

3. FIȘIERE

3.1. Noțiuni introductive

Un *fișier* este o colecție de date de același tip, memorate pe suport extern (hard-disc, dischetă, CD, etc).

Avantajele utilizării fișierelor sunt o consecință a proprietăților memoriilor externe. Fișierele permit memorarea unui volum mare de date persistente (care nu se „pierd” la încheierea execuției programului sau la închiderea calculatorului).

Până acum am lucrat numai cu date citite de la tastatură, memorate în variabile simple. Pe parcursul execuției unui program, se alocă memorie în RAM pentru variabilele utilizate în programul respectiv. Memoria alocată se eliberează fie pe parcursul execuției programului, fie la terminarea execuției acestuia. În orice caz, după terminarea programului, valorile variabilelor „s-au pierdut”, deoarece zona de memorie în care au fost memorate s-a eliberat și dacă doream să executăm încă o dată programul cu aceleași date de intrare, trebuia să le introducem din nou de la tastatură. Chiar și pentru programe cu un volum mic de date de intrare, așa cum am elaborat noi până acum, acest aspect reprezintă un inconvenient. Dar imaginați-vă ce s-ar întâmpla la o firmă, care, de exemplu, la fiecare rulare a programului de salarii ar trebui să introducă toate datele despre angajații firmei sau la o bibliotecă, la care, de exemplu, orice cerere de împrumut s-ar solda cu introducerea tuturor cărților existente în bibliotecă! Exemplele pot continua, dar ideea principală rămâne că în orice situație practică fișierele sunt indispensabile.

Fișierele pot fi clasificate după conținutul lor astfel:

- fișiere **text** – conțin o secvență de caractere ASCII, structurate pe linii;
 - fișiere **binare** – conțin o secvență de octeți, fără o structură predefinită.
- Noi vom studia modul de utilizare a fișierelor text.

3.2. Fișiere text în limbajul C++

Declararea fișierelor

Operațiile de intrare/ieșire în limbajul C++ au utilizat până acum *stream*-urile `cin` și `cout`, care erau automat asociate tastaturii, respectiv ecranului. Pentru ca un program să poată citi informații dintr-un fișier, respectiv să scrie informații într-un fișier, trebuie să asociem fișierul respectiv unui *stream* de intrare/ieșire.

În fișierul antet `fstream` sunt declarate clasele `ifstream`, `ofstream` și `fstream`. Pentru a utiliza într-un program operații de intrare/ieșire folosind fișiere trebuie să declarăm variabile de tipurile `ifstream`, `ofstream` sau `fstream`. Mai exact:

`ifstream` – declarare *stream* de intrare (numai pentru operații de citire);
`ofstream` – declarare *stream* de ieșire (numai pentru operații de scriere);
`fstream` – declarare *stream* de intrare/ieșire (în care se pot realiza atât operații de citire, cât și operații de scriere, în funcție de modul specificat la deschidere).

Exemple

```
ifstream f1; //am declarat un stream de intrare denumit f1
ofstream f2; //am declarat un stream de iesire denumit f2
fstream f3; //am declarat un stream denumit f3.
```

Deschiderea fișierelor

Pentru a utiliza *stream*-ul declarat trebuie să îl deschidem. La deschidere se asociază *stream*-ului un fișier fizic (existent pe suport extern) și, eventual, se precizează modul de deschidere, care determină operațiile permise cu fișierul respectiv. Deschiderea se poate realiza în două moduri:

- după declarare, prin apelul funcției membru `open`, precizând ca parametri un șir de caractere, care reprezintă specificatorul fișierului fizic, conform sintaxei sistemului de operare utilizat și (eventual) modul de deschidere;
- la declarare, specificând după numele *stream*-ului care se declară numai parametrii corespunzători funcției `open`, încadrați între paranteze rotunde.

La deschidere este obligatoriu să specificăm modul de deschidere dacă fișierul este declarat de tip `fstream`. Pentru tipul `ifstream` se utilizează implicit modul `ios::in`, iar pentru tipul `ofstream` se utilizează implicit modul `ios::out`.

