

FIȘIERE

- | | |
|--|--|
| <ul style="list-style-type: none"> 8.1. Caracteristicile generale ale fișierelor 8.2. Deschiderea unui fișier 8.3. Închiderea unui fișier 8.4. Prelucrarea fișierelor text <ul style="list-style-type: none"> 8.4.1. Prelucrarea la nivel de caracter 8.4.2. Prelucrarea la nivel de cuvânt | <ul style="list-style-type: none"> 8.4.3. Prelucrarea la nivel de șir de caractere 8.4.4. Intrări/ieșiri formate 8.5. Intrări/ieșiri binare 8.6. Poziționarea într-un fișier 8.7. Funcții utilitare pentru lucrul cu fișiere 8.8. Alte operații cu fișiere |
|--|--|

8.1. CARACTERISTICILE GENERALE ALE FIȘIERELOR

Noțiunea de **fișier** desemnează o colecție de informații memorată pe un suport permanent (de obicei discuri magnetice), percepută ca un ansamblu, căreia i se asociază un nume (în vederea conservării și regăsirii ulterioare).

Caracteristicile unui fișier (sub sistem de operare MS-DOS) sunt :

- ❑ Dispozitivul logic de memorare (discul);
- ❑ Calea (în structura de directoare) unde este memorat fișierul;
- ❑ Numele și extensia;
- ❑ Atributele care determină operațiile care pot fi efectuate asupra fișierului (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 - fișiere sistem asupra cărora numai sistemul de operare poate realiza operații operații, etc.).

Lucrul cu fișiere în programare oferă următoarele avantaje:

- ❑ Prelucrarea de unei cantități mari de informație obținută din diverse surse cum ar fi execuția prealabilă a unui alt program;
- ❑ Stocarea temporară pe suport permanent a informației în timpul execuției unui program pentru a evita supraîncărcarea memoriei de lucru;
- ❑ Prelucrarea aceleiași colecții de informații prin mai multe programe.

În limbajul C, operațiile asupra fișierelor se realizează cu ajutorul unor funcții din biblioteca standard (`stdio.h`). Transferurile cu dispozitivele periferice (tastatură, monitor, disc, imprimantă, etc.) se fac prin intermediul unor dispozitive logice identice numite **stream-uri** (fluxuri) și prin intermediul sistemului de operare. Un **flux de date** este un fișier 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 ieșire (monitorul) - ANSII C;
- ❑ `stderr` (standard error output device) - dispozitivul standard de eroare (de obicei un fișier care conține mesajele de eroare rezultate din execuția unor funcții) - ANSII C;
- ❑ `stdaux` (standard auxiliary device) - dispozitivul standard auxiliar (de obicei interfața 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 informații cu un program sunt percepute - în mod unitar - ca fluxuri de date. Datele introduse de la tastatură formează un **fișier de intrare (fișierul standard de intrare)**. Datele afișate pe monitor formează un **fișier de ieșire (fișierul standard de ieșire)**. Sfârșitul oricărui fișier este indicat printr-un marcaj de **sfârșit de fișier** (end of file). În cazul fișierului standard de intrare, sfârșitul de fișier 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

fișierul *stdio.h*), care are valoarea **-1**. Această valoare nu rămâne valabilă pentru fișierele binare, care pot conține pe o poziție oarecare caracterul `'\x1A'`.

De obicei, schimbul de informații dintre programe și periferice se realizează folosind zone tampon. O zonă tampon păstrează una sau mai multe înregistrări. Prin **operația 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 **operației de scriere**, înregistrarea se construiește în zona tampon, prin program, fiind apoi transferată pe suportul extern al fișierului. În cazul monitoarelor, înregistrarea se compune din caracterele unui rând. De obicei, o zonă tampon are lungimea multiplu de 512 octeți. Orice fișier trebuie **deschis** înainte 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 fișierele în linii separate prin caracterul `'\n'` (newline=linie nouă), putând fi citite ca orice fișier text. Fluxurile de tip binar transferă blocuri de octeți (fără nici o structură), neputând fi citite direct, ca fișierele text.

Prelucrarea fișierelor se poate face la *două niveluri*:

- ❑ Nivelul *superior* de prelucrare a fișierelor în care se utilizează funcțiile specializate în prelucrarea fișierelor.
- ❑ Nivelul *inferior* de prelucrare a fișierelor în care se utilizează direct facilitățile oferite de sistemul de operare, deoarece, în final, sarcina manipulării fișierelor revine sistemului de operare. Pentru a avea acces la informațiile despre fișierele cu care lucrează, sistemul de operare folosește câte un descriptor (bloc de control) pentru fiecare fișier.

