9. FIŞIERE

În marea majoritate a aplicațiilor, simple sau complexe, apare necesitatea păstrării informațiilor şi după terminarea programului. De exemplu, este necesar ca datele referitoare la candidații la concursul de admitere, despre care am discutat în lecția anterioară, să fie memorate sub formă de fişiere fie pe disc, fie pe dischetă, în vederea unor prelucrări ulterioare.

Un fişier este utilizat ca o unitate de memorare şi constă dintr-o succesiune de octeți căreia i se asociază un nume. Numele fişierului este format din două părți:

- prima parte, formată dintr-un şir de maximum 8 caractere, conține numele propriu zis al fişierului;
- a doua parte, numită **extensie**, constă dintr-un şir de maximum 3 caractere şi este separată de numele propriu-zis prin caracterul punct.

Prin intermediul unui set complet de funcții, limbajul C oferă utilizatorilor două modalități de programare a operațiilor de citire şi scriere a fişierelor sub controlul sistemului de operare.

O prima modalitate o constituie aşa-zisul mod standard de intrare/ieşire, cunoscut sub denumirea "**stream I/O**". Acest mod este uşor de utilizat deoarece o serie întreagă de activități privind efectuarea propriu-zisă a operațiilor sunt efectuate automat.

A doua modalitate o constituie programarea la nivelul sistemului de operare, cunoscută sub denumirea "**system-level I/O**" sau "**low-level I/O**". Acest sistem, mai dificil de utilizat, este folosit de programatorii experimentați pentru a crea sisteme standard de intrare/ieşire. Avînd în vedere acest aspect, în cadrul prezentei lecții ne vom ocupa numai de sistemul standard de intrare/ieşire.

9.1. Generalități privind modul standard de intrare-ieşire

Sistemul standard de intrare/ieşire permite efectuarea transferului de date (operații de citire-scriere sau intrare/ieşire) între program şi următoarele patru tipuri de fişiere:

- **fişiere caracter** - datele sunt citite sau scrise caracter cu caracter, similar cu citirea unui caracter de la tastatură sau scrierea acestuia pe ecran.
- **fişiere şir de caractere** - datele sunt citite sau scrise sub forma şirurilor de caractere, similar modului în care un şir de caractere este citit de la tastatură sau afişat pe ecran.
- **fişiere formatate** - datele citite sau scrise alcătuiesc colecții mixte de caractere şi valori numerice întregi sau reale transferate în conformitate cu descriptorii de format identici celor folosiți in cadrul funcțiilor **printf** şi **scanf**.
- **fişiere înregistrare** - datele sunt citite sau scrise sub forma unor blocuri de lungime fixă - numite **înregistrări**. În mod uzual, o inregistrare conține fie elementele unui tablou (matrice), fie ale unei structuri.
- În funcție de modul în care informația este reprezentată în cadrul fişierelor, acestea pot fi grupate în următoarele două categorii:
- **fişiere text** - în care informațiile sunt reprezentate sub forma unui text, pentru fiecare caracter fiind utilizat un octet. Aceste fişiere prezintă avantajul că pot fi vizualizate şi modificate cu ajutorul editoarelor de text, inclusiv cu ajutorul editorului mediului integrat Turbo C++/Borland C. Dezavantajul lor provine din faptul că, în cazul memorării datelor de tip numeric, este folosit un octet pentru fiecare cifră a numărului memorat şi, prin urmare, necesită un spațiu mare de memorie. De exemplu, valoarea -1756.3256 va fi memorata în 10 O (se alocă şi un octet pentru semn şi unul pentru punctul zecimal).
- **fişiere binare** - în care informațiile sunt memorate în formă binară intr-un mod similar cu memorarea valorilor variabilelor în memoria internă. Avantajul acestor fişiere provine din faptul că necesită un spațiu de memorie mai mic şi că nu necesită conversia/deconversia la scriere/citire. Astfel valoarea -1756.3256 va fi memorată în numai 4 O dacă este valoarea unei variabile de tipul real simplu sau în 8 O dacă este valoarea unei variabile de tipul real dublu. Dezavantajul lor provine din faptul că nu pot fi vizualizate sau modificate cu editoare de text.

Se precizează faptul că modurile caracter, şir şi format sunt destinate în general prelucrării fişierelor de tipul text, în timp ce modul înregistrare este destinat prelucrării fişierelor de tipul binar.

Un alt aspect legat de foloşirea unui fişier îl constituie deschiderea acestuia înaintea efectuării operațiilor propriu-zise de scriere şi citire.

Deschiderea unui fişier se realizează prin apelul funcției **fopen**. Aceasta are ca parametri două şiruri de caractere prin intermediul cărora transmitem sistemului de operare numele fişierului şi modul în care acesta urmează fi accesat şi returnează un pointer la o structură de tip **FILE**. Valoarea pointerului este furnizată de sistemul de operare şi poate fi 0 (NULL), dacă operația de deschidere a eşuat, sau o valoare diferită de 0, în cazul în care s-a reuşit deschiderea fişierului. Structura **FILE** este definită în fişierul antet **stdio.h** şi conține variabile în care se memorează informațiile referitoare la fişier.

