicit, evidența numărului de octeți din fișier cu ajutorul funcțiilor de I/O.

Observație: Ținând cont de existența caracterului special existent în fișierele create în mod text, dacă se crează un fișier în mod binar și se înscriu în el mai multe valori (octeți), este posibil ca printre acestea să se numere și un octet a cărui valoare să coincidă cu a „terminatorului”. Dacă vom încerca să citim fișierul în mod text, octetul respectiv va fi interpretat ca „sfârșit de fișier”, chiar dacă el se află în mijlocul și nu la sfârșitul acestuia!

Concluzie: Se recomandă, mai ales programatorilor începători, **să utilizeze fișierele în același mod în care le-au creat**. Combinarea celor două moduri de reprezentare/interpretare a informațiilor trebuie făcută în cunoștință de cauză, deoarece poate genera erori greu de depistat sau chiar deteriorări de informații.

10.3 Funcții de I/O (intrare/iesire) pentru lucrul cu fișiere

- ❖ Pointeri de tip „fișier”. Tipul *FILE*
- ❖ Deschiderea fișierelor
- ❖ Închiderea fișierelor. Erori
- ❖ Citirea/scrierea caracterelor. Funcția de testare a sfârșitului de fișier
- ❖ Citirea/scrierea șirurilor de caractere
- ❖ Funcțiile *fprintf()* și *fscanf()* (scriere/citire cu format)
- ❖ Citirea/scrierea unor blocuri de date (blocuri de octeți)

Observații:

- Pentru a asocia un **fișier intermediar (stream)** cu un anumit **fișier extern (fizic)**, se folosește operația de „deschidere fișier”.
- **Fișierul intermediar (stream)** este reprezentat în limbajul C printr-o entitate numită pointer la „fișier”

10.3.1. Pointeri de tip „fișier”. Tipul **FILE**

Funcțiile care lucrează cu fișiere au nevoie de anumite **informații**:

- nume fișier
- stare fișier
- poziția curentă în fișier
- etc.

Aceste informații sunt organizate într-o **structură** de **tip predefinit**, numit **FILE**, descris în **stdio.h**.

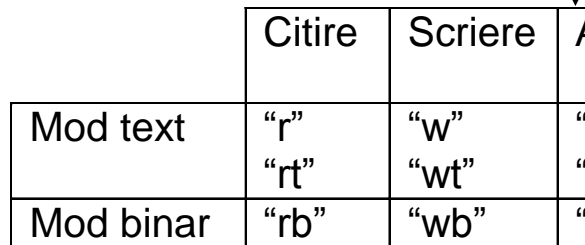
Funcțiile de bibliotecă specializate lucrează cu **adresa** informațiilor referitoare la un anumit fișier, care se memorează într-un așa-numit **pointer de tip „fișier”** (în fapt, adresa structurii de tip **FILE** ce conține informațiile despre fișierul în cauză):

```
FILE * fp;    /* fp este o variabilă de tip pointer la „fișier” */
```

10.3.2. Deschiderea fișierelor

Deschiderea unui fișier are drept rezultat crearea unei **legături (a unei asocieri)** între **fișierul extern (fizic)** și **fișierul intermediar (stream)**, reprezentat printr-un pointer la „fișier”.

FILE* fopen (**char*** NumeFisier, **char*** mod);



	Citire	Scriere	Adăugare	Citire și scriere	Scriere și citire	Adăugare și citire
Mod text	“r” “rt”	“w” “wt”	“a” “at”	“r+” “r+t”	“w+” “w+t”	“a+” “a+t”
Mod binar	“rb”	“wb”	“ab”	“r+b”	“w+b”	“a+b”

adresa informațiilor despre fișier sau **NULL** în caz de eroare

Obs:

- Pointerul returnat ca rezultat nu trebuie modificat prin program deoarece se pierde posibilitatea de a lucra cu informațiile despre fișier!!!
- Se recomandă să se testeze întotdeauna rezultatul operației de deschidere fișier, altfel funcționarea programului poate fi afectată de erori.

Exemplu:

```
FILE * fisier;
```

```
if ( ( fisier = fopen(“lucru.dat”, “w”) ) == NULL )  
    { puts(“Fișierul nu poate fi deschis!\n”);  
      exit(1);  
    }
```

Demonstratie:
deschidere.c
(cu si fara demo.txt)

O descriere sumară a efectului apelării funcției **exit()**:

- Este menționată în **stdlib.h**
- Determină închiderea tuturor fișierelor ce fuseseră deschise de către programul respectiv
- Determină încheierea execuției programului
- Determină „revenirea în sistemul de operare”

Valoarea pe care o folosește ca **parametru** poate fi utilizată de alte programe executabile.

Prin convenție, **valoarea 0** semnalează terminare normală a execuției iar **valoarea 1** – terminare cu eroare.