Ca urmare, există două abordări în privința lucrului cu fișiere:

- ❑ abordarea implementată în **stdio.h**, asociază referinței la un fișier un **stream** (flux de date), un pointer către o structură **FILE**.
- ❑ abordarea definită în header-ul **io.h** (input/output header) asociază referinței la un fișier un așa-numit **handle** (în cele ce urmează acesta va fi tradus prin indicator de fișier) care din punct de vedere al tipului de date este **in**;

Scopul lucrului cu fișiere este acela de a prelucra informația conținută. Pentru a putea accesa un fișier va trebui să-l asociem cu unul din cele două modalități de manipulare. Acest tip de operație se mai numește deschidere de fișier. Înainte de a citi sau scrie într-un fișier (neconectat automat programului), fișierul trebuie deschis cu ajutorul funcției **fopen** din biblioteca standard. Funcția primește ca argument numele extern al fișierului, negociază cu sistemul de operare și returnează un nume (identificator) intern care va fi utilizat ulterior la prelucrarea fișierului. Acest identificator intern este un pointer la o structură care conține informații despre fișier (poziția curentă în buffer, dacă se citește sau se scrie în fișier, etc.). Utilizatorii nu trebuie să cunoască detaliile, singura declarație necesară fiind cea pentru pointerul de fișier.

Exemplu: `FILE *fp;`

Operațiile care pot fi realizate asupra fișierelor sunt:

- ❑ *deschiderea unui fișier;*
- ❑ *scrierea într-un fișier;*
- ❑ *citirea dintr-un fișier;*
- ❑ *poziționarea într-un fișier;*
- ❑ *închiderea unui fișier.*

8.2. DESCHIDEREA UNUI FIȘIER

❑ Funcția **fopen**

Crează un flux de date între fișierul specificat prin numele extern (`nume_fișier`) și programul C. Parametrul `mod` specifică sensul fluxului de date și modul de interpretare a acestora. Funcția returnează un pointer spre tipul **FILE**, iar în caz de eroare - pointerul **NULL** (prototip în **stdio.h**).

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

Parametrul `mod` este o constantă șir de caractere, care poate conține caracterele cu semnificațiile:

- **r** : flux de date de intrare; deschidere pentru citire;
- **w** : flux de date de ieșire; deschidere pentru scriere (crează un fișier nou sau suprascrie conținutul anterior al fișierului existent);
- **a** : flux de date de ieșire cu scriere la sfârșitul fișierului, adăugare, sau crearea fișierului în cazul în care acesta nu există;
- **+** : extinde un flux de intrare sau ieșire la unul de intrare/ieșire; operații de scriere și citire asupra unui fișier deschis în condițiile r, w sau a.
- **b** : date binare;
- **t** : date text (modul implicit).

Exemple:

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

□ **Funcția freopen (stdio.h)**

Asociază un nou fișier unui flux de date deja existent, închizând legătura cu vechiul fișier și încercând să deschidă una nouă, cu fișierul specificat. Funcția returnează pointerul către fluxul de date specificat, sau NULL în caz de eșec (prototip în **stdio.h**).

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

□ **Funcția open**

Deschide fișierul specificat conform cu restricțiile de acces precizate în apel. Returnează un întreg care este un indicator de fișier sau -1 (în caz de eșec) (prototip în **io.h**).

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

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

```
O_RDONLY – citire;
O_WRONLY – scriere
O_RDWR – citire și scriere
O_CREAT – creare
O_APPEND – adăugare la sfârșitul fișierului
O_TEXT – interpretare CR-LF
O_BINARY – nici o interpretare.,
```

Restricțiile de mod de creare se realizează cu ajutorul constantelor:

```
S_IREAD – permisiune de citire din fișier
S_IWRITE – permisiune de scriere din fișier, eventual legate prin operatorul “|”.
```

□ **Funcția creat**

Crează un fișier nou sau îl suprascrie în cazul în care deja există. Returnează indicatorul de fișier sau -1 (în caz de eșec). Parametrul **un_mod** este obținut în mod analog celui de la funcția de deschidere (prototip în **io.h**).

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

□ **Funcția creatnew**

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

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

După cum se observă, informația furnizată pentru deschiderea unui fișier este aceeași în ambele abordări, diferența constând în *tipul de date* al entității asociate fișierului. Implementarea din **io.h** oferă un alt tip de control la nivelul comunicării cu echipamentele periferice (furnizat de funcția **ioctl**), asupra căruia nu vom insista, deoarece desfășurarea acestui tip de control este mai greoaie, dar mai profundă.

8.3. ÎNCHIDEREA UNUI FIȘIER

❑ Funcția `fclose`

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

Funcția închide un fișier deschis cu `fopen` și eliberează memoria alocată (zona tampon și structura `FILE`). Returnează valoarea 0 la închiderea cu succes a fișierului și -1 în caz de eroare (prototip în `stdio.h`).