Sintetizînd cele prezentate, concluzionăm ca pentru deschiderea unui fişier în cadrul unui program C se foloseşte următoarea secvență de instrucțiuni:

```
FILE *p_fis;
   .        .        .
   p_fis = fopen(nume_fişier,mod_acces);
   if (p_fis == NULL)
    {
          .        .        .
    }
```

care constă din:

- instrucțiunea de declarare a pointerului la fişier;
- instrucțiunea de apel a funcției **fopen**. Referitor la parametrii acestei funcții, se precizează faptul că variabila *nume_fişier* conține numele fişierului. Acesta trebuie sa respecte regulile prezentate la începutul lecției şi poate fi precedat de descrierea căii de acces la directorul în care se află fişierul. De exemplu, utilizand ca prim parametru al funcției **fopen** şirul: "*C:/lectii_c/text.txt*", se solicită deschiderea fişierului cu numele *text.txt* aflat pe discul *C* în directorul *lectii_c*. Variabila *mod_acces* este tot un şir de caractere care poate avea una din valorile prezentate în tabelul 9.1.

- instrucțiunea *if*, prin care se testează dacă operația de deschidere a fișierului a eșuat. În cazul în care apare o astfel de situație, se afișează un mesaj de eroare și se decide continuarea sau oprirea programului.

Tabelul 9.1.

Modul de acces			
Fişiere text	Fişiere binare	Efect	Observaţii
"r"	"rb"	Deschide fişierul pentru citire.	Fişierul trebuie să existe.
"w"	"wb"	Deschide fişierul pentru scriere.	Dacă fişierul există, conţinutul său va fi distrus, iar dacă nu există - va fi creat.
"a"	"ab"	Deschide fişierul pentru adăugare.	Dacă fişierul există se vor adăuga noi informatii la sfîrşitul său, iar dacă nu există - va fi creat.
"r+"	"rb+"	Deschide fişierul pentru citire cu posibilitatea de scriere.	Fişierul trebuie să existe. Prin operaţia de scriere conţinutul său nu este distrus.
"w+"	"wb+"	Deschide fişierul pentru scriere cu posibilitatea de citire.	Dacă fişierul există, conţinutul său va fi distrus, iar dacă nu există - va fi creat.
"a+"	"ab+"	Deschide fişierul pentru adăugare de informaţii, cu posibilitate de citire.	Dacă fişierul nu există, el va fi creat.

La terminarea operațiilor de scriere și/sau citire a unui fișier, acesta trebuie închis. Pentru aceasta se folosește funcția **fclose**, care are ca parametru pointerul de tipul structură **FILE** utilizat la deschiderea fișierului. Astfel, prin instrucțiunea *fclose(p_fis);* se solicită sistemului de operare închiderea fișierului având ca pointer variabila *p_fis*.

Operația de inchidere a fișierului are ca efect eliberarea zonei de memorie alocată structurii de tipul **FILE**, în care sunt memorate informațiile referitoare la fișier, precum și a bufferului de comunicație. Deoarece timpul de acces la suportul de memorie externă (disc sau dischetă) este mare, apare necesitatea optimizării transferului de date între program și fișier prin reducerea numărului de accesări. Pentru aceasta, se utilizează o zonă de memorie, numită **buffer** (memorie tampon), în care datele sunt memorate temporar. În momentul în care bufferul s-a umplut, conținutul său este transmis fișierului printr-un singur acces la unitatea de memorie externă. Este foarte posibil ca terminarea unei operații de scriere să se producă într-un moment în care bufferul este parțial ocupat, adică într-un moment în care conținutul său nu a fost încă transferat fișierului. Într-o astfel de situație, pentru a evita pierderea datelor conținute în buffer, funcția **fclose** va forța transferul acestora către fișier și apoi va elibera zona de memorie. În modul standard de intrare/ieșire toate aceste operații se desfășoară automat, atît existența lor cît și a bufferului fiind invizibile programatorului.

Toate funcțiile aferente sistemului standard de intrare/ieșire sunt declarate în fișierul antet <**stdio.h**>. Acesta trebuie inclus în fișierul sursă C ori de cîte ori efectuăm operații de scriere-citire.

9.2. Scrierea şi citirea fişierelor de tipul caracter

Pentru a exemplifica modul în care sunt efectuate operațiile de scriere și citire a fișierelor în modul caracter, considerăm următorul program de calcul:

Exemplu 9.1.

```
/* Programul ex_9_1*/
#include <stdio.h>
void scrie_c(char *)
void citeste_c(char *)
void main (argc,argv)
 int argc;     char *argv[ ];
 {
    if (argc != 2)
     {
            printf("\n Eroare! Tastati : nume_program nume_fisier");       exit(1);
     }
    scrie_c(argv[1]); citeste_c(argv[1]);
 }
void scrie_c(char *nume_fis)
 {
    char car;
    FILE *fis;          /* pointer la structura fisier */
    fis = fopen(nume_fis,"w");          /* deschide fisierul */
/* test deschidere reusita a fisierului */
    if (fis == NULL)
     {
            printf("\n Eroare! Fisierul %s nu poate fi deschis",nume_fis); exit(1);
     }
/* citirea caracterelor si scrierea lor in fisier */
    clrscr( );
    printf("Tastati succesiunea de caractere >");
    while (car = getche( )) != 13)        putc(car,fis);       fclose(fis);
 }
void citeste_c(char *nume_fis)
 {
    int car, nr_car=0, nr_cifre=0;
    FILE *fis;          /* pointer la structura fisier */
```

```
    fis = fopen(nume_fis,"r"); /* deschide fisierul */
/* test deschidere reusita a fisierului */
    if (fis == NULL)
     {
          printf("\n Eroare! Fisierul %s nu poate fi deschis",nume_fis); exit(1);
     }
/* citirea caracterelor din fisier */
    clrscr( );
    printf("Continutul fisierului: ");
    while (car = getc(fis)) != EOF)
     {
          nr_car++;          printf("%c",car);
          if (car >= 48) && car <= 57) nr_cifre++;
     }
    printf("\nNumarul total de caractere %d\nNumarul de cifre %d",nr_car,nr_cifre);
    fclose(fis);
}
```

Scopul acestui program constă în citirea de la tastatura a unei succesiuni de caractere pe care o scrie într-un fişier. Ulterior, conţinutul fişierului este analizat pentru a determina cîte caractere au fost introduse şi cate dintre acestea sunt cifre. Pentru aceasta, sunt definite şi utilizate funcţiile **scrie_c** şi **citeste_c**.

