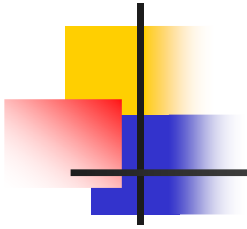
V. Fişiere



1. Caracteristici ale fişierelor
2. Operaţii de bază asupra fişierelor
3. Prelucrarea fişierelor text
4. Prelucrarea fişierelor binare
5. Funcţii speciale pentru tratarea erorilor
6. Actualizarea fişierelor în acces direct
7. Redirectarea intrării şi ieşirii standard

Fişiere
Definiţii

- Fişierul reprezintă o *grupare de informaţii omogene* din punct de vedere al semnificaţiei acestora şi al cerinţelor de prelucrare(similitudine cu tablourile). Componentele unui fişier se numesc **înregistrări** sau **articole**.
- Fişierul este o *colecţie de informaţii*, înregistrată complet pe un suport extern şi care are asociat un *nume extern* (independent de programele care îl prelucrează).
- Fişierul este *entitatea care stochează* un ansamblu de instrucţiuni, numere, texte sau imagini, *într-o structură coerentă*, astfel încât utilizatorul să le poată regăsi, modifica, şterge sau copia în altă parte. *Exemple tipice de fişiere* : programe sursă, programe executabile, fişiere de date, fişiere documente, imagini, muzică.

Fişiere
Caracteristici

- **Suportul fizic** pe care se înregistrează informațiile unui fişier este un suport extern, corespunzător unui echipament periferic (discul sau banda magnetică, discul flexibil, discul optic, memoria flash, hârtia de imprimantă, ecranul monitorului etc.). În acest fel fişierele pot înmagazina mult mai multe informații decât s-ar putea permite în cazul tablourilor.
- O consecință a faptului că unele dintre suporturile externe sunt nevolatile (informațiile nu se pierd la terminarea execuției programului şi deconectarea unității periferice respective) este aceea că informațiile dintr-un fişier pot fi reutilizate, fie la o nouă execuție a aceluiaşi program, fie la execuția altui program care lucrează cu acelaşi fişier, asigurând transmiterea de date de la un program la altul. Acelaşi avantaj apare şi pentru situațiile, destul de frecvente, în care fişierul nu înmagazinează date ci programe sau porțiuni de programe, fie în format sursă (de exemplu C), fie în format direct executabil(*.exe*).

Fişiere
Caracteristici

- **Numărul de componente** ale unui fişier, denumit şi **lungimea fişierului**, este oarecare şi se poate modifica la execuția programului. Acest număr este limitat doar de capacitatea de memorare a suportului utilizat şi, uneori, nici măcar de aceasta (fişierul se poate continua de pe un suport pe altul). În consecință, considerând structura de fişier ca tip, se poate aprecia că acest tip are cardinalitate infinită.

- Numărul variabil de componente atrage după sine necesitatea ca *sfârşitul fişierului* să fie marcat printr-o componentă specială notată ***End-Of-File*** care să poată să fie sesizată prin program, determinând încheierea procesului de parcurgere şi prelucrare a informațiilor din fişierul respectiv.

Fişiere
Moduri de organizare

În funcție de sistemul de operare şi limbajul de programare, există mai multe variante de „aranjare” a componentelor unui fişier pe suportul extern, determinând aşa numita **organizare** a fişierului:

- **secvențială**
- **secvențial-indexată**
- **aleatoare**
- **partiționată**
- **multiindexată**

În continuare prezentăm doar **organizarea secvențială** a fişierelor (cea mai simplă). În această organizare componentele sunt plasate pe suportul extern una după alta, secvențial, în ordinea în care au fost create.

Fişiere
Moduri de acces

Odată fişierul creat, limbajul de programare C permite două *modalităţi de acces* la componentele sale:

- **acces secvenţial** : componentele sunt prelucrate strict în ordinea în care sunt înregistrate în fişier.
- **acces direct** : componentele se pot prelucra în orice ordine, specificând poziţia în fişier a componentei ce urmează să fie prelucrată. Pentru a putea ca, pe baza poziţiei, să se calculeze locul din fişier de unde urmează să se continue prelucrarea, programatorul trebuie să cunoască formatul fişierului şi dimensiunea înregistrărilor.

Fişiere

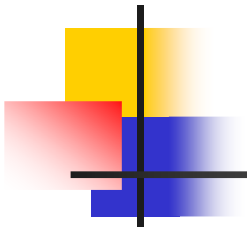
Fişiere speciale

- Din motive de uniformitate a tratării informaţiilor externe, în general în programare şi în particular în C, datele de intrare ale unui program, introduse de la tastatura calculatorului precum şi rezultatele sale, afişate pe ecran sunt interpretate ca nişte fişiere secvenţiale speciale, numite *fişierul standard de intrare*, notat în C cu **stdin** (implicit tastatura) şi respectiv *fişierul standard de ieşire*, notat cu **stdout** (implicit ecranul). În C există şi un *fişier standard pentru mesaje de eroare* numit **stderr** (implicit ecranul).
- Pe platformele MS-DOS şi Windows următoarele nume de fişiere au semnificaţie specială, desemnând dispozitive periferice : CON, PRN, AUX, NUL, COM1, COM2, ... COM9, LPT1, LPT2, ... LPT9. Aceste nume nu pot fi folosite ca nume de fişiere nici dacă li se adaugă o extensie (de exemplu *PRN.txt* este un nume neacceptat de sistem).

