LUCRAREA 11

Scopul lucrării *îl reprezintă studiul detaliat al operațiilor de intrare/ieşire (operații I/O) evidențiind capacitatea bibliotecii standard C de a lucra cu fişierele.*

I. OBSERVAȚII TEORETICE

1.1. Dificultatea programelor. Scurtă clasificare

Un program poate fi scris într-o serie de variante care acoperă plaja de dificultate [uşor, complex]. Între extreme există o clasificare de tip întreg, adică există câteva grade de dificultate precise (cuantificabile).

Astfel, un program este uşor în măsura în care operațiile pe care le cuprinde implică funcții sau operații de bază: citire/scriere la ecran, calcule aritmetice simple (sumă, produs, medii de diferite tipuri) sau probleme a căror rezolvare este imediată (prin folosirea directă a unei relații de calcul). Ca timp de calcul, măsurat în număr de operații astfel de probleme nu depăşesc o comportare liniară în timp, adică o lege de evoluție de forma:

$T(n) = c*n,$

unde c este o constantă, iar n este numărul de operații avut în vedere, în general entitatea care modelează cel mai bine problema în cauză, în sensul că rezolvarea depinde în mod esențial de acea entitate (de exemplu gradul polinomului - în situația operațiilor cu polinoame, sau ordinul vectorului sau matricei pentru operațiile cu vectori uni- şi bi-dimensionali). Dependența este, deci, liniară în funcție de gradul de intrare al problemei, reprezentat de n.

Alegerea acestui n cade în sarcina programatorului, acesta trebuind să-şi dea cel mai bine seama ce noțiune este definitorie pentru problema pe care o rezolvă.

Programele sunt considerate medii dacă pe lângă operațiile anterioare conțin şi construcții de funcții proprii, sunt compuse din module-program (adică părți care se ocupă cu operații specifice) şi atacă probleme cu un grad mai ridicat de dificultate (cum ar fi problemele legate de structurile de date abstracte: lista, stiva, coada, arborele, împreună cu operațiile specifice acestora). Comunicația modulelor programului nu este suficientă dacă nu există şi o interacțiune cu exteriorul a acelui program. Exterior înseamnă dispozitivele de stocare (FDD, HDD) sau legătura cu porturile de comunicație (serial, paralel, mai nou USB) prin care sunt conectate dispozitive externe de lucru (mouse, imprimanta, scanner, dispozitive de stocare externe ş.a.)

Gradul ridicat de dificultate este definit în timp drept polinomial adică timpul de calcul depinde polinomial de numărul de operații de la intrare, denumit de noi n.

$T(n) = c_1*n^k + c_2*n^{k-1} + \ldots + c_k*n$

La limită, pentru n foarte mare, termenul dominant este cel de grad maxim al lui n, adică primul în scrierea anterioară.

Programele dificile sunt cele pentru care timpul de calcul este exponențial, adică evoluția în timp este foarte costisitoare odată cu modificarea gradului de intrare.

Expresia timpului de calcul este:

$T(n) = c*exp(n)$

sau

$T(n) = c*b^n$

unde baza b este o valoare întreagă (2, 3 etc.), iar n poate fi şi număr real (de exemplu numărul e=2.71828).

Astfel de algoritmi sunt de exemplu algoritmii de grafuri, cei de sortare (de exemplu sortarea prin metoda bulelor sau sortarea Shell), cei de optimizare sau, mai recent, algoritmii legați de securitatea transmisiei prin InterNet (cum sunt cei implicați în criptare sau semnătura digitală strâns legați de aritmetica numerelor mari - numerele întregi având un număr de poziții (digiţi) relativ mare, de exemplu 25, 48 sau 52).

Aşadar, interacțiunea cu exteriorul este presupusă implicit pentru orice program începând cu gradul de dificultate mediu inclusiv. Justificarea acestei lucrări de laborator este, astfel, deplină.

1.2. Noțiunea de operație de intrare/ieşire

Fişierul este expresia comună a interacțiunii cu exteriorul, adică cu dispozitivele de stocare. Discuția acestei lucrări este preferențială, în sensul că se vor discuta operațiile specifice fişierelor stocate pe hard-disk (notat pe scurt HDD) sau floppy-disk (pe scurt FDD).

Fişierul este entitatea în care se pot scrie (salva) date, sau din care se pot citi (obține) informații. Locația fişierelor este indisolubil legată de mediile de stocare, anume HDD, FDD sau dispozitivele de stocare externă cu conectare pe port USB, adică oriunde se poate simula o organizare arborescentă, de tip tată-fiu. Prin tată se înțelege rădăcina (root) ierarhiei de stocare, iar prin fiu se înțelege o ramură în această ierarhie, denumită director (directory) sau dosar (folder). Fişierul (file) este un nod frunză al acestei ierarhii, constituind de fapt conținutul unui folder sau director. Un director (folder) poate conține la rândul său alte directoare (folder-e) adică structura arborescentă de care vorbim este recursivă (se defineşte şi prin termeni de aceeaşi natură cu cel de bază). Un director poate conține mai multe fişiere, situație în care un nod are numai frunze, referindu-ne la structura arborescentă utilizată pentru modelarea discuției.

Fişierul poate avea un conținut binar (de exemplu forma fişierelor obiect rezultate în urma compilării codului sursă) sau un conținut de tip ASCII (text obişnuit) cum este chiar fişierul sursă al unui program.

În funcție de tipul conținutului există anumite convenții de lucru, pe care le vom vedea în continuare.

1.3. Conceptul de flux (stream)

Compilatorul C/C++ lucrează cu fişierele indirect prin intermediul conceptului de flux (stream). Acest concept este definit ca o interfaţă între codul sursă (apelurile de funcții specifice fişierelor) şi entitatea fizică fişier, existentă pe HDD sau FDD. Interfața asigură şi legătura creată între programul C şi entitatea fizică. Această legătură există din momentul creării sale de către utilizator şi până în momentul închiderii sale, tot de către utilizator.

Această interfață reprezintă un mod flexibil de lucru, după cum se va vedea în continuare. Se asigură şi o independență (transparenţă) față de sistemul de operare pe care va rula programul. Există o operație de start a fluxului şi una de oprire. Între cele două momente obligatorii pentru sensul lucrului se desfăşoară operațiile de rutină asupra fişierelor, anume: scrieri (write), citiri (read), adăugări (append), înlocuiri (replace) sau actualizări (update).

Acestea sunt de altfel şi operațiile fundamentale asupra fişierelor, despre care ne ocupăm în paragraful următor.

Operațiile de citire/scriere şi adăugare modifică ceea ce se numeşte indicator de poziție al fişierului. Fiecare fişier are asociat un indicator (index) prin care se arată locul (poziția) în cadrul fişierului (calculat ca offset (deplasare) față de începutul fişierului) unde s-a ajuns în urma unei operații asupra acestuia. Există funcții care stabilesc acest indicator la o anumită poziție (fsetpos()), sau îl aduc la valoarea de start (adică atunci când indică începutul (rewind()), sau caută o anumită poziție în cadrul fişierului (fseek()).

Observație:
Există **regula** care spune că dacă un fişier a fost deschis pentru operații de citire/scriere atunci o operație de scriere nu poate fi urmată imediat de una de citire dacă nu există un apel la funcțiile legate de indicatorul poziției în fişier, ca nici reciproca: citire urmată imediat de scriere.

1.4. Constanta EOF

Numeroase funcții de lucru asupra fluxurilor, pe care le vom studia în cele ce urmează au ca valoare de retur pe caz de eroare constanta EOF (End Of File). Aceasta este o convenție prin care se marchează atingerea sfârşitului fişierului în urma diferitelor operații. O convenție similară a apărut în cazul vectorilor de caractere (şirurilor) unde exista constanta '\0' care marca sfârşitul şirului.

Sensul este clar: nu mai pot urma operații de prelucrare (citire, scriere sau adăugare) până ce nu se modifică valoarea indicatorului de poziție a fişierului.

Iată ce este prezentat în `stdio.h` pentru `Developer C++`:

```
/* Returned by various functions on end of file condition or error */
#define   EOF    (-1)
```

Iar pentru `Borland C/C++`:

```
* End-of-file constant definition*/
#define   EOF    (-1)   /* End of file indicator */
```

Se observă compatibilitatea perfectă între cele două definiții. Aceasta din cauza standardizării limbajului C.

1.5. Operațiile permise asupra fişierelor şi funcțiile asociate. Entitatea `FILE`

După cum am anunțat operațiile asociate fişierelor sunt:
a) citire (`read`),
b) scriere (`write`),
c) adăugare (`append`),
d) înlocuire/actualizare (`replace/update`).

Funcțiile asociate acestor operații sunt:
a) `fread()` sau `fopen()` cu argument `r/r+`
b) `fopen()` cu argument `w/w+`
c) `fwrite()` sau `fopen()` cu argument `a/a+`
d) `fopen()` cu argument `w/w+` (operație similară oarecum, cu scrierea).

Pentru fiecare dintre paragrafele următoare există câte un program asociat.

Rețineți!
Toate funcțiile ce urmează au prototip în `stdio.h`.

Toate informațiile despre un flux sunt stocate de către biblioteca standard într-o entitate de tip `FILE`. Această entitate este de fapt un tip utilizator creat folosind operatorul `typedef` împreună cu tipul de dată `struct`. Definiția lui apare în fişierul `stdio.h`.