Înainte de a analiza în detaliu aceste funcţii, să remarcăm utilizarea variabilelor *argc* şi *argv* ca parametri ai funcţiei **main**. În mod uzual, pentru lansarea în execuţie a unui program de calcul, se tastează, în continuarea mesajului sistemului de operare MS-DOS conţinînd directorul de lucru curent, numele programului urmat de apăsarea tastei <Enter>. Prin utilizarea parametrilor funcţiei **main**, limbajul C oferă posibilitatea transmiterii unor valori programului, în momentul lansării în execuţie. Aceste valori, constituite în şiruri de caractere, urmează numelui programului şi sunt separate de acesta şi între ele printr-un spatiu. Primul parametru al funcţiei **main** este de tipul întreg şi conţine numărul şirurilor de caractere ce alcătuiesc linia de lansare în execuţie a programului (inclusiv numele acestuia). Al doilea parametru este un tablou de pointeri la şiruri de caractere şi conţine adresele din memorie la care sunt memorate şirurile de caractere introduse de la tastatură în momentul lansării în execuţie (inclusiv numele programului). În cadrul programului ex_9_1 este folosit acest mecanism pentru a transmite numele fişierului în momentul lansării în execuţie. Astfel, dacă ne aflăm în directorul de lucru curent LECTII_C de pe discul sistem C: şi dorim ca fişierul pe care-l creăm să poarte numele "carac1er. txt", linia de lansare in execuţie a programului ex_9_1 este:

C:\LECTII_C> ex_9_1 caracter.txt<enter>

În corpul funcţiei **main**, mai întîi este verificată introducerea corectă a liniei de comandă pentru lansarea în execuţie a programului. Aceasta trebuie să conţină două şiruri de caractere (numele programului şi numele fişierului). Dacă valoarea variabilei *argv* este diferită de 2, este afişat un mesaj de eroare care sugerează forma corectă a comenzii de lansare în execuţie.

Din cele prezentate, deducem că nu este posibilă lansarea în execuţie a programului ex_9_1 din cadrul mediului integrat Turbo C++/Borland C cu ajutorul comenzii **Run**. Pentru lansarea în execuţie este necesară obţinerea mesajului MS-DOS.

Revenind la cele doua funcţii definite in programul ex_9_1, constatăm că fiecare dintre ele conţine în prima parte setul de instrucţiuni destinat deschiderii fişierului cu care lucrează. În funcţia **scrie_c**, caracterul tastat este preluat mai întîi prin apelul funcţiei **getche** şi memorat în variabila *ch* de tipul caracter. Apoi, în expresia logică a buclei **while**, este testat conţinutul acestei variabile pentru a detecta momentul în care s-a apăsat tasta <Enter>, adică momentul în care utilizatorul a terminat de introdus toate caracterele dorite. Atîta timp cît acesta este diferit de 13 (codul zecimal al tastei <Enter>), caracterul este scris în fişier prin apelul funcţiei **putc**. Forma generala de apel a funcţiei **putc** este:

putc (caracter,p_fis);

în care *caracter* este o constantă sau o variabilă de tipul caracter, iar *p_fis* este pointerul la fişierul în care aceasta urmează a fi scrisă. În funcţia **citeste_c** este utilizată funcţia **getc** pentru a citi conţinutul fişierului creat prin apelul funcţiei **scrie_c**.

Funcţia **getc**, care este perechea funcţiei **putc** pentru accesarea fişierelor de tipul caracter, primeşte ca parametru pointerul la fişierul din care se face citirea şi returnează codul caracterului citit. Forma generală de apel a acestei funcţii este:

rezultat = getc(p_fis);

Se precizează faptul că variabila *rezultat* trebuie să fie de tipul caracter fără sernn sau întreg. Această necesitate este impusă de mecanismul de detectare a sfîrşitului de fişier. După ce a fost citit ultimul caracter, la o nouă tentativă de citire sistemul de operare transmite programului un mesaj prin care se semnalează că nu mai există caractere în fişier. Mesajul de sfîrşit de fişier îl constituie valoarea întreaga -1 (constanta EOF - end of file - definită in fişierul antet **stdio.h**). Dacă am utiliza o variabilă de tipul caracter pentru a memora rezultatul returnat de funcţia **getc**, atunci caracterul avînd codul zecimal 255 (FF în hexazecimal) ar constitui un semnal fals al sfîrşitului de fişier.

