FIŞIERE

8

9.1. CARACTERISTICILE GENERALE ALE FIŞIERELOR

Noţiunea de **fişier** desemnează o colecţie de informaţii memorată pe un suport permanent (de obicei discuri magnetice), percepută ca un ansamblu, căreia i se asociază un nume (în vederea conservării şi regăsirii ulterioare).

Caracteristicile unui fişier (sub sistem de operare MS-DOS) sunt :
- Dispozitivul logic de memorare (discul);
- Calea (în structura de directoare) unde este memorat fişierul;
- Numele şi extensia;
- Atributele care determină operaţiile care pot fi efectuate asupra fişierului (de exemplu: `R`-read-only - citire; `W`-write-only scriere; `RW`-read-write citire/scriere; `H`-hidden - nu se permite nici măcar vizualizarea; `S`-system - fişiere sistem asupra cărora numai sistemul de operare poate realiza operaţii operaţii, etc.).

Lucrul cu fişiere în programare oferă următoarele avantaje:
- Prelucrarea de unei cantităţi mari de informaţie obţinută din diverse surse cum ar fi execuţia prealabilă a unui alt program;
- Stocarea temporară pe suport permanent a informaţiei în timpul execuţiei unui program pentru a evita supraîncărcarea memoriei de lucru;
- Prelucrarea aceleeaşi colecţii de informaţii prin mai multe programe.

În limbajul C, operaţiile asupra fişierelor se realizează cu ajutorul unor funcţii din biblioteca standard (`stdio.h`). Transferurile cu dipozitivele periferice (tastatură, monitor, disc, imprimantă, etc.) se fac prin intermediul unor dispozitive logice identice numite **stream-uri** (fluxuri) şi prin intermediul sistemului de operare. Un **flux de date** este un fişier sau un dispozitiv fizic tratat printr-un pointer la o structură de tip **FILE** (din header-ul **`stdio.h`**). Când un program este executat, în mod automat, se deschid următoarele fluxuri de date predefinite, dispozitive logice (în stdio.h):
- `stdin` (standard input device) - dispozitivul standard de intrare (tastatura) - ANSII C;
- `stdout` (standard output device) - dispozitivul standard de ieşire (monitorul) - ANSII C;
- `stderr` (standard error output device) - dispozitivul standard de eroare (de obicei un fişier care conţine mesajele de eroare rezultate din execuţia unor funcţii) - ANSII C;
- `stdaux` (standard auxiliary device) - dispozitivul standard auxiliar (de obicei interfaţa serială auxiliară) - specifice MS-DOS;
- `stdprn` (standard printer) - dispozitivul de imprimare - specifice MS-DOS.

În abordarea limbajului C (impusă de **stdio.h**), toate elementele care pot comunica informaţii cu un program sunt percepute - în mod unitar - ca fluxuri de date. Datele introduse de la tastatură formează un ***fişier de intrare (fişierul standard de intrare)***. Datele afişate pe monitor formează un ***fişier de ieşire (fişierul standard de ieşire)***. Sfârşitul oricărui fişier este indicat printr-un marcaj de ***sfârşit de fişier*** (end of file). În cazul fişierului standard de intrare, sfârşitul de fişier se generează prin **`Ctrl`+`Z`** (**`^Z`**) (sub MS-DOS) (sau **`Ctrl`+`D`** sub Linux). Acest caracter poate fi detectat prin folosirea constantei simbolice **`EOF`** (definită în

fişierul *stdio.h*), care are valoarea **-1**. Această valoare nu rămane valabilă pentru fişierele binare, care pot conţine pe o poziţie oarecare caracterul *'\x1A'*.
De obicei, schimbul de informaţii dintre programe şi periferice se realizează folosind zone tampon. O zonă tampon păstrează una sau mai multe înregistrări. Prin ***operaţia de citire***, înregistrarea curentă este transferată de pe suportul extern în zona tampon care îi corespunde, programul având apoi acces la elementele înregistrării din zona tampon. În cazul ***operaţiei de scriere***, înregistrarea se construieşte în zona tampon, prin program, fiind apoi transferată pe suportul extern al fişierului. În cazul monitoarelor, înregistrarea se compune din caracterele unui rând. De obicei, o zonă tampon are lungimea multiplu de 512 octeţi. Orice fişier trebuie ***deschis*** inainte de a fi prelucrat, iar la terminarea prelucrării lui, trebuie ***închis***.

