13

INTRĂRI/IEŞIRI

13.1. Principiile de bază ale sistemului de I/O din limbajul C++
13.2. Testarea şi modificarea stării unui flux
13.3. Formatarea unui flux
13.3.1. Formatarea prin manipulatori
13.3.2. Formatarea prin metode
13.4. Metodele clasei istream
13.5. Metodele clasei ostream
13.6. Manipulatori creaţi de utilizator
13.7. Fluxuri pentru fişiere
13.8. Fişiere binare

13.1. PRINCIPIILE DE BAZĂ ALE SISTEMULUI DE INTRĂRI/IEŞRI DIN LIMBAJUL C++

În limbajul C++ există două sisteme de intrări/ieşiri:
- Unul tradiţional, moştenit din limbajul C, în care operaţiile de intrare/ieşire se realizează cu ajutorul unor funcţii din biblioteca standard a sistemului (vezi capitolul 8);
- Unul propriu limbajului C++, orientat pe obiecte.

Sistemul de I/O orientat pe obiecte al limbajului C++ tratează în aceeaşi manieră operaţiile de I/O care folosesc consola şi operaţiile care utilizează fişiere (perspective diferite asupra aceluiaşi mecanism).

La baza sistemului de I/E din C++ se află:
- *două ierarhii de clase* care permit realizarea operaţiilor de I/O;
- *conceptul de* ***stream*** (stream-ul este un concept abstract care înglobează orice ***flux*** de date de la o sursă (canal de intrare) la o destinaţie (canal de ieşiere), la un consumator. Sursa poate fi tastatura (intrarea standard), un fişier de pe disc sau o zonă de memorie. Destinaţia poate fi ecranul (ieşirea standard), un fişier de pe disc sau o zonă de memorie (figura 13.1).

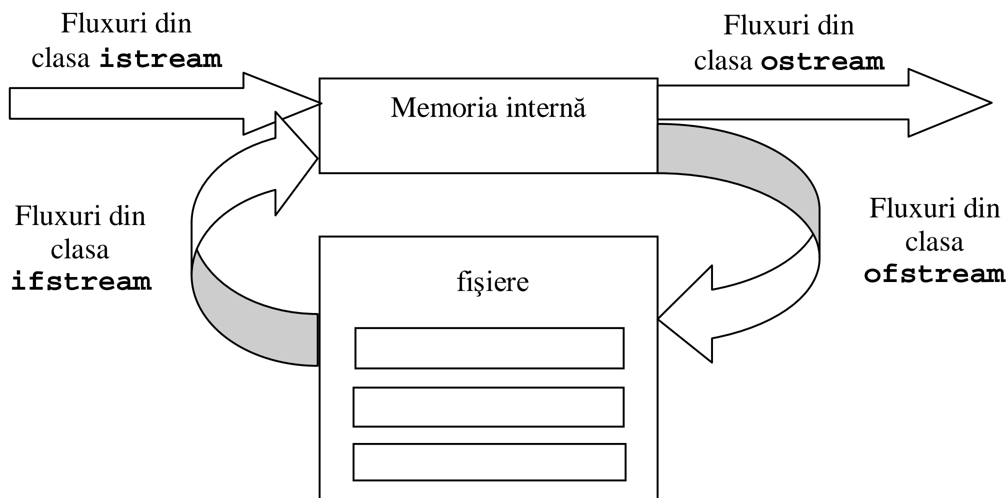

Figura 13.1. Stream-ul - baza sistemului de I/O

Avantajele utilizării stream-urilor sunt:
- Flexibilitate mare în realizarea operaţiilor de I/O (mai mare decât în cazul folosirii funcţiilor din C);
- Posibilitatea de eliminare a erorilor care apar în mod frecvent la apelul funcţiilor de I/O obişnuite, când numărul specificatorilor de format diferă de cel al parametrilor efectivi;
- Posibilitatea de a realiza operaţii de I/O nu numai cu date de tipuri predefinite, ci şi cu obiecte de tip abstract.

Biblioteca de clase de I/O a limbajului C++ utilizează moştenirea, polimorfismul şi clasele abstracte. Ierarhia are două clase de bază (figura 13.2.):

- Clasa virtuală **`ios`** care oferă informaţii despre fluxurile de date (variabile de stare, metode) şi facilităţi tratarea erorilor;
- Clasa **`streambuf`** (clasă prietenă cu `ios`), destinată operaţiilor de I/O cu format.