Exemplul 1

```
ifstream f1;
f1.open("date.in");
```

Am declarat un *stream* de intrare denumit `f1`, apoi l-am deschis cu ajutorul funcției `open`, asociindu-l fișierului `date.in`. Deoarece nu am precizat discul logic pe care se află acest fișier sau calea către directorul în care se află fișierul, se deduce că fișierul se află pe hard-disc în directorul curent. Același efect l-am fi obținut prin deschiderea fișierului la declarare, astfel:

```
ifstream f1("date.in");
```

Exemplul 3

```
fstream f3;
f3.open("nr.in", ios::in);
```

Am declarat un *stream* denumit `f3`, pe care la deschidere l-am asociat fișierului `nr.in`. Deoarece tipul `fstream` nu are asociat un mod de deschidere implicit, a fost necesar să precizăm modul de deschidere. În acest caz modul de deschidere

este `ios::in`, ceea ce înseamnă că fișierul a fost deschis ca fișier de intrare (numai pentru operații de citire).

Exemplul 4

```
fstream f4;  
f4.open("nr.out", ios::out);
```

Am declarat un *stream* denumit `f4`, pe care la deschidere l-am asociat fișierului `nr.out`. Modul de deschidere specificat este `ios::out`, ceea ce semnifică deschiderea fișierului ca fișier de ieșire (se creează un fișier vid cu numele specificat, în care se vor realiza doar operații de scriere).

Citirea datelor dintr-un fișier

După deschiderea fișierului ca fișier de intrare se pot realiza operații de citire. În acest scop se poate utiliza operatorul de citire `>>`, sau pot fi utilizate funcții membru specifice.

Una dintre particularitățile citirii cu operatorul `>>` este faptul că **ignoră caracterele albe**. Prin urmare dacă intenționăm să citim toate caracterele din fișier (inclusiv spațiu, tab, enter) acest mod de citire este inadecvat.

În acest scop putem utiliza funcția membru `get()`. Funcția `get()` are mai multe forme de apel. De exemplu, pentru citirea caracterelor (`char` sau `unsigned char`) se poate utiliza astfel:

```
ifstream fl("text.in");  
char c;  
fl.get(c);
```

Am citit primul caracter din fișier, indiferent dacă este sau nu caracter alb.

Scrierea datelor într-un fișier

După deschiderea fișierului ca fișier de ieșire, se pot realiza operații de scriere utilizând operatorul de scriere `<<`.

Exemplu

```
ofstream fout("date.out");  
fout<<"Salut";
```

Operații de test

Detectarea sfârșitului de fișier

În multe aplicații este necesar să citim toate datele existente într-un fișier, fără să avem informații prealabile despre numărul acestora.

La deschiderea unui fișier se creează un **pointer de fișier**, care indică poziția curentă în *stream*. Orice operație de citire determină deplasarea pointerului de citire

în *stream*-ul de intrare, respectiv orice operație de scriere determină deplasarea pointerului de scriere în *stream*-ul de ieșire.

Pentru a testa dacă pointerul de fișier a ajuns la sfârșitul fișierului putem utiliza funcția membru `eof()`. Funcția `eof()` returnează valoarea 0 dacă nu am ajuns la sfârșitul fișierului, respectiv o valoare diferită de 0 dacă am ajuns la sfârșit.

Exemplu

Vom citi și vom număra toate caracterele existente într-un fișier, până la sfârșitul acestuia.

```
ifstream fin("text.in");
char c;
long int nr=0;
while (!fin.eof()) {fin.get(c); nr++;}
cout<<"Fișierul contine "<<nr<<" caractere\n";
```

Funcția membru `get()` returnează pointerul de fișier (poziția curentă în *stream*). Prin urmare secvența de program ar putea fi rescrisă și astfel:

```
ifstream fin("text.in");
char c;
long int nr=0;
while (fin.get(c)) nr++;
cout<<"Fișierul contine "<<nr<<" caractere\n";
```

Testarea reușitei unei operații de intrare/ieșire

În aplicațiile didactice de cele mai multe ori presupunem că datele de intrare sunt corecte. În condiții practice însă trebuie să verificăm reușita oricărei operații de intrare/ieșire și mai mult, să verificăm și corectitudinea datelor citite.