Fişiere

Caracteristici ale fişierelor în C

- Prelucrarea fişierelor prin program implică o secvență de operații specifice. Orice fişier trebuie *deschis* înainte de a putea fi prelucrat. Prelucrările care se fac cel mai frecvent asupra unui fişier sunt *crearea (scrierea)* , *consultarea* (citirea înregistrărilor din fişier) şi *actualizarea* fişierului prin scrierea de noi înregistrări în fişier. La sfârşitul prelucrărilor, fişierul trebuie *închis*. Toate aceste operații se realizează prin funcții din biblioteca standard a limbajului C.
- Prelucrarea fişierelor în limbajul C se poate face la două niveluri: *nivelul inferior*, care utilizează direct funcții ale sistemului de operare, şi *nivelul superior* de prelucrare a fişierelor, care conține funcții specializate, specifice limbajului C. În continuare vor fi tratate doar funcțiile de nivel superior. Aceste funcții oferă mai multe facilități şi sunt mai uşor de utilizat.
- În funcție de modul de interpretare al datelor, în C fişierele pot fi prelucrate ca *fişiere text* sau ca *fişiere binare*.



Fişiere
Referirea unui fişier

- La nivelul suportului extern şi al sistemului de operare, un fişier este identificat prin *numele* său *extern*.

- La nivelul interfeței de programare C, un fişier este referit printr-o *structură de tip FILE*, un tip definit în *stdio.h şi* conținând informațiile necesare pentru accesul la fişier: dimensiunea înregistrării, poziția curentă a cursorului în fişier, modul de acces etc.

Fişiere
Referirea unui fişier

- În *Turbo C++ Lite*, *FILE* are următoarea structură:

```
typedef struct {
short level;              // Nivel plin/gol al bufferului
unsigned flags;           // Indicator de stare fisier
char fd;                  // Descriptor fisier
unsigned char hold; // ungetc char daca nu e buffer
short bsize;                // Dimensiune buffer
unsigned char *buffer;  // Buffer transfer date
 unsigned char *curp;   // Pointerul activ curent
unsigned istemp;        // Indicator de fisier temporar
short token;            // Folosit pentru testare validare
  } FILE;                   // Acesta este obiectul FILE
```

Operaţii de bază asupra fişierelor

Deschiderea unui fişier

- Pentru a putea manipula un fişier prin program, trebuie să se realizeze mai întâi operația de **deschidere** a fişierului, *operație care asociază numelui extern al fişierului o structură de tip FILE şi îl pregăteşte pentru prelucrările ulterioare.*

- Pentru deschiderea unui fişier se utilizează funcția *fopen*:

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

Parametrii acestei funcții sunt:

nume = un şir de caractere care reprezintă numele extern al fişierului; numele poate include şi specificarea căii.

mod = un şir de caractere care defineşte modul de prelucrare a fişierului după deschidere.

Operaţii de bază asupra fişierelor

Deschiderea unui fişier

mod	**Semnificație**
a	Deschide fişierul pentru operații de *adăugare* – dacă fişierul nu există, sistemul de operare îl crează
r	Deschide un fişier existent pentru operații de *citire*
w	Deschide un *fişier nou pentru scriere* – dacă există un fişier cu acelaşi nume, sistemul de operare îl suprascrie
r+	Deschide un *fişier existent pentru citire şi scriere*
w+	Deschide un *fişier nou pentru citire/scriere* – dacă există un fişier cu acest nume, sistemul de operare îl suprascrie
a+	Deschide un *fişier pentru operații de adăugare şi citire* – dacă fişierul nu există, sistemul de operare îl crează

Operaţii de bază asupra fişierelor

Deschiderea unui fişier

- La sfârşitul şirului de caractere reprezentând modul de acces, se poate adăuga, opțional, sufixul:

  t pentru fişiere text (implicit);

  b pentru *fişiere binare;*

  exemple: *rb wb ab r+b.*

- În caz de deschidere reuşită, funcția fopen returnează un pointer la structura de tip FILE care a fost asignată. În caz contrar se returnează pointerul NULL.

- Prin deschidere, se rezervă automat şi un *buffer* pentru transferul datelor.

Operaţii de bază asupra fişierelor

Deschiderea unui fişier

- Întotdeauna *trebuie să se testeze dacă rezultatul operației de deschidere* este un pointer nenul. Dacă rezultatul este pointerul NULL înseamnă că deschiderea fişierului nu a reuşit şi nu pot fi efectuate prelucrări ulterioare asupra sa.

De exemplu, dacă se doreşte deschiderea fişierului *date.txt* pentru citire, se poate utiliza o secvență ca şi cea de mai jos:

```
FILE *fs;
...
if ((fs=fopen("date.txt", "r"))==NULL) {
  printf("Fisierul nu se poate deschide! \n");
  exit(1);
}
```

Operaţii de bază asupra fişierelor
Închiderea unui fişier

- După terminarea prelucrării unui fişier acesta trebuie **închis**, utilizând funcția: ***int fclose(FILE * pf);***

Parametrul acestei funcții, *FILE * pf* este un pointer către structura *FILE* asignată la deschiderea fişierului.