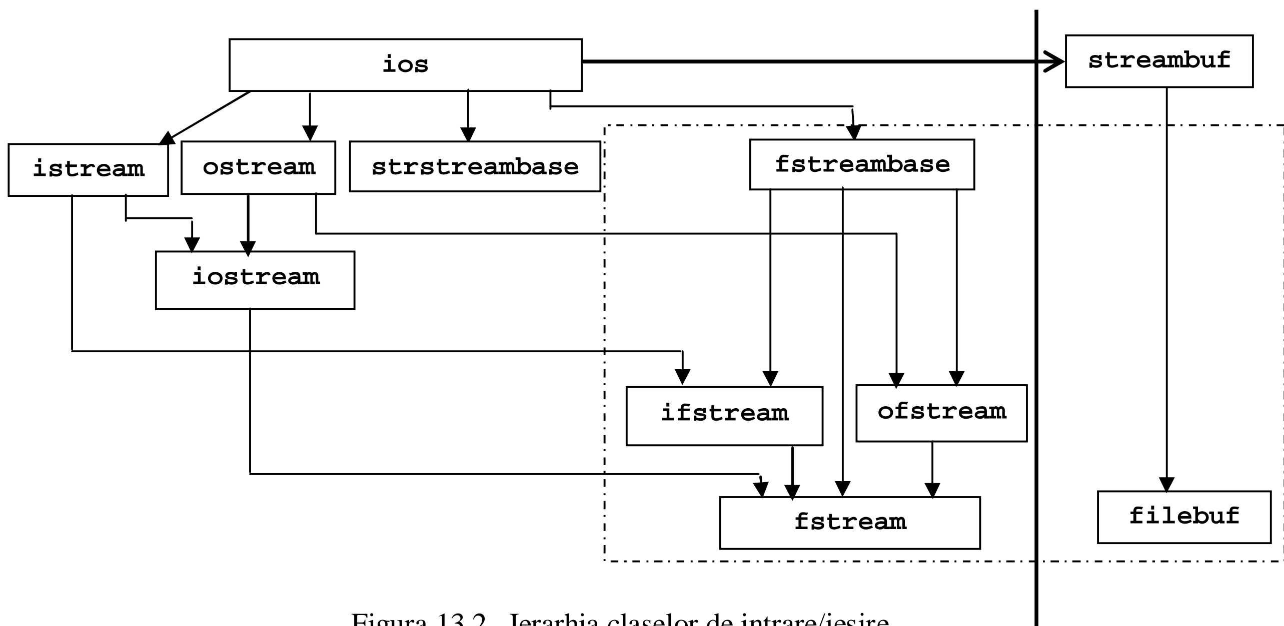

Figura 13.2. Ierarhia claselor de intrare/ieşire

Utilizarea ierarhiilor de clase de mai sus implică includerea headerului *iostream.h*.

- Clasa **`ios`** are un pointer către streambuf. Are date membru pentru a gestiona interfaţa cu streambuf şi pentru tratarea erorilor. Clasele derivate din clasa de bază **`ios`**:
  - Clasa **`istream`**, care gestionează intrările: **`class istream:virtual public ios`**
  - Clasa **`ostream`** gestionează ieşirile: **`class ostream: virtual public ios`**
    - Clasa **`iostream`**, derivată din istream şi ostream, gestionează intrările şi ieşirile. Fiecărui flux de date i se asociază în memorie o zonă tampon numită ***buffer***. Clasa furnizează funcţii generale pentru lucrul cu zonele tampon şi permite tratarea operaţiilor de I/O fără a avea în vedere formatări complexe.
      `class iostream:public istream, public ostream`

Clasele istream, ostream şi iostream sunt, fiecare, clase de bază pentru clasele derivate:

```
class istream_withassign:public istream
class ostream_withassign:public ostream
class iostream_withassign:public iostream
```

Clasele cu sufixul `_withassign` furnizează următoarele *fluxurile predefinite* (instanţe), deja cunoscute:

1. **`cout`** (console output) obiect al clasei `ostream_withassign`, similar fişierului standard de ieşire definit de pointerul `stdout` (în C). Se foloseşte cu operatorul insertor, supraîncărcat pentru tipurile predefinite:
   Exemplu: `int n; cout<<n;`
   Converteşte din format binar în zecimal valoarea lui n, şi trimite (inserează) în fluxul cout caracterele corespunzătoare fiecărei cifre obţinute.
2. **`cin`** (console input) obiect al clasei `istream_withassign`, similar fişierului standard de intrare definit de pointerul `stdin` (în C). A fost folosit cu operatorul extractor, supraîncărcat pentru tipurile predefinite:
   Exemplu: `int n; cin>>n;`
   Extrage din fluxul de intrare caracterele corespunzătoare, le converteşte din zecimal în binar şi le depune în memorie.
3. **`cerr:`** flux de ieşire conectat la ieşirea standard pentru erori (*stderr* în C) (fără buffer intermediar).
4. **`clog:`** flux de ieşire conectat la ieşirea standard pentru erori (fără buffer intermediar).

- Clasa **`streambuf`** este clasă prietenă cu `ios`. Ea furnizează funcţii generale pentru lucrul cu zonele tampon (buffere) şi permite tratarea operaţiilor de I/O fără formatări complexe. Din clasa streambuf deriva clasa **`filebuf`** .

Să urmărim care este rolul buffere-lor (zonă tampon asociată fiecărui flux de date), prin următorul exemplu:

Exemplu:
```
void main()
{int a; double x; char sir[20];
cin>>a>>x>>şir;
cout<<a<<' '<<x<<endl;
cout<<sir<<endl;
}
```

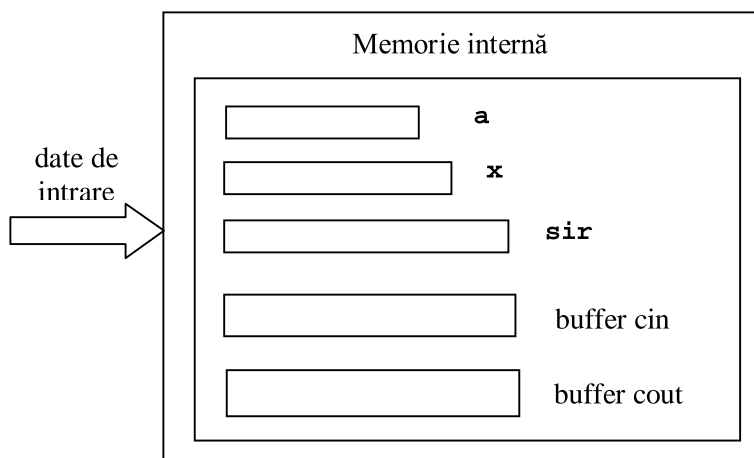

Figura 13.3. Rolul buffere-lor în operaţiile de I/O

Zona tampon (buffer-ul) este interfaţa dintre program şi sistemul de operare.