Fluxurile pot fi de tip **text** sau de tip **binar**. Fluxurile de tip text împart fişierele în linii separate prin caracterul '\n' (newline=linie nouă), putând fi citite ca orice fişier text. Fluxurile de tip binar transferă blocuri de octeţi (fără nici o structură), neputând fi citite direct, ca fişierele text.

Prelucrarea fişierelor se poate face la *două niveluri*:
- Nivelul *superior* de prelucrare a fişierelor în care se utilizează funcţiile specializate în prelucrarea fişierelor.
- Nivelul *inferior* de prelucrare a fişierelor în care se utilizează direct facilităţile oferite de sistemul de operare, deoarece, în final, sarcina manipulării fişierelor revine sistemului de operare. Pentru a avea acces la informaţiile despre fişierele cu care lucrează, sistemul de operare foloseşte câte un descriptor (bloc de control) pentru fiecare fişier.

Ca urmare, există două abordări în privinţa lucrului cu fişiere:
- abordarea implementată în **stdio.h**, asociază referinţei la un fişier un **stream** (flux de date), un pointer către o structură **FILE**.
- abordarea definită în header-ul **io.h** (input/output header) asociază referinţei la un fişier un aşa-numit **handle** (în cele ce urmează acesta va fi tradus prin indicator de fişier) care din punct de vedere al tipului de date este **in;**

Scopul lucrului cu fişiere este acela de a prelucra informaţia conţinută. Pentru a putea accesa un fişier va trebui să-l asociem cu unul din cele două modalităţi de manipulare. Acest tip de operaţie se mai numeşte deschidere de fişier. Înainte de a citi sau scrie într-un fişier (neconectat automat programului), fişierul trebuie deschis cu ajutorul funcţiei **fopen** din biblioteca standard. Funcţia primeşte ca argument numele extern al fişierului, negociază cu sistemul de operare şi retunează un nume (identificator) intern care va fi utilizat ulterior la prelucrarea fişireului. Acest identificator intern este un pointer la o structură care conţine informaţii despre fişier (poziţia curentă în buffer, dacă se citeşte sau se scrie în fişier, etc.). Utilizatorii nu trebuie să cunoască detaliile, singura declaraţie necesară fiind cea pentru pointerul de fişier.
Exemplu: `FILE *fp;`

Operaţiile care pot fi realizate asupra fişierelor sunt:
- *deschiderea unui fişier;*
- *scrierea într-un fişier;*
- *citirea dintr-un fişier;*
- *poziţionarea într-un fişier;*
- *închiderea unui fişier.*

9.2. DESCHIDEREA UNUI FIŞIER

- **Funcţia `fopen`**
  Crează un flux de date între fişierul specificat prin numele extern (`nume_fişier`) şi programul C. Parametrul `mod` specifică sensul fluxului de date şi modul de interpretare a acestora. Funcţia returnează un pointer spre tipul FILE, iar în caz de eroare - pointerul *NULL* (prototip în **stdio.h**).
  ```
  FILE *fopen(const char *nume_fişier, const char *mod);
  ```
  Parametrul `mod` este o constantă şir de caractere, care poate conţine caracterele cu semnificaţiile:

- **r** : flux de date de intrare; deschidere pentru citire;
- **w** : flux de date de ieşire; deschidere pentru scriere (crează un fişier nou sau suprascrie conţinutul anterior al fişierului existent);
- **a** : flux de date de ieşire cu scriere la sfârşitul fişierului, adăugare, sau crearea fişierului în cazul în care acesta nu există;
- **+** : extinde un flux de intrare sau ieşire la unul de intrare/ieşire; operaţii de scriere şi citire asupra unui fişier deschis în condiţiile r, w sau a.
- **b** : date binare;
- **t** : date text (modul implicit).

Exemple:

`"r+"` – deschidere pentru modificare (citire şi scriere);
`"w+"` – deschidere pentru modificare (citire şi scriere);
`"rb"` – citire binară;
`"wb"` – scriere binară;
`"r+b"` – citire/scriere binară.