Observații:

- În sisteme de operare **Windows**, specificarea numelui complet de fișier necesită folosirea caracterului **\ (backslash)**.
- **DAR...** acesta reprezintă în limbajul C un element ce permite scrierea secvențelor *escape*, deci, în cadrul constantelor șir de caracter trebuie scris dublat.
- De aceea, specificarea corectă a unui nume de fișier în programul sursă C este:

"D:\\Exemple\\Nou\\lucru.dat"

și nu

"D:\Exemple\Nou\lucru.dat"

Demonstratie:
deschidere.c
(cu demo.txt in alt
director)

- În sistemele de tip **UNIX** această problema nu apare, deoarece se folosește în scopul menționat un alt caracter - caracterul **/ (slash)**.

10.3.3. Închiderea fișierelor. Erori.

Operația de închidere a unui fișier este necesară atunci când dorim să specificăm că nu mai lucrăm cu informațiile din fișierul respectiv.

O astfel de situație poate să apară:

- înainte de încheierea execuției programului
- în timpul acesteia

din diferite motive care vor fi evidențiate pe parcursul detalierii efectului celorlalte funcții de bibliotecă.

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

↓
0 = normal

↘ obținut anterior prin **fopen()**

1 = eroare (ex: *memory-stick*-ul nu mai este conectat)

Fiecare operație de I/O se poate finaliza cu succes sau eșec.

Eșecul generează așa-numitul rezultat de eroare.

Pentru a analiza eventualele erori produse de o operație de I/O și a stabili acțiunile ce trebuie întreprinse prin program se poate utiliza o funcție specializată, descrisă prin:

int ferror (FILE *fisier);

↓ ↘ obținut anterior prin **fopen()**

- 0 = normal
- diferit de 0 = s-a produs o eroare la eroare la ultima operație (apelarea unei funcții de I/O) efectuată asupra fișierului

Pentru aflarea semnificației erorii se pot folosi, de exemplu, funcțiile:

void perror (const char *s); din *stdio.h*

char * strerror (size_t n); din *string.h*

sau constanta **errno** din *errno.h* și alte elemente cu rol în depistarea și tratarea erorilor din programele scrise cu ajutorul limbajului C.

10.3.4. Citirea/scrierea caracterelor. Funcția de testare a sfarsitului de fisier.

int putc (int caracter, FILE * fisier);

fputc

scrie în

- **caracter** (convertit la *unsigned char*)
- **EOF**, în caz de eroare.

Se reamintește că **EOF** este o constantă caracter specială, definită în *stdio.h* ca având valoarea întreagă -1 (niciun caracter nu are acest cod asociat în set!)

int getc (FILE * fisier);

fgetc

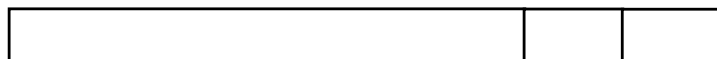
- caracterul pe care îl citește din **fisier**, convertit la *int*.
- **EOF**, în caz de eroare

Funcția de testare a sfârșitului de fișier

se poate utiliza atât în cazul **fișierelor binare** cât și în cazul **fișierelor text**.

int feof (FILE * fisier);

- diferit de 0 - a găsit sfârșit de fișier
- 0 – nu a găsit sfârșit de fișier



rezultat **feof(fisier) != 0**
rezultat **feof(fisier) == 0**

```

/*programul citeste de la tastatura un sir de caractere terminat cu punct si
scrie apoi toate caracterele intr-un fisier, pe disc magnetic*/
#include <stdio.h>
#include <stdlib.h> /*pentru functia exit()*/
void main (void)
{ FILE * fisier;
  char c;
  if ((fisier=fopen("exemplu.txt", "wt"))==NULL)
    {puts("Fisierul nu poate fi deschis \n");
     exit(1);
    }
  puts("Tastati o fraza ce se incheie cu punct \n");
  do
  { c=getchar(); /*citeste unul dintre caracterele tastate*/
    putc(c, fisier); /*scrie caracterul in fisier*/
  } while (c!='. ');
  fclose(fisier);
}

```

Demonstratie: FIS_3.C

Observatii:

- Deoarece fisierul creat este **de tip text**, puteti vedea ce contine deschizandu-l cu **orice editor de text simplu** (ex. Notepad).
- Daca doriti sa ii cititi continutul cu ajutorul unui **program scris in limbaj C**, puteti sa folositi exemplul urmator.

??? Ce se intampla daca fisierul **exemplu.txt** exista deja (fusesse creat si completat cu informatii anterior lansarii in executie a programului) ???

```

/*programul citeste de pe disc magnetic, caracter cu caracter, continutul unui fisier text si il afiseaza pe ecran*/
#include <stdio.h>
#include <stdlib.h>          /*pentru functia exit()*/

void main (void)
{ FILE * fisier;
  char c;
  if ((fisier=fopen("exemplu.txt", "rt"))==NULL)
    {puts("Fisierul nu poate fi deschis \n");
     exit(1);
    }
  puts("Continutul fisierului text este: \n");
  c=getc(fisier);    /*citeste primul caracter din fisier */
  while (c!=EOF)
  { putchar(c);      /*scrie caracterul pe ecran*/
    c=getc(fisier);  /*citeste urmatorul caracter din fisier*/
  };
  fclose(fisier);
}

```

Demonstratie: FIS_3_1.C

```

while ( ! feof(fisier) )
{ c=getc(fisier);  /*citeste din fisier*/
  putchar(c);      /*scrie pe ecran*/
};



```

Demonstratie: FIS_3_2.C

??? Ce se intampla daca fisierul **exemplu.txt** nu exista (nu fusese creat si completat cu informatii anterior lansarii in executie a programului) ???