Să urmărim cum se realizează transferul informaţiei în cazul operaţiilor multiple:
```
cin>>a>>x>>şir;
cin>>a;
```
Compilatorul verifică dacă numărul introdus este întreg (se opreşte când întâlneşte altceva decât o cifră: în acest caz - un blanc). Îl citeşte printr-o metodă a tipului int, îl converteşte în binar şi îl transmite în memoria internă în spaţiul rezervat pentru variabila a. În încheiere, cin>>a devine cin.
`cin>>a>>x;` // fluxul de date se transmite şi lui x
Presupunem că am introdus de la tastatură `325 -17.45e5 exemplu de şir`, valorile pentru `a`, `x` şi `sir`. La apăsarea tastei Enter, se transmite către sistemul de operare un octet cu semnificaţia de sfârşit introducere date. Sistemul de operare transmite un semnal zonei tampon şi abia acum se transmite valoarea (fluxul de date) lui a. La citirea şirului de caractere (până la primul blank), se transferă octet cu octet (i se adaugă automat caracterul `NULL`), din zona tampon în memoria internă.

Observaţii:
1. Stream-ul este secvenţial.
2. El poate realiza intrări/ieşiri formatate.
3. Este bine că în cazul unui flux de intrare să se verifice mai întâi dacă în zona tampon se găseşte ceva. Dacă nu, se poziţionează cursorul la începutul zonei tampon.
4. Informaţia de la buffer către memorie este gestionată prin program; informaţia de la zona tampon către alte periferice este gestionată de către sistemul de operare.

13.2. TESTAREA ŞI MODIFICAREA STĂRII UNUI FLUX

Clasa **`ios`**, clasă de bază a ierahiei claselor de I/O, ***defineşte o mulţime de tipuri, variabile şi metode comune tuturor tipurilor de stream-uri***.

Starea unui stream (rezultatul ultimului acces la acesta) este păstrată în **cuvântul de stare**, care este **dată membră** a clasei `ios`. Fiecărei instanţieri a unei clase de intrare/ieşire i se asociază propriul cuvânt de stare (o mulţime de indicatori de stare), care păstrează toate informaţiile aferente erorilor apărute în cursul operaţiilor cu stream-ul. Aceşti indicatori sunt memoraţi la nivel de bit în data membru `state.`

```
class ios{
//......
protected:
```

```
        int state;              //păstrează la nivel de bit valorile indicatorilor de stare
public:
        enum io_state{
                goodbit=0x00,   //ultima operaţie de intrare/ieşire corectă
                eofbit=0x01,    //s-a întâlnit sfârşitul de fişier într-o operaţie de intrare (lipsa
                                //caracterelor disponibile pentru citire)
                failbit=0x02,   //setat dacă ultima operaţie de intrare/ieşire a eşuat; stream-ul
                                //respectiv nu va mai putea fi folosit în operaţii de I/O până când
                                //bitul nu va fi şters
                badbit=0x04,    //ultima operaţie de intrare/ieşire invalidă; în urma resetării flag-ului
                                //este posibil ca stream-ul să mai poată fi utilizat
                hardbit=0x08    //eroare irecuperabilă
        };
};
```

Testarea valorii cuvântului de stare asociat unui stream se realizează cu ajutorul **metodelor**:

- **int good();**//Returnează valoare diferită de zero dacă cuvântul de stare este 0
- **int eof();**//Returnează valoare diferită de zero dacă `eofbit` este setat
- **int fail();**//Returnează valoare diferită de zero dacă `failbit`, `hardbit` sau `badbit` sunt setaţi
- **int bad();**//Funcţionează ca metoda fail, dar nu ia în considerare flagul `failbit`.

Modificarea valorii cuvântului de stare asociat unui stream se realizează cu ajutorul **metodelor**:

- **void clear(int i=0);**
  Setează data membru `state` la valoarea parametrului (implicit 0). Este capabilă să şteargă orice flag, cu excepţia flag-ului `hardfail`.
  Exemplu:: `a.clear(ios::badbit);`
  Setează bitul `failbit` şi anulează valorile celorlaţi biţi din cuvântul de stare a fluxului `a`. Pentru ca ceilalţi biţi să rămână nemodificaţi, se apelează metoda `rdstate`.
  Exemplu: `a.clear(a.rdstate()|val_f);`
  Se setează un singur flag, lăsându-le nemodificate pe celelalte; `val_f` are una din valorile definite de enumerarea `io_state`.
- **int rdstate();**
  Returnează valoarea cuvântului de stare, sub forma unui întreg (valoarea datei membru `state`).
  Exemplu: `a.clear(ios::badbit | a.rdstate());`
  Setează bitul `failbit`, fără a modifica valorile celorlalţi biţi:

Testarea cuvântului de stare se poate realiza şi prin folosirea **operatorilor !** şi **void*** :

- Operatorul **!** este supraîncărcat printr-o funcţie membră, cu prorotipul:

  int operator ! ();

  care returnează 0 dacă un bit de eroare din cuvântul de stare este setat (ca şi metoda `fail`).
  Exemplu:
  ```
  if (!cin)          //echivalent cu if (!cin.good())
          cout<<"Bit de eroare setat în cuvântul de stare!\n";
  else  cin>>a;
  ```
- Operatorul **void*** converteşte stream-ul într-un pointer generic. Conversia are ca rezultat zero dacă cel puţin un bit de eroare este setat:

  operator void *();

  Exemplu: O construcţie de forma: `cin>>s;` are ca valoare o referinţă la stream-ul `cin`, din clasa istream. Această referinţă poate fi utilizată sub forma: `if (cin>>s). . .`

Pentru scoaterea unui stream din starea de eroare, fie dispare cauza erorii, fie trebuie şterse flagurile care semnalizează eroarea.