- **Funcţia `freopen` (stdio.h)**

Asociază un nou fişier unui flux de date deja existent, închizând legătura cu vechiul fişier şi încercând să deschidă una nouă, cu fişierul specificat. Funcţia returnează pointerul către fluxul de date specificat, sau `NULL` în caz de eşec (prototip în **stdio.h**).

```
FILE*freopen(const char*nume_fiş,const char*mod,FILE *flux_date);
```

- **Funcţia open**

Deschide fişierul specificat conform cu restricţiile de acces precizate în apel. Returnează un întreg care este un indicator de fişier sau -1 (în caz de eşec) (prototip în **io.h**).

```
int open(const char *nume_fişier, int acces [,int mod]);
```

Restricţiile de acces se precizează prin aplicarea operatorului **|** (disjuncţie logică la nivel de bit) între anumite constante simbolice, definite în **`fcntl.h`**, cum sunt :

O_RDONLY - citire;
O_WRONLY - scriere
O_RDWR - citire şi scriere
O_CREAT - creare
O_APPEND - adăugare la sfârşitul fişierului
O_TEXT - interpretare CR-LF
O_BINARY - nici o interpretare.,

Restricţiile de mod de creare se realizează cu ajutorul constantelor:

S_IREAD - permisiune de citire din fişier
S_IWRITE - permisiune de scriere din fişier, eventual legate prin operatorul "|".

- **Funcţia `creat`**

Crează un fişier nou sau îl suprascrie în cazul în care deja există. Returnează indicatorul de fişier sau -1 (în caz de eşec). Parametrul `un_mod` este obţinut în mod analog celui de la funcţia de deschidere (prototip în **io.h**).

```
int creat(const char *nume_fişier, int un_mod);
```

- **Funcţia `creatnew`**

Crează un fişier nou, conform modului specificat. Returnează indicatorul fişierului nou creat sau rezultat de eroare (-1), dacă fişierul deja există (prototip în **io.h**).

```
int creatnew(const char *nume_fişier, int mod);
```

După cum se observă, informaţia furnizată pentru deschiderea unui fişier este aceeaşi în ambele abordări, diferenţa constând *în tipul de date* al entităţii asociate fişierului. Implementarea din `io.h` oferă un alt tip de control la nivelul comunicării cu echipamentele periferice (furnizat de funcţia **`ioctrl`**), asupra căruia nu vom insista, deoarece desfăşurarea acestui tip de control este mai greoaie, dar mai profundă.

9.3. ÎNCHIDEREA UNUI FIŞIER

- **Funcţia `fclose`**

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

Funcţia închide un fişier deschis cu `fopen` şi eliberează memoria alocată (zona tampon şi structura `FILE`). Returnează valoarea 0 la închiderea cu succes a fişierului şi -1 în caz de eroare (prototip în **stdio.h**).

- **Funcţia `fcloseall`**

```
int fcloseall(void);
```

Închide toate fluxururile de date şi returnează numărul fluxurilor de date închise (prototip în **stdio.h**).

- **Funcţia `close`**

```
int close(int indicator);
```

Închide un indicator de fişier şi returnează 0 (în caz de succes) sau -1 în caz de eroare (prototip în **io.h**).

9.4. PRELUCRAREA FIŞIERELOR TEXT

După deschiderea unui fişier, toate operaţiile asupra fişierului vor fi efectuate cu pointerul său. Operaţiile de citire şi scriere într-un fişier text pot fi:

- intrări/ieşiri la nivel de caracter (de octet);
- intrări/ieşiri la nivel de cuvânt (2 octeţi);
- intrări/ieşiri de şiruri de caractere;
- intrări/ieşiri cu formatare.

Comunicarea de informaţie de la un fişier către un program este asigurată prin ***funcţii de citire*** care transferă o cantitate de octeţi (unitatea de măsură în cazul nostru) din fişier într-o variabilă-program pe care o vom numi buffer, ea însăşi având sensul unei înşiruiri de octeţi prin declaraţia **`void *buf`**. Comunicarea de informaţie de la un program către un fişier este asigurată prin ***funcţii de scriere*** care transferă o cantitate de octeţi dintr-o variabilă-program de tip buffer în fişier.