Pentru a testa reușita unei operații de intrare/ieșire se poate utiliza de exemplu funcția membru `good()`. Funcția `good()` returnează o valoare diferită de 0 dacă operația precedentă de intrare/ieșire s-au efectuat cu succes și 0 în caz contrar.

Există și alte funcții de test asemănătoare: `bad()` (care returnează o valoare diferită de 0 dacă s-a efectuat o operație incorectă) sau `fail()` (care returnează o valoare diferită de 0 dacă a apărut o eroare în operațiile de intrare/ieșire până în acel moment).

Închiderea unui fișier

După realizarea tuturor operațiilor cu un fișier, acesta trebuie închis. Închiderea unui fișier se realizează prin apelarea funcției membru `close()`.

Exemplu

```
f1.close();
```

Operația de închidere este obligatorie, în special pentru fișierele de ieșire. *În cazul în care nu închideți fișierul de ieșire, există mari șanse să pierdeți informații!*

3.3. Fișiere text în limbajul C

Declararea fișierelor

Operațiile de intrare/ieșire în limbajul C au fost realizate până acum utilizând instrucțiunile `scanf()` și `printf()`, care implicit preiau informații de la tastatură și, respectiv, le transmit spre ecran. Pentru ca un program să poată prelucra informații dintr-un fișier, acesta trebuie declarat, indicându-se numele lui precum și operațiile permise asupra conținutului fișierului respectiv.

În fișierul antet `<stdio>` sunt declarate, printre altele, și funcțiile, constantele, tipurile de date și variabilele globale cu ajutorul cărora pot fi prelucrate informațiile conținute în fișiere. Pentru a utiliza într-un program operații de intrare/ieșire folosind fișiere trebuie să declarăm una sau mai multe variabile de tip `FILE *`. Declarația unui fișier se face astfel:

```
FILE * fisier;
```

Deschiderea fișierelor

Desigur, pentru a avea acces la informațiile din fișier trebuie să fie cunoscut fișierul fizic de pe suport (specificatorul de fișier conforma sintaxei sistemului de operare utilizat), ca și modul în care vor fi prelucrate informațiile din fișier. Acest lucru se realizează utilizând funcția `fopen`.

```
fisier = fopen(const char *nume, const char *mod)
```

unde `nume` este o constantă șir de caractere care reprezintă specificatorul fișierului fizic pe suport, iar `mod` este o constantă șir de caractere care indică modul în care vor fi accesate informațiile din fișier.

Exemple

```
FILE *f1, *f2, *f3; //am declarat trei fișiere f1, f2, f3
f1 = fopen("date.in", "r");
//am deschis fișierul de intrare (r - read) f1
f2 = fopen("date.out", "w");
//am deschis fișierul de ieșire (w - write) f2
```

Declarația și deschiderea unui fișier se pot face și simultan:

Exemplu 1

```
FILE *f1 = fopen("date.in", "r");
```

Am declarat fișierul de intrare denumit `f1`, apoi l-am deschis cu ajutorul funcției `fopen`, asociindu-l fișierului `date.in`. Deoarece în toate exemplele anterioare nu am precizat discul logic pe care se află fișierele sau calea către directorul în care se află acestea, se deduce că fișierele se află pe hard-disc în directorul curent.

Exemplul 2

```
FILE *f2;  
f2 = fopen("date.out", "w");
```

Am declarat un fișierul de ieșire, denumit `f2`, pe care l-am asociat fișierului fizic `date.out` în directorul curent.

Observație

Dacă, după operația de deschidere a fișierului, variabila corespunzătoare are valoarea `NULL`, deducem că operația de deschidere a eșuat (de exemplu, dacă dorim să deschidem un fișier de intrare care nu există). Este indicat să testăm succesul operației de deschidere înainte de a efectua orice altă operație cu fișierul.