13.3. FORMATAREA DATELOR DIN FLUXURILE DE INTRARE/IEŞIRE

Unul dintre principalele avantaje oferite de sistemul de I/O din limbajul C++ îl reprezintă ignorarea aspectului formatării, folosindu-se o ***formatare implicită***. În plus, se permite definirea unei formatări specifice pentru o anumită aplicaţie. Aşa cum s-a subliniat în cazul cuvântului de eroare, cuvântul de format poate fi privit ca un întreg, pentru care fiecare bit reprezintă o constantă predefinită din clasa ios. În cadrul acestui cuvânt sunt definite câmpuri de biţi (**cuvântul de format** este **dată membră** care conţine un număr de indici ce sunt biţi individuali).

```
class ios {
//……
protected:
      long flag_x;       //păstrează la nivel de bit indicatorii de format
      int x_width;       //numărul de caractere utilizate pentru afişarea unei valori pe ecran
public:
enum
{skipws = 0x0001,             // salt peste spaţiile albe de la intrare
left = 0x0002,                // aliniere la stânga la ieşire
right = 0x0004,               // aliniere la dreapta la iesire
internal = 0x0008,            // aliniere după semn sau specificator al bazei la ieşire
dec = 0x0010,                 // conversie în baza 10
oct = 0x0020,                 // conversie octală la intrare/ieşire
hex = 0x0040,                 // conversie hexa la intrare/ieşire
showbase = 0x0080,            // afişarea bazei la ieşire
showpoint = 0x0100,           // afişarea punctului zecimal pentru numere reale la ieşire
uppercase = 0x0200,           // afişarea cu majuscule a cifrelor hexa şi a literei E la ieşire.
showpos = 0x0400,             // afişarea semnului + pentru numerele pozitive la ieşire
scientific = 0x0800,          //folosirea formatului exponenţial (ştiinţific) pentru numerele reale
fixed = 0x1000,               // folosirea formatului în virgulă fixă pentru numere reale la
unitbuf = 0x2000,             // goleşte zona tampon după fiecare ieşire
stdio=0x4000                  // goleşte "stdout" şi "stdin" după fiecare inserare
};
};
```

În figura 13.4. sunt prezentate numele câmpurilor de biţi (acolo unde este cazul). În cadrul fiecărui câmp de biţi (**`adjustfield`**, **`basefield`**, **`floatfield`**) un singur bit poate fi activ.

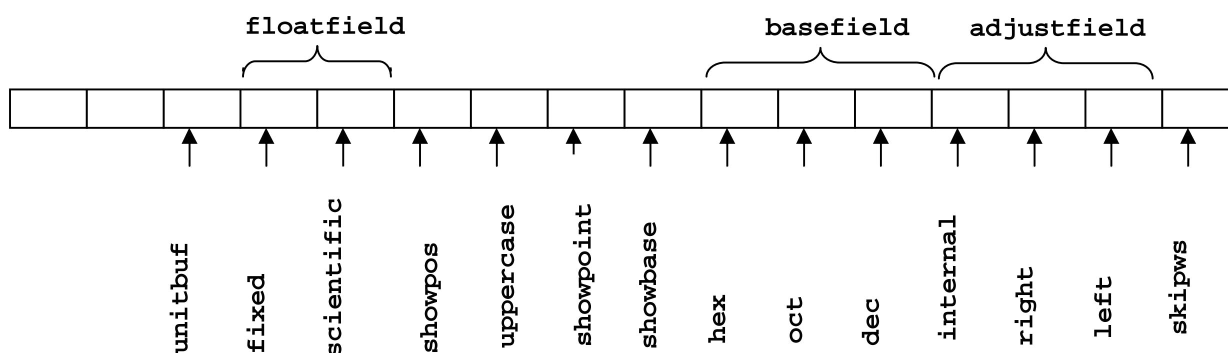

Figura 13.4. Câmpurile de biţi din cuvântul de stare

Modificarea cuvântului de format se poate realiza în următoarele moduri:
1. Cu ajutorul ***manipulatorilor*** (cu sau fără parametri);
2. Cu ajutorul unor ***funcţii membre*** ale claselor *istream* sau *ostream.*

13.3.1. FORMATAREA PRIN MANIPULATORI

Manipulatorii sunt *funcţii speciale*, asemănătoare operatorilor, care pot fi folosite împreună cu operatorii de inserţie într-un flux de ieşire sau de extracţie dintr-un flux de intrare, în scopul modificării caracteristicilor formatului informaţiilor de intrare/ieşire. Manipulatorii furnizează, ca rezultat, fluxul obţinut în urma acţiunii manipulatorilor, ceea ce permite ca aceştia să fie trataţi ca informaţii de transmis. Manipulatorii permit, deasemenea, înlănţuirea operatorilor insertori sau extractori care utilizează formate diferite.