Fişierele sunt percepute în limbajul C ca fiind, implicit, secvenţiale (informaţia este parcursă succesiv, element cu element). Pentru aceasta, atât fluxurile de date cât şi indicatorii de fişier au asociat un *indicator de poziţie curentă* în cadrul fişierului. Acesta este iniţializat la 0 în momentul deschiderii, iar operaţiile de citire, respectiv scriere, se referă la succesiunea de octeţi care începe cu poziţia curentă. Operarea asupra fiecărui octet din succesiune determină incrementarea indicatorului de poziţie curentă.

9.4.1. PRELUCRAREA UNUI FIŞIER LA NIVEL DE CARACTER

Fişierele pot fi scrise şi citite caracter cu caracter folosind funcţiile **`putc`** (pentru scriere) şi **`getc`** (citire).

- **Funcţia `putc`**

```
int putc (int c, FILE *pf);
```

`c` – este codul ASCII al caracterului care se scrie în fişier;
`pf` – este pointerul spre tipul `FILE` a cărui valoare a fost returnată de funcţia *fopen*.
Funcţia `putc` returnează valoarea lui `c` (valoarea scrisă în caz de succes), sau –1 (`EOF`) în caz de eroare sau sfârşit de fişier.

- **Funcţia `getc`**

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

Funcţia citeşte un caracter dintr-un fişier (pointerul spre tipul `FILE` transmis ca argument) şi returnează caracterul citit sau `EOF` la sfârşit de fişier sau eroare.

Exerciţiu: Să se scrie un program care crează **un fişier text** în care **se vor scrie caracterele** introduse de la tastatură (citite din fişierul standard de intrare), până la întâlnirea caracterului ^Z = Ctrl+Z.

```
#include <stdio.h>
#include <process.h>
void main()
{
int c, i=0; FILE *pfcar;
char mesaj[]="\nIntrodu caractere urmate de Ctrl+Z (Ctrl+D sub Linux):\n";
char eroare[]="\n Eroare deschidere fişier \n";
while(mesaj[i]) putchar(mesaj[i++]);
pfcar=fopen("f_car1.txt","w");       // crearea fişierului cu numele extern f_car1.txt
if(pfcar==NULL)
 {
  i=0;
  while(eroare[i])putc(eroare[i++],stdout);
  exit(1);
 }while((c=getchar())!=EOF)    // sau: while ((c=getc(stdin)) != EOF)
      putc(c,pfcar);           // scrierea caracterului în fişier
fclose(pfcar);                 // închiderea fişierului
}
```

Exerciţiu: Să se scrie un program care citeşte **un fişier text**, caracter cu caracter, şi afişează conţinutul acestuia.

```
#include <stdio.h>
#include <process.h>
void main()
{
int c, i=0;
FILE *pfcar;
char eroare[]="\n Eroare deschidere fişier \n";
pfcar=fopen("f_car1.txt","r");       //deschiderea fişierului numit f_car1.txt în citire
if(pfcar==NULL)
 {
  i=0;
  while(eroare[i])putc(eroare[i++],stdout);
  exit(1);
 } while((c=getc(pfcar))!=EOF)       //citire din fişier, la nivel de caracter
      putc(c,stdout);
      //scrierea caracterului citit în fişierul standard de ieşire (afişare pe monitor)
fclose(pfcar);
}
```

9.4.2. PRELUCRAREA UNUI FIŞIER LA NIVEL DE CUVÂNT

Funcţiile **putw** şi **getw** (putword şi getword) sunt echivalente cu funcţiile putc şi getc, cu diferenţa că unitatea transferată nu este un singur octet (caracter), ci un cuvânt (un int).

```
int getw(FILE *pf);
int putc (int w, FILE *pf);
```

Se recomandă utilizarea funcţiei feof pentru a testa întâlnirea sfârşitului de fişier.