10.3.5. Citirea/scrierea șirurilor de caractere

`int fputs (char * sir, FILE * fisier);`

scrie  în  , adică scrie toate caracterele (cu excepția lui '\0') și nu pune automat în fișier '\n', așa cum făcea funcția `puts()`

- ≥ 0 , normal (succes)
- **EOF** , în caz de eroare.

`char * fgets (char * sir, int lungime, FILE * fisier);`

- citește caractere din **fișier** și le memorează în **sir** până la întâlnirea caracterului „sfârșit de linie” sau până au fost citite **lungime-1** caractere.
- **Obs.:** dacă găsește „sfârșit de linie”, îl pune în sir, spre deosebire de funcția `gets()` care nu proceda așa!


↓ adresa de început a șirului în care se citește (**sir**) sau **NULL**, în cazul în care se depistează „sfârșit de fișier” sau apare o eroare.

Demonstratie (poate...):
[test_fisier_t_b.c](#)

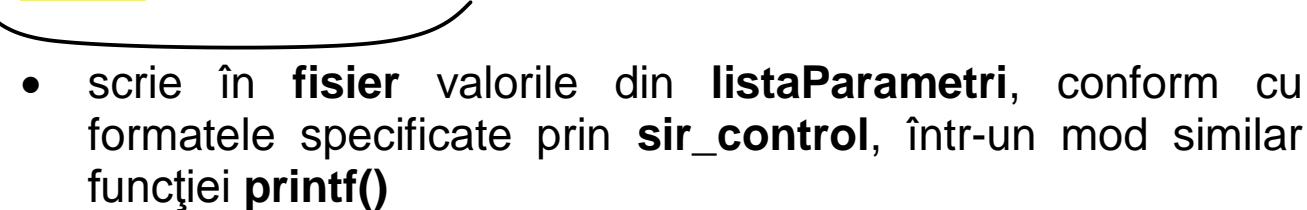
10.3.6 Funcțiile fprintf și fscanf (scriere/citire cu format)

Lucrează cu „fișiere disc” (în general, de tip text). „**Fișier disc**” înseamnă fișier stocat fizic pe disc magnetic.

int fscanf(FILE *fisier, char *sir_control, listaParametri);


numărul de elemente din **listaParametri** ce au putut fi convertite corect din șirul de caractere citit din **fisier** în valori de tip corespunzător formatului specificat în **sir_control** sau **EOF** (în caz de eroare sau de depistare a sfârșitului de fișier).

int fprintf(FILE *fisier, char *sir_control, listaParametri);



- scrie în **fisier** valorile din **listaParametri**, conform cu formatele specificate prin **sir_control**, într-un mod similar funcției **printf()**

numărul de caractere scrise sau **<0** în caz de eroare.

/*programul calculeaza si afiseaza in forma tabelata toate valorile din tabla inmultirii si a impartirii cu numere de la 1 la 10*/

```
#include <stdio.h>
#include <stdlib.h>
int main (void)
{ int l, j, vi, vf;
  FILE *f,*g;
  if ((f=fopen("INM.txt","wt"))==NULL)
    {puts("Fis. Pt. inmultire nu s-a deschis\n");
     exit(1);
    }
  if ((g=fopen("IMP.txt","wt"))==NULL)
    {puts("Fis. Pt. impartire nu s-a deschis\n");
     exit(1);
    }
  vi=1;
  vf=5;
  while(vi<11)
    {for (i=1;i<11;i++)
      {for (j=vi;j<=vf&&j<11;j++)
        fprintf(f,"%2d x%2d = %2d\t", j,i,i*j);
        fprintf(f,"\n");
      }
      vi=j;
      vf=j+5;
      fprintf(f,"\n\n");
    }
  fclose(f);
```

```
/*tabla impartirii*/
printf("\n\n");
vi=1;
vf=5;
while(vi<11)
  {for (i=1;i<11;i++)
    {for (j=vi;j<=vf&&j<11;j++)
      fprintf(g,"%2d :%2d = %2d\t", i*j,j,i);
      fprintf(g,"\n");
    }
    vi=j;
    vf=j+5;
    fprintf(g,"\n\n");
  }
fclose(g);
return 0;
}
```

Demonstratie: [TablaInm.C](#)

10.3.7 Citirea/scrierea unor blocuri de date (blocuri de octeți)

unsigned fread (**void ***buf, **int** nr_octeti_elem, **int** nr_elem, **FILE ***fisier);

↓
numărul de elemente efectiv citite din **fisier** în **buf**:

- \leq **nr_elem** (normal)
- **0** depistare sfârșit de fișier – **feof()** sau eroare – **ferror()**

unsigend fwrite (**void ***buf, **int** nr_octeti_elem, **int** nr_elem, **FILE ***fisier);

↓
numărul de elemente efectiv scrise din **buf** în **fisier**:

- \leq **nr_elem** (normal)
- **0** în caz de eroare

Observație:

dacă se deschide un fișier în mod binar,
cele două funcții pot lucra cu **orice tip de informații**.