Exemplul 3

```
#include <stdio.h>  
using namespace std;  
int main()  
{ FILE *numef;  
  if (!(numef = fopen("nr.in", "r")))  
    printf("EROARE la deschidere\n");  
  else printf("Deschidere corecta a fisierului\n");  
  return 0; }
```

Testarea erorii la deschiderea fișierului se poate face și astfel:

```
if ((numef = fopen("nr.in", "r")) == NULL)
```

Citirea datelor dintr-un fișier

După deschiderea fișierului ca fișier de intrare se pot realiza operații de citire. În acest scop se utilizează funcția `fscanf`.

```
int fscanf(fisier, format, adr_var1, ..., adr_varn);
```

Prototipul funcției `fscanf()` este asemănător cu prototipul funcției `scanf()`. Funcția `fscanf()` are în plus, ca prim parametru, fișierul din care se face citirea. În rest, semnificația parametrilor, rezultatului și efectul funcției sunt aceleași ca în cazul funcției `scanf()`.

Citirea datelor cu format specificat

Funcția `scanf()` permite citirea datelor sub controlul unui format specificat. Această funcție este declarată în fișierul antet `stdio.h` și are următorul format:

```
int scanf(format, adr_var1, adr_var2, ..., adr_varn);
```

Efect

Funcția parcurge succesiunea de caractere introdusă de la tastatură și extrage valorile care trebuie citite conform formatului specificat. Valorile citite sunt memorate în ordine în variabilele specificate prin adresa lor în lista de parametri ai

funcției `scanf()`. Adresa unei variabile se obține cu ajutorul operatorului de referențiere (`&`). Acest operator este unar (are un singur operand, care trebuie să fie o variabilă): `&variabilă`.

Funcția `scanf()` returnează numărul de valori citite corect. În cazul unei erori, citirea se întrerupe în poziția în care a fost întâlnită eroarea.

Observație

Dacă în lista de parametri ai funcției `scanf()` specificați doar numele variabilei, nu adresa acesteia, funcția va citi valoarea corespunzătoare acestei variabile, dar la ieșirea din funcție valoarea citită nu este atribuită variabilei respective.

Parametrul `format` este un șir de caractere care poate conține specificatori de format, caractere albe și alte caractere.

Caracterele albe vor fi ignorate. Celelalte caractere (care nu fac parte dintr-un specificator de format) trebuie să fie prezente la intrare în pozițiile corespunzătoare.

Specificatorii de format au următoarea sintaxă:

```
% [*] [lg] [l|L] literă_tip
```

Observați că orice specificator de format începe cu caracterul `%`, conține obligatoriu o literă care indică tipul valorii care se citește și eventual alte elemente opționale. Litera care indică tipul poate fi, de exemplu:

literă_tip	Semnificație
d	Se citește un număr întreg scris în baza 10, care va fi memorat într-o variabilă de tip <code>int</code>
o	Se citește un număr întreg scris în baza 8, care va fi memorat într-o variabilă de tip <code>int</code>
u	Se citește un număr întreg scris în baza 10, care va fi memorat într-o variabilă de tip <code>unsigned int</code>
x	Se citește un număr întreg scris în baza 16, care va fi memorat într-o variabilă de tip <code>int</code>
f, e sau g	Se citește un număr real scris în baza 10, care va fi memorat într-o variabilă de tip <code>float</code>
c	Se citește un caracter (în acest caz caracterele albe prezente la intrare nu se ignoră).
s	Se citește un șir de caractere (șirul începe cu următorul caracter care nu este alb și continuă până la primul caracter alb întâlnit sau până la

	epuizarea dimensiunii maxime <code>lg</code> din specificatorul de format) .
--	--

Opțional unele litere tip pot fi precedate de litera `l` sau de litera `L`. Literele `d`, `o` și `x` pot fi precedate de litera `l`, caz în care valoarea citită va fi convertită la tipul `long int`. Litera `u` poate fi precedată de litera `l`, caz în care valoarea citită va fi convertită la tipul `unsigned long int`. Literele `f`, `e`, `g` pot fi precedate de litera `l` (caz în care valoarea citită este convertită la tipul `double`) sau de litera `L` (caz în care valoarea citită este convertită la tipul `long double`).

Opțional poate fi specificată și `lg` – lungimea maximă a zonei din care se citește valoarea. Mai exact, funcția `scanf()` va citi maxim `lg` caractere, până la întâlnirea unui caracter alb sau caracter neconvertibil în tipul specificat de `literă_tip`.

Caracterul opțional `*` specifică faptul că la intrare este prezentă o dată de tipul specificat de acest specificator de format, dar ea nu va fi atribuită nici uneia dintre variabilele specificate în lista de parametri ai funcției `scanf()`.