9.3. Scrierea şi citirea fişierelor de tipul şir de caracter

Operaţiile de citire şi scriere a fişierelor de tipul şir sunt similare cu cele pentru citirea şi scrierea fişierelor de tipul caracter. În acest scop, se folosesc funcţiile **fputs** şi **fgets**. Pentru a exemplifica modul de utilizare a acestor funcţii, considerăm următorul program de calcul:

Exemplu 9.2

```
/* Programul ex_9_2*/
#include <stdio.h>
void main (void)
 {
    FILE *p_fis;
    char *buf[81];
    if (p_fis = fopen("text.txt","w")) == NULL)
```

```
    {
            printf("\n Eroare! Fisierul nu poate fi deschis");       exit(1);
    }
    clrscr( );
    printf("\nIntroduceti un text \n");
    while (strlen(gets(buf)) > 0)
    {
            fputs(buf,p_fis);  fputs("\n",p_fis);
    }
    fclose(p_fis);
    /* citirea fisierului */
    if (p_fis = fopen("text.txt","r")) == NULL)
    {
            printf("\n Eroare! Fisierul nu poate fi deschis");       exit(1);
    }
    clrscr( );
    printf("Continutul fisierului\n");
    while (fgets(buf,80,p_fis)) != NULL)          printf("%s",buf);
            if (car >= 48) && car <= 57) nr_cifre++;
    fclose(fis);          getch( );
}
```

Programul creează fişierul cu numele "*text.txt*" în care va scrie o succesiune de caractere citite de la tastatură, apoi conținutul este citit şi afişat pe ecran.

Scrierea fişierului se realizează în cadrul primei bucle while. Utilizatorul tastează pe fiecare linie a ecranului un şir de caractere (de exemplu, o propoziție) care se termină prin apăsarea tastei <Enter>. Fiecare şir este preluat prin apelul funcției **gets** şi memorat în variabila tablou de caractere *buf*. Aceasta, avînd dimensiunea 81, permite memorarea unor şiruri cu lungimea maximă de 80 de caractere, adică a unor şiruri care ocupă cel mult o linie a ecranului. Dacă apăsarea tastei < Enter > se face chiar la începutul liniei, atunci lungimea şirului este zero şi programul interpretează această acțiune ca semnal de incheiere a operației de scriere, deci ca semnal de părăsire a buclei şi închidere a fişierului.

Scrierea în fişier a fiecărui şir de caractere se realizează cu ajutorul funcției **fputs** avînd următoarea formă generala de apel: *fputs (sir_caractere,p_fis);*. Primul parametru este o constantă sau o variabilă de tipul şir de caractere, iar al doilea un pointer la fişierul în care se va scrie informația conținută în primul parametru.

În exemplul prezentat, corpul buclei while conține două apeluri ale funcției **fputs**. Primul realizează scrierea efectivă a şirului conținut în *buf*. Deoarece funcția **fputs** nu adaugă în mod automat caracterul new-line ("\n") la sfîrşitul şirului scris, acesta este adăugat în mod explicit prin cel de-al doilea apel. Această acțiune este necesară pentru a facilita operația de citire a fişierului, care se desfăşoară în cadrul celei de-a doua bucle while, prin apelul funcției **fgets**. Forma general5 de apel a acestei funcții este: *fgets (nume_sir,l_max,p_fis);* . Primul parametru este adresa (numele unui tablou de caractere sau numele unei variabile pointer la un şir de caractere), unde va fi memorat şirul de caractere citit. Al doilea parametru reprezintă lungimea maximă a şirului de caractere citit. Rolul său este de a împiedica citirea unor şiruri mai lungi decît capacitatea de memorare alocată prin intermediul primului parametru. Al treilea parametru este pointerul la fişierul din care se face citirea. Funcția **fgets** returnează o valoare întreagă care reprezintă lungimea şirului de caractere citit. În cadrul programului ex_9_2 valoarea returnată de **fgets** este utilizată în expresia logică a buclei while, pentru a detecta sfîrşitul fişierului. În momentul în care această valoare este 0 (constanta NULL) se părăseşte bucla şi se închide fişierul. Corpul buclei îl constituie instrucțiunea de apel a funcției **printf**, care afişează pe ecran şirurile citite din fişier.

9.4. Scrierea şi citirea fişierelor de tipul formatat

Spre deosebire de modul caracter şi modul şir, care permit scrierea şi citirea numai a fişierelor de tipul text, modul formatat permite scrierea şi citirea simultană atît a caracterelor, a şirurilor de caractere, cît şi a valorilor numerice. Datele, constituite în colecții mixte de caractere şi valori numerice, sunt transferate între program şi fişier în conformitate cu descriptorii de format, identici cu cei folosiți în cadrul funcțiilor **printf** şi **scanf**.

Pentru a exemplifica mecanismul de creare a fişierelor în modul formatat, considerăm programul din exemplul 9.3.

Exemplu 9.3

```
/* Programul ex_9_3*/
#include <stdio.h>
struct candidat
 {
    char nume[25], prenume[25];
    int nr_leg;
    float nota1, nota2, media;
 }
typedef struct candidat candidat;
candidat x;
int citeste (void);
void main (void)
 {
    FILE *p_fis;
    if (p_fis = fopen("baza.can","w")) == NULL)
     {
            printf("\n Eroare! Fisierul nu poate fi deschis");       exit( );
```

```
    }
    while (citeste( ))
    {
        fprintf(p_fis,"%s %s %d %f %f %f",x.nume,x.prenume,x.nr_leg,x.nota1,x.nota2,x.media);
        if (x.media >= 5) printf("%s %s admis",x.nume,x.prenume);
        else              printf("%s %s respins",x.nume,x.prenume);
        getch( );
    }
    fclose(p_fis);
int citeste( )
{
    float a,b;
    clrscr( ); fflush(stdin);
    printf("Numele candidatului:");     gets(x.nume);
    if (strlen(x.nume) == 0)    return(0);
    printf("Prenumele candidatului:"); gets(x.prenume);
    printf("Numarul legitimatiei de concurs:");  scanf("%d",&x.nr_leg);
    printf("Notele obtinute: ");        scanf("%f %f",&a,&b);
    x.nota1 = a;        x.nota2 = b;        x.media = (x.nota1+x.nota2)/2;
    return(1);
}
```

Acest program constituie o variantă îmbunătățită a programului ex_8_2, deoarece informațiile referitoare la un candidat la concursul de admitere sunt scrise în fişierul cu numele "*baza.can*" în vederea unor prelucrări ulterioare.

