3. FIŞIERE

3.1. Noţiuni introductive

Un ***fişier*** este o colecţie de date de acelaşi tip, memorate pe suport extern (hard-disc, dischetă, CD, etc).

Avantajele utilizării fişierelor sunt o consecinţă a proprietăţilor memoriilor externe. Fişierele permit memorarea unui volum mare de date persistente (care nu se „pierd" la încheierea execuţiei programului sau la închiderea calculatorului).

Până acum am lucrat numai cu date citite de la tastatură, memorate în variabile simple. Pe parcursul execuţiei unui program, se alocă memorie în RAM pentru variabilele utilizate în programul respectiv. Memoria alocată se eliberează fie pe parcursul execuţiei programului, fie la terminarea execuţiei acestuia. În orice caz, după terminarea programului, valorile variabilelor „s-au pierdut", deoarece zona de memorie în care au fost memorate s-a eliberat şi dacă doream să executăm încă o dată programul cu aceleaşi date de intrare, trebuia să le introducem din nou de la tastatură. Chiar şi pentru programe cu un volum mic de date de intrare, aşa cum am elaborat noi până acum, acest aspect reprezintă un inconvenient. Dar imaginaţi-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 angajaţii firmei sau la o bibliotecă, la care, de exemplu, orice cerere de împrumut s-ar solda cu introducerea tuturor cărţilor existente în bibliotecă! Exemplele pot continua, dar ideea principală rămâne că în orice situaţie practică fişierele sunt indispensabile.

Fişierele pot fi clasificate după conţinutul lor astfel:
- fişiere **text** – conţin o secvenţă de caractere ASCII, structurate pe linii;
- fişiere **binare** – conţin o secvenţă de octeţi, fără o structură predefinită.

Noi vom studia modul de utilizare a fişierelor text.

3.2. Fişiere text în limbajul C++

Declararea fişierelor

Operaţiile de intrare/ieşire în limbajul C++ au utilizat până acum *stream*-urile `cin` şi `cout`, care erau automat asociate tastaturii, respectiv ecranului. Pentru ca un program să poată citi informaţii dintr-un fişier, respectiv să scrie informaţii într-un fişier, trebuie să asociem fişierul respectiv unui *stream* de intrare/ieşire.

În fişierul antet `fstream` sunt declarate clasele `ifstream`, `ofstream` şi `fstream`. Pentru a utiliza într-un program operaţii de intrare/ieşire folosind fişiere trebuie să declarăm variabile de tipurile `ifstream`, `ofstream` sau `fstream`. Mai exact:

`ifstream` – declarare *stream* de intrare (numai pentru operaţii de citire);

`ofstream` – declarare *stream* de ieşire (numai pentru operaţii de scriere);

`fstream` – declarare *stream* de intrare/ieşire (în care se pot realiza atât operaţii de citire, cât şi operaţii de scriere, în funcţie 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 fişierelor

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

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

La deschidere este obligatoriu să specificăm modul de deschidere dacă fişierul 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 funcţiei `open`, asociindu-l fişierului `date.in`. Deoarece nu am precizat discul logic pe care se află acest fişier sau calea către directorul în care se află fişierul, se deduce că fişierul se află pe hard-disc în directorul curent. Acelaşi efect l-am fi obţinut prin deschiderea fişierului 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 fişierului `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 fişierul a fost deschis ca fişier de intrare (numai pentru operaţii 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 fişierului `nr.out`. Modul de deschidere specificat este `ios::out`, ceea ce semnifică deschiderea fişierului ca fişier de ieşire (se creează un fişier vid cu numele specificat, în care se vor realiza doar operaţii de scriere).

Citirea datelor dintr-un fişier

După deschiderea fişierului ca fişier de intrare se pot realiza operaţii de citire. În acest scop se poate utiliza operatorul de citire `>>`, sau pot fi utilizate funcţii membru specifice.

Una dintre particularităţile citirii cu operatorul **>>** este faptul că **ignoră caracterele albe**. Prin urmare dacă intenţionăm să citim toate caracterele din fişier (inclusiv spaţiu, tab, enter) acest mod de citire este inadecvat.

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

```
ifstream f1("text.in");
char c;
f1.get(c);
```

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

Scrierea datelor într-un fişier

După deschiderea fişierului ca fişier de ieşire, se pot realiza operaţii de scriere utilizând operatorul de scriere `<<`.

Exemplu

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

Operaţii de test

Detectarea sfârşitului de fişier

În multe aplicaţii este necesar să citim toate datele existente într-un fişier, fără să avem informaţii prealabile despre numărul acestora.

La deschiderea unui fişier se creează un ***pointer de fişier***, care indică poziţia curentă în *stream*. Orice operaţie 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 ieşire.