Exemplu: scrierea într-un fișier a tuturor elementelor unui tablou numeric, printr-o singură operație

```
#include<stdio.h>
#include<stdlib.h>
void main(void)
{   FILE *fisiere;
    float a[20];
    int k,n;
    if((fisiere=fopen("valori.dat", "wb"))==NULL)
        { puts("Fișierul nu poate fi deschis\n");
          exit(1);
        }
    printf("Cate valori contine tabloul? ");
    scanf("%d", &n);
    printf("Tastati valori pentru elementele tabloului:\n");
    for(k=0; k<n; k++)
        { printf("a[%d]=",k);
          scanf("%f", &a[k]);
        }
    /*scrierea in fisier a tuturor valorilor tabloului*/
    fwrite(a, sizeof(a[0]), n, fisiere);
        /* sau: fwrite(a, sizeof(float), n, fisiere); */
    fclose(fisiere);
}
```

Demonstratie: FIS_4.C

Putem deschide fisierul astfel creat cu un editor de text obisnuit?

Putem citi continutul fisierului cu un astfel de editor?

Informatiile citite sunt corecte ca semnificatie?

Demonstratie pe calculator...

Exemplu: citirea din fișier a tuturor valorilor tabloului, printr-o singură operație

```
#include<stdio.h>
#include<stdlib.h>
void main(void)
{ FILE *fisier;
  float a[20];
  int k,n;
  if((fisier=fopen("valori.dat", "rb"))==NULL)
    { puts("Fișierul nu poate fi deschis\n");
      exit(1);
    }
  printf("Cate valori continea tabloul in scris in fisier? ");
  scanf("%d", &n);
  /*citirea din fisier a tuturor valorilor tabloului*/
  fread(a, sizeof(a[0]), n, fisier);
  /*afisarea pe ecran a valorilor citite*/
  for(k=0; k<n; k++)
    printf("a[%d]=%f \n", k, a[k]);
  fclose(fisier);
}
```

Demonstratie: FIS_4_1.C

10.4 Fișiere cu acces direct și funcții de bibliotecă specifice

- Acces **secvențial** - specific fișierelor **text**
- Acces **direct** - caracteristic fișierelor **binare** (ele pot fi accesate **și secvențial**).
 - ↘ acces la un anumit octet, care se află (de exemplu) în „mijlocul” unui fișier, fără a fi nevoiți să parcurgem prin citire ceilalți octeți care îl preced.

Informațiile stocate pe **discul magnetic** pot fi privite ca un „tablou de octeți”. Prin urmare, **se poate accesa un anumit octet** dacă se precizează **poziția acestuia** în „tablou”. Trebuie menționat faptul că, în limbajul C, pozițiile asociate octeților ce compun un fișier sunt similare celor asociate elementelor tablourilor: **primul octet ocupă poziția zero**, al doilea ocupă poziția 1 și așa mai departe.

Funcția standard ce permite **poziționarea pe un anumit octet** în cadrul unui fișier are prototipul:

```
int fseek (FILE * fisier, long deplasare, int pozitie_start);
```

↓
=0 corect
≠0 incorect

numărul de octeți față de

{ 0 sau **SEEK_SET**
1 sau **SEEK_CUR**
2 sau **SEEK_END**

Constantele predefinite (descrise în *stdio.h*) au semnificația:

SEEK_SET sau **0** – începutul fișierului
SEEK_CUR sau **1** – poziția curentă în fișier
SEEK_END sau **2** – sfârșitul fișierului

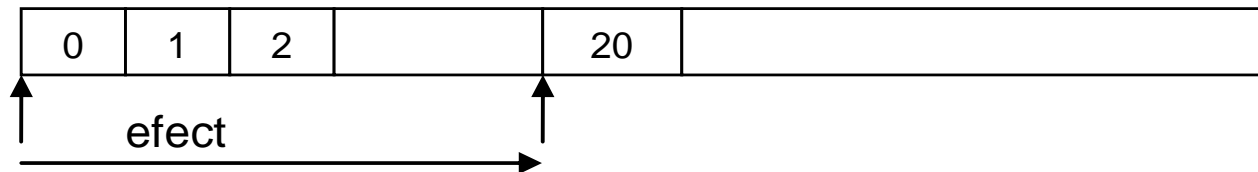
Exemple:

int pozitie;