❑ Funcția `fcloseall`

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

Închide toate fluxurile de date și returnează numărul fluxurilor de date închise (prototip în `stdio.h`).

❑ Funcția `close`

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

Închide un indicator de fișier și returnează 0 (în caz de succes) sau -1 în caz de eroare (prototip în `io.h`).

8.4. PRELUCRAREA FIȘIERELOR TEXT

După deschiderea unui fișier, toate operațiile asupra fișierului vor fi efectuate cu pointerul său. Operațiile de citire și scriere într-un fișier text pot fi:

- ❑ intrări/ieșiri la nivel de caracter (de octet);
- ❑ intrări/ieșiri la nivel de cuvânt (2 octeți);
- ❑ intrări/ieșiri de șiruri de caractere;
- ❑ intrări/ieșiri cu formatare.

Comunicarea de informație de la un fișier către un program este asigurată prin *funcții de citire* care transferă o cantitate de octeți (unitatea de măsură în cazul nostru) din fișier într-o variabilă-program pe care o vom numi buffer, ea însăși având sensul unei înșiruri de octeți prin declarația `void *buf`. Comunicarea de informație de la un program către un fișier este asigurată prin *funcții de scriere* care transferă o cantitate de octeți dintr-o variabilă-program de tip buffer în fișier.

Fișierele sunt percepute în limbajul C ca fiind, implicit, secvențiale (informația este parcursă succesiv, element cu element). Pentru aceasta, atât fluxurile de date cât și indicatorii de fișier au asociat un *indicator de poziție curentă* în cadrul fișierului. Acesta este inițializat la 0 în momentul deschiderii, iar operațiile de citire, respectiv scriere, se referă la succesiunea de octeți care începe cu poziția curentă. Operarea asupra fiecărui octet din succesiune determină incrementarea indicatorului de poziție curentă.

8.4.1. PRELUCRAREA UNUI FIȘIER LA NIVEL DE CARACTER

Fișierele pot fi scrise și citite caracter cu caracter folosind funcțiile `putc` (pentru scriere) și `getc` (citire).

❑ Funcția `putc`

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

`c` – este codul ASCII al caracterului care se scrie în fișier;

`pf` – este pointerul spre tipul `FILE` a cărui valoare a fost returnată de funcția `fopen`.

Funcția `putc` returnează valoarea lui `c` (valoarea scrisă în caz de succes), sau -1 (EOF) în caz de eroare sau sfârșit de fișier.

❑ Funcția `getc`

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

Funcția citește un caracter dintr-un fișier (pointerul spre tipul `FILE` transmis ca argument) și returnează caracterul citit sau EOF la sfârșit de fișier sau eroare.

Exercițiu: Să se scrie un program care crează **un fișier text** în care **se vor scrie caracterele** introduse de la tastatură (citite din fișierul standard de intrare), până la întâlnirea caracterului $\text{^Z} = \text{Ctrl+Z}$.

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

Exercițiu: Să se scrie un program care citește **un fișier text**, caracter cu caracter, și afișează conținutul acestuia.

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

8.4.2. PRELUCRAREA UNUI FIȘIER LA NIVEL DE CUVÂNT

Funcțiile **putw** și **getw** (putword și getword) sunt echivalente cu funcțiile **putc** și **getc**, cu diferența 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 funcției **feof** pentru a testa întâlnirea sfârșitului de fișier.

Exemplu:

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

```

        tab[i]=getw(pf);
        //citire din fișier la nivel de cuvânt și memorare în vectorul tab
        //...
    }
}
printf("Sfarșit de fișier\n");

```

8.4.3. PRELUCRAREA UNUI FIȘIER LA NIVEL DE ȘIR DE CARACTERE

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

Funcția **fgets** este identică cu funcția **gets**, cu deosebirea că funcția **gets** citește din fișierul standard de intrare (**stdin**). Funcția **fputs** este identică cu funcția **puts**, cu deosebirea că funcția **puts** scrie în fișierul standard de ieșire (**stdout**).

□ Funcția **fputs**

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

Funcția scrie un șir de caractere într-un fișier și primește ca argumente pointerul spre zona de memorie (buffer-ul) care conține șirul de caractere (**s**) și pointerul spre structura **FILE**. Funcția returnează ultimul caracter scris, în caz de succes, sau **-1** în caz de eroare.

□ Funcția **fgets**

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

Funcția citește maximum **dim-1** octeți (caractere) din fișier, sau până la întâlnirea sfârșitului de linie. Pointerul spre zona în care se face citirea caracterelor este **s**. Terminatorul null ('`\0`') este plasat automat la sfârșitul șirului (buffer-ului de memorie). Funcția returnează un pointer către buffer-ul în care este memorat șirul de caractere, în caz de succes, sau pointerul **NULL** în cazul eșecului.