Pentru a testa dacă pointerul de fişier a ajuns la sfârşitul fişierului putem utiliza funcția membru `eof()`. Funcția `eof()` returnează valoarea `0` dacă nu am ajuns la sfârşitul fişierului, respectiv o valoare diferită de `0` dacă am ajuns la sfârşit.

Exemplu

Vom citi şi vom număra toate caracterele existente într-un fişier, până la sfârşitul acestuia.

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

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

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

Testarea reuşitei unei operații de intrare/ieşire

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

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

Există şi 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/ieşire până în acel moment).

Închiderea unui fişier

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

Exemplu

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

Operația de închidere este obligatorie, în special pentru fişierele de ieşire. *În cazul în care nu închideți fişierul de ieşire, există mari şanse să pierdeți informații*!

3.3. Fişiere text în limbajul C

Declararea fişierelor

Operaţiile de intrare/ieşire în limbajul C au fost realizate până acum utilizând instrucţiunile `scanf()` şi `printf()`, care implicit preiau informaţii de la tastatură şi, respectiv, le transmit spre ecran. Pentru ca un program să poată prelucra informaţii dintr-un fişier, acesta trebuie declarat, indicându-se numele lui precum şi operaţiile permise asupra conţinutului fişierului respectiv.

În fişierul antet `<cstdio>` sunt declarate, printre altele, şi funcţiile, constantele, tipurile de date şi variabilele globale cu ajutorul cărora pot fi prelucrate informaţiile conţinute în fişiere. Pentru a utiliza într-un program operaţii de intrare/ieşire folosind fişiere trebuie să declarăm una sau mai multe variabile de tip `FILE *`. Declaraţia unui fişier se face astfel:

```
FILE * fisier;
```

Deschiderea fişierelor

Desigur, pentru a avea acces la informaţiile din fişier trebuie să fie cunoscut fişierul fizic de pe suport (specificatorul de fişier conforma sintaxei sistemului de operare utilizat), ca şi modul în care vor fi prelucrate informaţiile din fişier. Acest lucru se realizează utilizând funcţia `fopen`.

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

unde `nume` este o constantă şir de caractere care reprezintă specificatorul fişierului fizic pe suport, iar `mod` este o constantă şir de caractere care indică modul în care vor fi accesate informaţiile din fişier.

Exemple

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

Declararea şi deschiderea unui fişier se pot face şi simultan:

Exemplu 1

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

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

Exemplul 2

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

Am declarat un fişierul de ieşire, denumit `f2`, pe care l-am asociat fişierului fizic `date.out` în directorul curent.

Observaţie

Dacă, după operaţia de deschidere a fişierului, variabila corespunzătoare are valoarea `NULL`, deducem că operaţia de deschidere a eşuat (de exemplu, dacă dorim să deschidem un fişier de intrare care nu există). Este indicat să testăm succesul operaţiei de deschidere înainte de a efectua orice altă operaţie cu fişierul.

Exemplul 3

```
#include <cstdio>
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 fişierului se poate face şi astfel:

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

Citirea datelor dintr-un fişier

După deschiderea fişierului ca fişier de intrare se pot realiza operaţii de citire. În acest scop se utilizează funcţia `fscanf`.

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

Prototipul funcţiei `fscanf()` este asemănător cu prototipul funcţiei `scanf()`. Funcţia `fscanf()` are în plus, ca prim parametru, fişierul din care se face citirea. În rest, semnificaţia parametrilor, rezultatului şi efectul funcţiei sunt aceleaşi ca în cazul funcţiei `scanf()`.

Citirea datelor cu format specificat

Funcţia `scanf()` permite citirea datelor sub controlul unui format specificat. Această funcţie este declarată în fişierul antet `cstdio` şi are următorul format:

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

Efect

Funcţia parcurge succesiunea de caractere introdusă de la tastatură şi 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 ieşirea din funcție valoarea citită nu este atribuită variabilei respective.

Parametrul `format` este un şir de caractere care poate conține specificatori de format, caractere albe şi 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 citeşte şi eventual alte elemente opționale. Litera care indică tipul poate fi, de exemplu:

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

	epuizarea dimensiunii maxime `lg` din specificatorul de format) .

Opţional unele litere tip pot fi precedate de litera `l` sau de litera `L`. Literele `d`, `o` şi `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`).

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

Caracterul opţional `*` 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 funcţiei `scanf()`.

Observaţie

Caracterul `%` are o semnificaţie 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 construcţia sintactică `%%`.

Exemple

1. Să considerăm următoarele declaraţii de variabile:

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

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

```
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 declaraţie de variabile:

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

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

```
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 declaraţii de variabile:

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

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

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

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

Apelând funcţia:

```
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 declaraţie de variabile:

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

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

```
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 funcţia

```
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 şi anul putem apela funcţia:

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

Scrierea datelor într-un fişier

După deschiderea fişierului ca fişier de ieşire, se pot realiza operaţii de scriere utilizând funcţia `fprintf()`.