Prezentăm această definiție pentru două generații de compilatoare: `BorlandC/C++ 3.1` şi `Developer C++`, acesta din urmă fiind mediul predominant de lucru din laborator.

Pentru `BorlandC/C++` versiunea 3.1 `stdio.h` ne oferă următoarele:

```
/* Definition of the control structure for streams*/
typedef struct  {
   int level;               /* fill/empty level of buffer */
   unsigned flags;          /* File status flags */
```

```
    char fd;                    /* File descriptor */
    unsigned char  hold;        /* Ungetc char if no buffer   */
    int bsize;                  /* Buffer size */
    unsigned char  _FAR *buffer;     /* Data transfer buffer */
    unsigned char  _FAR *curp;/* Current active pointer */
    unsigned istemp;            /* Temporary file indicator */
    short token;                /* Used for validity checking */
} FILE;         /* This is the FILE object */
```

Iar pentru mediul - mai nou - de programare `Developer C++`:

```
typedef struct _iobuf
{
   char*  _ptr;
   int    _cnt;
   char*  _base;
   int    _flag;
   int    _file;
   int    _charbuf;
   int    _bufsiz;
   char*  _tmpfname;
} FILE;
```

Câmpurile structurii reprezintă noțiuni tipice mediilor de programare dar mai ale sistemului de operare, care interacționează, la rândul lui, tot indirect cu fișierele. Pentru fiecare fișier cu care se lucrează există deschis un `stream`, operație echivalentă cu modificarea în timp real a câmpurilor structurii prezentate. Nu detaliem aceste noțiuni ele nefiind esențiale pentru discuția purtată în acest moment asupra fișierelor.

1.5.1. Funcția `fopen()`

Crearea legăturii virtuale între program și entitatea fizică denumită fișier se face prin apelul funcției `fopen()`, care are definiția următoare în fișierul header `stdio.h`. Această funcție arată astfel:

```
FILE* fopen (const char *filename, const char *mode);
```

Această funcție generează o adresă a unei structuri de tip `FILE`. Ceea ce înseamnă că ea pregătește operațiile tipice fișierelor. Acest canal de interacțiune devine valabil și rămâne astfel până în momentul închiderii lui explicite, cu funcția `fclose()`, sau până în momentul încheierii programului, situație în care, neexistând un apel la `fclose()`, este prevăzut un mod de tratare implicit din partea mediului de programare, anume închiderea automată prin apel de `fclose()` a tuturor fluxurilor (`stream`) rămase deschise.

Argumentele acestei funcții sunt:

- `filename` - arată calea către/de unde va fi salvat/există fișierul în cauză;
- `mode` - specifică varianta în care se fac prelucrările asupra fișierului, anume:
    - `r` = citire (`read`);
    - `w` = scriere (`write`);

- a = adăugare (append) la sfârşitul fişierului; un fişier în mod append poate fi scris numai la sfârşitul său;
- r+ = deschide fişier existent pentru actualizare (scriere şi citire);
- w+ = creează fişier pentru actualizare;
- a+ = deschidere pentru actualizarea la sfârşitul fişierului.

Există un comportament predefinit al mediului de programare în ce priveşte lucrul asupra fişierelor. Astfel, dacă un fişier cu nume specificat nu există pe disc, atunci: dacă argumentul mode este w sau a (cu variantele w+ şi a+) el este nou creat, iar dacă argumentul mode este r se generează eroare (nu pot citi dintr-un fişier inexistent). Dacă fişierul există, atunci deschiderea cu mode w conduce la golirea sa (pierderea conţinutului său).

Un fişier se poate prelucra în modul text dacă se adaugă sufixul t la variantele mode existente. De exemplu: wt sau a+t sau at+, sau în mod binar dacă este adăugat, pe acelaşi principiu, sufixul b (de exemplu rb, wb sau w+b sau ab+).

Dacă nu se specifică modul de deschidere, atunci compilatorul va lua decizia pe baza valorii variabilei globale _fmode (cu valori O_BINARY sau O_TEXT) din headerul fcntl.h.

Iată ce se specifică , în fişierul fcntl.h al compilatorului Developer C++:

```
/* NOTE: Text is the default even if the given _O_TEXT bit is not on. */
#define    _O_TEXT 0x4000        /* CR-LF in file becomes LF in memory. */
#define    _O_BINARY     0x8000  /* Input and output is not translated. */
#define    _O_RAW _O_BINARY
```

Se vede că modul implicit de lucru asupra fişierelor este modul text. Se poate schimba aceasta prin specificarea sufixului b, după cum am arătat.

Iar pentru compilatoarele din familia Borland C/C++ există următoarele definiţii:

```
/* MSDOS special bits */
…
#define O_TEXT         0x4000       /* CR-LF translation */
#define O_BINARY       0x8000       /* no translation */
```

La fişierele text o succesiune de linii este separată prin caractere newline (NL) care pot fi convertite în perechea CR-LF (carriage return împreună cu line feed), în timp ce pentru fişierele binare nu se face această conversie.

Returnează un pointer NULL în caz de eroare (generată de diferite cauze) sau pointerul la FILE în caz de succes.

1.5.2. Funcţia fclose()

Este funcţia opusă lui fopen(). Rolul său este de a elibera canalul de comunicaţie existent (fluxul) către o anumită structură de tip FILE. Argumentul pe care o specificăm este pointerul la acea structură.

Sintaxa este preluată din fişierul `stdio.h`:

```
int fclose(FILE *__stream);         // Borland C/C++
```

sau

```
intfclose (FILE *flux);             // Developer C++
```

Argumentul este deci numele fluxului deschis anterior cu `fopen()`. Este de menţionat un amănunt tehnic. La orice deschidere de flux (`stream` sau canal de comunicaţie) se rezervă şi o `zonă de memorie tampon` pentru acel flux. Închiderea unui flux are ca efect şi golirea acelui tampon (cu termen de specialitate `flushing`), adică scrierea în fişier a datelor din memorie, odată cu eliberarea zonei de memorie astfel rezervate.

După cum se observă închiderea unui flux este importantă, având un dublu efect dorit: eliberarea zonei de memorie pentru alte operaţii, împreună cu scrierea pe disc a informaţiilor rămase în memorie (`flushing of buffer`).

În caz de reuşită funcţia întoarce `0`. La eşec returnează `EOF` (`End Of File`).

1.5.3. Funcţia `freopen()`

Cu ajutorul acestei funcţii se schimbă asocierea unui flux existent. Astfel, dacă se deschid cu `fopen()` două fluxuri `A` şi `B`, atunci prin apelul `freopen()` se poate asocia fluxul `B` celui dedicat iniţial lui `A`. Adică se atribuie un flux existent altui fişier. Înainte de schimbarea propriu-zisă, `freopen()` încearcă închiderea fişierului asociat în mod curent fluxului căruia îi va schimba asocierea. Şi în caz de reuşită şi în caz de eşec a încercării de închidere a fluxului căruia îi încearcă schimbarea asocierii, `freopen()` va deschide celălalt fişier.

Ca utilitate principală este `redirecţionarea` fluxurilor standard `stdin` (tastatura), `stdout` (ecranul) şi `stderr` (dispozitivul. standard de eroare) către un (alt) fişier.

Sintaxa este:

```
FILE* freopen (const char *numeFis, const char *mode, FILE *flux);
```

Noul nume de fişier este `numeFis`, modul de acces este dat prin `mode`, iar fluxul ce urmează a fi reatribuit este `flux`.

Dacă există vreo eroare, funcţia întoarce un pointer `NULL`. La succes se întoarce pointerul către `flux`.

1.5.4. Funcţia `fgetc()`

Funcţia `fgetc()` întoarce următorul caracter (octet) din fluxul de intrare specificat ca argument:

```
int fgetc(FILE flux);
```

Incrementează apoi indicatorul de poziţiei al fişierului. Caracterul este citit ca un `unsigned char`, convertit apoi la un întreg.

La întâlnirea sfârşitului de fişier funcţia întoarce `EOF` (`End Of File`), la fel cu întâlnirea unei erori neaşteptate.

Observaţii:

În cazul fişierelor `binare`:

- deoarece `EOF` este codificat ca un întreg (v. `paragraful 1.4.`), iar un fişier binar este o mulţime de octeţi, deci prin asimilare variabile de tip întreg cu semn, trebuie folosită funcţia `feof()` pentru testarea sfârşitului de fişier;
- pentru situaţiile de eroare neaşteptate, deoarece funcţia întoarce tot `EOF` (ca şi în cazul întâlnirii sfârşitului de fişier) trebuie folosită funcţia `ferror()` pentru detectarea erorilor de fişier.

Observaţie:

În unele implementări apare şi funcţia `getc()`. Aceasta din urmă este identică cu `fgetc()`.

Nu uitaţi de observaţia cu care se încheie `paragraful 1.3`.

1.5.5. Funcţia `fputc()`

Scrie caracterul preluat ca prim argument în fişierul al cărui flux este preluat drept al doilea argument. Incrementează apoi indicatorul de poziţie în cadrul fişierului.

```
int fputc (int ch, FILE *flux);
```

Intern, funcţia converteşte caracterul preluat ca `int` la `unsigned char`. Funcţia întoarce valoarea caracterului scris, în caz de succes şi `EOF` în caz de eroare.

Ca şi la `paragraful 1.4.4.` trebuie făcută aceeaşi `observaţie`: pentru `fişierele binare`, `EOF` este un caracter valid, deci trebuie folosită funcţia `ferror()` pentru detectarea eventualelor erori legate de fişier.