Manipulatorii pot fi:
- Manipulatori fără parametri;
- Manipulatori cu parametri;

13.3.1.1. Manipulatori fără parametri

Prototipul manipulatorilor fără parametri este:

```
ostream & nume_manipulator(ostream &);
istream & nume_manipulator(istream &);
```

Manipulatorii fără parametri (prezentaţi în tabelul 13.1.) se folosesc astfel:

```
flux_ieşire<<manipulator;
flux_intrare>>manipulator;
```

Tabelul 13.1

Manipulator	Intrare/ Ieşire	Acţiune
dec	I/O	Formatează datele numerice în zecimal (activează bitul de conversie zecimală)
hex	I/O	Formatează datele numerice în hexa (activează bitul de conversie hexazecimală)
oct	I/O	Formatează datele numerice în octal (activează bitul de conversie octală)
ws	I	Ignoră caracterele "spaţii albe" (activează bitul de salt peste spaţiile albe)
endl	O	Afişează (inserează) un caracter '\n' şi eliberează fluxul
ends	O	Inserează un caracter null, de sfârşit de flux (\0)
flush	O	Videază (goleşte) buffer-ul, eliberează fluxul

13.3.1.2. Manipulatori cu parametri

Prototipul manipulatorilor fără parametri (prezentaţi în tabelul 13.2.) este:

```
istream & manipulator (argument);
ostream & manipulator (argument);
```

Tabelul 13.2.

Manipulator	Intrare/ Ieşire	Acţiune
setbase(int baza)	I/O	Stabileşte baza de conversie
resetiosflags(long f)	I/O	Atribuie valoarea 0 tuturor biţilor indicaţi de argument, lăsând restul biţilor nemodificaţi (dezactivează indicatorii specificaţi de f)
setiosflags (long f)	I/O	Atribuie valoarea 1 tuturor biţilor indicaţi de argument, lăsând restul biţilor nemodificaţi (activează indicatorii specificaţi de f)
setfill (int c)	I/O	Defineşte caracterul de umplere (cel implicit este spaţiul liber, blank-ul)
setprecision (int p)	I/O	Defineşte precizia pentru numerele reale
setw (int w)	I/O	Defineşte lăţimea câmpului (numărul de octeţi care vor fi citiţi sau afisaţi)

Utilizarea manipulatorilor impune includerea header-ului **`iomanip.h`**.

Exerciţiu: Exemplificarea modului de folosire a manipulatorilor pentru intrări/ieşiri formatate.

```
#include <iostream.h>
#include <iomanip.h>
void main()
{int a,b,c;double alfa,x,y;long ll;char xx,yy;
float u,v; unsigned int w; unsigned char ab,db;
a=125,b=10;  ll=100*a*b; cout<<"100*a*b="<<ll<<endl; //100*a*b=-6072
alfa=75.50;y=1.345678;
xx=35;yy=6; x=y/alfa-xx/yy; cout<<"x="<<x<<endl; //x=-4.982176
      //setarea mărimea câmpului de afişare
cout<<"x="<<setw(8)<<x<<endl;                    //x=-4.982176
cout<<"x="<<setw(20)<<x<<endl;                   //x=        -4.982176
      //setarea lungimii câmpului de afişare şi a caracterului de umplere
cout<<"x="<<setw(20)<<setfill('*')<<x<<endl;     //x=***********-4.982176
       // precizie
cout<<"x="<<setw(8)<<setprecision(2)<<x<<endl;   //x=***-4.98
cout<<"x="<<setw(8)<<setprecision(0)<<x<<endl;   //x=-4.982176
       // afisarea octetului semnificativ var & 0377 si a celui mai putin semnificativ (var>>8) & 0377
w=34; ab='34' & 0377; db=('34'>>8)&0377;
cout<<"w="<<w<<"  ab="<<ab<<"  db="<<db<<endl;   //w=34  ab=3  db=4
u=2579.75;v=12345.567E+10;
      //formatare in octal si hexa
cout<<"a in baza 10="<<a<<" in octal="<<oct<<a<<" in hexa="<<hex<<a<<endl;
      //a in baza 10=125 in octal=175 in hexa=7d
cout<<"b="<<dec<<b<<endl;                        //b=10
}
```

13.3.2. FORMATAREA PRIN METODE

Formatarea fluxurilor de intrare/ieşire se poate realiza şi prin **metode** ale clasei **`ios`**.

- Funcţia membră **`setf`** permite modificarea cuvântului de stare a formatării. Ea este supradefinită astfel:

  `long setf (long) ;`

  Primeşte ca parametru un număr întreg long şi setează pe 1 biţii specificaţi prin argument. Returnează valoarea anterioară a cuvântului de stare a formatării. Ea poate fi folosită numai pentru flag-urile care nu fac parte dintr-un câmp de biţi.

  Exemplu: `cout. setf (ios:: showpos);` //setează bitul showpos pe 1

  `long setf (long flags, long field) ;`

  Setează pe 1 biţii specificaţi prin primul argument, doar în câmpul de biţi definit ca al doilea argument. Ea modifică pe 0 toţi biţii din câmpul respectiv, după care setează (pe 1) bitul respectiv.

  Exemple:

  ```
  cout. setf (ios:: showpos | ios :: uppercase | ios:: internal)
        //toţi biţii vor fi setaţi pe valoarea 1
  cout.setf (ios :: hex, ios :: basefield) //setează indicele hex din câmpul basefield
  cout. setf (ios :: skipws | ios :: hex | ios:: showbase, ios :: basefield)
  ```

- Funcţia membră **`fill`** permite testarea sau modificarea caracterului de umplere şi returnează codul caracterului curent de umplere:

  `int fill ( ) ;`

  Exemplu: `cout.fill ( );` // întoarce codul caracterului de umplere pt. cout

  `int fill (char) ;`

  Setează noul caracter de umplere şi returnează codul vechiului caracter de umplere.

  Exemplu: `cout.fill('# ');` // setează un alt caracter de umplere ('#').