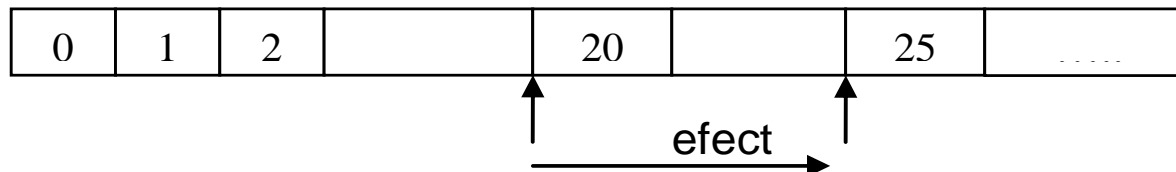
pozitie = fseek (fisier, 3L, SEEK_END);

(am cerut deplasare din poziția curentă la destinația „cu 3 octeți dincolo de sfârșitul fișierului”, ceea ce este imposibil, deci **pozitie** va primi o **valoare diferită de zero** - eroare).

fseek (fisier, 20L, SEEK_SET);



fseek (fisier, 5L, SEEK_CUR);



O altă funcție deosebit de utilă din biblioteca standard a limbajului C este:

long ftell (**FILE** *fisier);

↙ în caz de succes, reprezintă **numărul de octeți** existenți în fișier, calculați **de la începutul fișierului și până la poziția curentă**
↓
în caz de eroare, este valoarea **-1L** (long)

Obs:

- La fel ca la tablouri, pentru fișiere **primul octet** are asociată **poziția 0**
- Funcțiile **fseek()** și **ftell()** se folosesc, în general, pentru fișiere binare, dar pot fi utilizate (în cunoștință de cauză) și pentru fișiere text.

Altă variantă de rezolvare pentru „Exemplu: citirea din fișier a tuturor valorilor tabloului, printr-o singură operație”

```
#include<stdio.h>
#include<stdlib.h>
int main(void)
{ FILE *fisier;
  float a[20];
  int k,n;
  if((fisier=fopen("valori.dat", "rb"))==NULL)
    { puts("Fișierul nu poate fi deschis\n");
      exit(1);
    }
  fseek(fisier,0L,2); /*pozitionare la sfarsitul fisierului*/
  n=ftell(fisier)/sizeof(a[0]);
  fseek(fisier,0L,0); /*pozitionare la inceputul fisierului*/
  /*citirea din fisier a tuturor valorilor tabloului*/
  fread(a, sizeof(a[0]), n, fisier);
  /*afisarea pe ecran a valorilor citite*/
  for(k=0; k<n; k++)
    printf("a[%d]=%f \n", k, a[k]);
  fclose(fisier);
}
```

Demonstratie: FIS_4_2.C

Exemplu: Program de copiere fișiere (cu argumente în linia de comandă).

Fișierele se deschid în mod binar, iar componentele sunt considerate ca având dimensiune de 1 octet. Programul poate fi folosit pentru orice tip de fișiere. Programul sursă va fi salvat cu numele **cp.c**, iar pentru lansarea sa în execuție din linia de comandă va trebui scris: **cp NumeFisSursa NumeFisDestinatie<ENTER>**

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char *argv[ ])
{ FILE *sursa, *dest;
  char buffer[1000];
  size_t n;
  if(argc != 3)
    {fprintf(stderr, "cp: Sintaxa este: cp sursa dest\n");
     /* stderr este stream standard (ecranul, in mod implicit)
      dar poate fi asociat altui dispozitiv, in mod explicit*/
     exit(1);
    }
  /*se folosesc argumentele introduse in linia de comanda*/
  if((sursa=fopen(argv[1], "rb"))==NULL)
    {fprintf(stderr, "cp: Eroare deschidere fisier %s\n", argv[1]);
     exit(1);
    }
  if((dest=fopen(argv[2], "wb"))==NULL)
    { fprintf(stderr, "cp: Eroare deschidere fisier %s\n", argv[2]);
     exit(1);
    }
  n=fread(buffer, 1, 1000, sursa);
  while(n>0)
    { fwrite(buffer, 1, n, dest);
      n = fread(buffer, 1, 1000, sursa);
    }
  fclose(sursa);
  fclose(dest);
  return 0;
}
```

Demonstratie:
cp.C

10.5 Alte funcții de bibliotecă specializate în prelucrarea fișierelor

freopen() – reassignarea fișierelor (dispozitivelor de I/O) prin program

fflush() – forțează scrierea conținutului *buffer*-elor în fișierele date ca parametru (deschise pentru scriere). Unele implementări permit apelarea și pentru fișiere deschise pentru citire (ex: **stdin**, asociat tastaturii), efectul fiind „ștergerea”(golirea) conținutului *buffer*-ului asociat fișierului.

remove() – șterge fișier

Demonstratie (eventual...):
[test_reopen.c](#)

rename() – schimbă nume fișier

rewind() – echivalent cu { **fseek(fp, 0L, SEEK_SET); clear(fp);** }, adică „rebobinare” fișier (revenire la începutul fișierului) și resetarea indicatorilor de eroare.

fgetpos() – memorează poziția curentă din fișier

fsetpos() – setează poziția curentă din fișier

Obs: există și alte funcții interesante și utile pentru studierea cărora se recomandă citirea literaturii de specialitate, a documentației *on-line* și a altor surse similare.