Observaţie:

În unele implementări apare şi funcţia `putc()`. Aceasta din urmă este identică cu `fputc()`.

Nu uitaţi nici de observaţia cu care se încheie `paragraful 1.3`.

1.5.6. Funcţia `fgets()`

Sintaxa este:

```
char *fgets (char *sir, int num, FILE *flux);
```

Această funcţie preia maxim `num-1` caractere din fluxul de intrare `flux` şi le salvează într-un vector de caractere indicat prin `sir`. După ultima scrierea în vectorul de caractere se face automat adăugarea caracterului de sfârşit de şir (`'\0'`).

În caz de reuşită se întoarce pointerul la vectorul de caractere. În caz de eroare returnează un pointer NULL. Atunci când atinge sfârşitul fişierului această funcţie întoarce tot un pointer NULL. Se impune aşadar folosirea funcţiilor de tratare a erorilor ferror() şi feof(), pentru distingerea cazului real întâlnit. Aceste funcţii speciale le analizăm în paragrafele 1.5.12.1, 1.5.12.2. şi 1.5.12.3.

Nu uitaţi nici de observaţia cu care se încheie paragraful 1.3.

1.5.7. Funcţia fputs()

Sintaxa:

```
int fputs(const char *sir, FILE *flux);
```

Scrie în flux conţinutul vectorului de caractere specificat prin sir. Valoarea de sfârşit de şir nu este scrisă în fişier.

La succes returnează valori non-negative. La eroare întoarce EOF.

Pentru fişiere de tip text din cauza conversiilor amintite (CR-LF în NL) există posibilitatea ca numărul de caractere existent iniţial în şir să nu corespundă numărului caracterelor efectiv scrise în fişier.

Nu uitaţi nici de observaţia cu care se încheie paragraful 1.3.

1.5.8. Funcţia fread()

Efectul funcţiei este citirea unui număr specificat de obiecte de dimensiune stabilită tot de programator.

Sintaxa este următoarea:

```
size_t fread (void *buf, size_t octeti, size_t numar, FILE *flux);
```

Numărul obiectelor se specifică prin parametrul numar. Dimensiunea unui obiect, în octeţi este dată prin parametrul octeti. Acestea sunt citite din fluxul flux şi sunt stocate în zona tampon indicată prin buf. Ca şi celelalte funcţii de citire/scriere se modifică valoarea indicatorului de poziţie, cu numărul de octeţi citit.

Returnează numărul octeţilor efectiv citiţi. Dacă au fost returnate mai puţine elemente decât cele citite efectiv este sem de eroare. Cauzele sunt fie necunoscute, fie s-a atins sfârşitul fişierului. De aceea este indicată utilizarea funcţiilor de eroare feof() sau ferror().

Dacă se lucrează cu fişiere text pot apărea neconcordanţe între numărul de octeţi cerut şi cel efectiv întors, ceea ce se datorează conversiilor tipice anunţate (combinaţiile CR-LF se convertesc în NL - new line).

1.5.9. Funcţia `fwrite()`

Complementara lui `fread()`. Aceasta scrie `numar` obiecte de dimensiune `octeti`. Elementele sunt preluate din zona tampon (buffer) indicată prin `buf`. Fluxul în care se scrie este `flux`.

Sintaxa este:

```
size_t fwrite(const void *buf, size_t octeit, size_t numar, FILE
*flux);
```

Returnează numărul obiectelor scrise efectiv, care în caz de succes este egal cu numărul solicitat. În situaţia în care nu apare coincidenţă între ce se cere scris şi ce s-a scris înseamnă apariţia unei erori.

Pentru fişiere deschise `în mod text` deşi apar conversiile amintite, identitatea trebuie să se menţină.

1.5.10. Funcţia `fprintf()`

Scrie în fluxul specificat drept argument valorile argumentelor preluate ca parametri, pe baza indicaţiilor din şirul de format.

Sintaxa:

```
int fprintf(FILE *flux, const char *format, …);
```

În caz de succes returnează numărul caracterelor efectiv afişate, iar în caz de eroare generează o `valoare negativă`.

Este o funcţie cu număr variabil de argumente. Numărul maxim al acestora depinde de sistemul pe care este instalat compilatorul. În rest comportamentul şi stilul de lucru sunt aceleaşi ca la clasica funcţie `printf()`.

Nu uitaţi de observaţia cu care se încheie `paragraful 1.3`.

1.5.11. Funcţia `fscanf()`

Are un comportament identic cu celebra `scanf()`, doar că citirea se face de la fluxul specificat.

Sintaxa este:

```
int fscanf (FILE 8flux, const char *format, …);
```

Cele trei puncte arată că funcţia `fscanf()` (ca şi `scanf()`) este o funcţie cu număr variabil de argumente.

Returnează numărul argumentelor cărora li s-au atribuit efectiv valori. Apariţia la ieşire lui `EOF` indică apariţia unei erori înainte de efectuarea primei atribuiri.

1.5.12. Funcţii de tratare a erorilor

1.5.12.1. Funcţia `feof()`

Sintaxa acesteia este:
```
int feof(FILE *flux);
```

Această funcţie verifică poziţia indicatorului de fişier. Dacă acest indicator de poziţie a atins sfârşitul fişierului funcţia returnează o valoare diferită de `0`. O valoare intermediară a indicatorului generează `zero`. Astfel că o parcurgerea unui fişier (`binar`) s-ar putea face într-un ciclu, pe baza condiţiei:

```
…
// deschiderea fluxului asociat fişierului fisier
FILE fisier = fopen("c:\\fisier.c", r);
if(fisier != NULL){
   while(!feof(fisier)) {    // not(0) = 1
         // instrucţiuni de prelucrare asupra fluxului fisier
   }
}
else {    // tratarea erorii necunoscute
   printf("\n Eroare deschidere fişier!");

   // afişarea codului de eroare asociat
   printf("\n Cod eroare: %u", ferror(fisier));
   // părăsirea absolută (imediată) a programului. prototip în stdlib.h:
   // void exit (int) _ATTRIB_NORETURN;
   exit(1);
   }
…
```

Odată ajunşi la sfârşitul unui fişier, eventualele operaţii de citire ulterioare vor returna `EOF` (în mod evident) până la apelarea funcţiei `rewind()` (cu semnificaţia de derulare la început) sau până în momentul în care indicatorului de poziţie i se modifică valoarea, de exemplu prin una din funcţiile `fseek()` sau `fsetpos()`.

Observaţie:
`feof()` este utilă mai ales în lucrul cu `fişiere binare`, în situaţia cărora valoarea `EOF` este şi una validă (octet valid în cadrul fişierului, pe o poziţie oarecare), pe lângă rolul caracterului de control sfârşit de fişier.

1.5.12.2. Funcţia `ferror()`

Sintaxa este:
```
int ferror(FILE *flux);
```
Această eroare încearcă detectarea unei erori de fişier pentru fluxul preluat ca parametru. Dacă returnează zero atunci nu a apărut nici o eroare. O valoare diferită de zero indică o eroare.

Indicatorii de eroare sunt valabili până în momentul în care fluxul asociat fişierului este închis sau până în momentul în care se apelează una din funcţiile `rewind()` sau `clearerr()`.

Pentru natura exactă a erorii se foloseşte funcţia `perror()`.

1.5.12.3. Funcţia `perror()`

Sintaxa:

```
void perror(const char *sir);
```

Realizează o corespondenţă biunivocă între valoarea variabilei globale errno şi un fişier. Apoi scrie şirul `sir` la dispozitivul standard de eroare (`stderr`). Dacă valoarea lui `sir` nu este nulă atunci el este afişat, urmat de două puncte (`:`) şi de mesajul de eroare tipic mediului de programare pentru eroarea apărută. De obicei `stderr` este 'deviat' către ecran (`stdout`).

Şirul `sir` este dat de programator.

1.6. Detalii de programare

Pentru înţelegerea programelor utilizate în lucrare studiaţi `ANEXA` lucrării.

1.7. Comenzile compilatorului

Înainte de compilare codul sursă trebuie salvat (cu combinaţia de taste `CTRL+S`), sau, echivalent, din meniul `File`, cu comanda `Save as…`.

Compilarea se va face cu comanda `CTRL+F9`, iar rularea cu comanda `CTRL+F10`. Compilarea şi rularea se pot executa, succesiv, printr-o singură tastă, anume `F9`. Acţiunile echivalente din meniu: meniul `Execute` comanda `Compile`, respectiv `Compile + Run`.

Observaţii:

1. Orice nouă modificare a codului-sursă trebuie urmată obligatoriu de salvarea acestuia (`CTRL+S`). Abia după aceea pot urma celelalte acţiuni dorite.
2. Dacă nu se realizează aducerea la zi a programului, compilarea şi rularea ce urmează se fac tot pe varianta veche, adică cea existentă pe `HDD` înaintea ultimelor modificări.

II. DESFĂŞURAREA LUCRĂRII

2.1. Lucrul cu fişierele - anume deschiderea unui flux asociat unui fişier - este studiată în programul `lab11_1.cpp`. Sunt exemplificate lucrul atât cu fişiere în `mod text` cât şi în `mod binar`. Conţinutul fişierului în studiu este în prealabil stabilit de programator. Pentru ca operaţiile prezentate să aibă sens atributul `read-only` al fişierului (în caz că este setat) trebuie dezactivat. Aceasta se face cu ajutorul comenzilor `Windows`: click dreapta pe numele fişierului, apoi alegerea comenzii `Properties`, după care debifarea opţiunii `Read-only` (din zona `Attributes`). Un atribut rămas pe `Read-only` generează erori ale mediului de programare.