- Funcţia membră **precision** permite testarea sau modificarea preciziei. Numim precizie numărul de cifre semnificative (în GNU) sau numărul de cifre după virgulă (în TurboC)

  `int precision ( );`

  Returnează valoarea actuală a preciziei numerice.

  `int precision (int);`

  Setează precizia la valoarea parametrului şi returnează vechea valoare a preciziei.
- Funcţia membră **width** returnează valoarea actuală a lungimii câmpului de date sau setează valoarea lungimii câmpului de date.

  `int width ( ) ;`

  `.int width (int) ;`

  Exemplu: `cout. width (10);` //data se va afişa într-un câmp de cel puţin 10 caractere.

  Astfel, dacă argumentul este 0, la inserţia într-un flux de ieşire se transmit în stream atâţia octeţi câţi are data respectivă. Dacă argumentul este diferit de 0, dar lungimea necesară afişării este mai mare, se transmite numărul de octeţi necesari; iar dacă lungimea necesară afişării este mai mică, se transmite numărul de octeţi necesari, iar restul se completează cu caracterul de umplere.

Exerciţiu:

```
#include <iostream.h>
void main()
{int i=123, j=456; double x=1234.567890123456, y=5678;
cout.width(5); cout<<i<<' '<<j<<endl;       //123 456
cout<<i;      //toate metodele sunt persistente, cu excepţia metodei width
cout.width(5); cout<<j<<endl;               //123  456
cout.setf(ios::showpos);       //afis. +
cout<<i<<' '<<j<<endl;         //+123 +456
cout.setf(ios::hex, ios::basefield);
cout.width(20); cout.precision(15); cout.fill('*');
cout.setf(ios::showpos | ios::showpoint);
cout<<x<<' '<<y<<endl;         //***+1234.56789012346 +5678.00000000000
cout<<i<<' '<<j<<endl;         //7b 1c8
cout.setf(ios::dec, ios::basefield);
cout.unsetf(ios::showpos); cout.fill(' ');
cout<<i<<' '<<j<<endl;         //123 456
cout<<x<<' '<<y<<endl;         //1234.56789012346 5678.00000000000
}
```

Observaţie: Toate metodele sunt persistente, cu excepţia metodei *width* care este valabilă numai pentru operaţia următoare, după care se setează la 0.

13.4. METODELE CLASEI istream

Clasa **`istream`** este derivată din clasa `ios`: ***ios:istream.***

- Supradefinirea **operatorului de extracţie** >>

  Operatorul de extracţie din stream-ul (fluxul) de intrare este supraîncărcat printr-o *funcţie membră*, pentru toate tipurile de bază. El are rolul de a extrage din fluxul de intrare caracterele necesare pentru a obţine o valoare dintr-un tip de bază.

  `istream & operator >> (&tip_de_bază);`

  Primul argument este implicit (`this`): clasa care îl apelează. Al doilea argument este o referinţă către un `tip_ de_bază`. Operatorul poate fi supraîncărcat printr-o *funcţie prietenă* (vezi capitolul 11), pentru a extrage din fluxul de intrare informaţiile despre un obiect dintr-un tip definit de utilizator (clasă):

  `friend istream & operator >>(istream &,  clasa &);`
- Metoda **`get`**

  `istream & get (char &);`

  Metoda extrage din fluxul de intrare un caracter şi îl memorează în variabila referinţă, transmisă ca parametru. Întoarce o referinţă către istream.

Exemplu: `char c;` `cin.get(c);`

Metoda `get` este supraîncărcată şi astfel:

`int get ( );`

Extrage din fluxul de intrare un singur caracter şi îl returnează sub forma unui întreg. Această metodă este utilizată, în special, la testarea sfârşitului de fişier (`EOF`, -1).

Metoda get este supraîncărcată şi astfel:

`istream & get (char * şir, int lungime, char delimitator = '\n')`

Se extrage din fluxul de date de intrare un şir de caractere (`char * şir` = pointer către şir), cu lungimea maximă specificată prin argumentul `lungime`, până la întâlnirea delimitatorului (`delimitator` nu se extrage din fluxul de date de intrare). Rezultatul se depune în variabila ş*ir*.

- Metoda **`getline`**

  Metoda determină preluarea din fluxul de intrare a unui şir de caractere, terminat cu un caracter cunoscut.

  `istream & getline (char * şir, int lungime, char delimitator = EOF);`

  Acţionează asemănător cu metoda `get` supraîncărcată prin ultima formă, cu deosebirea că din fluxul de intrare se extrage şi delimitatorul. Delimitatorul nu se introduce în ş*ir*.

  Exemplu: `char şir[50];cin.get (şir, 50, '\ n');` *// sau* `cin.get(şir, 50);`

  Din fluxul de date de intrare se extrag caracterele până la sfârşit de linie, cel mult 50 de caractere. Extragerea caracterelor din fluxul de intrare se termină fie la întâlnirea terminatorului, fie atunci când s-a citit un număr de caractere egal cu *lungime-1* .

- Metoda `gcount` returnează un întreg care reprezinată numărul efectiv de caractere preluat prin `getline`.

  `int gcount();`

- Metoda **`read`** extrage din fluxul de intrare un şir de caractere (octeţi) de `lungime` impusă şi-l depozitează în zona de memorie `şir`.

  `istream & read (char * şir, int lungime);`

  Eemplu: `char t[30]; cin.read(t, 20);`

- Metoda **`ignore`** este utilizată la golirea zonei tampon a stream-ului de intrare.

  `istream & ignore (int lungime, char delimitator = ' | ');`

  Se extrag din fluxul de intrare caracterele până la *delimitator,* dar nu mai multe decât numărul indicat de parametru *lungime*. Caracterele extrase sunt *eliminate*, nu sunt memorate.