Exercițiu: Să se scrie un program care crează un fișier text în care se vor scrie șirurile 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 fișierului f_sir.txt pentru scriere
    psir=fgets(sir,n,stdin); // citirea șirurilor din fișierul standard de intrare
    while(psir!=NULL)
    {
        fputs(sir,pfsir); // scrierea în fișierul text
        psir=fgets(sir,n,stdin);
    }
    fclose(pfsir);
}

```

Exercițiu: Să se scrie un program care citește un fișier text, linie cu linie, și afișează conținutul 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);
//afișarea (scrierea în fișierul standard de ieșire) șirului (liniei) citit din fișierul text
psir=fgets(sir, n, pfsir); //citirea unei linii de text din fișier
}
fclose(pfsir);

```

8.4.4. INTRĂRI/IEȘIRI FORMATATE

Operațiile de intrare/ieșire formate permit citirea, respectiv scrierea într-un fișier text, impunând un anumit format. Se utilizează funcțiile **fscanf** și **fprintf**, similare funcțiilor `scanf` și `printf` (care permit citirea/scrierea formatată de la tastatură/monitor).

□ Funcția **fscanf**

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

□ Funcția **fprintf**

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

Funcțiile primesc ca parametri ficși pointerul (`pf`) spre tipul `FILE` (cu valoarea atribuită la apelul funcției `fopen`), și specificatorul de format (cu structură identică celui prezentat pentru funcțiile `printf` și `scanf`). Funcțiile returnează numărul câmpurilor citite/scrise în fișier, sau -1 (EOF) în cazul detectării sfârșitului fișierului sau al unei erori.

8.5. INTRĂRI/IEȘIRI BINARE

Reamintim că fluxurile de tip binar transferă blocuri de octeți (fără nici o structură), neputând fi citite direct, ca fișierele text (vezi paragraful 8.1.). Comunicarea de informație dintre un program și un fișier este asigurată prin funcții de citire/scriere care transferă un număr de octeți, *prin intermediul unui buffer*.

Funcțiile de citire

□ Funcția **fread**

Citește date dintr-un flux, sub forma a `n` blocuri (entități), 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);
```

□ Funcția **read**

Citește dintr-un fișier (precizat prin indicatorul său, `indicator`) un număr de `n` octeți și îi memorează în bufferul `buf`. Funcția returnează numărul de octeți citați efectiv (pentru fișierele deschise în mod text nu se numără simbolurile de sfârșit de linie), sau -1 în caz de eroare (prototip în **io.h**).

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

Funcțiile de scriere

Fișierele organizate ca date binare pot fi prelucrate cu ajutorul funcțiilor *fread* și *fwrite*. În acest caz, se consideră că înregistrarea este o *colecție 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.

□ Funcția **fwrite**

Scrie informația (preluată din buffer, `buf` este pointerul spre zona tampon care conține articolele citite) într-un flux de date, sub forma a `n` entități de dimensiune `dim`. Returnează numărul de entități 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);
```

□ Funcția **write**

Scrie într-un fișier (desemnat prin indicatorul său, `indicator`) un număr de `n` octeți preluați dintr-un buffer (`buf` este pointerul spre acesta). Returnează numărul de octeți scriși efectiv sau -1 în caz de

eroare (prototip în **io.h**).

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

Exercitiu: Să se scrie un program care crează un fișier 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 fișierul test_nrb.dat"<<"\n";
    return 1;
  }
  cout<<"\nIntroduceți numere(diferite de 0) terminate cu un 0:"<<"\n";
  cin>>nr;
  while(nr!=0)
  {
    x=fwrite(&nr, sizeof(nr), 1, f); //scriere în fișier
    cin>>nr;
  }
  fclose(f);
  return 0;
}
```

Exemplu: Să se scrie un program ce citește dintr-un fișier 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 fișierul test_nrb.dat"<<"\n";
    return 1;
  }
  cout<<"\nNumerele nenule citite din fișier sunt:"<<"\n";
  while((fread(&buf, sizeof(buf), 1, f))==1)
    // funcția sizeof(buf) care returneaza numarul de octeți necesari variabilei buf.
    cout<<buf<<" ";
  fclose(f);
  cout<<"\n";
  return 0;
}
```

8.6. POZIȚIONAREA ÎNTR-UN FIȘIER

Pe lângă mecanismul de poziționare implicit (asigurat prin operațiile de citire și scriere) se pot folosi și operațiile de poziționare explicită.