Exemplu:

```
int tab[100];
FILE *pf;
// . . . deschidere fişier
while (!feof(pf)){
      for (int i=0; i<100; i++){
            if (feof(pf))
                  break;
```

```
                tab[i]=getw(pf);
                //citire din fişier la nivel de cuvânt şi memorare în vectorul tab
                //...
        }
    }
    printf("Sfarşit de fişier\n");
```

9.4.3. PRELUCRAREA UNUI FIŞIER LA NIVEL DE ŞIR DE CARACTERE

Într-un fişier text, liniile sunt considerate ca linii de text separate de sfârşitul de linie (`'\n'`), iar în memorie, ele devin şiruri de caractere terminate de caracterul nul (`'\0'`). Citirea unei linii de text dintr-un fişier se realizează cu ajutorul funcţiei **fgets**, iar scrierea într-un fişier - cu ajutorul funcţiei **fputs**.

Funcţia `fgets` este indentică cu funcţia `gets`, cu deosebirea că funcţia gets citeşte din fişierul standard de intrare (`stdin`). Funcţia `fputs` este indentică cu funcţia `puts`, cu deosebirea funcţia puts scrie în fişierul standard de ieşire (`stdout`).

- **Funcţia `fputs`**

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

Funcţia scrie un şir de caractere într-un fişier şi primeşte ca argumente pointerul spre zona de memorie (buffer-ul) care conţine şirul de caractere (`s`) şi pointerul spre structura `FILE`. Funcţia returnează ultimul caracter scris, în caz de succes, sau -1 în caz de eroare.

- **Funcţia `fgets`**

```
char *fgets(char *s, int dim, FILE *pf);
```

Funcţia citeşte maximum `dim-1` octeţi (caractere) din fişier, sau până la întâlnirea sfarşitului de linie. Pointerul spre zona în care se face citirea caracterelor este `s`. Terminatorul null (`'\0'`) este plasat automat la sfârşitul şirului (buffer-lui de memorie). Funcţia returnează un pointer către buffer-ul în care este memorat şirul de caractere, în caz de succes, sau pointerul `NULL` în cazul eşecului.

Exerciţiu: Să se scrie un program care crează un fişier text în care se vor scrie şirurile de caractere introduse de la tastatură.

```
#include <stdio.h>
void main()
{
int n=250;  FILE *pfsir;
char mesaj[]="\nIntrodu siruri car.urmate de Ctrl+Z(Ctrl+D sub Linux):\n";
char sir[250],*psir;    fputs(mesaj,stdout);
pfsir=fopen("f_sir.txt","w"); //deschiderea fişierului f_şir.txt pentru scriere
psir=fgets(sir,n,stdin);              // citirea şirurilor din fişierul standard de intrare
while(psir!=NULL)
 {
  fputs(sir,pfsir);            // scrierea în fişierul text
  psir=fgets(sir,n,stdin);
 }
fclose(pfsir);
}
```

Exerciţu: Să se scrie un program care citeşte un fişier text, linie cu linie, şi afişează conţinutul acestuia

```
#include <stdio.h>
void main()
{
int n=250; FILE *pfsir; char sir[250],*psir;
pfsir=fopen("f_sir.txt","r"); psir=fgets(sir,n,pfsir);
while(psir!=NULL)
 {
```

```
        fputs(sir,stdout);     //sau: puts(sir);
             //afişarea (scrierea în fişierul standard de ieşire) şirului (liniei) citit din fişierul text
        psir=fgets(sir,n,pfsir);      //citirea unei linii de text din fişier
       }
     fclose(pfsir);}
```

9.4.4. INTRĂRI/IEŞIRI FORMATATE

Operaţiile de intrare/ieşire formatate permit citirea, respectiv scrierea într-un fişier text, impunând un anumit format. Se utilizează funcţiile **`fscanf`** şi **`fprintf`**, similare funcţiilor `scanf` şi `printf` (care permit citirea/scrierea formatată de la tastatură/monitor).

- Funcţia **`fscanf`**

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

- Funcţia **`fprintf`**

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

Funcţiile primesc ca parametri ficşi pointerul (`pf`) spre tipul `FILE` (cu valoarea atribuită la apelul funcţiei `fopen`), şi specificatorul de format (cu structură identică celui prezentat pentru funcţiile `printf` şi `scanf`). Funcţiile returnează numărul câmpurilor citite/scrise în fişier, sau -1 (`EOF`) în cazul detectării sfârşitului fişierului sau al unei erori.