- Metoda **`peck`** furnizează primul caracter din fluxul de intrare, fără a-l extrage însă din flux. Caracterul va fi primul caracter extras la următoarea citire.

  `int peck( );`

  Exemplu: `c = cin.peck ( );`

- Metoda **`putback`** inserează în fluxul de intrare caracterul trimis ca argument.

  `istream & putback(char &);`

Observaţii:

1) `int a;` `cin.get (a)` ⇔ `cin >> a;`

2) Metodele pot fi folosite în serie: ***Exemplu:*** `cin.get(a).get(b).get (c);`

13.5. METODELE CLASEI ostream

- **Supradefinirea operatorului de inserţie <<**

  Operatorul de inserţie în fluxul de ieşire este supraîncărcat printr-o *funcţie membră* pentru toate tipurile de bază. El are rolul de a introduce (insera) în fluxul de ieşire caracterele necesare pentru a afişa o valoare dintr-un tip de bază.

  `ostream & operator << (tip_de_bază);`

  Primul argument este implicit (`this`). Al doilea argument este o expresie de tip de bază.

  Operatorul poate fi supraîncărcat printr-o *funcţie prietenă*, pentru a insera în fluxul de ieşire informaţiile despre un obiect dintr-un tip definit de utilizator (clasă):

  `friend ostream & operator << (ostream &, clasa &);`

- Metoda **`put`** inserează în fluxul de date de ieşire caracterul transmis ca parametru.

  `ostream put (char);`

Exemple:

`char c='S'; cout . put ( c );` // echivalent cu `cout << c;`

`char c1='A',c2='B',c3='C';cout.put(c1).put(c2).put(c3);`

//echivalent cu:`cout.put(c1);cout.put(c2); cout.put(c3);`

// echivalent cu `cout<<c1<<c2<<c3;`

- Metoda **`write`** inserează în fluxul de date de ieşire un număr de caractere, de `lungime` impusă, existente în zona `tablou`.

ostream & write (char * tablou, int lungime);

Exemplu: `char t[]="Bună ziua!\n";  cout.write(t, 4);`
//Inserează în fluxul de ieşire 4 caractere, începând de la adresa t.

Exerciţiu: Se ilustrează formatarea unui flux de intrare/ieşire atât prin funcţii membre, cât şi cu ajutorul manipulatorilor (cu sau fără parametri).

```
#include <iostream.h>
#include <iomanip.h>
void main()
{int i=123; char car, mesaj[]=" Apasă tasta Enter\n";
cout<<setw(5)<<resetiosflags(ios::internal |ios::right);
cout<<setiosflags(ios::left)<<setfill('#')<<i<<endl;
cout<<setw(5)<<setiosflags(ios::showpos)<<setfill(' ')<<i<<endl;
cout<<setw(5)<<resetiosflags(ios::left)<<setfill('0')<<setiosflags(ios::right);
cout<<i<<endl;
cout<<"\n hexagesimal:"<<setw(5)<<hex<<i<<endl;
cout<<"\n octal:"<<setw(5)<<oct<<i<<endl;
cout<<"\n zecimal:"<<setw(5)<<dec<<i<<endl;
cout<<mesaj; cin.get(car);
double d=123.456789012345678901234567890123456789;
cout<<"Afisare d cu precizia implicită:"<<d<<endl;
cout<<"Afisare d cu lung 25 si precizie
15:"<<setw(25)<<setprecision(15)<<d<<endl;
cout<<"Schimbare caracter umplere &:"<<setw(25)<<setfill('&')<<d<<endl;
cout<<endl;
}
```

13.6. MANIPULATORI CREAŢI DE UTILIZATOR

Manipulatorii sunt funcţii speciale care pot fi utilizate alături de operatorii de inserţie/extracţie pentru a formata datele de ieşire/intrare. Pe lângă manipulatorii predefinţi, utilizatorul îşi poate crea manipulatori proprii.

Crearea manipulatorilor fără parametri

`ostream &nume_manipulator (ostream &);` //pentru un flux de ieşire

`istream &nume_manipulator (istream &);` //pentru un flux de intrare

Crearea manipulatorilor cu parametri

Crearea manipulatorilor fără parametri necesită folosirea a două funcţii:

`ostream & nume_manipulator (ostream &, int);`

`omanip <int> nume_manipulator (int n)` // ***funcţie şablon (template)*** cu parametru de tip int

`{return omanip <int> (nume_manipulator, n);}`

Deci, manipulatorii sunt funcţii ale căror corpuri sunt stabilite de utilizator.

Exemplu: În exemplul următor se definesc manipulatorii `indentare` (indentare = înaintea liniei propriu-zise, se lasă un număr de spaţii albe), `convhex` (realizează conversia din zecimal în hexa), `init` (iniţializează lungimea câmpului la 10 caractere, stbileşte precizia la 4 şi caracterul de umplere $).

```
#include <iostream.h>
#include <iomanip.h>
ostream &convhex (ostream &ies)
      {ies.setf(ios::hex, ios::basefield); ies.setf(ios::showbase); return ies;}
ostream &indentare (ostream &ies, int nr_spaţii)
      {for (int i=0; i<nr_spatii; i++)    ies<<' '; return ies;}
ostream &init (ostream &ies)
      {ies.width(10); stream.precision(4); ies.fill('$'); return ies;}
omanip <int> indentare (int nr_spaţii)
      {return omanip <int> (indentare, nr_spaţii);}
void main()
      {cout<<712<<' '<<convhex<<712<<endl;
      cout<<indentare(10)<<"Linia1 text\n"<<indentare(5)<<"Linia2 text\n";
      cout init<<123.123456;}
```

13.7. FLUXURI DE DATE PENTRU FIŞIERE

Un flux de intrare/ieşire poate fi asociat unui fişier.