2.2. Închiderea unui `stream` este analizată în programul `lab11_2.cpp`. Se observă situaţiile în care `fclose()` este prezentă şi absentă. Pe linia `38` există un comentariu care maschează execuţia funcţiei `fclose()`. Renunţaţi la acest comentariu şi reluaţi programul. Veţi observa că nu este marcată nici o eroare. Totuşi, amintiţi-vă de avantajele închiderii explicite a unui flux (v. `paragraful 1.5.2.`): eliberarea zonei de memorie pentru alte operaţii şi scrierea pe disc a informaţiilor rămase în memorie (`flushing of buffer`).

2.3. Schimbarea asocierii unui flux existent este exemplificată în programul `lab11_3.cpp`. Unul dintre fluxurile iniţiale este atribuit celuilalt, după care se efectuează operaţii tipice. În final ambele sunt închise.

Rulaţi programul de cel puţin două ori. Remarcaţi pe `HDD` cum fişierul `test1.txt` conţine mereu textul cel mai recent, în timp ce fişierul `test2.txt` toate celelalte texte date de utilizator la fiecare rulare. Atenţie la modul de deschidere din fiecare caz al fluxurilor `pFis1` şi `pFis2`. Reţineţi aceste aspecte.

Creaţi două noi fluxuri `flux1` şi `flux2`. Interschimbaţi-le asocierea. Citiţi din `flux1` şi scrieţi în `flux2`. Închideţi apoi ambele fluxuri fişier.

2.4. În programul `lab11_4.cpp` se studiază comportamentul funcţie `fgetc()` pentru un `stream` creat de programator. Citirea caracterelor se face până la întâlnirea sfârşitului de fişier (`EOF`) atât pentru `tip text` cât şi pentru `tip binar`.

Observaţi rezultatele finale prin citirea fişierelor rezultate. Marcaţi concluziile pe care le consideraţi utile.

2.5. Programul `lab11_5.cpp` arată comportamentul funcţiei complementare lui `fgetc()`, anume `fputc()`. Fişierul trebuie deschis în `mod actualizare` pentru ca o asemenea operaţie să poată fi valabilă. Se observă şi valorile indicatorului de poziţie în fişier, înainte şi după scrierea caracterelor în fişierul respectiv. Prin citirea fişierelor create pe `HDD` se poate observa cum caracterul `'\n'` fin şirul de scris este interpretat în mod diferit în funcţie de natura fişierului: `text` sau `binar`.

Pentru ca operaţiile prezentate să aibă sens atributul `read-only` al fişierului în caz că există, trebuie înlăturat. Aceasta se face cu ajutorul comenzilor `Windows`: click dreapta pe numele fişierului, apoi alegerea comenzii `Properties`, după care de-bifarea opţiunii `Read-only`. În final `Apply` şi `OK` (sau direct `OK`).

2.6. Preluarea unui şir de caractere dintr-un fişier (funcţia `fgets()`) şi observaţiile asupra indicatorului de poziţie al fişierului sunt urmărite în programul `lab11_6.cpp`. Trebuie avută în vedere conversia posibilă din situaţia fişierelor de `tip text`. De aceea şirul destinaţie trebuie declarat de o dimensiune acoperitoare. Dacă la fiecare linie există un caracter de control `NL`, acesta va fi convertit la citire în două caractere de control: `CR-LF`. Puteţi calcula astfel, aproximativ, numărul de caractere al şirului destinaţie.

Prin observarea rezultatului final se concluzionează: un şir dintr-un fişier (`text` sau `binar`) ce conţine un caracter `new-line` este citit până la întâlnirea acelui caractere de control. `Nu` se trece mai departe, pe linia următoare pentru citirea restului şirului. Modificaţi conţinutul fişierelor şi observaţi comportamentul programului, prin urmărirea în final a conţinutului celor două fişiere.

2.7. Funcţia complementară lui `fgets()` este `fputs()`. Utilizarea acesteia este studiată în programul `lab11_7.cpp`. Ca şi în cazul lui `fgets()` indicatorul de poziţie în fişier este modificat în consecinţă.

Observaţi diferenţa între numărul de caractere scris efectiv şi lungimea reală a celor două şiruri ce au fost scrise.

2.8. Funcţia `fread()` este urmărită în programul `lab11_8.cpp`. Atenţie aici la primul argument - `void *buf`. Trebuie să exise operatorul de conversie explicită, către tipul de dată pe care doriţi să-l citiţi. După cum se poate concluziona, tipul `void*` reprezintă o generalizare, adică 'pointer către orice'. Trebuie remarcate şi conversiile între caracterele de control `NL` şi `CR-LF`.

2.9. Complementara lui `fread()` este `fwrite()`. Studiul său se face în programul `lab11_9.cpp`. Efectele sunt aceleaşi ca la orice funcţie de citire/scriere: se modifică indicatorul de poziţie în fişier, iar conversiile de caractere de control sunt inevitabile pentru fişierele de `tip text`.

2.10. Funcţiile `fprintf()` şi `fscanf()` apar în studiu în programul `lab11_10.cpp`. După cum se cunoaşte de la funcţiile consacrate `printf()` şi `scanf()` comportamentul celor dedicate fişierelor lor este similar. Observaţi şi aici valoarea indicatorului de poziţie.

Condiţia ca `fscanf()` să funcţioneze corect este ca fişierul din care se face citirea acelor numere să conţină pe prima linie valori valide specifice tipului de dată pentru care se face citirea. Aici sunt citite `3` valori reale

simplă precizie şi 2 valori întregi scurt cu semn. Fişierul `test2.txt` trebuie să conţină pe prima linie valori valide.

2.11. Funcţiile de tratare a erorilor legate de fişiere sunt urmărite în programul `lab11_11.cpp`. Erorile sunt provocate pentru a putea urmări diferitele aspecte ale funcţionării acestor funcţii. Fişierul fie nu există şi se doreşte citirea din acesta, fie are atributul `read-only` activ şi se doreşte scrierea, fie se cer mai multe caractere de citit decât există în mod real, depăşindu-se dimensiunea acestuia (atingerea sfârşitului de fişier `EOF`).

Goliţi fişierele `test1.txt` şi `test2.txt` unplute în urma rulărilor multiple ale programului (sau programelor anterioare) şi reluaţi `lab11_11.cpp`. Observaţi în ce situaţii testul de depăşire a indicatorului de poziţie în fişier are condiţia adevărată. Reţineţi apsectele şi situaţiile practice întâlnite.

Modificaţi modul de acces la deschiderea fluxurilor şi încercaţi operaţii opuse modului ales, din punct de vedere logic (de exemplu fişier deschis pentru `citire` în care se doreşte `scrierea`). Observaţi cum operaţiile aflate în contradicţie cu modul de deschidere al fişierului sunt ignorate (nu au efect la rulare).

Activaţi atributul `read-only` al fişierelor, apoi alegeţi un mod de deschidere impropriu şi observaţi comportamentul mediului integrat de programare. Reţineţi situaţiile practice de eroare.

Citiţi şi comentariile programului `lab11_11.cpp`, din secţiunea **V - ANEXĂ** şi urmaţi-le cursul.

III. ÎNTREBĂRI

3.1. Definiţi entitatea fişier. Ce dificultate are programul ce conţine lucrul cu fişiere?
3.2. Cum lucrează `compilatorul C` cu fişierele?
3.3. Ce operaţii sunt permise asupra fişierelor?
3.4. Care sunt tipurile de fişiere şi prin ce diferă între ele? Daţi două exemple din fiecare tip de fişier.
3.5. Ce semnifică un pointer `NULL` în contextul funcţiilor de lucru asupra fişierelor?
3.6. Definiţi constanta `EOF`. Explicaţi sensul acesteia.
3.7. Cum se poate face diferenţa între valoarea utilă `EOF` şi o eventuală eroare de fişier care generează aceeaşi valoare?
3.8. Ce rol are `void*` în situaţia funcţiilor `fread()` şi `fwrite()`? Cum se lucrează corect cu un astfel de pointer? Daţi un exemplu.
3.9. Definiţi indicatorul de poziţie în fişier. Unde este util acest indicator?
3.10. Ce tip de dată special are indicatorul de poziţie?
3.11. Cum puteţi genera eroarea `EOF`? Dar o eroare oarecare în lucrul cu fişierele?

3.12. În ce condiţii o operaţie începută asupra unui fişier se poate continua asupra altuia?

3.13. Concatenaţi două fişiere. Rezolvaţi două situaţii:
- fişierul rezultat este unul nou creat;
- fişierul rezultat este unul dintre cele existente.

IV. PROIECT DE LABORATOR

Creaţi un `editor de text` rudimentar. Singurele operaţii pe care trebuie să le realizeze sunt:
- `deschidere` fişier;
- `operaţii asupra conţinutului` (`înlocuire` de caracter sau cuvânt);
- `salvare` fişier;
- `căutare` în fişier.

Se presupune că se lucrează în `mod text` (`ecran DOS` cu dimensiunile: `80` caractere pe orizontală (`număr coloane`, adică numărul maxim de litere pe un rând) şi `25` pe verticală (`numărul de linii` de text posibile). Liniile sunt separate prin caracterul `NL`, care va fi recunoscut drept caracter de control.