Observație

Caracterul `%` are o semnificație specială în parametrul format, el indică începutul unui specificator de format. Dacă dorim să specificăm în parametrul format că la intrare trebuie să apară caracterul `%`, vom utiliza construcția sintactică `%%`.

Exemple

1. Să considerăm următoarele declarații de variabile:

```
int a; unsigned long b; double x;
```

Să presupunem că de la tastatură sunt introduse caracterele: 312 -4.5 100000. Apelând funcția:

```
scanf("%d %lf %lu", &a, &x, &b);
```

Variabilei `a` i se atribuie valoarea 312, variabilei `x` i se atribuie valoarea -4.5, iar variabilei `b` i se atribuie valoarea 100000.

2. Să considerăm următoarea declarație de variabile:

```
char c1, c2, c3;
```

Să presupunem că de la tastatură sunt introduse caracterele: 312. Apelând funcția:

```
scanf("%c%c%c", &c1, &c2, &c3);
```

Variabilei `c1` i se atribuie caracterul '3', variabilei `c2` i se atribuie caracterul '1', iar variabilei `c3` i se atribuie caracterul '2'.

3. Să considerăm următoarele declarații de variabile:

```
char c1, c2; unsigned a;
```

Să presupunem că de la tastatură se introduce: 312 xd. Apelând funcția:


```
scanf("%u%c%c", &a, &c1, &c2);
```

Variabilei `a` i se atribuie valoarea 312, variabilei `c1` i se atribuie caracterul ' ' (spațiu), iar variabilei `c2` i se atribuie caracterul 'x'.

Apelând funcția:

```
scanf("%u %c %c", &a, &c1, &c2);
```

Variabilei `a` i se atribuie valoarea 312, variabilei `c1` i se atribuie caracterul 'x', iar variabilei `c2` i se atribuie caracterul 'd'.

4. Să considerăm următoarea declarație de variabile:

```
int a, b, c;
```

Să presupunem că de la tastatură se introduce: 3=1+2. Apelând funcția:

```
scanf("%d=%d+%d", &a, &b, &c);
```

Variabilei `a` i se atribuie valoarea 3, variabilei `b` i se atribuie valoarea 1, iar variabilei `c` i se atribuie valoarea 2.

Apelând funcția

```
scanf("%d=%*d+%d", &a, &b);
```

Variabilei `a` i se atribuie valoarea 3, variabilei `b` i se atribuie valoarea 2. Valoarea 1 a fost citită dar nu a fost atribuită nici unei variabile.

1. Să presupunem că de la tastatură se introduce o dată forma `zz-ll-aa`. Pentru a citi ziua, luna și anul putem apela funcția:

```
scanf("%d-%d-%d",&zi, &luna, &an);
```

Scrierea datelor într-un fișier

După deschiderea fișierului ca fișier de ieșire, se pot realiza operații de scriere utilizând funcția `fprintf()`.

```
int fprintf(fisier, format, expresie1, ..., expresie_n);
```

Prototipul funcției `fprintf()` este asemănător cu prototipul funcției `printf()`. Funcția `fprintf()` are în plus, ca prim parametru, fișierul în care se face afișarea. În rest, semnificația parametrilor, rezultatului și efectul funcției sunt aceleași ca în cazul funcției `printf()`.

Afișarea datelor cu format

Afișarea datelor pe ecran cu un format specificat se realizează apelând funcția `printf()`. Această funcție este declarată în fișierul antet `cstdio` și are următoarea sintaxă:

```
int printf(format, expresie1, expresie2, ..., expresie_n);
```

Efect

Se evaluează expresiile și se scriu în ordine valorile acestora, în forma specificată de parametrul `format`.

În caz de succes, funcția returnează numărul de caractere afișate.

Parametrul `format` este un șir de caractere care poate conține specificatori de format și eventual alte caractere. În parametrul `format` trebuie să existe câte un specificator de format pentru fiecare expresie din lista de parametri (specificatorul 1 corespunde expresiei 1, specificatorul 2, corespunde expresiei 2, etc).

Celelalte caractere din parametrul `format` vor fi afișate pe ecran în pozițiile corespunzătoare.