Funcția returnează valoarea *0* dacă operația a decurs normal sau *EOF (-1)* dacă operația nu a reuşit.

- Deoarece numărul de fişiere ce poate fi deschis simultan e limitat de sistemul de operare, se recomandă ca programatorul să închidă fişierul de îndată ce s-a terminat prelucrarea lui.
- Programatorul nu trebuie să închidă sau să deschidă fişierele standard (*stdin*, *stdout*, *stderr*). Ele sunt deschise automat la lansarea în execuție a programului şi sunt închise la terminarea programului prin apelul funcției *exit*.

Operaţii de bază asupra fişierelor

Schema de lucru cu fişiere

- **Secvența tipică** de lucru cu un fişier:

```
FILE *fp;
char *nume="f.text";  /* sau din argv[ ], sau de la tastatura */
if ( !(fp=fopen(nume, "r")))    {  /* trateaza eroarea  */ }
else    { /* lucreaza cu fisierul */ }
if (fclose(fp))  /*eroare la inchidere */;
```

- **Alte funcții** referitoare la fişiere:

int feof(FILE *fp);

returnează o valoare nenulă dacă s-a întâlnit sfârşitul de fişier la ultima operație de intrare şi 0 în caz contrar.

int fflush(FILE *fp);

determină golirea buffer-ului unui fişier.

Fişiere text

Prelucrarea pe caractere a unui fişier

- Cele mai simple **funcţii de prelucrare a fişierelor** sunt cele pe caractere, ***getc*** (***fgetc*** - macrodefiniţie) şi ***putc*** (***fputc*** -macro).

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

Funcţiile primesc ca şi parametru un pointer *pf* la un fişier deschis în modul corespunzător; *pf* trebuie să fi fost deschis:

- pentru citire (în modul *r)* – *getc*;
- pentru scriere (în modul *w* sau *a*) - *putc*.

- Funcţia ***getc*** returnează următorul caracter din fişierul indicat ca şi parametru. În caz de eroare sau când se ajunge la sfârşitul fişierului, returnează valoarea *EOF*.
- Funcţia ***putc*** scrie caracterul *c* în fişierul indicat de parametrul *pf* şi returnează caracterul scris, sau *EOF* în caz de eroare.

Fişiere text

Prelucrarea pe caractere a unui fişier

- ***getchar*** şi ***putchar*** sunt realizate ca macrodefiniţii pornind de la *getc* şi *putc*:

  getchar este echivalent cu *getc(stdin)*;

  putchar(c) este echivalent cu *putc(c, stdout).*

- *Exemplu:* Un program care crează un fişier copie (*nume_dest)*, caracter cu caracter, al unui alt fişier dat (*nume_sursa)*.

  - Se declară doi pointeri de tip FILE, *fs* şi *fd*, pt. prelucrarea acestor două fişiere. Fişierul sursă *fs* este deschis în mod *r* pentru citire, iar fişierul copie *fd* este deschis în modul *w* pentru scriere.

  - Dacă a existat anterior un alt fişier cu numele *nume_dest* acesta va fi rescris.

  - Pentru fiecare operaţie de deschidere de fişier se verifică faptul că operaţia a reuşit (pointerul la FILE returnat de funcţia de deschidere este diferit de NULL).

Fişiere text

Prelucrarea pe caractere a unui fişier

Copierea se face citind câte un caracter din fişierul *fs*, atâta timp cât acesta este diferit de marcajul de sfârşit de fişier, EOF. Caracterul este imediat scris în fişierul *fd*. La sfârşit se închid ambele fişiere.

Observaţie: funcţia gets () nu este protejată la depăşire ceea ce inseamnă că poate să apară o eroare de execuţie gravă dacă informaţia citită depăşeşte dimensiunea tamponului (80).

```
#include <stdio.h>
#include <stdlib.h>
int main() {
 int c;
 char nume_sursa[80], nume_dest[80];
 FILE *fs,*fd;
 printf("Introduceti numele fisierului sursa \n");
 gets(nume_sursa);
 printf("Introduceti numele fisierului copie \n");
 gets(nume_dest);
```

Fişiere text

Prelucrarea pe caractere a unui fişier

```
if ((fs=fopen(nume_sursa, "r"))==NULL) {
    printf("Nu se poate deschide fisierul %s \n",
                                    nume_sursa);
    exit(1);   }
if ((fd=fopen(nume_dest, "w"))==NULL) {
    printf("Nu se poate deschide fisierul %s \n",
                                    nume_dest);
    exit(1);   }
 while ((c=getc(fs))!=EOF)
    putc(c, fd);
fclose(fs);  fclose(fd);
 return 0;
}
```

Fişiere text

Prelucrarea pe caractere a unui fişier

- *Exemplu:* afişarea conţinutului unor fişiere ale căror nume se dau în linia de comandă.

```
#include  <stdio.h>
void citeste(FILE *f) {
   int c; while((c=fgetc(f)) != EOF) putchar(c); }
int main(int argc, char *argv[ ]) {
  FILE *fp;
  if (argc = = 1) citeste(stdin); /* citeste de la intrare */
  else while (- - argc >0)      /* pt. fiecare argument */
              if (!(fp = fopen(*++argv, “r”)))
                  printf(“fisierul %s nu se poate deschide”, *argv);
              else { citeste(fp); fclose(fp); }
   return 0; }
```

Fişiere text

Aplicaţie de prelucrare a textelor

- *Exemplu :* Să se scrie un program care citeşte un text conținut într-un fişier şi afişează fiecare cuvânt întâlnit pe o nouă linie pe ecran. Se consideră că două cuvinte pot fi despărțite printr-un număr oarecare de separatori. Separatorii sunt caracterele *spațiu*, *tab* şi *linie nouă*. Orice altă grupare de caractere se consideră a fi un cuvânt([20], pag. 212).