V. ANEXĂ - sursele complete ele programelor

```
// lab11_1.cpp
// Lucrul cu fişierele: deschiderea si inchiderea unui stream
// (flux) - partea 1.
#include<stdio.h>
#include<stdlib.h>     // pentru functia exit()

int main(void){
 char ch, *sirEroare = "Eroare la deschidere fisier";
 int nrChar = 0;// contor al caracterelor
 FILE *pFisier; // pointerul la fisier; numele fluxului

 pFisier = fopen("c:\\test.txt", "W");
 // deschidere pentru scriere. Daca fiserul nu exista el este creat
 // Este util sa fie deschis in mod 'append' (adaugare la sfarsit),
 // pentru ca fisierul sa-si pastreze continutul intre doua
// deschideri.
// Un caracter Enter la intrare se traduce in perechea CR-LF la
// scriere in fisier.
 if (pFisier == NULL){ // eroare la deschidere fisier
   if(ferror(pFisier)) {
         perror(sirEroare);
         exit(1);           // iesire din program
         }
   }
```

```
    else {   // corp de instructiuni_else
      while((ch = getchar()) != 'q'){
            fputc((int)ch, pFisier);
            nrChar++;
            }
    }
    printf("\nNumarul caracterelor scrise: %i", nrChar);
    fclose(pFisier);      //  inchiderea  fluxului  asociat  fisierului
'test.txt'

   // incheiere
    int i;
    scanf("%i", &i); return 0;
   }
```

// lab11_2.cpp

```
   //  lucrul  cu  fisierele:  deschiderea  si  inchiderea  unui  stream
(flux) - partea 2
   #include<stdio.h>
   #include<conio.h>
   #include<stdlib.h>     // pentru functia exit()
   int main(void){
    char ch, *sirEroare = "Eroare la deschidere fisier";
    int nrChar = 0;// contor al caracterelor
    FILE *pFisier; // pointerul la fisier; numele fluxului

    pFisier = fopen("c:\\test1.txt", "w+");
      //  Deschidere pentru scriere si actualizare.
      //   Un caracter Enter la intrare se traduce in perechea
CR-LF la scriere in fisier.

   if (pFisier == NULL){  // eroare la deschidere fisier
      if(ferror(pFisier)) {
            perror(sirEroare);
            exit(1);           // iesire din program
            }
      }
    else {   // corp de instructiuni_else
      while((ch = getchar()) != 'q'){
            fputc((int)ch, pFisier);
            nrChar++;
            }
    }
    printf("\nNumarul caracterelor scrise: %i", nrChar);

   // apar erori in acest moment?
    if(ferror(pFisier)) perror(sirEroare);
   // fclose(pFisier);          //    inchiderea    fluxului    asociat
fisierului 'test.txt'
   // incheiere
    getch(); // prototip in conio.h
   }
```

```
   // lab11_3.cpp
   // lucrul cu fisierele: schimbarea asocierii unui flux fisier - freopen()
   #include<stdio.h>
   #include<conio.h>
   #include<stdlib.h>           // pentru functia exit()

   int main(void){
    char ch;
    char *sirErr1 = "Eroare la deschidere fisier 1";
    char *sirErr2 = "Eroare la deschidere fisier 2";
    int nrCh1 = 0, nrCh2 = 0;          //  contoare  ale  caracterelor (octetilor)
    int contor = 20;            // numarul de executii al celei de-a doua bucle while()

    FILE *pFis1, *pFis2;  // pointerii la fisiere; numele fluxurilor

   // deschiderea fluxurilor şi testele de eroare aferente.
    pFis1 = fopen("c:\\test1.txt", "wt+");
    pFis2 = fopen("c:\\test2.txt", "at+");
    // Un caracter Enter la intrare se traduce in perechea CR-LF la
   // scriere in fisier.

    if (pFis1 == NULL){   // eroare la deschidere fisier1
      if(ferror(pFis1)) {
            perror(sirErr1);
            exit(1);            // iesire din program
            }
      }

    if (pFis2 == NULL){   // eroare la deschidere fisier2
      if(ferror(pFis2)) {
            perror(sirErr2);
            exit(1);            // iesire din program
            }
      }

   // scriere in fisierul 1
    printf("\n Se scrie in fisierul test1.txt... \n");
    while((ch = getchar()) != 'q'){
      fputc((int)ch, pFis1);
      nrCh1++;
       }
    printf("\nNumarul caracterelor scrise in fisierul >> test1.txt <<: %i", nrCh1);

   // citire din fisierul 2. Acesta este un fisier de proba existent apriori la
   // locatia anuntata. Trebuie sa contina cel putin 20 caractere (!).
    printf("\n\n\n Se citeste din fisierul test2.txt ...\n");
    while(contor){
      printf("%c", fgetc(pFis2));
      nrCh2++;     // incrementare a numarului de caractere citite.
```

```
    contor--;    // decrementare la fiecare carcter citit.
    }
 printf("\nNumarul caracterelor citite din fisierul >>test2.txt<<:\
                                                        %i", nrCh2);

// Inchiderea fluxului 2. Schimbarea asocierii fluxului 1, cu
// fluxul 2.
 fclose(pFis2);
 pFis2 = freopen("c:\\test2.txt", "at+", pFis1);
// Asociez pFis1 fisierului 'test2.txt'. Acesta nu mai este asociat
// fisierului test1.txt (!)

// Scriu in fisierul 2, in mod adaugare, dupa schimbarea
// atribuirii.
// reset pe contorul de caractere pe care l-am stabilit,
                // conventional, fisierului 'test1.txt'
 nrCh1 = 0;
printf("\n\n\n Se scrie in fisierul test2.txt, dupa reatriuirea fluxurilor\
                                                                    \n");

 while((ch = getchar()) != 'q'){
   fputc((int)ch, pFis2);
   nrCh1++;
    }
 printf("\nNumarul caracterelor scrise in fisierul >> test2.txt << dupa \
                                            reatribuire: %i", nrCh1);

// inchiderea definitiva a fluxurilor
 fclose(pFis1); fclose(pFis2);

// incheiere
 getch(); // prototip in conio.h
}
```

// **lab11_4.cpp**

```
//  Lucrul cu fisierele: citirea dintr-un fisier cu ajutorul
// functiei fgetc() - text şi binar.
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>     // pentru functia exit()

int main(void){
 char ch, *sirEroare = "Eroare la deschidere fisier";

// numarul caracterelor ce vor fi citite
 int nrCharT = 32 , nrCharB = 32;

// contoare de citire pentru fiecare dintre fluxuri
 int contorT = 0, contorB = 0;
FILE *pFisT, *pFisB;          // pointerii la fisier;
                              // numele fluxurilor: text si binar

 pFisT = fopen("c:\\test1.txt", "rt");
```

```
// deoarece urmeaza o citire din acest fisier se presupune ca el
// exista deja pe HDD, la locatia anuntata. Altfel sunt generate
// erori de compilator.
//  Un caracter Enter la intrare se traduce in perechea CR-LF la
// scriere in fisier.
 pFisB = fopen("c:\\test2.txt", "rb");    // citire in mod binar

 if (pFisT == NULL){   // eroare la deschidere fisier?
   if(ferror(pFisT)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
 }
 if (pFisT == NULL){   // eroare la deschidere fisier?
   if(ferror(pFisB)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
 }

// urmeaza operatiile specifice fisierelor
// pentru fisierul text
  printf("\nCitire din fisierul test1.txt ...");

  while(nrCharT){
   if(ferror(pFisT)) perror(sirEroare);
// citesc din fisierul test1.txt, prin intermediul fluxului asociat
// pointerului 'pFisier'
   printf("%c", fgetc(pFisT));
   nrCharT--;   // cat mai raman in bucla?
   contorT++;   // cate caractere am scris pana acum?
   }
 printf("\n\nNumarul caracterelor citite: %i", contorT);

// pentru fisierul binar
  printf("\n\nCitire din fisierul test2.txt ...");

  while(nrCharB){
   if(ferror(pFisB)) perror(sirEroare);
// Citesc din fisierul test1.txt, prin intermediul fluxului asociat
// pointerului 'pFisier'
   printf("%c", fgetc(pFisB));
   nrCharB--;   // cat mai raman in bucla?
   contorB++;   // cate caractere am scris pana acum?
   }

printf("\n\nNumarul caracterelor citite: %i", contorB);


// inchiderea celor doua fluxuri
 fclose(pFisT);   // inchiderea fluxului asociat fisierului 'test1.txt'
 fclose(pFisB);   // inchiderea fluxului asociat fisierului 'test2.txt'

// incheiere
```

```
 puts("\n\nIncheiati apasand o tasta alfa-numerica...");
 getch(); // prototip in conio.h
}
```