În cadrul buclei while, informațiile citite de la tastatură cu ajutorul funcției **citeste** sunt prelucrate pentru a afişa rezultatul obținut de candidat şi apoi sunt scrise în fişier prin apelul funcției **fprintf**. Aşa cum se poate constata, această funcție este similară funcției **printf** şi are următoarea formă generală de apel:

fprintf (p_fis, sirul_descriptorilor_de_format,lista_de_variabile)

Singura deosebire față de funcția **printf** o constituie prezența ca prim argument a pointerului p_fis la fişierul în care se va face scrierea.

La părăsirea buclei while, moment indicat prin apăsarea tastei < Enter > la solicitarea numelui candidatului, fişierul este închis astfel încît informațiile pot fi regăsite şi după terminarea programului. Pentru a ilustra acest fapt, considerăm programul din exemplul 9.4 în cadrul căruia conținutul fişierului *baza.can*, creat cu ajutorul programului ex_9_3, este citit şi afişat pe ecran.

Exemplu 9.4

```
/* Programul ex_9_4*/
#include <stdio.h>
struct candidat
{
    char nume[25], prenume[25];
    int nr_leg;
    float nota1, nota2, media;
}
typedef struct candidat candidat;
void main (void)
{
    candidat x;
    FILE *p_fis;
    if (p_fis = fopen("baza.can","r")) == NULL)
    {
        printf("\n Eroare! Fisierul nu poate fi deschis");      exit( );
    }
    clrscr( );
    while (fscanf(p_fis,"%s %s %d %f %f %f",x.nume,x.prenume,x.nr_leg,x.nota1,x.nota2,x.media)) != EOF)
    {
        printf("Numele candidatului:%s %s\nNumarul legitimatiei de concurs: %d",x.nume,x.prenume,x.nr_leg);
        printf("Notele obtinute: %f %f\nMedia: %f",x.nota1,x.nota2,x.media);
        if (x.media >= 5) printf(" (admis)");
        else              printf(" (respins)");
        getch( );
    }       fclose(p_fis);
}
```

Citirea înregistrărilor din fişier se realizează în cadrul unei bucle while cu ajutorul funcției **fscanf**. Această funcție, similară funcției **scanf**, constituie perechea funcției **fprintf** şi are următoarea formă generală de apel:

fscanf (p_fis, sirul_descriptorilor_de_format, lista_adreselor_variabilelor)

Se remarcă faptul că singura deosebire față de funcția **scanf** o constituie prezența ca prim argument a pointerului *p_fis* la fişierul din care se face citirea. Atît funcția **fprintf**, cît şi funcția **fscanf** returnează un întreg a cărui valoare reprezintă numărul de octeți scrişi, respectiv citiți din fişier. În cazul funcției **fscanf**, după citirea ultimei înregistrări, la o nouă tentativă de citire aceasta va returna valoarea -1, adică indicatorul de sfîrşit de fişier EOF. Avînd în vedere acest aspect, pentru a detecta sfîrşitul fişierului în cadrul

programului ex_9_4, apelul funcției **fscanf** este utilizat ca expresie logică a buclei while. Atît timp cît valoarea returnată de **fscanf** este diferită de valoarea -1 (EOF), programul afişează pe ecran informațiile citite din fişier.

Se precizează faptul că funcția **fscanf** nu poate fi utilizată pentru citirea şirurilor de caractere ce conțin spații albe,deoarece acestea sunt interpretate ca terminator de şir şi deci citirea se face incorect. Din acest motiv, în programele ex_9_3 şi ex_9_4 structura candidat a fost modificată prin introducerea unui nou membru. Acesta este de tip tablou de caractere şi este destinat citirii şi memorării prenumelui candidatului, separat de numele său.

9.5. Scrierea şi citirea fişierelor de tipul înregistrare

Modul formatat de scriere şi citire a datelor descris în paragraful anterior prezintă dezavantaje importante. Primul dezavan-taj constă într-o utilizare ineficientă a spațiului de memorare al discului, datorită faptului că fiecare cifră a datelor numerice este memorată pe 1 O (ca un caracter). Al doilea dezavantaj rezultă din faptul că pentru scrierea şi citirea datelor de tipul structură şi tablou este necesară accesarea fiecărui element în parte al acestora, nefiind posibilă tratarea lor ca o entitate.

Aceste inconveniente sunt eliminate de către **modul înregistrare de scriere şi citire a fişierelor**, numit şi **modul bloc de intrare/ieşire**. În acest mod de exploatare a fişierelor, datele sunt memorate în formă binară şi este permisă citirea şi scrierea oricărei cantități de informație în cadrul unei singure instrucțiuni.

Pentru a exemplifica mecanismul de creare a fişierelor de tip înregistrare, considerăm programul din exemplul 9.5.