9.5. INTRĂRI/IEŞIRI BINARE

Reamintim că fluxurile de tip binar transferă blocuri de octeţi (fără nici o structură), neputând fi citite direct, ca fişierele text (vezi paragraful 9.1.). Comunicarea de informaţie dintre un program şi un fişier este asigurată prin funcţii de citire/scriere care transferă un număr de octeţi, *prin intermediul unui buffer*.

Funcţiile de citire

- **Funcţia `fread`**

  Citeşte date dintr-un flux, sub forma a `n` blocuri (entităţi), fiecare bloc având dimensiunea `dim`, într-un buffer (`buf`). Returnează numărul de blocuri citite efectiv, sau -1 în caz de eroare (prototip în **stdio.h**).

```
size_t fread(void *buf, size_t dim, size_t n, FILE *flux_date);
```

- **Funcţia `read`**

  Citeşte dintr-un fişier (precizat prin indicatorul său, `indicator`) un număr de `n` octeţi şi îi memorează în bufferul `buf`. Funcţia returnează numărul de octeţi citiţi efectiv (pentru fişierele deschise în mod text nu se numără simbolurile de sfirşit de linie), sau -1 în caz de eroare (prototip în **io.h**).

```
int read(int indicator, void *buf, unsigned n);
```

Funcţiile de scriere

Fişierele organizate ca date binare pot fi prelucrate cu ajutorul funcţiilor *fread* şi *fwrite*. În acest caz, se consideră că înregistrarea este o *colecţie de date structurate* numite **articole**. La o citire se transferă într-o zonă specială, numită **zona tampon**, un număr de articole care se presupune că au o lungime fixă.

- **Funcţia `fwrite`**

  Scrie informaţia (preluată din buffer, `buf` este pointerul spre zona tampon care conţine articolele citite) într-un flux de date, sub forma a `n` entităţi de dimensiune `dim`. Returnează numărul de entităţi scrise efectiv, sau -1 în caz de eroare (prototip în **stdio.h**).

```
size_t fwrite(const void *buf, size_t dim, size_t n, FILE *flx_date);
```

- **Funcţia `write`**

  Scrie într-un fişier (desemnat prin indicatorul său, `indicator`) un număr de `n` octeţi preluaţi dintr-un buffer (`buf` este pointerul spre acesta). Returnează numărul de octeţi scrişi efectiv sau -1 în caz de

eroare (prototip în **io.h**).

```
int write(int indicator, void *buf, unsigned n);
```

Exercițu: Să se scrie un program care crează un fişier binar în care se vor introduce numere reale, nenule.

```
#include <iostream.h>
#include <stdio.h>
int main()
 { FILE *f; double nr;  int x;
  if ((f= fopen("test_nrb.dat", "wb")) == NULL) //deschidere flux binar, scriere
   {      cout<<"\nNu se poate deschide fişierul test_nrb.dat"<<'\n';
      return 1;
   }
   cout<<"\nIntroduceţi numere(diferite de 0) terminate cu un 0:"<<'\n';
   cin>>nr;
   while(nr!=0)
   {
    x=fwrite(&nr, sizeof(nr), 1, f);       //scriere în fişier
    cin>>nr;
   }
   fclose(f);
   return 0;
}
```

Exemplu: Să se scrie un program ce citeşte dintr-un fişier binar numere reale, nenule.

```
#include <iostream.h>
#include <stdio.h>
int main()
 { FILE *f; double buf;
  if ((f= fopen("test_nrb.dat", "rb")) == NULL)
   {
      cout<<"\nNu se poate deschide fişierul test_nrb.dat"<<'\n';
      return 1;
   }
   cout<<"\nNumerele nenule citite din fişier sunt:"<<'\n';
   while((fread(&buf, sizeof(buf), 1, f))==1)
            // funcţia sizeof(buf) care returneaza numarul de octeţi necesari variabilei buf.
       cout<<buf<<"  ";
   fclose(f);
   cout<<'\n';
   return 0;
}
```