// lab11_5.cpp

// lucrul cu fisierele: scrierea intr-un fisier cu ajutorul functiei fputc() **-** text ***şi*** binar

```
#include<stdio.h>
#include<conio.h>          // pentru functia getch()
#include<stdlib.h>         // pentru functia exit()
#include<math.h>           // pentru functia pow()

typedef  unsigned long int ULI;

int main(void){
 char *sirEroare = "Eroare la deschidere fisier";
 char *sirDeScris = " Un mesaj de proba\n Pe doua randuri";
 char *pCh = sirDeScris;            // pointer la inceputul sirului de scris
 FILE *pFisT, *pFisB;               // pointerii la fisier;
                                    // numele fluxurilor: text si binar

 pFisT = fopen("c:\\test1.txt", "wt+");       // actualizare in mod text.
    //  Un caracter Enter la intrare se traduce in perechea CR-LF la scriere in
    // fisier.
 pFisB = fopen("c:\\test2.txt", "wb+");       // actualizare in mod binar.

 if (pFisT == NULL){       // eroare la deschidere fisier?
    if(ferror(pFisT)) {
            perror(sirEroare);
            exit(1);          // iesire din program
            }
 }
 if (pFisB == NULL){       // eroare la deschidere fisier?
    if(ferror(pFisB)) {
            perror(sirEroare);
            exit(1);          // iesire din program
            }
 }

 else {       // incep scrierea in fisiere
  ULI i;      // variabila locala corpului de instructiuni 'else'
  ULI contor = 0;
  printf("\n Se scrie in fisiere...\n");
  while(*pCh++ != '\0') {
    i =  (ULI) pow(2, 25);  // numar natural posibil pentru acest tip de data (ULI)
                            // 2^25 = 32 Mega
```

```
      fputc(*pCh, pFisT);
      fputc(*pCh, pFisB);

      printf(" %i", contor);
        contor++;
        for(; i>0; i--);   // delay
        }
    }

   // inchiderea fluxurilor
    fclose(pFisT);
    fclose(pFisB);

   // incheiere
    printf("\n\n Dati un caracter alfa-numeric... ");
    getch();
   }
```

// lab11_6.cpp

```
// Lucrul cu fisierele: citirea intr-un fisier cu ajutorul
//functiei fgets() - text si binar
#include<stdio.h>
#include<conio.h>      // pentru functia getch()
#include<stdlib.h>     // pentru functia exit()

typedef unsigned long int ULI;
// tip de data 'intreg lung fara semn'. Poate contine valori din [0 2^32-1]

int main(void)
{
 char *sirEroare = "Eroare la deschidere fisier";
 char sirCititT[32], sirCititB[32];

//  Sirurile citite sunt copiate in acesti vectori de caractere (text,
// respectiv binar)
 const int nrMax = 30; // numarul maxim de caractere ce urmeaza sa
fie citit

 FILE *pFisT, *pFisB;        // pointerii la fisier;
                             // numele fluxurilor: text si binar

 pFisT = fopen("c:\\test1.txt", "rt+");   // citire in mod text.
//  Un caracter Enter la intrare se traduce in perechea CR-LF la
// scriere in fisier.
 pFisB = fopen("c:\\test2.txt", "rb+");   // citire in mod binar.
 if (pFisT == NULL){   // eroare la deschidere fisier?
   if(ferror(pFisT)) {
         perror(sirEroare);
         exit(1);           // iesire din program
         }
```

```
    }

    if (pFisB == NULL){   // eroare la deschidere fisier?
      if(ferror(pFisB)) {
             perror(sirEroare);
             exit(1);           // iesire din program
             }
    }

    else {   // incep citirea din fisiere
     printf("\n S-au citit din fisiere urmatoarele siruri...\n");

     puts(fgets(sirCititT, nrMax, pFisT));
     puts(fgets(sirCititB, nrMax, pFisB));

    } // end_else

   // afisarea continutului sirurilor buffer
    printf("\n\n  s  Ce  s-a  scris  in  sirul  sirCititT:  \n  %s",
sirCititT);
    printf("\n  Iar  ce  s-a  scris  in  sirul  sirCititB:  \n  %s",
sirCititB);

   // inchiderea fluxurilor
    fclose(pFisT);
    fclose(pFisB);

   // incheiere
    printf("\n\n Dati un caracter alfa-numeric... ");
    getch();
   }
```

// lab11_7.cpp

```
   //  Lucrul cu fisierele: scrierea intr-un fisier cu ajutorul
   // functiei fputs()
   //  Urmarirea indicatorilor de pozitie inainte si dupa scriere. –
   // text şi binar
   #include<stdio.h>
   #include<conio.h>      // pentru functia getch()
   #include<stdlib.h>     // pentru functia exit()
   #include<string.h>     // pentru functia strlen()

   typedef unsigned long int ULI;

   int main(void){
    char *sirEroare = "Eroare la operatie asupra fisier";
    char *sirDeScrisT = "\n Un mesaj de proba, \n in mod text";
    char *sirDeScrisB = "\n Un mesaj de proba, \n in mod binar";

    FILE *pFisT, *pFisB;        // pointerii la fisier;
                                // numele fluxurilor: text si binar
```

```
// deschiderea fluxurilor, impreuna cu testele de eroare aferente
 pFisT = fopen("c:\\test1.txt", "wt+");
// actualizare prin stergerea continutului anterior, in mod
// text.
//  Un caracter Enter la intrare se traduce in perechea CR-LF la
// scriere in fisier.
 pFisB = fopen("c:\\test2.txt", "wb+");
// actualizare prin stergerea continutului anterior, in mod binar.

 if (pFisT == NULL){   // eroare la deschidere fisier?
   if(ferror(pFisT)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
 }

 if (pFisB == NULL){   // eroare la deschidere fisier?
   if(ferror(pFisB)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
 }

// operatia efectiva asupra fisierelor
 else {         // nu este eroare => incep scrierea in fisiere.
  ULI lenT = strlen(sirDeScrisT), lenB = strlen(sirDeScrisB);
   // lungimile celor doua siruri de scris.
  fpos_t pozT, pozB;

  printf("\n Lungimile celor doua siruri ce urmeaza sa fie scrise:\
                              text = %i, binar = %i", lenT, lenB);

// Test eroare indicator fisier text: inainte de scriere
  if(!fgetpos(pFisT, &pozT))
   printf("\n Indicatorul de pozitie al fisierului text (inainte):\
                                                          %u", pozT);
  else {
   if(ferror(pFisT)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
  }

// test eroare indicator fisier binar: inainte de scriere
  if(!fgetpos(pFisB, &pozB) )
  printf("\n Indicatorul de pozitie al fisierului binar (inainte):\
                                                          %u", pozB);
  else {
   if(ferror(pFisB)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
  }
```

```
//  Scrierea efectiva. Teste de eroare la scriere (fputs() intoarce
// valori non-negative in caz de succes. Intoarce EOF in caz
// contrar)
  printf("\n\n Se scrie in fisiere...\n");

  if(fputs(sirDeScrisT, pFisT) == EOF) {
   if(ferror(pFisT)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
  }

  if(fputs(sirDeScrisB, pFisB) == EOF)  {
   if(ferror(pFisB)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
  }

// test eroare indicator fisier text: dupa scriere
  if(!fgetpos(pFisT, &pozT))
   printf("\n Indicatorul de pozitie al fisierului text (dupa): \
                                                          %u", pozT);
  else {
   if(ferror(pFisT)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
  }


// test eroare indicator fisier binar: dupa scriere
  if(!fgetpos(pFisB, &pozB))
  printf("\n Indicatorul de pozitie al fisierului binar (dupa):\
                                                          %u", pozB);
  else {
   if(ferror(pFisB)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
  }
 } // end_else

// inchiderea fluxurilor
 fclose(pFisT);
 fclose(pFisB);

// incheiere
 printf("\n\n Dati un caracter alfa-numeric... ");
 getch();
}
```

```
// lab11_8.cpp
//  Lucrul cu fisierele: scrierea intr-un fisier cu ajutorul
// functiei fwrite()
//  Urmarirea indicatorilor de pozitie inainte si dupa scriere. –
// text şi binar
#include<stdio.h>
#include<conio.h>      // pentru functia getch()
#include<stdlib.h>     // pentru functia exit()
#include<string.h>     // pentru functia strlen()

typedef  unsigned long int ULI;

int main(void){
 char *sirEroare = "Eroare la operatie asupra fisier";
 char *sirDeScrisT = " Un mesaj de proba, in mod text.";
 char *sirDeScrisB = " Un mesaj de proba, in mod binar.";

 const int nrObiecte = 8;     // voi scrie/citi 8 obiecte ...
 const int dimObiect = 2;     // ... de lungime 2 octeti, fiecare.

 FILE *pFisT, *pFisB;         // pointerii la fisier;
                              // numele fluxurilor: text si binar
// deschiderea fluxurilor, impreuna cu testele de eroare aferente
 pFisT = fopen("c:\\test1.txt", "wt+");
 // actualizare prin stergerea continutului anterior, in mod text.
 //  Un caracter Enter la intrare se traduce in perechea CR-LF la
 // scriere in fisier.
 pFisB = fopen("c:\\test2.txt", "wb+");
// actualizare prin stergerea continutului anterior, in mod binar.

 if (pFisT == NULL){   // eroare la deschidere fisier?
   if(ferror(pFisT)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
 }

 if (pFisB == NULL){   // eroare la deschidere fisier?
   if(ferror(pFisB)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
 }

// operatia efectiva asupra fisierelor
 else {         // nu este eroare => incep scrierea in fisiere.
  ULI lenT = strlen(sirDeScrisT), lenB = strlen(sirDeScrisB);
   // lungimile celor doua siruri de scris.
  fpos_t pozT, pozB;

  printf("\n Lungimile celor doua siruri din care urmeaza sa se \
                        scrie: text = %i, binar = %i", lenT, lenB);

// test eroare indicator fisier text: inainte de scriere
```

```
 if(!fgetpos(pFisT, &pozT) )
   printf("\n Indicatorul de pozitie al fisierului text (inainte):\
                                                          %u", pozT);
  else  {
   if(ferror(pFisT)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
  }

// test eroare indicator fisier binar: inainte de scriere
  if(!fgetpos(pFisB, &pozB) )
   printf("\n Indicatorul de pozitie al fisierului binar (inainte):
                                                          %u", pozB);
  else {
   if(ferror(pFisB)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
  }

// Scrierea efectiva, impreuna cu testele de eroare aferente legate
// de functia folosita.
 {
  ULI scriseT, scriseB;
// numarul obiectelor scrise efectiv - raportat de functia fwrite()

  printf("\n\n Se scrie in fisiere...\n");

if((scriseT = fwrite((char*)sirDeScrisT, dimObiect, nrObiecte,
                                                 pFisT)) == nrObiecte)
   printf("\n Succes! Numarul efectiv de caractere scrise (text):\
                                           %u", scriseT*dimObiect);

  else  {
   if(ferror(pFisT)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
  }

// aceleasi operatii pentru fisierul binar
if((scriseB = fwrite((char*)sirDeScrisB, dimObiect, nrObiecte,
                                                 pFisB)) == nrObiecte)
 printf("\n Succes! Numarul efectiv de caractere scrise (binar): \
                                           %u", scriseB*dimObiect);
else  {
   if(ferror(pFisB)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
  }
// test eroare indicator fisier text: dupa scriere
  if(!fgetpos(pFisT, &pozT))
```

```
  printf("\n\n Indicatorul de pozitie al fisierului text (dupa): \
                                                          %u", pozT);
  else {
   if(ferror(pFisT)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
  }

// test eroare indicator fisier binar: dupa scriere
  if(!fgetpos(pFisB, &pozB))
printf("\n Indicatorul de pozitie al fisierului binar (dupa): %u",
                                                              pozB);
  else {
   if(ferror(pFisB)) {
          perror(sirEroare);
          exit(1);           // iesire din program
          }
  }
 } // end_else

 } // end_scriereEfectiva
// inchiderea fluxurilor
 fclose(pFisT);
 fclose(pFisB);

// incheiere
 printf("\n\n Dati un caracter alfa-numeric... ");
 getch();
}
```