- Se definesc funcțiile *separator* şi *incuvant* care testează dacă un caracter dat este separator sau poate face parte din cuvinte.

- Principiul algoritmului de găsire a cuvintelor, realizat de funcția *cuvinte*, este următorul:

atata timp cat nu s-a terminat fisierul
* *elimina separatorii*
* *colecteaza caracterele unui cuvant*

Fişiere text

Aplicaţie de prelucrare a textelor

```
#include <stdio.h>
#include <stdlib.h>
#define MAXLIT 80
int separator(int c) {
/* testeaza daca c este separator */
   if ((c==' ')||(c=='\t')||(c=='\n')) return 1;
   return 0;
}

int incuvant(int c) {
/*testeaza daca c poate apare in interiorul unui cuv. */
      return  !separator(c);
}
```

Fişiere text

Aplicaţie de prelucrare a textelor

```
void cuvinte(FILE * fp){
   char s[MAXLIT];
   int l,c;
   c=getc(fp);
   while (c!=EOF) {
       while (separator(c)) {
          /* elimina separatorii */
          c=getc(fp);  }
       if (c==EOF)
          break;
       /* incepe un cuvant */
       l=0; /* initializeaza contorul literelor */
       while ((c!=EOF)&&incuvant(c)) {
          s[ l++ ]=(char)c;  /* depune litera */
          c=getc(fp);   }
       s[ l ]='\0';  /* depune terminatorul de sir */
       printf("Cuvantul: %s \n", s);   } }
```

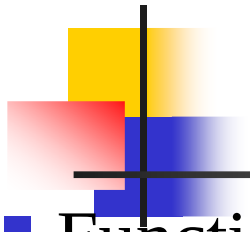
Fişiere text

Aplicaţie de prelucrare a textelor

```
void main(void) {
  char numefis[80];
  FILE *fp;
  printf("Introduceti numele fisierului \n");
  scanf("%80s", numefis);
  if ((fp=fopen(numefis, "r"))==NULL) {
     printf("Eroare deschidere fisier ! \n");
     exit(1);   }
  cuvinte(fp);
  fclose(fp);
}
```

Fişiere text

Aplicaţie de prelucrare a textelor

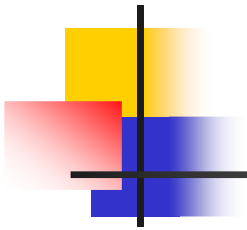


- Funcţia cuvinte primeşte ca parametru un pointer la structura FILE asociată fişierului care va fi prelucrat şi care a fost deschis pentru citire.

- Fişierul este citit caracter cu caracter. Întâi se elimină separatorii, apoi se colectează caracterele care formează un cuvânt.

- Primul caracter diferit de un separator, care a determinat ieşirea din ciclul while (separator(c)), poate fi primul caracter parte din cuvânt, sau poate fi EOF, dacă s-a ajuns la sfârşit de fişier.

Fişiere text
Aplicaţie de prelucrare a textelor



- Dacă s-a ajuns la sfârşit de fişier, instrucțiunea break determină ieşirea din ciclul while deci se termină şi funcția cuvinte. Altfel, caracterul *c* este primul caracter al cuvântului care urmează şi se inițializează *l*, contorul literelor cuvântului curent. Atâta timp cât *c* este un caracter care nu este separator şi nu s-a ajuns la sfârşit de fişier, se depune *c* în tamponul colector *s*. Când s-a ajuns la sfârşitul cuvântului, se adaugă la poziția curentă din *s* şi terminatorul de şir, *\0*.

- Procedeul continuă atâta timp cât nu s-a ajuns la sfârşit de fişier.

Fişiere text

Operaţiile de intrare-ieşire cu format

- Asupra fişierelor se pot aplica operaţii de intrare-ieşire cu format utilizând funcţiile ***fscanf*** şi ***fprintf***. Aceste funcţii sunt similare funcţiilor scanf şi printf, având în plus un parametru de tip pointer la fişier:

int fscanf(FILE*pf,char *format ,adresa1,adresa2, ...);

int fprintf (FILE *pf, char *format, arg1, arg2, ...);

- pentru funcţia fscanf, primul parametru este un pointer la un fişier care a fost deschis pentru *citire*;
- pentru funcţia fprintf primul parametru este un pointer la un fişier care a fost deschis pentru *scriere*.