□ Funcția **fseek**

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

Funcția deplasează capul de citire/scriere al discului, în vederea prelucrării înregistrărilor fișierului într-o ordine oarecare. Funcția setează poziția curentă în fluxul de date la *n* octeți față de referință):

deplasament – definește numărul de octeți peste care se va deplasa capul discului;

referința – poate avea una din valorile:

0 - începutul fișierului (**SEEK_SET**);

1 - poziția curentă a capului (**SEEK_CUR**);

2 - sfârșitul fișierului (**SEEK_END**).

Funcția returnează valoarea zero la poziționarea corectă și o valoare diferită de zero în caz de eroare (prototip în **stdio.h**).

□ **Funcția lseek**

```
int lseek(int indicator, long n, int referinta);
```

Setează poziția curentă de citire/scriere în fișier la *n* octeți față de referință. Returnează valoarea 0 în caz de succes și diferită de zero în caz de eroare (prototip în **io.h**).

□ **Funcția fgetpos**

```
int fgetpos(FILE *flux_date, fpos_t *poziție);
```

Determină poziția curentă (pointer către o structură, *fpos_t*, care descrie această poziție în fluxul de date). Înscrie valoarea indicatorului în variabila indicată de *poziție*. Returnează 0 la determinarea cu succes a acestei poziții sau valoare diferită de zero în caz de eșec. Structura care descrie poziția poate fi transmisă ca argument funcției *fsetpos* (prototip în **stdio.h**).

□ **Funcția fsetpos**

```
int fsetpos(FILE *flux_date, const fpos_t *poziție);
```

Setează poziția curentă în fluxul de date (atribuie indicatorului valoarea variabilei indicate *poziție*), la o valoare obținută printr apelul funcției *fgetpos*. Returnează valoarea 0 în caz de succes, sau diferită de 0 în caz de eșec (prototip în **stdio.h**).

Există funcții pentru modificarea valorii indicatorului de poziție și de determinare a poziției curente a acestuia.

□ **Funcția ftell**

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

Indică poziția curentă a capului de citire în fișier. Funcția returnează o valoare de tip *long int* care reprezintă poziția curentă în fluxul de date (deplasamentul în octeți a poziției capului față de începutul fișierului) sau -1L în caz de eroare (prototip în **stdio.h**).

□ **Funcția tell**

```
long tell(int indicator);
```

Returnează poziția curentă a capului de citire/scriere în fișier (exprimată în număr de octeți față de începutul fișierului), sau -1L în caz de eroare (prototip în **io.h**).

□ **Funcția rewind**

```
void rewind(FILE *flux_date);
```

Poziționează indicatorul la începutul fluxului de date specificat ca argument (prototip în **stdio.h**).

8.7. FUNCȚII UTILITARE PENTRU LUCRUL CU FIȘIERE

Funcții *de testare a sfârșitului de fișier*

□ **Funcția feof**

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

Returnează o valoare diferită de zero în cazul întâlnirii sfârșitului de fișier sau 0 în celelalte cazuri (prototip în **stdio.h**).

□ **Funcția eof**

```
int eof(int indicator);
```

Returnează valoarea 1 dacă poziția curentă este sfârșitul de fișier, 0 dacă indicatorul este poziționat în altă parte, sau -1 în caz de eroare (prototip în **io.h**).

Funcții *de golire a fluxurilor de date*

□ **Funcția fflush**

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

Golește un fluxul de date specificat ca argument. Returnează 0 în caz de succes și -1 (EOF) în caz de eroare (prototip în **stdio.h**).

□ **Funcția flushall**

```
int flushall(void);
```

Golește toate fluxurile de date existente, pentru cele de scriere efectuând și scrierea în fișiere. Returnează numărul de fluxuri asupra cărora s-a efectuat operația (prototip în **stdio.h**).

8.8. ALTE OPERAȚII CU FIȘIERE

Funcții care *permit operații ale sistemului de operare asupra fișierelor*

□ **Funcția remove**

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

Șterge un fișier. Returnează valoarea 0 pentru operație reușită și -1 pentru operație eșuată (prototip în **stdio.h**).

□ **Funcția rename**

```
int rename(const char *nume_vechi, const char *nume_nou);
```

Redenumeste un fișier. Returnează 0 pentru operație reușită și -1 în cazul eșecului (prototip în **stdio.h**).

□ **Funcția unlink**

```
int unlink(const char *nume_fișier);
```

Șterge un fișier. Returnează 0 la operație reușită și -1 la eșec; dacă fișierul are permisiune read-only, funcția nu va reuși operația (prototip în **io.h, stdio.h**).