```

/*programul permite concatenarea mai multor fisiere text*/
#include<stdio.h>
#include<stdlib.h>
void main(void)
{ FILE * tf[10], *rez;
  int i, nr, n;
  char nume[20], buffer[1000], mesaj[ ]="Nu pot deschide fisierul %s";
  puts("Programul concateneaza numai fisiere cu extensia .txt \n");
  puts("Tastati numele fisierului in care doriti sa pastrati rezultatul: ");
  gets(nume);
  if((rez=fopen(nume, "at"))==NULL)
  { printf(mesaj, nume);
    exit(1);
  }
  puts("Cate fisiere doriti sa concatenati? ");
  scanf("%d", &nr);      /*se citeste o valoare numerica !!!*/
  fflush(stdin);      /*daca nu se foloseste, in buffer-ul asociat tastaturii
                        ramane "sfarsitul de linie"*/

  i=0;
  while (i<nr)
  { puts("Tastati numele urmatorului fisier (ce trebuie concatenat):");
    gets(nume); /*se citeste un sir de caractere*/
    if ((tf[i]=fopen(nume, "rt"))==NULL)
    { printf(mesaj, nume);
      exit(1);
    }
    /*citeste continutul fisierului precizat prin nume si il adauga in fisierul rezultat*/
    n=fread(buffer, 1, 1000, tf[i]);
    while(n>0)
    { fwrite(buffer, 1, n, rez);
      n = fread(buffer, 1, 1000, tf[i]);
    }
    fclose(tf[i]);
    i++;
  }
  fclose(rez);
}

```

Demonstratie: FIS_6.C
 unu.txt doi.txt trei.txt

```

/*alta varianta pentru programul ce permite concatenarea mai multor fisiere text*/
#include<stdio.h>
#include<stdlib.h>
void main(void)
{ FILE * f, *rez;
  int i, nr, n;
  char nume[20], buffer[1000], mesaj[ ]="Nu pot deschide fisierul %s";
  puts("Programul concateneaza numai fisiere cu extensia .txt \n");
  puts("Tastati numele fisierului in care doriti sa pastrati rezultatul: ");
  gets(nume);
  if((rez=fopen(nume, "at"))==NULL)
    { printf(mesaj, nume);
      exit(1);
    }
  puts("Cate fisiere doriti sa concatenati? ");
  scanf("%d", &nr);      /*se citeste o valoare numerica !!!*/
  fflush(stdin);      /*daca nu se foloseste, in buffer-ul asociat tastaturii
                        ramane "sfarsitul de linie"*/