Exemplul 9.5.

```
/* Programul ex_9_5 */
#include <stdio.h>
#include "candidat"
candidat x;
int citeste(void);
void main(void)
 {
    FILE *p_fis;
    if((p_fis=fopen("baza.can","w"))==NULL)
     {
            printf("EROARE! Nu poate fi deschis fişierul");      exit( );
     }
    while(citeste( ))          fwrite(&x,sizeof(x),1,p_fis);
    fclose(p fis);
 }
```

Acest program este şirnilar celui din exemplul 9.3, în sensul că citeşte de la tastatură informațiile referitoare la candidații la un concurs de admitere, folosind funcția **citeste**, şi apoi le depune într-un fişier. Totuşi, cele două programe sunt total diferite datorită faptului că fişierul "*baza.can*" din programul ex_9_5 este un fişier binar, iar scrierea informațiilor în el se realizează în modul înregistrare prin intermediul funcției **fwrite** a cărei formă generală de apel este:

```
fwrite (&inreg, lung_inreg, nr_inreg, p_fis);
```

Primul parametru este adresa de memorie la care se află înregistrările care vor fi scrise în fişier. Aceasta poate fi adresa unei structuri sau a unui tablou. Al doilea parametru reprezintă lungimea înregistrării exprimată în octeți. De obicei aceasta se află prin apelul funcției **sizeof**. Al treilea parametru reprezintă numărul de înregistrări de acelaşi tip care vor fi scrise în fişier. De exemplu, dacă *inreg* este un tablou de structuri, iar o structură reprezintă o înregistrare, atunci *nr_inreg* va avea valoarea egală cu dimensiunea tabloului. Ultimul parametru este pointerul la fişierul în care se face scrierea.

Avînd în vedere aceste precizări, înțelegem că prin instrucțiunea *fwrite(&x,sizeof(x),1,p_fis);* din exemplul 9.5 se va scrie în fişieru! p_fis o singură înregistrare (*nr_inreg=1*), constituită din informațiile memorate în variabila de tipul structură *candidat x*, a cărei dimensiune este furnizată de apelul funcției **sizeof**.

La părăsirea buclei while, moment indicat prin apăsarea tastei <Enter> la solicitarea numelui candidatului, fişierul este închis, astfel încît informațiile pot fi regăsite şi după terminarea programului. Pentru a ilustra acest fapt, considerăm programul din exemplul 9.6 în cadrul căruia conținutul fişierului *baza.can*, creat de programul ex_9_5, este citit şi prelucrat pentru a afişa pe ecran rezultatele obținute de candidați.

Exemplul 9.6.

```
/* Programul ex_9_6 */
#include <stdio.h>
#include "candidat"
candidat x;
void main(void) {
    FILE *p_fis;
    if((p_fis=fopen("baza.can","w"))==NULL)
     {
            printf ("EROARE! Nu poate f i deschis fişierul");     exit( );
     }
    while (citeste( ))
     {
            fwrite(&x, sizeof(x),1,p_fis);
            if(x.media >= 5)  printf("%s admis",x.nume);
                    else      printf ("%s respins",x.nume);
            fclose(p fis);
     }
 }
```

Citirea înregistrărilor din fişier se realizează cu ajutorul funcţiei **fread**, a cărei formă generală de apel este:

fread (&inreg, lung_inreg, nr_inreg, p_fis);

şi în care parametrii au aceeaşi semnificaţie ca şi parametrii funcţiei **fwrite**. Singura deosebire constă în faptul că *&inreg* constituie adresa de memorie în care vor fi depuse cele *nr_inreg* citite din fişier.

Funcţia **fread** returnează un întreg care reprezintă numărul de înregistrări citite efectiv. În condiţiile unor citiri corecte, valoarea returnată este egală cu valoarea celui de-al treilea parametru de apel.Dacă în tentativa sa de citire funcţia **fread** întîlneşte indicatorul EOF (sfîrşit de fişier), atunci valoarea returnată este mai mică decît valoarea specificată prin cel de-al treilea parametru de apel.

Avînd în vedere cele prezentate, înţelegem că prin instrucţiunea *fread(&x,sizeof(x),1,p_fis);* folosită în cadrul expresiei de control a buclei while din programul ex_9_6, se citeşte o înregistrare (*nr_inreg=1*) din fişierul p_fis atîta timp cît valoarea returnată de **fread** este 1. Înregistrarea citită este memorată în variabila *x* de tipul structură *candidat* care este apoi prelucrată pentru a decide dacă respectivul candidat a obţinut sau nu o medie mai mare decît 5.

9.6. Accesarea directă a înregistrărilor unui fişier

În exemplele prezentate pînă acum, scrierea şi citirea înregistrărilor s-au realizat în mod secvenţial. Aceasta înseamnă că atunci cînd scriem înregistrările (caractere, şiruri de caractere, valori nurnerice sau structuri), acestea sunt plasate în fişier una după alta în ordinea în care au fost introduse. În rnod similar, la citirea fişierului se porneşte cu citirea prirnei înregistrări şi se continuă în mod secvenţial cu citirea următoarelor înregistrări, pînă cînd se ajunge la sfîrşitul fişierului. Acest mecanism de scriere şi citire prezintă dezavantajul că pentru a accesa o înregistrare aflată într-o poziţie oarecare în interiorul fişierului este necesară accesarea tuturor înregistrărilor anterioare acesteia, fapt ce duce la o creştere substanţială a timpului de calcul.

În numeroase aplicaţii, cum ar fi cele de creare, actualizare şi prelucrare a bazelor de date, modul de acces secvenţial este ineficient datorită faptului că, prin timpul mare de acces la o înregistrare, reduce viteza de execuţie a programului. Pentru a elimina acest dezavantaj, sistemul standard de intrare/ieşire al limbajului C oferă utilizatorilor, ca o alternativă a modului de acces secvenţial, posibilitatea accesării aleatoare a înregistrărilor unui fişier. Aceasta înseamnă de fapt posibilitatea de a accesa în mod direct o anume înregistrare, indiferent de poziţia pe care o ocupă aceasta în fişier.