Funcții care *permit manipularea aceluiași fișier prin două indicatoare de fișier independente*

□ **Funcția dup**

```
int dup(int indicator);
```

Duplică un indicator de fișier. Returnează noul indicator de fișier pentru operație reușită sau -1 în cazul eșecului (prototip în **io.h**).

□ **Funcția dup2**

```
int dup2(int indicator_vechi, int indicator_nou);
```

Duplică un indicator de fișier la valoarea unui indicator de fișier deja existent. Returnează 0 în caz de succes și -1 în caz de eșec (prototip în **io.h**).

Funcții pentru *aflarea sau modificarea dimensiunii în octeți a fișierelor*

□ **Funcția chsize**

```
int chsize(int indicator, long lungime);
```

Modifică dimensiunea unui fișier, conform argumentului `lungime`. Returnează 0 pentru operație reușită sau -1 în caz de eșec (prototip în **stdio.h**).

□ **Funcția filelength**

```
long filelength(int indicator);
```

Returnează lungimea unui fișier (în octeți) sau -1 în caz de eroare (prototip în **io.h**).

Funcții de *lucru cu fișiere temporare* care oferă facilități de lucru cu fișiere temporare prin generarea de nume unice de fișier în zona de lucru.

□ **Funcția tmpfile**

```
FILE *tmpfile(void);
```

Deschide un fișier temporar, ca flux de date, în mod binar (w+b). Returnează pointerul către fișierul deschis în cazul operației reușite, sau NULL în caz de eșec (prototip în **stdio.h**).

□ **Funcția tmpnam**

```
char *tmpnam(char *sptr);
```

Crează un nume unic pentru fișierul temporar (prototip în **stdio.h**).

□ **Funcția creattemp**

```
int creattemp(char *cale, int attrib);
```

Crează un fișier unic ca nume, cu atributele specificate în argumentul `attrib` (prin `_fmode`, `O_TEXT` sau `O_BINARY`), în directorul dat în argumentul `cale`. Returnează indicatorul (handler-ul) către fișierul creat sau -1 (și setarea `errno`) în cazul eșecului (prototip în **io.h**).

Exemplu: Să se creeze un fișier binar, care va conține informațiile despre angajații unei întreprinderi: nume, marca, salariu. Să se afișeze apoi conținutul fișierului.

```
#include<iostream.h>
#include <stdio.h>
#include <ctype.h>
typedef struct
    { char nume[20];int marca;double salariu;
    }angajat;
union
{angajat a;char sbinar[sizeof(angajat)];}buffer;

int main()
{angajat a; FILE *pf; char cont;char *nume_fis;
cout<<"Nume fisier care va fi creat:"; cin>>nume_fis;
    if ((pf= fopen(nume_fis, "wb")) == NULL)
        { cout<<"\nEroare creare fișier "<<nume_fis<<"!\n";
          return 1;    }
do
    {cout<<"Marca : ";cin>>a.marca;
      cout<<"Nume : ";cin>>a.nume;
      cout<<"Salariu : ";cin>>a.salariu;
      buffer.a=a;
      fwrite(buffer.sbinar,1,sizeof(angajat),pf);
      cout<<"Continuati introducerea de date (d/n) ?";
      cin>>cont;
    } while(toupper(cont)!='N');
fclose(pf);
//citirea informatiilor
    if ((pf= fopen(nume_fis, "rb")) == NULL)
        { cout<<"\nEroare citire fișier "<<nume_fis<<"!\n";
          return 1;    }
for(;;)
    {
      fread(buffer.sbinar,1,sizeof(a),pf);
      a=buffer.a;
      if(feof(pf)) exit(1);
      cout<<" Marca : "<<a.marca;
      cout<<" Numele : "<<a.nume<<"\n";
      cout<<" Salariul : "<<a.salariu<<"\n";
    }
fclose(pf);
}
```

Exemplu: Aplicație pentru gestiunea materialelor dintr-un depozit. Aplicația va avea un meniu principal și va permite gestiunea următoarelor informații: codul materialului (va fi chiar "numărul de ordine"), denumirea acestuia, unitatea de măsură, prețul unitar, cantitatea contractată și cea recepționată (vectori cu 4 elemente). Memorarea datelor se va face într-un fișier de date (un fișier binar cu structuri), numit "material.dat". Aplicația conține următoarele funcții:

1. `help()` - informare privind opțiunile programului
2. Funcții pentru fișierele binare, care să suplinească lipsa funcțiilor standard pentru organizarea directă a fișierelor binare:
 - `citireb()` - citire în acces direct din fișier;

scrieb() - scriere în acces direct în fișier;
citmat() - citirea de la terminal a informațiilor despre un material;
afismat() - afișarea informațiilor despre un material (apelată de list);
lungfisis() - determinarea lungimii fișierului existent;
crefis() - creare fișier.

