Capitolul IB.11. Operaţii cu fişiere în limbajul C

Cuvinte cheie
Fişiere text, fişiere binare,
deschidere/ închidere fişiere, citire-scriere fişiere,
acces direct, fişiere predefinite, funcţia fflush, redirectarea fişierelor
standard, fişiere în C++

IB.11.1. Noţiunea de fişier

Un fişier este o colecţie de date memorate pe un suport extern şi care este identificată printr-un nume.

Conţinutul fişierelor poate fi foarte variat:
- texte, inclusiv programe sursă
- numere
- alte informaţii binare: programe executabile, numere în format binar, imagini sau sunete codificate numeric ş.a.

Numărul de elemente ale unui fişier este variabil (poate fi nul).

Fişierele de date se folosesc pentru:
- date iniţiale mai numeroase
- rezultate mai numeroase
- păstrarea permanentă a unor date de interes pentru anumite aplicaţii.

Fişierele sunt entităţi ale sistemului de operare şi ca atare ele au nume care respectă convenţiile sistemului, fără legătură cu un anume limbaj de programare. Operaţiile cu fişiere sunt realizate de către sistemul de operare, iar compilatorul unui limbaj traduce funcţiile de acces la fişiere în apeluri ale funcţiilor de sistem.

De obicei prin *fişier* se subînţelege un fişier disc (pe suport magnetic sau optic), dar noţiunea de fişier este mai generală şi include orice flux de date din exterior spre memorie sau dinspre memoria internă spre exterior. De aceea s-a introdus cuvântul *stream*, tradus prin flux de date si sinonim cu fişier logic, deci orice sursă sau destinaţie externă a datelor.
Stream (flux de date, canal) este în acest context sinonim cu *file* (fişier): pune accent pe aspectul dinamic al transferului de date.

Programatorul se referă la un fişier printr-o variabilă; tipul acestei variabile depinde de limbajul folosit şi chiar de funcţiile utilizate (în C). Asocierea dintre numele extern (un şir de caractere) şi variabila din program se face la deschiderea unui fişier, printr-o funcţie standard.

IB.11.2. Tipuri de fişiere în C

Fişiere text
- **conţin o succesiune de linii, separate prin NewLine**
- **fiecare linie are 0 sau mai multe caractere tipăribile şi/sau tab**

Fişiere binare
- **conţin o succesiune de octeţi**

Un fişier text conţine numai caractere ASCII, grupate în linii de lungimi diferite, fiecare linie terminată cu unul sau două caractere terminator de linie.

Caracter terminator de linie:

- **fişierele Unix/Linux: un singur caracter terminator de linie '\n'**
- **fişierele Windows şi MS-DOS: caracterele '\r' şi '\n' (CR,LF) ca terminator de linie**

Un fişier text poate fi terminat printr-un *caracter terminator de fişier* (*Ctrl-Z* = *EOF*)
Valoarea efectivă este dependentă de sistem, dar în general este -1.

Acest terminator nu este însă obligatoriu. Sfârşitul unui fişier disc poate fi detectat şi pe baza lungimii fişierului (număr de octeți), memorată pe disc.

Funcțiile de citire sau de scriere cu format din/în fişiere text realizează conversia automată din format extern (şir de caractere) în format intern (binar virgulă fixă sau virgulă mobilă) la citire şi conversia din format intern în format extern, la scriere pentru numere întregi sau reale.

Fişierele binare pot conține:

- numere în reprezentare internă (binară)
- articole (structuri de date)
- fişiere cu imagini grafice, în diverse formate, etc

Citirea şi scrierea se fac fără conversie de format.
Pentru fiecare tip de fişier binar este necesar un program care să cunoască şi să interpreteze corect datele din fişier (structura articolelor). Este posibil ca un fişier binar să conțină numai caractere, dar funcțiile de citire şi de scriere pentru aceste fişiere nu cunosc noțiunea de linie; ele specifică numărul de octeți care se citesc sau se scriu.

Consola şi imprimanta sunt considerate fişiere text.
Fişierele disc trebuie deschise şi închise, dar fişierele consolă şi imprimanta nu trebuie deschise şi închise.

IB.11.3. Operarea cu fişiere

Pentru operarea cu un fişier (text sau binar) se defineşte o variabilă de tip ***FILE **** pentru accesarea fişierului:

FILE * - tip structură definită în stdio.h

Conține informații referitoare la fişier şi la tamponul de transfer de date între memoria centrală şi fişier:

- adresa
- lungimea tamponului
- modul de utilizare a fişierului
- indicator de sfârşit de fişier
- indicator de poziție în fişier

Etapele pentru operarea cu un fişier în limbajul C sunt:

- se deschide fişierul pentru un anumit mod de acces, folosind funcţia de biblioteca ***fopen***,
  - realizează şi asocierea între variabila fişier şi numele extern al fişierului
- se prelucrează fişierul
  - operaţii citire/scriere
- se închide fişierul folosind funcţia de biblioteca ***fclose***.