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

Prototipul funcţiei `fprintf()` este asemănător cu prototipul funcţiei `printf()`. Funcţia `fprintf()` are în plus, ca prim parametru, fişierul în care se face afişarea. În rest, semnificaţia parametrilor, rezultatului şi efectul funcţiei sunt aceleaşi ca în cazul funcţiei `printf()`.

Afişarea datelor cu format

Afişarea datelor pe ecran cu un format specificat se realizează apelând funcţia `printf()`. Această funcţie este declarată în fişierul antet `cstdio` şi are următoarea sintaxă:

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

Efect

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

În caz de succes, funcţia returnează numărul de caractere afişate.

Parametrul `format` este un şir de caractere care poate conţine specificatori de format şi 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 afişate pe ecran în poziţiile corespunzătoare.

Un specificator de format are următoarea sintaxă:

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

Observaţi că un specificator de format începe cu caracterul `%`, trebuie să conţină obligatoriu o literă care să indice tipul expresiei corespunzătoare şi eventual alte elemente opţionale. Litera care indică tipul poate fi, de exemplu:

literă_tip	**Semnificaţie**
d	Expresia se converteşte din tipul `int` şi se afişează în baza 10.
o	Expresia se converteşte din tipul `int` şi se afişează în baza 8.
u	Expresia se converteşte din tipul `unsigned` şi se afişează în baza 10
x, X	Expresia se converteşte din tipul `int` şi se afişează în baza 16 (literele care reprezintă 10,11,12,13,14,15 sunt majuscule pentru `X` sau minuscule pentru `x`).
f	Expresia se converteşte din tipul `float` şi se afişează în formatul `parte_întreagă.parte_zecimală`
e, E	Expresia se converteşte din tipul `float` şi se afişează în format exponenţial (ştiinţific) `cifră.parte_zecimală`**`e`**±`exponent` Pentru `E` litera care precedă exponentul este **`E`**.
g	Se aplică una dintre conversiile corespunzătoare literei `f` sau literei `e` (cea care se reprezintă cu număr minim de caractere)
c	Expresia are ca valoare codul ASCII al unui caracter şi se afişează caracterul corespunzător.
s	Expresia este un şir de caractere care va fi afişat pe ecran.

Opţional, 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 afişarea. Dacă numărul de caractere necesar pentru a afişa valoarea expresiei depăşeşte `lg`, se vor utiliza atâtea caractere câte sunt necesare. Dacă numărul de caractere necesare pentru a afişa valoarea expresiei este mai mic decât `lg`, se vor utiliza `lg` caractere (restul zonei pe care se realizează afişarea 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 (şi completarea cu spații se va realiza la dreapta).

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

Pentru şiruri de caractere, precizia indică numărul de caractere din şir care se afişează (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 afişează:**
`printf("a=%d sau a=%u\n", a, a);`	`a=-1 sau a=65535`

`printf("x=%f sau\n x=%e sau\n x=%g\n", x, x, x);`	`x=-123.147000 sau`
	`x=-1.231470e+02 sau`
	`x=-123.147`

`printf("y=%f sau\n y=%e sau\n y=%g\n", y, y, y);`	`y=0.000080 sau y=8.000000e-05` `sau y=8e-05`
`printf("b=%d sau b=%o sau b=%x \n", b, b, b);`	`b=375 sau b=567 sau b=177`
`printf("c=%c sau c=%d\n", c, c);`	`c=x sau c=120`
`printf("d=%d sau d=%x sau\n d=%u\n", d, d, d);`	`d=-4070 sau d=f01a sau` `d=61466`
`printf("x=%.2f\n", x);`	`x=-123.15`
`printf("b=%6dxb=%-6dx\n", b, b);`	`b=   375xb=375   x`

Închiderea fişierelor

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

```
fclose(fisier);
```

Exemplul 1

Să scriem un program care citeşte din fişierul de intrare `nr.in` un număr natural `n` (`0<n≤100`) şi apoi `n` valori naturale şi afişează în fişierul de ieşire `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 şi 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 fişierului `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 fişierul `test.out` pentru `n=31` şi `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. Fişierul `nr.in` conține numere întregi, scrise pe mai multe linii, numerele de pe aceeaşi linie fiind separate prin spații. Scrieți un program care să numere câte valori întregi sunt pe fiecare dintre liniile fişierului.

3. Scrieți un program care să testeze dacă două fişiere ale căror nume se citesc de la tastatură sunt identice.

4. În fişierul `NR.TXT` se află numere naturale din intervalul `[0, 5000]`, separate prin spații. Să se creeze fişierul `PARE.TXT` care să conțină doar valorile pare din fişierul `NR.TXT`, câte o valoare pe linie. (variantă bacalaureat 2000)

5. Scrieți un program care să numere câte linii conține fişierul text `text.in`.