3. Funcții pentru adaugarea, modificarea, ștergerea și listarea de materiale.

```
#include <process.h>
#include <iostream.h>
#include <stdio.h>
#include <ctype.h>
typedef struct material
{ int codm,stoc,cant_c[4],cant_r[4];
  char den_mat[20],unit_mas[4];
  float preț;
};
material mat;
FILE *pf;
void crefis(),adaug(),modif(),sterg(),list(),help();
void main()
{
  char opțiune;
  do //afișarea unui meniu de opțiuni și selecția opțiunii
  {
    cout<<'\n'<<"Opțiunea Dvs. de lucru este"<<'\n'
      <<"(c|m|s|l|h pentru help) : ";
    cin>>opțiune;
    switch(opțiune)
    {
      case 'c':case 'C':crefis();break;
      case 'a':case 'A':adaug();break;
      case 'm':case 'M':modif();break;
      case 's':case 'S':șterg();break;
      case 'l':case 'L':list();break;
      case 'h':case 'H':help();break;
      case 'e':case 'E': break;
      default:help(); break;
    }
  }while(toupper(opțiune)!='E');
}
void help() // afișare informații despre utilizarea meniului și opțiunile acestuia
{cout<<"Opțiunile de lucru sunt :"<<'\n';
  cout<<" C,c-creare fisier"<<'\n';
  cout<<" A,a-adaugare"<<'\n';
  cout<<" M,m-modificare"<<'\n';
  cout<<" L,l-listare"<<'\n';
  cout<<" S,s-ștergere"<<'\n';
  cout<<" H,h-help"<<'\n';
  cout<<" E,e-exit"<<'\n';
}
long int lungfis(FILE *f) // returnează lungimea fișierului
{long int posi,posf;
  posi=ftell(f); fseek(f,0,SEEK_END);
  posf=ftell(f); fseek(f,posi,SEEK_SET);
  return posf;
}
void scrieb(int nr,void *a,FILE *f) //scriere în fișierul binar
{long depl=(nr-1)*sizeof(material);
  fseek(f,depl,SEEK_SET);
  if(fwrite(a,sizeof(material),1,f)!=1)
    {cout<<"Eroare de scriere in fișier !"<<'\n';
```

```

        exit(1); }
    }
void citireb(int nr,void *a,FILE *f) //citire din fișierul binar
{long depl=(nr-1)*sizeof(material);
 fseek(f,depl,SEEK_SET);
 if(fread(a,sizeof(material),1,f)!=1)
    {cout<<"Eroare de citire din fișier !"<<"\n";
    exit(2); }
}
void afismat(material *a) //afișarea informațiilor despre un anumit material
{
    int i;
    if(a->codm)
        {cout<<"Cod material : "<<a->codm<<"\n";
        cout<<"Denumire material: "<<a->den_mat<<"\n";
        cout<<"Cantități contractate:"<<"\n";
        for(i=0;i<4;i++)
            cout<<"Contractat "<<i<<" : "<<a->cant_c[i]<<"\n";
        cout<<"Cantități recepționate:"<<"\n";
        for(i=0;i<4;i++)
            cout<<"Recepționat "<<i<<" : "<<a->cant_r[i]<<"\n";
        cout<<"Stoc : "<<a->stoc<<"\n";
        cout<<"Unitate de masura: "<<a->unit_mas<<"\n";
        cout<<"Preț unitar : "<<a->preț<<"\n";
        }
    else cout<<"Acest articol a fost șters !"<<"\n";
}
void citmat(material *a) //citirea informațiilor despre un anumit material
{
    int i;float temp;
    cout<<"Introduceți codul materialului (0=End): ";cin>>a->codm;
    if(a->codm==0) return;
    cout<<"Introduceți denumirea materialului : ";cin>>a->den_mat;
    cout<<"Introduceți unitatea de măsură : ";cin>>a->unit_mas;
    cout<<"Introduceți prețul : ";cin>>temp;a->preț=temp;
    cout<<"Introduceți cantitățile contractate : "<<"\n";
    for(i=0;i<4;i++)
        {cout<<"Contractat "<<i+1<<" : ";cin>>a->cant_c[i]; }
    cout<<"Introduceți cantitățile recepționate : "<<"\n";
    for(i=0;i<4;i++)
        {cout<<"Recepționat "<<i+1<<" : ";cin>>a->cant_r[i]; }
}
void crefis() //deschidere fișier
{
    if((pf=fopen("material.dat","r"))!=NULL)
        cout<<"Fișierul exista deja !"<<"\n";
    else
        pf=fopen("material.dat","w");
    fclose(pf);
}
void adaug() //adăugare de noi materiale
{
    int na; pf=fopen("material.dat","a");//deschidere pentru append
    na=lungfis(pf)/sizeof(material);
    do
        {citmat(&mat);
        if(mat.codm) scrieb(++na,&mat,pf);
        } while(mat.codm);
    fclose(pf);
}