IB.11.4. Funcţii pentru deschidere şi închidere fişiere

Funcţiile standard pentru acces la fişiere sunt declarate în *stdio.h*. După cum spuneam mai devreme, funcţiile de citire/scriere/pozitionare în fişier folosesc pentru identificarea unui fişier o variabilă pointer la o structură predefinită *FILE*.

IB.11.4. 1 Deschiderea unui fişier

Pentru a citi sau scrie dintr-un/într-un fişier disc, acesta trebuie mai întâi deschis folosind funcţia *fopen*.

FILE *fopen (const char *numefisier, const char *mod);

- Deschide fişierul cu numele dat pentru acces de tip mod
- Returnează pointer la fişier sau NULL dacă fişierul nu poate fi deschis
- Valoarea returnată este memorată în variabila fişier, care a fost declarată (FILE *) pentru accesarea lui.

unde:

- ***numefisier***: numele fişierului
- ***mod***: şir de caractere (între 1 şi 3 caractere):
  - ***r*** - readonly , este permisă doar citirea dintr-un fişier existent
  - ***w*** - write, crează un nou fişier, sau dacă există deja, distruge vechiul conţinut
  - ***a*** - append, deschide pentru scriere un fişier existent (scrierea se va face în continuarea informaţiei deja existente în fişier, deci pointerul de acces se plasează la sfârşitul fişierului)
  - **+** - permite scrierea şi citirea din acelasi fişier - actualizare (ex: "r+", "w+", "a+").
  - ***t*** sau ***b*** - tip fişier ("text", "binary"), implicit este ***t***

Primul argument al funcţiei *fopen* este numele extern al fişierului scris cu respectarea convenţiilor limbajului C.

Numele fişier extern poate include următoarele:

- **Numele unităţii de disc sau partiţiei disc (** ex: A:, C:, D:, E:)
- **Calea spre fişier**: succesiune de nume de fişiere catalog (director), separate printr-un caracter ('\' în MS-DOS şi MS-Windows, sau '/' în Unix şi Linux)
- **Numele propriu-zis al fişierului**
- **Extensia,** care indică tipul fişierului şi care poate avea între 0 şi 3 caractere în MS-DOS.

Sistemele MS-DOS şi MS-Windows nu fac deosebire între litere mari şi litere mici, în cadrul numelor de fişiere.

Atenţie! pentru separarea numelor de cataloage dintr-o cale se vor folosi:

- \\ - pentru a nu se considera o secvență de caractere e*scape* sau:
- caracterul /

Exemple:

```
char *numef = "C:\\WORK\\T.TXT";
char *numef = "c:/work/t.txt";
```

La deschiderea unui fişier se iniţializează variabila pointer asociată, iar celelalte funcţii se referă la fişier numai prin intermediul variabilei pointer.
Funcția *fopen* are rezultat NULL (0) dacă fişierul specificat nu este găsit după căutare în directorul curent sau pe calea specificată.

Exemplu:

```
//exemplu 1
char *numef = "C:\\WORK\\T.TXT";   // sau c:/work/t.txt
FILE * f;                          // pentru referire la fişier
if ( (f=fopen(numef,"r")) == NULL) {
  printf("Eroare la deschidere fişier %s \n", numef);
  return;
}

//exemplu 2
#include <stdio.h>
int main ( ) {
  FILE * f; // pentru referire la fişier

  // deschide un fişier binar ptr citire
  f = fopen ( "c:\\t.txt", "rb" );

  printf ( f == NULL ? "Fişier negasit" : " Fişier gasit");
  ...
  if (f)          // dacă fişier existent
     fclose(f);   // închide fişier
  return 0;
}
```

Diferența dintre *b* şi *t* este aceea că la citirea dintr-un fişier binar toţi octeţii sunt consideraţi ca date şi sunt transferaţi în memorie, iar la citirea dintr-un fişier text anumiţi octeţi sunt interpretaţi ca terminator de linie (\0x0a) sau ca terminator de fişier (\0x1a). Nu este obligatoriu ca orice fişier text să se termine cu un caracter special cu semnificaţia *sfârşit de fişier* (*CTRL-Z* , de exemplu) .

Pentru fişierele text sunt folosite modurile:

- *w* pentru crearea unui nou fişier
- *r* pentru citirea dintr-un fişier
- *a* pentru adăugare la sfârşitul unui fişier existent
- Modul *w+* poate fi folosit pentru citire după creare fişier.