Pentru a exemplifica modul de accesare directă a înregistrărilor unui fişier considerăm următorul program de calcul:

Exemplul 9.7.

```
/* Programul ex_9_7 */
#include <stdio.h>
void main(void) {
    long int offset;
    int i;
    int mat[4][4] = {{11,12,13,14},{31,32,33,34},{21,22,23,24},{41,42,43,44}};
    FILE *p_fis;
    if((p_fis=fopen("stoc.mat","wb"))==NULL)
     {
            pzintf ("EROARE! Fişierul nu poate f i deschis");    exit( );
     }
/* scrie liniile matricei in fisier */
    foz(i=o; i<4; i++)          fwrite(&mat[i][0],sizeof(int),5,p_fis);
    fcloee(p fis);
    if ((p_fis=fopen("stoc.mat","rb") )==NULL)
     {
            printf ("EROARE! Fisierul nu poate fi deschis");     exit( );
     }
/* citizea aleatoare a inregistrarilor */
    /* citeste a patra inregistrare si o depune in a patra linie a matricei */
    offset=3*4*sizeof(int);     fseek(p_fis,offset,0);
    fread(&mat[3][0],sizeof(int),4,p_fis);
    /* citeste prima inregistrare si o depune in prima linie a matricei */
    offset=0;                   fseek(p_fis,offset,0);
    fread(&mat[0][0],sizeof(int),4,p_fis);
    /* citeste a treia inregistrare si o depune in a doua linie a matricei */
    offset=2*4*sizeof(int);     fseek(p_fis,offset,0);
    fread(&mat[1][0],sizeof(int),4,p_fis);
    /* citeste a doua inregistrare si o depune in a treia linie a matricei */
    offset=l*4+sizeof(int);     fseek(p_fis,offset,0);
    fread(&mat[2][0],sizeof(int),4,p_fis);
    fclose(p fis);      clrscr( );
    printf("MATRICEA CITITA DIN FIŞIER");
    for (i=0; i<4; i++)
            printf ("\n%d %d %d %d", mat[i][0],mat[i][1],mat[i][2],mat[i][3]);
    getch ();
 }
```

În prima parte a programului este creat fişierul binar cu numele "*stoc.mat*" în care se vor scrie elementele matricei de tipul întreg *mat*, avînd patru linii şi patru coloane. Scrierea în fişier se face în mod secvenţial, prin apelul funcţiei **fwrite** în cadrul buclei for. Analizînd parametrii funcţiei **fwrite**, constatăm că la fiecare apel se scriu în fişier cele patru elemente de tipul întreg ce alcătuiesc linia *i* din matricea *mat*, constituite într-o înregistrare. Se menţionează faptul că acest mod de scriere a elementelor matricei într-un

fişier nu este cel mai eficient din punct de vedere al codului scris, al necesarului de memorie şi al timpului de calcul, dar a fost adoptat cu scopul de a facilita explicarea şi îngelegerea modului de acces direct la înregistrările unui fişier.

În general, metoda cea mai eficientă de scriere a unei matrice într-un fişier o constituie scrierea printr-un singur apel al funcfiei **fwrite** a tuturor elementelor sale. Astfei, în locul secvenţei de instrucţiuni:

```
for (i=0; i<4; i++)
    fwrite (&mat[i][0], sizeof(int),4,p_fis);
```

din exemplul prezentat, se poate folosi instrucţiunea:

```
fwrite (&mat[0][0],sizeof(int),16,p_fis);
```

care are acelaşi efect, dar care conduce la reducerea necesarului de memorie şi a timpului de calcul, prin eliminarea setului de instrucţiuni aferent buclei for.

Revenind la programul ex_9_7, se precizează faptul că valorile cu care s-a făcut iniţializarea elementelor matricei au fost alese astfet încît prima cifră constituie indicele liniei, iar a doua cifră indicele coloanei. Avînd în vedere acest aspect, constatăm că liniile 2 şi 3 sunt schimbate între ele. Din acest motiv, în partea a doua a programului este folosit modul de acces direct la fişierul *stoc.mat*, pentru a citi liniile matricei şi a le depune în tabloul *mat* în ordinea corectă a indicilor.

Pentru a întelege modul în care se realizează acest lucru, vom analiza mai întîi modul în care sistemul standard de intrare/ieşire gestionează spaţiul de memorie externă alocat fişierului. În acest sens, reamintim faptul că pointerul *p_fis* de tip FILE este un pointer la o structură definită in <**stdio.h**>, care conţine printre elementele sale un pointer în care este memorată adresa de început a fişierului. Adresa unei inregistrări din fişier (adresa unui octet) se obţine adăugînd la adresa de început deplasamentul acesteia, numit **offset**. Desigur, deplasamentul primei înregistrări este 0, iar deplasamentul unei inregistrări oarecare *n* se obţine cu expresia *(n-1)*lungime_inregistrare*.

Dacă jn modul de acces secven.ial poziţionarea pointerului în fişier se face în mod automat în modul de acces direct, programatorul mai întîi va poziţiona pointerul în fişier la înregistrarea dorită şi apoi va efectua operaţia de scriere sau citire. Acest lucru se realizează cu ajutorul funcţiei **fseek**, a cărei formă generală de apel este:

```
fseek (p_fis,deplasament,mod);
```

în care primul parametru este pointerul la structură de tip FILE asociată fişierului la deschiderea acestuia. Al doilea parametru este o variabilă de tipul **long int**, în care se memorează deplasamentul. Ultimul parametru este o valoare întreagă care indică faţă de cine se măsoară deplasamentul. Valorile posibile ale acestuia şi semnificaţia lor sunt următoarele:

0 - deplasament relativ la începutul fişierului;
1 - deplasament relativ la poziţia curentă în fişier;
2 - deplasament relativ la sfîrşitul fişierului.