Un specificator de format are următoarea sintaxă:

`% [ind] [lg] [.prec] [l|L] literă_tip`

Observați că un specificator de format începe cu caracterul `%`, trebuie să conțină obligatoriu o literă care să indice tipul expresiei corespunzătoare și eventual alte elemente opționale. Litera care indică tipul poate fi, de exemplu:

literă_tip	Semnificație
d	Expresia se convertește din tipul <code>int</code> și se afișează în baza 10.
o	Expresia se convertește din tipul <code>int</code> și se afișează în baza 8.
u	Expresia se convertește din tipul <code>unsigned</code> și se afișează în baza 10
x, X	Expresia se convertește din tipul <code>int</code> și se afișează în baza 16 (literele care reprezintă 10,11,12,13,14,15 sunt majuscule pentru X sau minuscule pentru x).
f	Expresia se convertește din tipul <code>float</code> și se afișează în formatul <code>parte_întreagă.parte_zecimală</code>
e, E	Expresia se convertește din tipul <code>float</code> și se afișează în format exponențial (științific) <code>cifră.parte_zecimalăe±exponent</code> Pentru E litera care precedă exponentul este E .
g	Se aplică una dintre conversiile corespunzătoare literei <code>f</code> sau literei <code>e</code> (cea care se reprezintă cu număr minim de caractere)
c	Expresia are ca valoare codul ASCII al unui caracter și se afișează caracterul corespunzător.
s	Expresia este un șir de caractere care va fi afișat pe ecran.

Opțional, înainte de `literă_tip` poate să apară litera `l` sau litera `L`. Dacă litera `l` apare înainte de `d`, `o`, `x`, `X` sau `u` conversia se realizează din tipul `long int` (respectiv `unsigned long int`). Dacă litera `l` apare înainte de `f`, `e`, `E`

sau `g` conversia se realizează din tipul `double`. Litera `L` poate să preceadă doar literele `f`, `e`, `E` sau `g` (caz în care conversia se realizează din tipul `long double`).

Opțional poate fi specificat `lg` – numărul minim de caractere pe care se va realiza afișarea. Dacă numărul de caractere necesar pentru a afișa valoarea expresiei depășește `lg`, se vor utiliza atâtea caractere câte sunt necesare. Dacă numărul de caractere necesare pentru a afișa valoarea expresiei este mai mic decât `lg`, se vor utiliza `lg` caractere (restul zonei pe care se realizează afișarea completându-se cu spații). Modul de aliniere implicit este la dreapta (deci completarea cu spații se realizează în stânga). Dacă se specifică indicatorul – (minus) atunci alinierea se va face la stânga (și completarea cu spații se va realiza la dreapta).

Pentru expresiile de tip real sau șir de caractere se poate specifica și precizia `prec`. Pentru valorile reale aceasta indică numărul de cifre de la partea zecimală care vor fi afișate (implicit 6 zecimale). Dacă precizia este mai mică decât numărul de zecimale ale numărului real afișat, atunci valoarea afișată va fi rotunjită (dacă prima zecimală care nu se afișează este mai mare sau egală cu 5, atunci ultima zecimală afișată va fi mai mare cu 1).

Pentru șiruri de caractere, precizia indică numărul de caractere din șir care se afișează (primele `prec`).

Exemple

Să considerăm următoarele declarații de variabile

```
int a=-1, b=0567, d=0xf01a;
char c='x';
float x=-123.147, y=0.00008;
```

Să urmărim efectul următoarelor apeluri ale funcției `printf()`:

Apel	Pe ecran se afișează:
<code>printf("a=%d sau a=%u\n", a, a);</code>	<code>a=-1 sau a=65535</code>
<code>printf("x=%f sau\n x=%e sau\n x=%g\n", x, x, x);</code>	<code>x=-123.147000 sau x=-1.231470e+02 sau x=-123.147</code>
<code>printf("y=%f sau\n y=%e sau\n y=%g\n", y, y, y);</code>	<code>y=0.000080 sau y=8.000000e-05 sau y=8e-05</code>
<code>printf("b=%d sau b=%o sau b=%x \n", b, b, b);</code>	<code>b=375 sau b=567 sau b=177</code>
<code>printf("c=%c sau c=%d\n", c, c);</code>	<code>c=x sau c=120</code>
<code>printf("d=%d sau d=%x sau\n d=%u\n", d, d, d);</code>	<code>d=-4070 sau d=f01a sau d=61466</code>
<code>printf("x=%.2f\n", x);</code>	<code>x=-123.15</code>
<code>printf("b=%6dxb=%-6dx\n", b, b);</code>	<code>b= 375xb=375 x</code>