Deschiderea în modul *w* şterge orice fişier existent cu acelaşi nume, fără avertizare, dar programatorul poate verifica existența unui fişier în acelaşi director înainte de a crea unul nou.
Pentru fişierele binare se practică actualizarea pe loc a fişierelor, fără inserarea de date între cele existente, deci modurile *r+*, *a+*, *w+*. (literele *r* şi *w* nu pot fi folosite simultan).

Fişierele standard de intrare-ieşire (tastatura şi ecranul consolei) au asociate variabile de tip pointer cu nume predefinit (*stdin* şi *stdout*); care pot fi folosite în funcţiile destinate tuturor fisierelor, cum ar fi *fflush*.

Pentru închiderea unui fişier disc se foloseşte funcţia ***fclose***:

int fclose(FILE *fp);

- închide fişierul şi eliberează zona tampon
- în caz de succes întoarce 0, altfel, întoarce **EOF**.

Închiderea este absolut necesară pentru fişierele în care s-a scris ceva, dar poate lipsi dacă s-au făcut doar citiri din fişier.

IB.11.4. Operaţii uzuale cu fişiere text

Accesul la fişiere text se poate face

- **fie la nivel de linie**
- **fie la nivel de caracter**

dar numai secvenţial.

Deci nu se pot citi/scrie linii sau caractere decât în ordinea memorării lor în fişier şi nu pe sărite (aleator)!
Nu se pot face modificări într-un fişier text fără a crea un alt fişier, deoarece nu sunt de conceput deplasări de text în fişier!

Pentru citire/scriere din/în fişierele standard stdin/stdout se folosesc funcţii cu nume puţin diferit şi cu mai puţine argumente, dar se pot folosi şi funcţiile generale destinate fişierelor disc. Urmează câteva perechi de funcţii:

Sintaxa	Descriere
int fgetc (FILE * f); **// sau getc (FILE*)**	Citeşte un caracter din ***f*** şi îl întoarce ca un unsigned char convertit la int, Returnează ***EOF*** dacă s-a întâlnit sfârşitul de fişier sau în caz de eroare.
int fputc (int c, FILE * f); **// sau putc (int, FILE*)**	Scrie caracterul cu codul ascii ***c*** în fişier
char * fgets(char * line, int max, FILE *f);	Citeşte maxim n-1 caractere sau până la ***\n*** inclusiv, şi le depune în ***s***, adaugă la sfârşit ***\0*** dar nu elimină terminatorul de linie \n. Returnează adresa şirului. La eroare întoarce valoarea ***NULL***.
int fputs (char * line, FILE *f);	Scrie şirul ***line*** în fişier, fără caracterul '\0'. La eroare întoarce ***EOF***.

Detectarea sfârşitului de fişier se poate face şi cu ajutorul funcţiei ***feof*** (*Find End of File*):

int feof (FILE *fp);

- testează dacă s-a ajuns la *end-of-file* al fişierului referit de fp
- returnează 0 dacă nu s-a detectat sfârşit de fişier la ultima operaţie de citire, respectiv o valoare nenulă (adevarată) pentru sfârşit de fişier.

Atenţie! Rezultatul lui *feof* se modifică după încercarea de a citi după sfârşitul fişierului!

Se va scrie în fişierul de ieşire şi –1, rezultatul ultimului apel al funcţiei *fgetc*:

```
while ( ! feof(f1))
    fputc(fgetc(f1),f2);
```

Soluţia preferabilă pentru ciclul de citire-scriere caractere este următoarea:

```
while ( (ch=fgetc(f1)) != EOF)
            fputc ( ch, f2);
```

Exemplu:

```
// citire şi afişare linii dintr-un fişier
#include<stdio.h>
#include<stdlib.h>

int main()
{
   FILE *fp;
   char s[80];
   if ( (fp=fopen("c:\\test.c","r")) == NULL ) {
      printf ( "Nu se poate deschide la citire fişierul!\n" );
      exit (1);
   }
   while ( fgets(s,80,fp) != NULL )
      printf ( "%s", s);
   fclose (fp);
   return 0;
}

/*
Scriere sub formă de litere mici caracterele dintr-un fişier în alt fişier
numele sursei şi destinaţiei transmise în linia de comandă.

Lansarea în execuţie a programului:
copiere fişier_sursa.dat fişier_dest.dat
*/
#include<stdio.h>
#include<ctype.h>

int main(int argc, char** argv){
   FILE * f1, * f2;  int ch;
   f1= fopen (argv[1], "r");
   f2= fopen (argv[2], "w");
   if ( f1==0 || f2==0) {
      puts (" Eroare la deschidere fişiere \n");
```

```
    return 1;
  }
  while ( (ch=fgetc(f1)) != EOF)        // citeste  din f1
    fputc ( tolower(ch),f2);            // scrie în f2
  fclose(f1);
  fclose(f2);
  return 0;
}
```

În principiu se poate citi integral un fişier text în memorie, dar în practică se citeşte o singură linie sau un număr de linii succesive, într-un ciclu repetat până se termină fişierul (pentru a se putea prelucra fişiere oricât de mari).