// lab11_9.cpp

```
// lucrul cu fisierele: citirea dintr-un fisier cu ajutorul functiei fread()
// urmarirea indicatorilor de pozitie inainte si dupa scriere. - text şi binar
#include<stdio.h>
#include<conio.h>    // pentru functia getch()
#include<stdlib.h>   // pentru functia exit()
#include<string.h>   // pentru functia strlen()

typedef  unsigned long int ULI;
void finalSir(const char*, ULI*, const ULI);

int main(void){
 char *sirEroare = "Eroare la operatie asupra fisier";
 void *sirCititT[32], *sirCititB[32];
            // cele doua zone tampon (buffer) in care
            // se salveaza textul citit din fisiere (text respectiv binar).

 const int nrObiecte = 8;    // voi citi 8 obiecte ...
 const int dimObiect = 3;    // ... de lungime 2 octeti, fiecare.

 FILE *pFisT, *pFisB;                // pointerii la fisier;
                                     // numele fluxurilor: text si binar
// deschiderea fluxurilor, impreuna cu testele de eroare aferente
 pFisT = fopen("c:\\test1.txt", "rt");
// mod citire si actualizare, in mod text.
//  Un caracter Enter la intrare se traduce in perechea CR-LF la scriere in fisier.
 pFisB = fopen("c:\\test2.txt", "rb");    // mod citire si actualizare, in mod binar.
```

```
 if (pFisT == NULL){ // eroare la deschidere fisier?
    if(ferror(pFisT)) {
            perror(sirEroare);
            exit(1);         // iesire din program
            }
}

 if (pFisB == NULL){ // eroare la deschidere fisier?
    if(ferror(pFisB)) {
            perror(sirEroare);
            exit(1);         // iesire din program
            }
 }

// operatia efectiva asupra fisierelor
 else {              // nu este eroare => incep citirea din fisiere.
  fpos_t pozT, pozB; // indicatorii de pozitie ai celor doua tipuri de fisiere.
// test eroare indicator fisier text: inainte de scriere
 if(!fgetpos(pFisT, &pozT) )
    printf("\n Indicatorul de pozitie al fisierului text (inainte): %u", pozT);
  else  {
    if(ferror(pFisT)) {
            perror(sirEroare);
            exit(1);         // iesire din program
            }
  }

// test eroare indicator fisier binar: inainte de scriere
  if(!fgetpos(pFisB, &pozB) )
    printf("\n Indicatorul de pozitie al fisierului binar (inainte): %u", pozB);
  else {
    if(ferror(pFisB)) {
            perror(sirEroare);
            exit(1);         // iesire din program
            }
  }

//  Citirea efectiva, impreuna cu testele de eroare aferente legate de functia
// folosita.
 {  // un nou corp de instructiuni
  ULI cititeT, cititeB;
            // numarul obiectelor citite efectiv - raportat de functia fread()
  char *pVar;
  int contor;

  printf("\n\n Se citeste din fisiere...\n");

  if((cititeT = fread((char*)sirCititT, dimObiect, nrObiecte, pFisT)) == nrObiecte)
    {
    printf("\n Succes! Numarul efectiv de caractere citite (text): %u",
                                            cititeT*dimObiect);
    finalSir((const char*)sirCititT, &cititeT, dimObiect);
    printf("\n Ceea ce s-a citit (text): %s", (char*)sirCititT);
    }
  else  {
    if(ferror(pFisT)) {
            perror(sirEroare);
            exit(1);         // iesire din program
            }
  }

// aceleasi operatii pentru fisierul binar
 if((cititeB = fread((char*)sirCititB, dimObiect, nrObiecte, pFisB)) == nrObiecte)
    {
    printf("\n\n Succes! Numarul efectiv de caractere citite (binar): %u",
                                            cititeB*dimObiect);
    finalSir((const char*)sirCititB, &cititeB, dimObiect);
    printf("\n Ceea ce s-a citit (binar): %s", (char*)sirCititB);
    }
```

```
  else  {
    if(ferror(pFisB)) {
            perror(sirEroare);
            exit(1);         // iesire din program
            }
  }

// test eroare indicator fisier text: dupa scriere
  if(!fgetpos(pFisT, &pozT))
    printf("\n\n Indicatorul de pozitie al fisierului text (dupa): %u", pozT);
  else {
    if(ferror(pFisT)) {
            perror(sirEroare);
            exit(1);         // iesire din program
            }
  }
// test eroare indicator fisier binar: dupa scriere
  if(!fgetpos(pFisB, &pozB))
    printf("\n Indicatorul de pozitie al fisierului binar (dupa): %u", pozB);
  else {
    if(ferror(pFisB)) {
            perror(sirEroare);
            exit(1);         // iesire din program
            }
  }
 } // end_else

 }  // end_citireEfectiva

// inchiderea fluxurilor
 fclose(pFisT);
 fclose(pFisB);

// incheiere
 printf("\n\n Dati un caracter alfa-numeric... ");
 getch();
}

// implementarea functiei declarate local (functie proprie programatorului).
// Se marcheza explicit sfarsitul sirului tampon, deoarece fread() nu
// face acest lucru automat.
void finalSir(       const char *sirCitit,
                     /* text sau binar, nu conteaza din cauza apelului referinta*/
                     ULI *citite,
                     const ULI dimObiect) {
 char *pVar;    // pointer variabil, pentru deplasarea in interiorul sirului preluat
                // ca argument.
 int contor;         // variabila locala

// instructiunile proprii functiei
 contor = *citite*dimObiect;
 pVar = (char*) sirCitit;

 while(contor--) pVar++;
// variabila contor se decrementeaza dupa ce se testeaza valoarea de
// adevar a conditiei logice.
 *pVar = '\0';       // marchez explicit sfarsitul sirului
}
```