Fişiere text
Operaţiile de intrare-ieşire cu format

- Exemplu: crearea unui fişier prin *scriere cu format* :

```
#include <stdio.h>
void main(void) {
FILE * fp;
char nume_fis[15]="fis.txt";
int i=5;
char c='z';
float x=7.56;
if ((fp=fopen(nume_fis, "w"))==NULL) {
   printf("Nu se poate deschide fisierul %s \n",nume_fis);
       exit(1);  }
  fprintf( fp, "%d %c %.2f \n", i, c, x);
  fclose( fp );  }
```

Fişiere text

Intrări-ieşiri de şiruri de caractere

- Ca urmare a execuției programului, se crează fişierul cu numele *fis.txt* având conținutul:

  5 z 7.56

- Se pot utiliza funcțiile ***fgets*** şi ***fputs***, similare funcțiilor cunoscute gets şi puts.

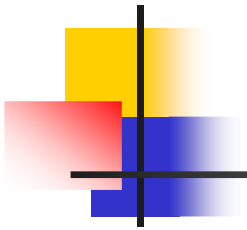
  char *fgets(char *s, int n, FILE *pf);

Funcția fgets citeşte max. *n-1* caractere din fişier sau până la *'\n'* inclusiv, şi le depune în s, adaugă la sfârşit terminatorul *'\0'* şi returnează adresa şirului. La eroare, întoarce valoarea NULL.

Observație: corelând valoarea lui n cu dimensiunea bufferului s, se evită potențiala eroare de programare prezentă la apelul lui gets ().

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

Funcția fputs scrie şirul în fişier, fără a scrie şi caracterul *'\0'*. În caz de eroare, returnează EOF.



Fişiere text
Testul de sfârşit de fişier

- Pentru a testa dacă parcurgerea unui fişier a ajuns la sfârşitul acestuia se poate utiliza funcția ***feof***: ***int feof(FILE * pf);***

Funcția returnează o valoare nenulă (adevărat) dacă s-a atins sfârşitul de fişier.

- *Exemplu:* Dintr-un fişier text se citesc rezultatele la examen ale unei grupe de studenți. Fişierul este organizat pe linii de forma:

nume prenume nota

Se cere să se afişeze câti studenți au note de 9 sau 10, câți au nota între 6 şi 8, câți au 5 şi câți au note sub 5.

- Deoarece inițial nu se cunoaşte numărul de studenți din fişier, prelucrarea (citirea) datelor se face atâta timp cât nu s-a ajuns la sfârşitul fişierului de date de intrare.

Fişiere text

Testul de sfârşit de fişier

```
#include <stdio.h>
#include <stdlib.h>
void main(void) {
   FILE * pf;
   char numefis[100];
   char nume[80],pren[80];
   int nota;
   int n1=0, n2=0, n3=0, n4=0;
   printf("Introduce numele fisierului");
   gets(numefis);
   pf=fopen(numefis, "r");
   if (pf==NULL) {
     printf("Nu pot deschide fisierul %s \n", numefis);
     exit(1);
   }
```

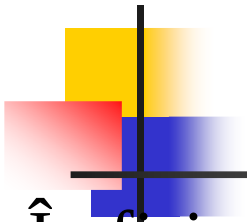
Fişiere text

Testul de sfârşit de fişier

```
   while (!feof(pf)) {
      fscanf(pf, "%s %s %d", nume, pren, &nota);
      if ((nota==10)||(nota==9)) n1++;
      else if ((nota >=6)&&(nota <=8)) n2++;
      else if (nota ==5) n3++;
      else n4++;
    }
    fclose(pf);
    printf("%d %d %d %d", n1, n2, n3, n4);
}
```

Fişiere binare

- Fişierele de tip text nu sunt adecvate pentru a stoca pe disc baze de date în mod eficient. Într-un fişier text, octeţii sunt consideraţi coduri de caractere. Scrierea unei valori numerice într-un fişier text presupune o operaţie de conversie a numărului în şirul de caractere care reprezintă numărul conform formatului de ieşire ales.
- Pentru scrierea valorii numerice în fişierul de tip text se consumă mai mult spaţiu decât pentru reprezentarea în formatul intern din memorie (unde, de ex., variabila de tip *int* ocupă doar 2 sau 4 octeţi). De asemenea, pentru scrierea şi citirea valorilor numerice cu format se fac operaţii de conversie între formatul intern de reprezentare a numerelor şi cel extern (şir de caractere), ceea ce consumă şi timp de execuţie.



Fişiere binare

- În fişierele organizate ca date binare, octeții nu sunt considerați ca fiind coduri de caractere.
- Se consideră că *o înregistrare a fişierului este o colecție de date structurate numite articole*.
- La o citire se transferă din fişier într-o zonă tampon un număr de articole care au lungime fixă. La scriere se transferă din zona tampon în fişier un număr de articole de lungime fixă.
- Operațiile de citire şi de scriere într-un fişier se execută *secvențial*, *implicând poziția curentă din fişier*. La fiecare apel al funcțiilor de citire se citeşte înregistrarea curentă din fişier, iar la scriere se scrie întotdeauna la poziția curentă.