Pentru actualizarea unui fişier text prin modificarea lungimii unor linii, ştergerea sau inserţia de linii se va scrie un alt fişier şi nu se vor opera modificările direct pe fişierul existent.

IB.11.5. Intrări/ieşiri cu conversie de format

Datele numerice pot fi scrise în fişiere disc fie în format intern (mai compact), fie transformate în şiruri de caractere (cifre zecimale, semn ş.a).
Un fişier text ocupă mai mult spaţiu deoarece formatul şir de caractere necesită şi caractere separator între numere. Avantajul este că un fişier text poate fi citit cu programe scrise în orice limbaj sau cu orice editor de texte sau cu alt program utilitar de vizualizare fişiere!

Funcţiile de citire-scriere cu conversie de format şi editare sunt:

int fscanf (FILE * f, char * fmt, ...)
realizează citirea cu format dintr-un fişier; analog scanf

int fprintf (FILE * f, char * fmt, ...)
identică cu printf cu deosebirea că scrie într-un fişier

Pentru aceste funcţii se aplică toate regulile de la funcţiile *scanf* şi *printf*.

Un fişier text prelucrat cu funcţiile *fprintf* şi *fscanf* conţine mai multe câmpuri de date separate între ele prin unul sau mai multe spaţii albe (*blanc, tab, linie nouă*). Conţinutul câmpului de date este scris şi interpretat la citire conform specificatorului de format pentru acel câmp.
Exemplu de creare şi citire fişier de numere:

```
FILE * f;                           // f = pointer la fişier
int x;

// creare fişier de date:
f = fopen("num.txt", "w");          // deschide fişier
for (x=1;x<=100;x++)
  fprintf(f,"%4d", x);              // scrie un numar

fclose(f);                          // inchidere fişier

// citire şi afişare fişier creat:
f=fopen("num.txt", "r");
if ( (f == NULL ) {
    printf ( "Nu se poate deschide la citire fişierul!\n‟ );
    exit (1);
}
```

```
while (fscanf(f,"%d", &x) == 1)          // pana la sfirsit  fişier
  printf("%4d", x);                      // afişare numar citit
```

Exemplu:

Într-un fişier de tip text sunt păstrate valorile reale ale unei măsuratori sub forma:

nr_măsuratori
val1
val2
val3 ...

Să se scrie programul care afişează numărul de măsurători şi valorile respective.
Se vor adăuga la fişier noi măsuratori până la introducerea valorii 0.
Rezolvare:

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define MAX  100

void afiseaza(double *mas,int nrm){
  int i;
  for (i=0; i<nrm; i++)
    printf ("Masuratoarea %d = %6.2e\n", i+1, mas[i]);
}

void loadmas (FILE *fp, int *nrm, double *mas){
  int i=0;
  fscanf (fp,"%d", nrm);
  if (*nrm>MAX) *nrm = MAX;
  for ( ; i<*nrm; i++)
    fscanf (fp, "%lf", &mas[i]);
}

int main(){
  FILE *fp;
  double masur[MAX], mas_noua=1;
  char nume_fis[12];
  int nr_mas;

  printf ("nume fisier:");
  gets (nume_fis);
  if ( (fp = fopen(nume_fis, "r+") ) == 0 ){
    printf ("nu exista fisierul\n");
    exit (1);
  }
  loadmas (fp,&nr_mas,masur);
  afiseaza (masur,nr_mas);
  fclose(fp);

  if ( (fp = fopen(nume_fis, "a") ) == 0 ){
    printf ("nu exista fisierul\n");
    exit (1);
```

```
   }
   printf ("\nmasuratori noi:\n”);
   while( nr_mas++<MAX-1){
      scanf(“%lf”,&mas_noua);
      if(mas_noua) fprintf (fp, "%lf\n", mas_noua);
      else break;
   }
   fclose(fp);

   if ( (fp = fopen(nume_fis, "r+") ) == 0 ){
      printf ("nu exista fisierul\n");
      exit (1);
   }
   fprintf (fp, "%d", --nr_mas);
   fclose (fp);
   return 0;
}
```

IB.11.6. Funcţii de citire-scriere pentru fişiere binare

Un fişier binar este format în general din articole de lungime fixă, fără separatori între articole. Un articol poate conţine:

- un singur octet
- un număr binar (pe 2, 4 sau 8 octeţi)
- structură cu date de diferite tipuri

Funcţiile de acces pentru fişiere binare *fread* şi *fwrite* pot citi sau scrie unul sau mai multe articole, la fiecare apelare. Transferul între memorie şi suportul extern se face fără conversie sau editare (adăugare de caractere la scriere sau eliminare de caractere la citire). Prototipuri funcţii intrare/iesire (fişiere binare – b):

size_t fread (void *ptr, size_t size, size_t nmemb, FILE *fp)
Citeşte la adresa **ptr** cel mult **nmemb** elemente de dimensiune **size** din fişierul referit de **fp**

size_t fwrite (void *ptr, size_t size, size_t nmemb, FILE *fp)
Scrie în fişierul referit de **fp** cel mult **nmemb** elemente de dimensiune **size** de la adresa **ptr**

Exemple:

```
int a[10];
fread (a, sizeof(int), 10, fp);
fwrite(a, sizeof(int),10,fp);
```

De remarcat că primul argument al funcţiilor *fread* şi *fwrite* este o adresă de memorie (un pointer): adresa unde se citesc date din fişier sau de unde se iau datele scrise în fişier.
Al doilea argument este numărul de octeţi pentru un articol, iar al treilea argument este numărul de articole citite sau scrise. Numărul de octeţi citiţi sau scrişi este egal cu produsul dintre lungimea unui articol şi numărul de articole.

Rezultatul funcţiilor este numărul de articole efectiv citite sau scrise şi este diferit de argumentul 3 numai la sfârşit de fişier (la citire) sau în caz de eroare de citire/scriere!

Dacă ştim lungimea unui fişier şi dacă este loc în memoria RAM atunci putem citi un întreg fişier printr-un singur apel al funcţiei *fread* sau putem scrie integral un fişier cu un singur apel al funcţiei *fwrite*. Citirea mai multor date dintr-un fişier disc poate conduce la un timp mai bun faţă de repetarea unor citiri urmate de prelucrări, deoarece se pot elimina timpii de aşteptare pentru poziţionarea capetelor de citire – scriere pe sectorul ce trebuie citit (rotaţie disc plus comandă capete).

Programul următor scrie mai multe numere întregi într-un fişier disc (unul câte unul) şi apoi citeşte conţinutul fişierului şi afişează pe ecran numerele citite.

```
int main () {
  FILE * f; int x;  char * numef ="num.bin";
  // creare fişier
  f=fopen(numef,"wb"));                  // fisier in directorul curent
  for (x=1; x<=100; x++)
    fwrite (&x,sizeof(float),1,f);
  fclose(f);

   // citire fişier pentru verificare
  if ( (f=fopen(numef,"rb")) == NULL ) { // fisier in directorul curent
    printf ( "Nu se poate deschide la citire fişierul!\n" );
    exit (1);
  }
  printf("\n");
  while (fread (&x,sizeof(float),1,f)==1)
    printf ("%4d ",x);
  fclose(f);
  return 0;
}
```

Lungimea fişierului *num.bin* este de 200 de octeţi, câte 2 octeţi pentru fiecare număr întreg, în timp ce lungimea fişierului *num.txt* creat anterior cu funcţia *fprintf* este de 400 de octeţi (câte 4 caractere ptr fiecare număr). Pentru alte tipuri de numere diferenţa poate fi mult mai mare.

De obicei articolele unui fişier au o anumită structură, în sensul că fiecare articol conţine mai multe câmpuri de lungimi şi tipuri diferite. Pentru citirea sau scrierea unor astfel de articole în program trebuie să existe (cel puţin) o variabilă structură care să reflecte structura articolelor.
Exemplu de definire a structurii articolelor unui fişier simplu cu date despre elevi şi a funcţiilor ce scriu sau citesc articole ce corespund unor variabile structură:

```
typedef struct {
  char nume[25];
  float medie;
} Elev;

// creare fişier cu nume dat
void creare(char * numef) {
  FILE * f;  Elev s;
  f=fopen(numef,"wb");
  printf ("Nume şi medie ptr. fiecare student: \n");
  while (scanf ("%s %f ", s.nume, &s.medie) != EOF)
    fwrite(&s,sizeof(s),1,f);
  fclose (f);
}
```

```
// afişare conţinut fişier pe ecran
void listare (char* numef) {
   FILE * f;  Elev e;
   if ( (f=fopen(numef,"rb")) == NULL ) { // fisier in directorul curent
     printf ( "Nu se poate deschide la citire fişierul!\n" );
     exit (1);
   }
   while (fread (&e,sizeof(e),1,f)==1)
     printf ("%-25s %6.2f \n",e.nume, e.medie);
   fclose (f);
}