// **lab11_10.cpp**

```
// lucrul cu fisierele: citirea/scrierea formatate intr-un/dintr-un fisier cu
// ajutorul functiilor pereche fprintf() / fscanf().
// urmarirea indicatorilor de pozitie inainte si dupa scriere. - text şi binar

#include<stdio.h>
#include<conio.h>    // pentru functia getch()
#include<stdlib.h>   // pentru functia exit()
typedef  unsigned long int ULI;

int main(void){
 char *sirEroare = "Eroare la operatie asupra fisier";
 char *sirFormat = "\n Se scriu %f, %f, %f drept valori reale si %i, %i ca
valori intregi.";

 const int nrObiectReal = 3; // voi scrie 3 obiecte reale...
 const int nrObiectInt = 2;          // ... si 2 obiecte intregi.

 FILE *pFisT, *pFisB;                // pointerii la fisier;
                                     // numele fluxurilor: text si binar

// deschiderea fluxurilor, impreuna cu testele de eroare aferente
 pFisT = fopen("c:\\test1.txt", "wt+");
    // actualizare prin adaugare, in mod text.
    //  Un caracter Enter la intrare se traduce in perechea CR-LF la scriere in
    // fisier.
 pFisB = fopen("c:\\test2.txt", "at+");
    // actualizare prin adaugare, in mod binar.

 if (pFisT == NULL){ // eroare la deschidere fisier?
    if(ferror(pFisT)) {
             perror(sirEroare);
             exit(1);         // iesire din program
             }
 }

 if (pFisB == NULL){ // eroare la deschidere fisier?
    if(ferror(pFisB)) {
             perror(sirEroare);
             exit(1);         // iesire din program
             }
 }
// operatia efectiva asupra fisierelor
 else {              // nu este eroare => incep citirea din fisiere.
  fpos_t pozT, pozB; // indicatorii de pozitie ai celor doua tipuri de fisiere.

// test eroare indicator fisier text: inainte de scriere
 if(!fgetpos(pFisT, &pozT) )
    printf("\n Indicatorul de pozitie al fisierului text (inainte): %u", pozT);
  else  {
    if(ferror(pFisT)) {
             perror(sirEroare);
             exit(1);         // iesire din program
             }
  }
// test eroare indicator fisier binar: inainte de scriere
  if(!fgetpos(pFisB, &pozB) )
    printf("\n Indicatorul de pozitie al fisierului binar (inainte): %u", pozB);
  else {
    if(ferror(pFisB)) {
             perror(sirEroare);
             exit(1);         // iesire din program
             }
  }

// operatiile efective, impreuna cu testele de eroare legate de functiile folosite.
 {
  ULI scriseT, cititeB;      // numarul caaracterelor scrisein mod text, si
                                                    // citite in mod binar
```

```
  int i1, i2;                          // variabilele intregi citite binar.
  float r1, r2, r3;  // variabilele reale citite binar.

  printf("\n\n Se scrie in fisierul text...");

  scriseT = fprintf(pFisT, sirFormat, 2.71828, -3.141592, -0.17, 3, 4);
  printf("\n Numarul efectiv de caractere scrise in fisier (!) text): %u", scriseT);

// operatiile pentru fisierul binar
  printf("\n\n Se citeste din fisierul binar...");

  cititeB = fscanf(pFisB, "%f,%f,%f,%i,%i", &r1, &r2, &r3, &i1, &i2);
  printf("\n Numarul efectiv de obiecte atribuite(!) (binar): %u", cititeB);
  printf("\n Valorile variabilelor dupa citirea din fisier: %f, %f, %f si %i, %i",
                    r1, r2, r3, i1, i2);

// test eroare indicator fisier text: dupa scriere
  if(!fgetpos(pFisT, &pozT))
    printf("\n\n Indicatorul de pozitie al fisierului text (dupa): %u", pozT);
  else {
    if(ferror(pFisT)) {
             perror(sirEroare);
             exit(1);         // iesire din program
             }
  }
// test eroare indicator fisier binar: dupa scriere
  if(!fgetpos(pFisB, &pozB))
    printf("\n Indicatorul de pozitie al fisierului binar (dupa): %u", pozB);
  else {
    if(ferror(pFisB)) {
             perror(sirEroare);
             exit(1);         // iesire din program
             }
  }
 } // end_else
 }  // end_citireEfectiva
// inchiderea fluxurilor
 fclose(pFisT);
 fclose(pFisB);
// incheiere
 printf("\n\n Dati un caracter alfa-numeric... ");
 getch();
}
```

// lab11_11.cpp
// lucrul cu fisierele: functiile de tratare a erorilor

```
#include<stdio.h>
#include<conio.h>    // pentru functia getch()
#include<stdlib.h>   // pentru functia exit()

int main(void){
 char *sirEroareT = "\nLa operatia asupra fisierului text";
 char *sirEroareB = "\nLa operatia asupra fisierului binar";
 char bufferText[128], bufferBin[128];
                      // numarul maxim de caractere (octeti) cititi poate
                      // fi cel specificat de dimensiunea vectorilor.

 FILE *pFisT, *pFisB;                // pointerii la fisier;
                                     // numele fluxurilor: text si binar

// deschiderea fluxurilor, impreuna cu testele de eroare aferente
 pFisT = fopen("c:\\test1.txt", "rt");
    //  Deschis in mod citire, pentru tip text.
    //  Ramane sa modificati pe rand aceste moduri de deschidere si sa observati
    //efectele, vizualizand de fiecare data continutul fisierelor.
    //  Daca fisierul test1.txt contine un numar dat de caractere, atunci incercarea
```

```
      // de a citi mai multe caractere genereaza EOF.
      // Modificati modul de deschidere in "wt". Observati comportamentul
      // compilatorului, care se blocheaza.

// pFisB = fopen("c:\\test2.txt", "wb");
// pFisB = fopen("c:\\test2.txt", "ab");
 pFisB = fopen("c:\\test2.txt", "rb");
      //  Deschidere in mod adaugare, sau scriere sau citire. Ramane sa modificati
      // pe rand aceste moduri de deschidere si sa observati efectele, vizualizand
      // de fiecare data continutul fisierelor.
      //  Stabiliti pentru fisierul de pe HDD atributul read-only pe ON.
      // Compilatorul va genera erori grave si se va blocha.
      //  Dezactivati apoi atributul read-only pentru a obtine o functionare normala
      // a programului. Sau schimbati modul de deschidere al fisierului, astfel
      // incat acesta sa fie compatibil cu atributul read-only.
      //  Daca modul de deschidere este 'append' si se doresc operatii de citire
      // acestea nu se executa cu succes.

 if (pFisT == NULL){ // eroare la deschidere fisier?
    if(ferror(pFisT)) {
             perror(sirEroareT);
             exit(1);         // iesire din program
             }
 }

 if (pFisB == NULL){ // eroare la deschidere fisier?
    if(ferror(pFisB)) {
             perror(sirEroareB);
             exit(1);         // iesire din program
             }
 }

// operatia efectiva asupra fisierelor
 else {              // nu este eroare => incep operatiile pe fisiere.
  fpos_t pozT, pozB; // indicatorii de pozitie ai celor doua tipuri de fisiere.

// test eroare indicator fisier text: inainte de scriere
 if(!fgetpos(pFisT, &pozT) )
    printf("\n Indicatorul de pozitie al fisierului text (inainte): %u", pozT);
  else  {
    if(ferror(pFisT)) {
             perror(sirEroareT);
             exit(1);         // iesire din program
             }
  }
// test eroare indicator fisier binar: inainte de scriere
  if(!fgetpos(pFisB, &pozB) )
    printf("\n Indicatorul de pozitie al fisierului binar (inainte): %u", pozB);
  else {
    if(ferror(pFisB)) {
             perror(sirEroareB);
             exit(1); // iesire din program
             }
  }
// operatiile efective, impreuna cu testele de eroare legate de functiile
// folosite.
  {
  // numarul de obiecte ce urmeaza sa fie citite (variabila locala)
  int nrObiecte = 4;
  int dimObiect = 2; // numarul de octeti al unui obiect (variabila locala)
  printf("\n\n Se citeste din fisierul text...");
  printf("\n S-au citit efectiv %u obiecte", fread(bufferText, dimObiect, nrObiecte,
                                                                          pFisT));

  if(feof(pFisT)) perror(sirEroareT); // test de EOF.

  int i = 0;
  while(i < nrObiecte*dimObiect) i++;
  bufferText[i] = '\0';      // marcarea explicita a sfarsitului de sir.
```

```
//  puts(bufferText);
  printf("\n Buffer-ul contine sirul: %s", bufferText);

// operatie improprie modului de deschidere ales
 fprintf(pFisT, "\n Operatie interzisa, conform modului de deschidere ales");
    // aceasta operatie este ignorata (!)
 }

// operatiile pentru fisierul binar
  {
  int nrObiecte = 24;        // numarul de obiecte ce urmeaza sa fie citite
                             // (vaiabila locala)
  int dimObiect = 4; // numarul de octeti al unui obiect (variabila locala)
  printf("\n\n Se citeste din fisierul binar...");

// o citire cu un numar de caractere peste dimensiunea fisierului, considerand
// fisierul initializat cu mai putine caractere decat valoarea nrObiecte*dimObiect
// din acest moment. O rulare multipla a programului va schimba continutul
// fisierului si deci si valoarea logic a conditiei din if(feof()) de mai jos.
  fread(bufferBin, dimObiect, nrObiecte, pFisB);

  if(feof(pFisB)) perror(sirEroareB); // test de EOF.

  int i = 0;
  while(i < nrObiecte*dimObiect) i++;
  bufferText[i] = '\0';      // marcarea explicita a sfarsitului de sir.

  printf("\n Buffer-ul contine sirul: %s", bufferBin);

// o scriere fara modificarea indicatorului de pozitie, dar cu schimbarea fluxului
// asociat
  fclose(pFisT);
  pFisT = freopen("c:\\test2.txt", "ab", pFisB);      // reatribui pFisB lui pFisT
    // voi modifica acum fisierul test2.txt, cu ajutorul fluxului 'pFisT'
//
  fprintf(pFisT, "\n Un text oarecare\n pentru proba. Fisier binar");
  if(feof(pFisT)) perror(sirEroareB); // test de EOF.
  if(feof(pFisB)) perror(sirEroareB); // test de EOF.
  }
// test eroare indicator fisier text: dupa operatii
  if(!fgetpos(pFisT, &pozT))
    printf("\n\n Indicatorul de pozitie al fisierului text (dupa): %u", pozT);
  else {
    if(ferror(pFisT)) {
             perror(sirEroareT);
             exit(1);         // iesire neconditionata din program
             }
  }

// test eroare indicator fisier binar: dupa operatii
  if(!fgetpos(pFisB, &pozB))
    printf("\n Indicatorul de pozitie al fisierului binar (dupa): %u", pozB);
  else {
    if(ferror(pFisB)) {
             perror(sirEroareB);
             exit(1); // iesire neconditionata din program
             }
  }
 } // end_else

// inchiderea fluxurilor
 fclose(pFisT);
 fclose(pFisB);

// incheiere
 printf("\n\n Dati un caracter alfa-numeric... "); getch();
}
```