Fişiere binare

Citirea şi scrierea în fişiere binare

- În C, fişierele binare pot fi prelucrate folosind funcțiile ***fread*** şi ***fwrite***.

unsigned fread(void *ptr, unsigned dim, unsigned nrart, FILE *pf);

Citeşte dintr-un fişier un număr de articole de aceeaşi dimensiune şi le stochează într-o zonă de memorie alocată pentru aceasta.

ptr = pointerul spre zona tampon ce va conține articolele citite

dim = lungimea unui articol

nrart = numărul articolelor care se transferă

pf = pointer spre structura *FILE* care defineşte fişierul din care se face citirea şi care trebuie să fi fost deschis în modul *rb*.

Fişiere binare

Citirea şi scrierea în fişiere binare

unsigned fwrite(void *ptr, unsigned dim, unsigned nrart, FILE *pf);

- Parametrii funcției *fwrite* au semnificații analoage cu parametrii funcției *fread*.

- Fişierul *pf* trebuie să fi fost deschis pentru scriere în mod binar, în modurile *wb* sau *ab*.

Fişiere binare

Citirea şi scrierea în fişiere binare

- Ambele funcții returnează numărul articolelor transferate sau -1, în caz de eroare. De obicei, numărul de octeți transferați este *dim*nrart*.

- Dacă în fişierul binar se transferă valorile unor variabile, locațiile de memorie corespunzătoare sunt tratate ca o zonă tampon de scriere:

```
int x=5;  double y=9.3;
fwrite(&x, sizeof(int), 1, fp);
fwrite(&y, sizeof(double), 1, fp);
```

Fişiere binare

Citirea şi scrierea în fişiere binare

- *Exemplu:* Copierea unui fişier (se reia programul de la fişiere text, [19], pag.218). Este mai eficient să se transfere deodată blocuri de dimensiuni mai mari decât un singur caracter. Se defineşte *MAX* ca fiind numărul maxim de octeți transferați deodată.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 500
void main() {
   char nume_sursa[80], nume_dest [80];
   FILE *fs,*fd;
   char buf[MAX];
   int nr;
    printf("Introduceti numele fisierului care se copiaza \n");
    gets(nume_sursa);
   printf("Introduceti numele fisierului copie \n");
   gets(nume_dest);
```

Fişiere binare

Citirea şi scrierea în fişiere binare

```
if ((fs=fopen(nume_sursa, "rb"))==NULL) {
   printf("Nu pot deschide fisierul %s \n", nume_sursa);
   exit(1);
}
if ((fd=fopen(nume_dest, "wb"))==NULL) {
   printf("Nu pot deschide fisierul %s \n", nume_dest);
   exit(1);
 }
 while (!feof(fs)) {
     nr=fread(buf, 1, MAX, fs);
     fwrite(buf,  1, nr,   fd);
  }
  fclose(fs);   fclose(fd);  }
```

Fişiere binare

Citirea şi scrierea în fişiere binare

- Se alocă o zonă de memorie *buf* capabilă să conțină *MAX* octeți. Operația de citire din fişierul sursă este realizată de instrucțiunea: nr=fread(buf, 1, MAX, fs);
- Aceasta transferă din fişierul *fs* cel mult *MAX* articole de *1* octet înspre zona *buf*.
- În general, operația de citire va transfera exact *MAX* octeți, mai puțin în ultima iterație a ciclului când este posibil ca numărul octeților care au mai rămas de transferat să fie mai mic.
- Numărul exact de octeți citiți este returnat de funcția *fread* în variabila *nr*. Funcția *fwrite* va scrie primii *nr* octeți din zona *buf* în fişierul destinație.

Tratarea erorilor
Macroinstrucţiuni specifice

- Când un program execută operații de I/E cu un fişier, trebuie să se testeze întotdeauna valoarea returnată de funcțiile *fopen*, *fputs*, *fgets* şi celelalte, pentru a verifica dacă operația a reuşit.
- Limbajul C dispune de *macroinstrucțiunea* ***ferror***, care examinează un flux de I/E(corespunzător unui fişier) pentru a detecta erorile de citire sau de scriere; macroinstrucțiunea ferror returnează valoarea true dacă apare o eroare şi valoarea false în caz contrar: int ferror(FILE *pf);
- După apariție unei erori, macroinstrucțiunea ferror rămâne adevărată până când nu se invocă în program, pentru fluxul(fişierul) dat, *macroinstrucțiunea* ***clearerr***:

```
void clearerr(FILE *pf);
```

Tratarea erorilor
Coduri şi mesaje de eroare

- Unele apeluri de funcţii sistem(inclusiv de I/E), în plus faţă de valoarea returnată, poziţionează în caz de eroare şi un cod în variabila globală ***errno***, declarată în *errno.h:*
```
extern int errno;
```
- Descrierea în cuvinte a erorii este furnizată de funcţia ***strerror*** din *string.h*:
```
char *strerror(int errnum);
```
- Un rol asemănător are funcţia ***perror*** din *stdio.h*, care tipăreşte mesajul *s* dat de utilizator, caracterul **:** şi apoi descrierea erorii dată de sistem:
```
void perror(const char *s);
```
- *Exemplu:* un program care citeşte conţinutul unui fişier şi-l afişează pe ecran. După fiecare operaţie de I/E se testează existenţa erorilor. Dacă apare o eroare, programul se întrerupe şi afişează un mesaj de eroare la *stderr*.

Tratarea erorilor

Exemplu de program: citirea şi afişarea unui fişier