Închiderea fișierelor

După terminarea tuturor operațiilor care implică lucrul cu fișiere acestea trebuie închise. Funcția care realizează acest lucru este:

```
fclose(fisier);
```

Exemplul 1

Să scriem un program care citește din fișierul de intrare `nr.in` un număr natural n ($0 < n \leq 100$) și apoi n valori naturale și afișează în fișierul de ieșire `nr.out` numerele citite descompuse în factori primi, pe fiecare linie fiind scris câte un număr urmat de perechi de forma (d, p) , indicând divizorul și puterea lui.

```
#include <cstdio>
using namespace std;
int main()
{ int n, i, d, p, er;
  unsigned long x;
  FILE *fin = fopen("nr.in", "r");
  if (!fin)
    {printf("Eroare la deschiderea fisierului nr.in\n");
     return 1;}
  FILE *fout = fopen("nr.out", "w");
  er = fscanf(fin, "%d", &n); //citesc n din fisierul fin
  if (!er || n>100 || n<0)
    {printf("Numarul de valori este incorect\n");
     return 2;}
  for (i=0; i<n; i++) //citesc cele n valori
    { er = fscanf(fin, "%lu", &x);
      if (!er)
        { printf("Eroare la citirea valorilor\n");
         return 3;}
      fprintf(fout, "%10lu ", x);
      d=2;
      while (x>1)
        { p=0;
          while (x%d==0)
            { p++; x/=d; }
          if (p) fprintf(fout, "(%d,%d)", d, p);
          d++;}
      fprintf(fout, "\n");
    }
  fclose(fout);
  return 0;
}
```

Exemplul 2

Scrieți un program care creează o copie a fișierului `intrare.txt` cu numele `iesire.txt`.

```
#include <cstdio>
using namespace std;
int main()
{
  FILE *fisin, *fisout;
  char x;
  if (!(fisin = fopen("intrare.txt", "r")))
```

```
{printf("EROARE la deschiderea fisierului de intrare");
  return 1;}
if (!(fisout = fopen("iesire.txt", "w")))
  {printf("EROARE la deschiderea fisierului de iesire");
   return 2;}
while (!feof(fisin))
  if (fscanf(fisin, "%c", &x) != EOF)
    fprintf(fisout, "%c", x);
fclose(fisin);
fclose(fisout);}
return 0; }
```

3.4. Probleme propuse

1. Ce va conține fișierul `test.out` pentru `n=31` și `a=3`? Care credeți că este efectul programului?

```
#include <fstream.h>
#include <iomanip.h>
int main()
{
  ofstream fout("test.out");
  int a, n, i, j;
  cout<<"n (28<=n<=31), a (1<=a<=7) = "; cin >> n >> a;
  for (i=1; i<= 7; i++)
    {j=i-a+1;
     if (j>0) fout<<setw(4)<<j; else fout << " ";
     for (j += 7; j <= n; j+=7) fout<<setw(4)<<j;
     fout << endl;}
  fout.close();
  return 0;
}
```

2. Fișierul `nr.in` conține numere întregi, scrise pe mai multe linii, numerele de pe aceeași linie fiind separate prin spații. Scrieți un program care să numere câte valori întregi sunt pe fiecare dintre liniile fișierului.
3. Scrieți un program care să testeze dacă două fișiere ale căror nume se citesc de la tastatură sunt identice.
4. În fișierul `NR.TXT` se află numere naturale din intervalul $[0, 5000]$, separate prin spații. Să se creeze fișierul `PARE.TXT` care să conțină doar valorile pare din fișierul `NR.TXT`, câte o valoare pe linie. (variantă bacalaureat 2000)
5. Scrieți un program care să numere câte linii conține fișierul `text.in`.