// adaugare articole la sfârşitul unui fişier existent

void adaugare (char * numef) {
   FILE * f;  Elev e;
   if ( (f=fopen(numef,"ab")) == NULL ) { // fisier in directorul curent
     printf ( "Nu se poate deschide la citire fişierul!\n" );
     exit (1);
   }
   printf ("Adaugare nume şi medie:\n");
   while (scanf ("%s%f", e.nume, &e.medie) != EOF)
     fwrite (&e, sizeof(e), 1, f);
   fclose (f);
}
```

IB.11.7. Funcţii pentru acces direct la datele dintr-un fişier

Accesul direct la date dintr-un fişier este posibil numai pentru un fişier cu articole de lungime fixă şi înseamnă posibilitatea de a citi sau scrie oriunde într-un fişier, printr-o poziţionare prealabilă înainte de citire sau scriere. Fişierele mari care necesită regăsirea rapidă şi actualizarea frecventă de articole vor conţine numai articole de aceeaşi lungime.

În C poziţionarea se face pe un anumit octet din fişier, iar funcţiile standard permit accesul direct la o anumită adresă de octet din fişier. Funcţiile pentru acces direct din *stdio.h* permit operaţiile următoare:

- Poziţionarea pe un anumit octet din fişier (*fseek*).
- Citirea poziţiei curente din fişier (*ftell*).
- Memorarea poziţiei curente şi poziţionare (*fgetpos*, *fsetpos*).

Poziţia curentă în fişier este un număr de tip *long*, pentru a permite operaţii cu fişiere foarte lungi. Poziţia se obţine printr-un apel al funcţiei *ftell*:

long int ftell (FILE *fp)

Întoarce valoarea indicatorului de poziţie

- pentru fişier binar: numărul de octeţi de la începutul fişierului
- pentru fişier text: o valoare ce poate fi utilizată de *fseek* pentru a seta indicatorul de poziţie în fişier la această poziţie.

Funcţia *fseek* are prototipul următor :

int fseek (FILE *fp, long int offset, int poziţie)

poziţionează indicatorul de poziţie la valoarea dată de *offset* faţă de:

- SEEK_SET sau 0 ⇔ **începutul fişierului**

- SEEK_CUR sau 1⇔ **poziţia curentă**
- SEEK_END sau 2 ⇔ **sfârşitul fişierului**

Offset reprezintă numărul de octeţi faţă de punctul de referinţă.

Exemple:

- poziţionarea la sfârşitul fişierului: **fseek (fp, 0, SEEK_END)**
- poziţionarea la caracterul precedent: **fseek (fp, -1, SEEK_CUR)**
- poziţionarea la inceputul fişierului: **fseek (fp, 0, SEEK_SET)**

Atenţie! Poziţionarea relativă la sfârşitul unui fişier nu este garantată nici chiar pentru fişiere binare, astfel că ar trebui evitată!

Ar trebui evitată şi poziţionarea faţă de poziţia curentă cu o valoare negativă, care nu funcţionează în toate implementările!

Funcţia *fseek* este utilă în următoarele situaţii:

- Pentru repoziţionare pe început de fişier după o căutare şi înainte de o altă căutare secvenţială în fişier (fără a închide şi a redeschide fişierul)
- Pentru poziţionare pe începutul ultimului articol citit, în vederea scrierii noului conţinut (modificat) al acestui articol, deoarece orice operaţie de citire sau scriere avansează automat poziţia curentă în fişier, pe următorul articol.
- Pentru acces direct după conţinutul unui articol (după un câmp cheie), după ce s-a calculat sau s-a găsit adresa unui articol cu cheie dată.

Într-un fişier text poziţionarea este posibilă numai faţă de începutul fişierului, iar poziţia se obţine printr-un apel al funcţiei *ftell*.

Modificarea conţinutului unui articol (fără modificarea lungimii sale) se face în mai mulţi paşi:

- Se caută articolul ce trebuie modificat şi se reţine adresa lui în fişier (înainte sau după citirea sa);
- Se modifică în memorie articolul citit;
- Se readuce poziţia curentă pe începutul ultimului articol citit;
- Se scrie articolul modificat, peste conţinutul său anterior.

Exemplu de secvenţă pentru modificarea unui articol:

```
pos=ftell (f);
fread (&e,sizeof(e),1,f );          // poziţia inainte de citire
. . .
      // modifica ceva in variabila e
fseek (f,pos,0);                    // repoziţionare pe articolul citit
fwrite (&e,sizeof(e),1,f);          // rescrie ultimul articol citit
```

Memorarea poziţiei curente sau poziţionarea se pot realiza utilizând următoarele funcţii:

int fgetpos (FILE *fp, fpos_t *poziţie)

- **memorează starea curentă a indicatorului de poziţie al fluxului referit de *fp* în poziţie;**
- **întoarce 0 dacă operaţia s-a realizat cu succes!**

int fsetpos (FILE *fp, const fpos_t *poziţie)

- **setează indicatorul de poziţie al fluxului referit de *fp* la valoarea data de poziţie**

void rewind (FILE *fp)

- **setează indicatorul de poziţie al fluxului referit de *fp* la începutul fişierului**

Exemplu:

Funcţie care modifică conţinutul mai multor articole din fişierul de elevi creat anterior.