```

```

void modif()          //modificarea informațiilor despre un material existent
{
    int na; char ch; pf=fopen("material.dat","r+");
    do
        {cout<<"Numarul articolului de modificat este (0=END): ";cin>>na;
        if(na)
            {citireb(na,&mat,pf);
            afismat(&mat);
            cout<<"Modificați articol (D/N) ? :";
            do
                { cin>>ch;
                ch=toupper(ch);
                } while(ch!='D' && ch!='N');
            if(ch=='D')
                {citmat(&mat);
                scrieb(na,&mat,pf);
                }
            }
        }while(na);
    fclose(pf);
}

void sterg()          //ștergerea din fișier a unui material
{ int n;long int na; pf=fopen("material.dat","r+");
  mat.codm=0; na=lungfis(pf)/sizeof(material);
  do
    {
    do
        {cout<<"Numarul articolului de șters este (0=END): ";cin>>n;
        if(n<0||n>na) cout<<"Articol eronat"<<'\n';
        }while(!(n>=0 && n<=na));
        if(n) scrieb(n,&mat,pf);
    }while(n);
    fclose(pf);
}

void list()           //afișare informații despre un anumit material
{
    int na; pf=fopen("material.dat","r");
    do
        {cout<<"Numarul articolului de listat este (0=END): ";cin>>na;
        if(na)
            {citireb(na,&mat,pf);
            afismat(&mat);
            cout<<'\n';
            }
        }while(na);
    fclose(pf);
}

```

ÎNTREBĂRI ȘI EXERCIIU

Chestiuni practice

1. Scrieți un program de tipărire a conținuturilor mai multor fișiere, ale căror nume se transmit ca parametri către funcția main. Tipărirea se face pe ecran (lungimea paginii = 22) sau la imprimantă (lungimea paginii = 61). Conținutul fiecărui fișier va începe pe o pagină nouă, cu un titlu care indică numele fișierului. Pentru fiecare fișier, paginile vor fi numerotate (cu ajutorul unui contor de pagini).
2. Scrieți un program care citește un fișier text. Pornind de la conținutul acestuia, se va crea un alt fișier, prin înlocuirea spațiilor consecutive cu unul singur. Se vor afișa pe ecran conținutul fișierului de la care s-a pornit și conținutul fișierului obținut.

3. Să se consulte conținutul unui fișier și să se afișeze următoarele informații statistice: numărul de cuvinte din fișier, numărul de caractere, numărul de linii, numărul de date numerice (nu cifre, numere!).
4. Scrieți un program care să compare conținutul a două fișiere, și afișați primele linii care diferă și poziția caracterelor diferite în aceste linii.
5. Scrieți un program care citește conținutul unui fișier sursă scris în limbajul C și afișează în ordine alfabetică fiecare grup al numelor de variabile care au primele n caractere identice (n este citit de la tastatură).
6. Scrieți un program care consultă un fișier text și afișează o listă a tuturor cuvintelor din fișier. Pentru fiecare cuvânt se vor afișa și numerele liniilor în care apare cuvântul.
7. Scrieți un program care citește un text introdus de la tastatură și afișează cuvintele distincte, în ordinea crescătoare a frecvenței lor de apariție. La afișare, fiecare cuvânt va fi precedat de numărul de apariții.
8. Scrieți un program care citește un text introdus de la tastatură, ordonează alfabetic liniile acestuia și le afișează.
9. Scrieți o aplicație pentru gestiunea informațiilor despre cărțile existente într-o bibliotecă. Aplicația va avea un meniu principal care va permite:
 - a) Memorarea datelor într-un fișier (un fișier binar cu structuri), al cărui nume se introduce de la tastatură. Fișierul va conține informațiile: nume carte, autor, editura, anul apariției, preț. Pentru fiecare carte, se va genera o cotă (un număr unic care să constituie cheia de căutare).
 - b) Adaugarea de noi cărți;
 - c) Afișarea informațiilor despre o anumită carte;
 - d) Căutarea titlurilor după un anumit autor;
 - e) Modificarea informațiilor existente;
 - f) Lista alfabetică a tuturor autorilor;
 - g) Ștergerea unei cărți din bibliotecă;
 - h) Ordonarea descrescătoare după anul apariției;
 - i) Numele celei mai vechi cărți din bibliotecă;
 - j) Numele celei mai scumpe cărți din bibliotecă;
 - k) Numele autorului cu cele mai multe cărți;
 - l) Valoarea totală a cărților din bibliotecă.