```
#include<stdio.h>
#include<stdlib.h>
void main(int argc, char *argv[ ]) {
   FILE *pointer_fisier;   char linie[256];
   if (pointer_fisier=fopen(argv[1],"r")) {
     while(fgets(linie, sizeof(linie), pointer_fisier)) {
       if (ferror(pointer_fisier)) {
          fprintf(stderr, "Eroare la citire %s\n", argv[1]); exit(1); }
       else {fputs(linie, stdout);
              if (ferror(stdout)) {
          fprintf(stderr,"Eroare la scriere in stdout\n"); exit(1);}}}}
   else printf("Eroare la deschidere %s\n", argv[1]);  }
```

Tratarea erorilor

Exemplu: copierea a două fişiere(PC2-M.Minea)

```
#include<stdio.h>
#include<errno.h>
#define MAX 512
int filecopy(FILE *fi, FILE *fo) {
   char buf[MAX];
   int dim;  /* nr. octeti cititi */
   while (!feof(fi)) {
      dim = fread(buf, 1, MAX, fi);
      fwrite( buf, 1, dim, fo );
      if (ferror( f i ) || ferror( fo ))  return errno;  }
    return 0;  }
```

Tratarea erorilor

Exemplu de program: copierea a două fişiere

```
int main(int argc, char *argv[ ]) {
   FILE *fi, *fo;
   if (argc != 3) {
      fprintf(stderr, “Nr. de argumente eronat\n”);
      return -1;
   }
   else { if (!(fi=fopen(argv[1], ”r”))) {
      fprintf(stderr,“%s: Er. la deschidere %s:”,argv[0],
                  argv[1]);
      perror( NULL ); /* mesajul s-a scris deja */
      return errno; }
```

Tratarea erorilor

Exemplu de program: copierea a două fişiere

```
    if (!(fo=fopen(argv[2], ”w”))) {
        fprintf(stderr,“%s: Er.la deschidere %s:”,argv[0],
                                    argv[2]);
        perror( NULL );
        return errno;
    }
    if (filecopy( fi, fo ))  perror(“Eroare la copiere”);
    if (fclose( fi ) | | fclose( fo))  perror(“Err. la inchidere”);
  }
  return errno;
}
```

Tratarea erorilor

Redirectarea *stderr* prin redeschidere

- Un pointer de fişier deschis se poate modifica(suprascrie) pentru a reprezenta un alt fişier, prin operația de **redeschidere:** funcția ***freopen*** (din *stdio.h*).

```
FILE  *freopen(const char *numefisier, const char
                                *mod_acces, FILE *pf);
```

- Funcția freopen se deosebeşte de funcția fopen prin faptul că primeşte un pointer de fişier a cărui valoare se doreşte a fi suprascrisă: refolosită pentru a desemna alt fişier.
- Dacă funcția reuseşte suprascrierea, va returna un pointer la fluxul originar de fişier. Dacă apare o eroare, va returna *NULL*.

Tratarea erorilor
Redirectarea *stderr* prin redeschidere

- *Exemplu*: un program care redirectează fişierul *stderr* către fişierul *standard.err*, şi nu spre ecran, utilizând operația de redeschidere.

```
#include <stdio.h>
void main(void)
  {
    if (freopen("standard.err", "w", stderr))
      fputs("stderr a fost redirectat", stderr);
    else
      printf("Eroare la redeschidere\n");
  }
```

Actualizarea unui fişier

Prelucrarea fişierelor în acces direct

- Dacă se doreşte accesul aleator într-un fişier, se poate utiliza funcția de poziționare forțată, ***fseek***.

int fseek(FILE *pf, long deplasament, int origine);

pf = pointer către structura *FILE* care defineşte fişierul în care se face poziționarea

deplasament = diferența, în octeți, între poziția de referință față de care se face deplasarea, indicată de *origine*, şi noua poziție care se doreşte a fi obținută.

origine = un cod reprezentând poziția de referința față de care se face deplasarea:

- începutul fişierului, codificat cu 0 (SEEK_SET);
- poziția curentă, codificată cu 1 (SEEK_CUR);
- sfârşitul fişierului, codificat cu 2 (SEEK_END).

- Funcția *fseek* returnează valoarea 0 dacă poziționarea a fost corectă şi o valoare nenulă în caz contrar.

Actualizarea unui fişier
Poziţionarea într-un fişier

- De exemplu, următoarea instrucțiune poziționează capul de citire/scriere la începutul fişierului:

```
fseek(pf, 0, 0);
```

- Pentru aflarea poziției curente în fişier se poate utiliza funcția ***ftell:***

long ftell(FILE *pf);

Funcția returnează o valoare reprezentând deplasamentul, în octeți, a poziției curente a capului de citire/scriere față de începutul fişierului.

Actualizarea unui fişier
Exemplu de program

- Să se realizeze un program care actualizează evidența unei grupe de studenți. Datele despre studenți (numele, vârsta, media) se păstrează sub forma unui fişier text. Programul trebuie să permită următoarele opțiuni:

  a, A – adăugarea unui nou student în fişier;

  l,L – listarea datelor tuturor studenților;

  m,M – modificarea datelor unui student;

  x,X – terminarea programului.

Actualizarea unui fişier

Exemplu de program

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#define LNUME 20 /* lungimea maxima a numelor */
typedef struct {
    char nume[LNUME];
    int varsta;
    float medie;
} student;
char fisier[13];    /* numele fisierului */
```