```
// modificare conţinut articole, dupa cautarea lor
void modificare (char * numef) {
   FILE * f;
   Elev e;
   char nume[25];
   long pos;
   int  ef;

   if ( (f=fopen(numef,"rb+")) == NULL ) {// fisier in directorul curent
     printf ( "Nu se poate deschide la citire fişierul!\n" );
     exit (1);
   }

   do {
     printf ("Nume cautat: ");
     scanf ("%s",nume);
     if (strcmp(nume, ".") == 0)  break; // datele se termină cu un punct

     // cauta "nume" în fişier
     fseek (f, 0, 0);                    // readucere pe inceput de fişier
     while ( (ef=fread (&e, sizeof(e), 1, f)) ==1 )
       if (strcmp (e.nume, nume)==0) {
            pos= ftell(f) - sizeof(e);
            break;
       }
     if ( ef < 1) break;

     printf ("noua medie: ");
     scanf ("%f", &e.medie);

     fseek (f, pos, 0);             // pozit. pe inceput articol gasit
     fwrite (&e, sizeof(e), 1, f);  // rescrie articol modificat
   } while (1);

   fclose (f);
}

int main(){
   char name[]="c:elev.txt";
   creare (name);
   listare (name);
   adaugare (name);
   modificare (name);
   listare (name);
   return 0;
}
```

IB.11.8. Fişiere predefinite

Există trei ***fluxuri predefinite***, care se deschid automat la lansarea unui program:

- ***stdin*** - fişier de intrare, text, este intrarea standard - tastatura
- ***stdout*** - fişier de ieşire, text, este ieşirea standard - ecranul monitorului.
- ***stderr*** - fişier de iesire, text, este ieşirea standard de erori - ecranul monitorului.

Ele pot fi folosite în diferite funcţii, un exemplu practic este funcţia *fflush* care goleşte zona tampon (*buffer*) asociată unui fişier.

Observaţii

- Nu orice apel al unei funcţii de citire sau de scriere are ca efect imediat un transfer de date între exterior şi variabilele din program!
- Citirea efectivă de pe suportul extern se face într-o zonă tampon asociată fişierului, iar numărul de octeţi care se citesc depind de suport: o linie de la tastatură, unul sau câteva sectoare disc dintr-un fişier disc, etc.
- Cele mai multe apeluri de funcţii de I/E au ca efect un transfer între zona tampon (anonimă) şi variabilele din program.
- Este posibil ca să existe diferenţe în detaliile de lucru ale funcţiilor standard de citire-scriere din diferite implementări (biblioteci), deoarece standardul C nu precizează toate aceste detalii!

Funcţia *fflush*

Are rolul de a goli zona tampon folosită de funcţiile de I/E, zonă altfel inaccesibilă programatorului C. Are ca argument variabila pointer asociată unui fişier la deschidere, sau variabilele predefinite *stdin* şi *stdout*.

```
fflush (FILE* f);
```

Exemple de situaţii în care este necesară folosirea funcţiei *fflush*:

- Citirea unui caracter după citirea unui câmp sau unei linii cu *scanf* :

```
int main () {
   int n;
   char s[30];
   char c;

   scanf ("%d",&n);                  // sau  scanf("%s",s);
   // fflush(stdin);                 // pentru corectare

   c = getchar();                    // sau  scanf ("%c", &c);
   printf ("%d \n",c);               // afiseaza codul lui c
   return 0;
}

/*  va afişa 10 care este codul numeric al caracterului terminator de
linie \n, în loc să  afişeze codul caracterului c, deoarece după o
citire cu scanf în zona tampon rămân unul sau câteva caractere
separator de câmpuri ('\n', '\t', ' '), care trebuie scoase de acolo
prin fflush(stdin) sau prin alte apeluri scanf.  */
```

- Funcţia *scanf* opreşte citirea unei valori din zona tampon ce conţine o linie la primul caracter separator de câmpuri sau la un caracter ilegal în câmp (de ex. literă într-un câmp numeric)!
  In cazul repetării unei operatii de citire (cu *scanf*) după o eroare de introducere în linia anterioară (caracter ilegal pentru un anumit format de citire) în zona tampon rămân caracterele din linie care urmau după cel care a produs eroarea!