  i=0;
  while (i<nr)
    { puts("Tastati numele urmatorului fisier (ce trebuie concatenat):");
      gets(nume); /*se citeste un sir de caractere*/
      if ((f=fopen(nume, "rt"))==NULL)
        { printf(mesaj, nume);
          exit(1);
        }
      /*citeste continutul fisierului precizat prin nume si il adauga in fisierul rezultat*/
      n=fread(buffer, 1, 1000, f);
      while(n>0)
        { fwrite(buffer, 1, n, rez);
          n = fread(buffer, 1, 1000, f);
        }
      fclose(f);
      i++;
    }
  fclose(rez);
}

```

Demonstratie: FIS_6_bis.C
 unu.txt doi.txt trei.txt

Exemplu: preluat din “Inițiere în limbajul de programare C”- D. Irimescu-Litografia UPB-1992 și adaptat la forma prezentată în fișierele sursă numite Scr_fis.c, Cit_fis.c și Sort_fis.c

Exemplu de sortare a informațiilor conținute de un fișier binar:

- Dorim să creăm un fișier binar în care să memorăm informații despre angajații unei firme.
- Informația despre un angajat va fi reprezentată folosind o definiție de **tip structură (PERSONA)**.
- Analizând datele problemei, se observă ușor că sunt „voluminoase” și că, probabil, vor fi introduse în fișier aleator, nu ordonate după un anumit criteriu.
- Presupunem că ne propunem ca obiectiv final sortarea și afișarea pe ecran a informațiilor în ordinea alfabetică a numelor angajaților.

Variantă de rezolvare:

- Informațiile vor fi aduse (toate și în întregime!) în memoria internă, vor fi sortate și rescrise pe disc, eventual în alt fișier sau în același, redenumit.
- Va fi necesar un număr mare de operații și se va folosi multă memorie. Dacă volumul de informații este foarte mare este posibil să nu existe suficientă memorie pentru a le prelucra pe toate odată și atunci ele vor trebui prelucrate „pe bucăți”.
- Dacă, ulterior, se dorește sortarea informațiilor după un alt criteriu, se va efectua din nou un mare număr de operații.
- În concluzie, o astfel de abordare nu este eficientă pentru că generează o serie de probleme legate de prelucrarea informațiilor.

Se va adopta o soluție care nu modifică fișierul ci creează un fișier ajutător pe care îl vom numi **fișier index**.

De exemplu, dacă fișierul dat (original) conține numai 4 înregistrări:

Gheorghiu Florin	21	01	1980	Ploiești
Toma Ion	03	11	1974	Reșița
Alexandrescu Victor	15	03	1954	Tîrgoviște
Botez Cristian	11	11	1960	București

pentru a realiza sortarea se va citi din fișier numai acea parte din informația referitoare la persoană în raport cu care se face sortarea, pe care o vom numi „**cheie**” – în cazul nostru va fi numele persoanei. Fiecărei **chei** i se va asocia un indicator, numit „**index**”, ce arată poziția pe care o ocupă informația completă în fișierul dat (original).

<u>Index</u>	<u>Cheie</u>
--------------	--------------

0	Gheorghiu Florin
1	Toma Ion
2	Alexandrescu Victor
3	Botez Cristian

Index

Cheie

0	Gheorghiu Florin
1	Toma Ion
2	Alexandrescu Victor
3	Botez Cristian

În continuare se va face **sortarea** perechilor **index-cheie** în ordinea alfabetică a **cheilor** (numelor persoanelor). Cantitatea de date cu care se lucrează este limitată la valorile cheilor:

Index

Cheie

2	Alexandrescu Victor
3	Botez Cristian
0	Gheorghiu Florin
1	Toma Ion

- Informațiile din fișierul original nu se rearanjează.
- Se lucrează numai asupra fișierului **index**, ceea ce simplifică mult prelucrarea informațiilor (de obicei, sunt mai puține informații de prelucrat).
- Perechile **index-cheie** pot fi reprezentate folosind o definiție de **tip structură (INDEX)** care are 2 câmpuri – poziția înregistrării în fișierul original și cheia (câmpul din înregistrare după care se face sortarea).
- După sortare ele vor fi memorate în **fișierul index**.

Observații:

- În exemplele de program prezentate în fișierele sursă următoare se lucrează cu un volum mic de date. De aceea, diferența (ca număr de prelucrări și volum de informații) dintre sortarea fișierului „index” și a celui original nu este foarte semnificativă.
- În situații reale însă, metoda prezentată reduce în mod semnificativ volumul de prelucrări și cantitatea de memorie necesară execuției.
- Evident, există și alte metode de prelucrare a fișierelor! Acesta este doar un exemplu pentru creare, afișare și sortare a componentelor unui fișier binar în una dintre variantele posibile!!!

Observație finală

NU încercați să învățați pe de rost rezolvarea prezentată ca exemplu! Important este să o înțelegeți!!!

Demonstratie:

SCR_FIS.C

CIT_FIS.C

Sort_fis.c