Actualizarea unui fişier
Exemplu de program

```
void AdaugStudent(void) {
   FILE *f;
   student s;
   if (!(f=fopen(fisier, “a”))) {
      puts(“\nFisierul nu poate fi deschis.”);
      return;  }
   printf(“\nNumele, varsta, media:”);
   scanf(“%s %d %f”, s.nume, &s.varsta, &s.medie);
   fflush(stdin); /* se goleste buffer-ul tastaturii */
   fprintf(f,“%20s %2d %6.2f\n”,s.nume,s.varsta,s.medie);
   fclose(f);
}
```

Actualizarea unui fişier
Exemplu de program

```
void ListezStudenti(void) {
   FILE *f;
   student s;
   if(!(f=fopen(fisier, “r”))) {
      puts(“\nFisierul nu poate fi deschis. “);
      return;  }
   while(fscanf(f,”%s %d %f”, s.nume, &s.varsta,
&s.medie)!=EOF)
       printf(“\n%-20s %2d %6.2f\n”, s.nume, s.varsta,
                  s.medie);
   fclose(f);
 }
```

Actualizarea unui fişier
Exemplu de program

```
void ModificStudent(void) {
 int gasit=0;
 FILE *f; student s;
 char n[LNUME]; /* numele stud. ale carui date se modifica  */
 if(!(f=fopen(fisier, “r+”))) {
    puts(“\nFisierul nu poate fi deschis. “);   return; }
 printf(“\nNume student: “);  fgets(n, LNUME, stdin);
 n[strlen(n)-1] = ‘\0’;
 while(fscanf(f,”%s %d %f”,s.nume,&s.varsta,&s.medie)!=EOF)
    if (!strcmp(n, s.nume))  { /* stud. gasit, se afiseaza datele */
        printf(“\n%-20s %2d %6.2f\n”, s.nume, s.varsta, s.medie);
        gasit=1;  break; }
```

Actualizarea unui fişier
Exemplu de program

```
  if (!gasit)
     printf("\nStudentul %s nu exista in fisier.",n);
  else {
    printf("\nNumele, varsta, media:");
    scanf("%s %d %f", s.nume, &s.varsta, &s.medie);
    fflush(stdin);
    /* pozitionare la inceputul inregistrarii */
    fseek(f, -32, SEEK_CUR);
    fprintf(f,"%20s %2d %6.2f\n",s.nume,s.varsta,s.medie);
  }
  fclose(f);
}
```

Actualizarea unui fişier
Exemplu de program

```
void AfisezMeniu(void) {
    puts("\na,A ----- adaugare student");
    puts("\nm,M ----- modificare date student");
    puts("\nl,L ----- listare studenti");
    puts("\nx,X ----- parasire program");
}
void main(void) {
   char opt;
   puts("Nume fisier: ");
   fgets(fisier, 13, stdin);
```

Actualizarea unui fişier
Exemplu de program

```
while(1) {
    AfisezMeniu();
    opt=tolower(getche());
    switch(opt) {
        case ‘a’: AdaugStudent(); break;
        case ‘m’: ModificStudent(); break;
        case ‘l’: ListezStudenti(); break;
        case ‘x’: exit(0);
        default: puts(“Comanda eronata\n”); }
  }
}
```

Redirectarea intrării şi ieşirii standard

- Datele introduse de la un terminal se consideră că formează fişierul standard de intrare. În mod analog, datele care se afişează pe terminal se consideră că formează fişierul standard de ieşire.
- În cazul fişierelor de intrare de la tastatură, sfârşitul de fişier se generează prin *<CTRL>Z*.
- Intrarea şi ieşirea standard pot fi *redirectate*. În felul acesta, toate funcțiile de intrare-ieşire din biblioteca *stdio* (***scanf, printf, putchar, getchar***, etc.) pot fi folosite pentru a realiza operații de intrare-ieşire cu alte periferice decât terminalul standard.
- Redirectarea se indică la lansarea programului din linia de comandă.

Redirectarea intrării şi ieşirii standard

- Redirectarea ieşirii în fişierul *fis_out*: ***prog>fis_out***
- Redirectarea intrării din fişierul *fis_in* se face: ***prog<fis_in***
- Redirectarea ieşirii unui program *prog1* ca intrare pentru un alt program *prog2*: ***prog1|prog2***

Exemplu: Redirectarea intrării şi ieşirii standard.

Programul *copiaza.c* realizează copierea, caracter cu caracter, a intrării standard la ieşirea standard. Programul poate fi folosit pentru realizarea unei copii a unui fişier *f1*, dacă intrarea standard a programului este redirectată către acesta, iar ieşirea standard către fişierul copie care se crează.

Redirectarea intrării şi ieşirii standard

```
/* copiaza.c */
#include <stdio.h>
 void main() {
  int c;
  while ((c=getchar())!=EOF)
    putchar(c);
}
```

- În linia de comandă, se va da comanda:

  copiaza <f1 >copie

- Rezultatul va fi crearea fişierului cu numele *copie*, având acelaşi conținut ca şi fişierul *f1*.