Pe baza acestui mecanism, în cadrul programului ex_9_7 mai întîi este citită ultima înregistrare (a patra) din fişierul *stoc.mat* şi memorată pe ultima linie a matricei. Pentru aceasta, se atribuie deplasamentului valoarea 24 (cu instrucţiunea *offset=3*4*sizeof(int);*), se poziţionează pointerul în fişier (apelul funcţiei **fseek** cu modul 0) şi se efectuează citirea (apelul funcţiei **fread**). În continuare, prin atribuirea de valori corespunzătoare deplasamentului şi apelul funcţiilor **fseek** şi **fread**, sunt citite şi celelalte înregistrări. Acestea sunt memorate în liniile corespunzătoare ale matricei *mat*, aşa cum rezultă din comentariile inserate în program. În urma execuţiei programului, pe ecran va fi afişat noul conţinut al matricei *mat*, în care se constată că liniile 2 şi 3 au fost schimbate între ele.

9.7. Fişiere predefinite. Mesaje de eroare

Pentru a efectua operaţii de intrare/ieşire (citire/scriere), la nivel de fişiere, este necesară mai întîi definirea unui pointer de tipul structură **FILE** şi apoi deschiderea fişierului prin apelul funcţiei **fopen**. În cadrul sistemului de operare MS-DOS sunt predefiniţi cinci pointeri la aşa-zisele **fişiere standard**, pe care îi putem utiliza, în cadrul programelor C, fără a mai fi necesare operaţiile de definire a lor şi de deschidere a fişierelor asociate. Denumirile acestor pointeri, fişierele pe care le accesează şi dispozitivele fizice carora le sunt asociate sunt următoarele:

stdin - fişierul standard de intrare asociat tastaturii;
stdout - fişierul standard de ieşire asociat monitorului;
stderr - fişierul standard de afisare a erorilor, asociat monitorului;
stdaux - fişierul standard de ieşire asociat interfeţei seriale;
stdprn - fişierul standard de ieşire asociat imprimantei.

Pentru a exemplifica modul în care pot fi utilizate fişierele standard MS-DOS, considerăm programul din exemplul 9.8, în cadrul căruia utilizăm funcţiile **fgets** şi **fputs** pentru a citi de la tastatură un mesaj, pe care apoi îl scriem la imprimantă.

Exemplul 9.8.

```
/* Programul ex_9_8 */
#include <stdio.h>
void main(void)
 {
    char buf[81];
    clrscr( );
    printf("Intioduceti mesajul adresat imprimantei:");
    fgets (buf,80,stdin);
    fputs (buf, stdprn);
 }
```

În cadrul celor cinci fişiere standard MS-DOS întîlnim şi fişierul **stderr**. El este destinat preluării mesajelor de eroare şi este de obicei asociat monitorului. Acest fişier este folosit şi de sistemul de intrare/ieşire standard al limbajuiui C, pentru afişarea mesajelor de eroare.

În marea lor majoritate, funcţiile standard de citire/scriere nu returnează în mod explicit un cod de eroare. De exemplu, dacă funcţia **getc** întoarce valoarea *-1*, aceasta poate semnifica fie mesajul *EOF*, fie apariţia unei erori. În mod similar, dacă funcţia **fgets** întoarce valoarea *NULL*, aceasta poate semnifica fie mesajul *EOF*, fie o eroare.

Pentru a determina apariția unei erori în timpul operațiilor de scriere/citire a fișierelor se utilizează funcțiile **ferror** și **perror**. Funcția **ferror** are ca argument pointerul la structura de tipul **FILE** asociată fișierului și returnează valoarea *0* (*FALSE*) dacă nu a apărut o eroare, respectiv o valoare diferită de zero (*TRUE*) dacă s-a produs o eroare. În cazul apariției unei erori, detectată prin apelul funcției **ferror**, pentru a obține pe ecran mesajul explicit asociat acesteia se utilizează funcția **perror**. Aceasta are ca argument o constantă șir de caractere, care constituie mesajul programului, la care se adaugă caracterul ":" urmat de mesajul de eroare al sistemului.

Pentru a exemplifica modul de utilizare a funcțiilor **ferror** și **perror**, în programul din exemplul 9.9 se prezintă o altă variantă de a testa încheierea cu succes a operației de deschidere a unui fișier.

Exemplul 9.9.

```
/* Programul ex_9_9 */
#include <stdio.h>
void main (void) {
    FILE *p_fis;
    p_fis=fopen("test_err.err","r");
    if (ferror(p_fis))
     {
            clrscr( );
            perror("EROARE! Fişierul nu poate f i deachis");
            getch( ); exit( );
     }
 }
```

Dacă fișierul cu numele *test_err.err* nu există în directorul de lucru curent, la rularea acestui program operația de deschidere și citire genereazu o eroare. În acest caz, valoarea întoarsă de funcția **ferror** este diferită de zero (*TRUE*) și are ca efect executarea corpu1ui instrucțiunii de decizie **if**, adică afișarea prin apelul funcției **perror** a următorului mesaj:

EROARE! Fişierul nu poate fi deschis: No such file or directory.

Textut din stînga caracterului ":" este mesajul programului (constanta șir de caractere cu care a fost apelată funcția **perror**), iar cel din dreapta este mesajul sistemului de operare care indică absența fișierului în directorul de lucru curent.