```

/* Scr_fis.c - Program de citire a informatiilor despre persoane de la tastatura si memorare a
lor in fișierul binar numit exemplu.dat, aflat in acelasi director cu programul executabil*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define LUNG_NUME 20
typedef struct
{
    char nume[LUNG_NUME];
    int zi,luna,an;
    char loc_nastere[LUNG_NUME];
} PERSOANA;
int main(void)
{
    FILE *f;
    int n,i;
    PERSOANA p;
    if ((f=fopen("exemplu.dat","wb")) ==NULL)
        {printf("Fișierul %s nu poate fi deschis \n", "exemplu.dat");
        exit(1);
        }
    printf("introduceti date de la tastatura:\n");
    printf("Cate persoane inregistrati?: ");
    scanf("%d", &n);
    for (i=1; i<=n;i++)
        { printf("persoana %d\n", i);
          printf("numele=");
          fflush(stdin);
          gets(p.nume);
          printf("zi-luna-an nastere=");
          scanf("%d%d%d", &p.zi, &p.luna, &p.an);
          printf("loc nastere=");
          fflush(stdin);
          gets(p.loc_nastere);
          fwrite(&p, sizeof(PERSOANA),1,f);
        }
    fclose(f); return 0;
}

```

/* **Cit_fis.c** - Program de citire a informatiilor despre persoane din **fișierul binar** numit **exemplu.dat**, aflat in acelasi director cu programul executabil, si afisare a lor pe ecran */

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define LUNG_NUME 20

typedef struct
{
    char nume[LUNG_NUME];
    int zi,luna,an;
    char loc_nastere[LUNG_NUME];
}PERSOANA;

int main (void)
{
    FILE *f;
    int n,i;
    PERSOANA p;
    if ((f=fopen("exemplu.dat","rb")) ==NULL)
        {printf("Fișierul %s nu poate fi deschis \n", "exemplu.dat");
        exit(1);
        }
    fseek(f,0L,2);
    n=ftell(f)/sizeof(PERSOANA);
    fseek(f,0L,0);
    for (i=1; i<=n;i++)
        {fread(&p,sizeof(PERSOANA),1,f);
        printf("%20s, %d %d %d, %20s \n",
                p.nume,p.zi,p.luna,p.an,p.loc_nastere);
        }
    fclose(f);
    return 0;
}
```

```
/* Sort_fis.c - Program de ordonare si afisare a informatiilor despre persoane, conform  
cerintelor din enuntul problemei. */
```

```
#include <stdlib.h>  
#include <stdio.h>  
#include <string.h>  
#define LUNG_NUME 20
```

```
typedef struct
```

```
{char nume[LUNG_NUME];  
  int zi,luna,an;  
  char loc_nastere[LUNG_NUME];  
}PERSOANA;
```

```
typedef struct
```

```
{ int index;  
  char cheie[LUNG_NUME];  
}INDEX;
```

```
int constr_tablou_index (FILE *fis, INDEX tabl[], int nr_max)
```

```
{ /*citeste componentele din fisierul dat, extrage chei (nume),  
   returneaza nr. componente gasite*/  
  int nr_gasite=0;  
  int depl=0;   /*pt. trecerea la componenta urmatoare*/  
  int k;  
  fseek(fis, 0L,SEEK_END);  
  nr_gasite=ftell(fis)/sizeof(PERSOANA);  
  fseek(fis,0L,SEEK_SET);  
  for (k=0; k<nr_gasite && nr_gasite<nr_max;k++)  
  { tabl[k].index=k; /*se completeaza cp. indice*/  
    fgets(tabl[k].cheie, LUNG_NUME,fis);  
    depl += sizeof(PERSOANA);  
    fseek(fis, depl, SEEK_SET);  
  }  
  return nr_gasite;  
}
```

```
/*se continua pe slide-ul urmator...*/
```

```
/*continuate...*/
```

```
/*compara 2 elemente de tip INDEX*/
```

```
int compara (const void * a, const void *b)
```

```
{ return strcmp (((INDEX *)a)->cheie,((INDEX *)b)->cheie);  
}
```

```
/*sorteaza tabloul index ce are "nr" elemente cu "QSORT"*/
```

```
void sort_tablou (INDEX tabl[], int nr)
```

```
{ qsort(tabl,nr,sizeof(INDEX),compara);  
}
```

```
/*citeste inf. din fis. original si le scrie pe ecran in ordinea indicata de tabloul index  
sortat conform criteriului stabilit*/
```

```
void scrie_inf_din_fisier (FILE *fis, INDEX tabl[], int val_max_indice)
```

```
{ PERSOANA p;  
  int k;  
  for(k=0; k < val_max_indice; k++)  
    { if (fseek(fis,sizeof(PERSOANA)*tabl[k].index,0))    /* vezi definiție fseek() !!! */  
      exit(1);  
      fread(&p,sizeof(PERSOANA),1,fis);  
      printf("%20s, %d %d %d, %20s \n", p.nume,p.zi,p.luna,p.an,p.loc_nastere);  
    }  
}
```

```
/*se continua pe slide-ul urmator...*/
```

```
/*continuare...*/
```

```
/*functia "main"*/
```

```
#define NR_MAX_COMPONENTE 100
```

```
int main (void)
```

```
{FILE * fisier;
```

```
INDEX tabl[NR_MAX_COMPONENTE];
```

```
int nr_inreg_citite;
```

```
if ((fisier=fopen("exemplu.dat","rb")) ==NULL)
```

```
{printf("Fisierul %s nu poate fi deschis \n", "exemplu.dat");
```

```
exit(1);
```

```
}
```

```
nr_inreg_citite=constr_tablou_index (fisier,tabl,NR_MAX_COMPONENTE);
```

```
sort_tablou(tabl,nr_inreg_citite);
```

```
scrie_inf_din_fisier(fisier,tabl,nr_inreg_citite);
```

```
return 0;
```

```
}
```

NU încercați să învățați pe de rost rezolvarea prezentată ca exemplu!

Important este să o înțelegeți!

Doua probleme propuse

1. Să se scrie programul pentru concatenarea a două sau mai multe fișiere ce conțin numere reale. Se va afișa pe ecran informația din fișierul astfel rezultat.
2. Două firme își păstrează informațiile referitoare la stocul de mărfuri (cod produs, denumire, cantitate, preț unitar) în fișierele "marfuri1.dat" și respectiv "marfuri2.dat", ordonate crescător după cod. Prin fuzionarea celor două firme, rezultă un stoc comun care trebuie memorat în fișierul "marfuri.dat", ordonat tot după cod.
 - a) Să se creeze fișierele inițiale, pe baza datelor introduse de la tastatură și apoi să se creeze fișierul de stoc comun "marfuri.dat". (Pentru mărfuri cu cod comun, se consideră că denumirea și prețul unitar corespund.)
 - b) Fișierul "marfuri.dat" se va parcurge secvențial, tipărind pentru fiecare componentă denumirea și cantitatea.
 - c) Pentru o componentă dorită (specificată prin denumire, de la tastatură), se va modifica **direct în fișier** prețul său unitar.