```
do {
   printf ("x, y = ");
   err = scanf ("%d%d", &x, &y);
   if ( err == 2 ) break;
   fflush (stdin);
} while (err != 2);
```

  Observaţie: După citirea unei linii cu funcţiile "gets" sau "fgets" nu rămâne nici un caracter în zona tampon şi nu este necesar apelul lui "fflush"!

- Se va folosi periodic *fflush* în cazul actualizării unui fişier mare, pentru a evita pierderi de date la producerea unor incidente (toate datele din zona tampon vor fi scrise efectiv pe disc):

```
int main () {
     FILE * f;
     int c;
     char numef[]="TEST.DAT";
     char x[ ] = "0123456789";

     f=fopen (numef,"w");
     for (c=0;c<10;c++)
       fputc (x[c], f);
     fflush (f);      // sau fclose(f);

     f=fopen (numef,"r");
     while ( (c=fgetc(f)) != EOF)
       printf ("%c", c);
     return 0;
}
```

IB.11.9. Redirectarea fişierelor standard

Trebuie observat că programele C care folosesc funcţii standard de I/E cu consola pot fi folosite, fără modificări, pentru preluarea de date din orice fişier şi pentru trimiterea rezultatelor în orice fişier, prin operaţia numită redirectare (redirecţionare) a fişierelor standard. Prin redirectare, fişierele standard se pot asocia cu alte fişiere.

Redirectarea se face prin adăugarea unor argumente în linia de comandă la apelarea programului.
Exemplu:

```
fişier_exe < fişier_1  > fişier_2
```

În acest caz, preluarea informaţiilor se face din fişier_1, iar afişarea informaţiilor de ieşire se face în fişier_2.
Exemple:

```
/*
* Copierea conținutului unui fişier în alt fişier utilizand redirectarea
* Folosind redirectarea fişierelor standard, se va lansa printr-o linie de
* comandă de forma:
*    copiere1 <fişier_sursa.dat >fişier_dest.dat
* Şi atunci următorul program va avea acelaşi rezultat ca si când s-ar citi
* din fisier_sursa.dat   * şi s-ar scrie în fişier_dest.dat
*/

#include <stdio.h>
int main(void){
  char c;
  while ( (c=getchar()) != EOF )
    putchar(c);
  return 0;
}
```

```
/*
* Exemplu de program "filter":
* filter este numele unui program (fişier executabil) care aplică un filtru
* oarecare pe un text pentru a produce un alt text
* Folosind redirectarea fişierelor standard, se va lansa printr-o linie de
* comandă de forma:
* filter <input  -  citire din input şi scriere pe ecran
* filter >output  - citire de la consola şi scriere în output
* filter <input  >output -  citire din input şi scriere în output
*/

#include <stdio.h>
// pentru funcțiile gets,puts
int main () {
  char line[256];                        // aici se citeşte o linie
  while ( gets(line) != NULL)            // repeta citire linie
  if ( line[0]=='/' && line[1]=='/')     // daca linie comentariu
                puts (line);             // atunci se scrie linia
  return 0;
}
```

Utilizarea comenzii *filter* fără argumente citeşte şi afişează la consolă; utilizarea unui argument de forma <*input* redirectează intrările către fişierul *input*, iar un argument de forma >*output* redirectează ieşirile către fişierul *output*.

Redirectarea se poate aplica numai programelor care lucrează cu fişiere text.

IB.11.10. Anexa. Fişiere în C++

Fişierele sunt în C++ variabile de tipurile ***ifstream*** (*input file stream*), ***ofstream*** (*output file stream*) sau ***fstream*** (care permit atât citire cât şi scriere din fişier).
Operaţiile de citire/scriere se pot realiza fie prin funcţii specifice, fie prin operatorii de inserţie în flux (<<) sau extragere din flux (>>). Pentru a putea fi folosite, fişierele disc trebuie deschise, utilizând funcţia *open* şi închise după folosire utilizând funcţia *close*.

Exemplu de scriere într-un fişier text:

```
ofstream ofile;
char nr[4];
ofile.open ("numere.txt");
for (int i=1;i<100;i++)
  ofile << itoa (i,nr,10)<< endl;
ofile.close();
```

La citirea dintr-un fişier text există două diferenţe faţă de scriere:

- Trebuie detectat sfârşitul de fişier cu una din funcţiile ***eof(), good()*** sau ***bad().***
- Citirea cu operatorul >> repetă ultima linie citită din fişier şi de aceea se preferă funcţia ***getline*** (cu argument de tip *string* şi nu vector de caractere).

Exemplu de citire din fişierul text creat anterior:

```
ifstream ifile; char nr[4];
ifile.open ("numere.txt");
while (ifile.good()) {              // while ( ! ifile.eof()) {
  ifile >> nr;
  cout << nr << endl;
}
ifile.close();                      // poate lipsi
```

Se poate verifica dacă deschiderea fişierului a reuşit cu funcţia ***is_open()*** :

```
if (! ifile.is_open()) cout << "eroare la deschidere\n";
```