Pentru a asocia un **flux de ieşire** unui fisier, este necesară **crearea unui obiect** de tipul **ofstream**. Clasa ofstream este *derivată* din clasele **ostream** (moşteneşte metodele care permit inserarea informaţiilor într-un flux de ieşire) şi **fstreambase** (moşteneşte operaţiile privitoare la asocierea fişierelor).

Pentru a asocia un **flux de intrare** unui fişier, este necesară crearea unui obiect de tipul **ifstream**. Clasa ifstream este *derivată* din clasele **istream** (moşteneşte metodele care permit extragerea informaţiilor dintr-un flux de intrare) şi **fstreambase** (moşteneşte operaţiile privitoare la asocierea fişierelor).

Pentru crearea unor obiecte din clasele ifstream sau ofstream se impune includerea headere-lor **iostream.h** şi **fstream.h**.

Constructorii clasei **ofstream** sunt:

ofstream();

Fluxul nu este asociat nici unui fişier

ofstream(const char *nume_fisier, int mod_acces=ios:out);

Parametri constructorului sunt numele fişierului şi modul de acces la acesta (scriere, fişier de ieşire).

ofstream (int fd, char *buffer, int lung_buffer);

Parametrii sunt numărul intern al fisierului (fd), zona tampon de intrări/ieşiri (buffer) şi lungimea zonei tampon (lung_buffer).

Constructorii clasei **ifstream** sunt:

ifstream();

Fluxul nu este asociat nici unui fişier

ifstream(const char *nume_fisier, int mod_acces=ios:in);

Parametri constructorului sunt numele fişierului şi modul de acces la acesta (citire, fişier de intrare).

ifstream (int fd, char *buffer, int lung_buffer);

Parametrii sunt numărul intern al fişierului (fd), zona tampon de intrări/ieşiri (buffer) şi lungimea zonei tampon (lung_buffer).

Exemplu:

Se declară obiectul numit fis_ies, de tipul ofstream, obiect asociat unui fişier cu numele DATE.txt, deschis apoi pentru ieşire (scriere). Scrierea în fişierul asociat obiectului se face printr-un flux care beneficiază de toate "facilităţile" clasei ostream.

   ofstream fis_ies("DATE.txt", ios::out);   //sau: ofstream fis_ies("DATE.txt");

Scrierea în fişierul DATE.txt: fis_ies<< . . . << . . .;

Pentru scriere formatată sau scriere binară în fişierul asociat: fis_ies.write(. . .);

Pentru examinarea stării fluxului de eroare corespunzător: `if (fis_ies) . . .;`

Se declară obiectul numit `fis_intr`, de tipul `ifstream`, obiect asociat unui fişier cu numele `NOU.txt`, deschis apoi pentru intrare (citire). Citirea din fişierul asociat obiectului se face printr-un flux care beneficiază de toate "facilităţile" clasei `istream`.

```
ifstream fis_intr("DATE.dat", ios::in);    //sau: ifstream fis_intr("NOU.txt");
```

Citirea în fişierul `NOU.txt`: `fis_intr>> . . . >> . . .;`
Pentru citire formatată sau citire binară din fişierul asociat: `fis_intr.read(. . . );`
Pentru examinarea stării fluxului de eroare corespunzător: `if (fis_intr) . . .;`

Observaţii: Conectarea (asocierea) unui flux unui fişier presupune:
- Fie existenţa a două tipuri de obiecte: un flux şi un fişier;
- Fie declararea unui flux care va fi asociat ulterior unui fişier.

Clasa **`fstream`** moşteneşte atât clasele ifstream, cât şi ofstream. Ea permite accesul în citire/scriere. Constructorul cu parametri al clasei fstream:

```
fstream(char *nume_fişier,int mod_deschid,int protec=filebuf::openprot);
```

Modul de deschidere (`mod_deschid`) a unui fişier poate fi:

ios::in	fişier de intrare
ios::out	fişier de ieşire
ios::ate	după deschiderea fişierului, poziţia curentă este sfârşitul acestuia
ios::app	mod append (de scriere la sfârşitul unui fişier existent)
ios::trunc	dacă fişierul există, datele existente se pierd
ios::nocreate	dacă fişierul asociat nu există, nu va fi creat decât la scriere
ios::binary	fişier binar
ios::noreplace	fişierul nu trebuie să existe

Modul de deschidere este definit printr-un cuvânt de stare, în care fiecare bit are o semnificaţie particulară. Pentru setarea (activarea) mai multor biţi, se foloseşte operatorul | (sau logic, pe bit), ca în exemplu.

```
class ios{
//……..
      public:
      enum open_mode{
            in=0x01, out=0x02, ate=0x04,
            app=0x08, trunc=0x10, nocreate=0x20,
            noreplace=0x40, binary=0x80
      };
};
```

Exemple: `ifstream f1("DATE.txt", ios::in|ios::nocreate);`
`fstream f2("pers.dat", ios::in|ios::out);`

- Metoda **`open`**
  Prelucrarea unui fişier începe cu deschiderea acestuia, care se poate realiza:
  - La instanţierea obiectelor din clasele ifstream, ofstream sau fstream, cu ajutorul constructorului cu parametri (constructorul apelează automat funcţia open);
  - Prin apelul explicit al funcţiei open.

  Metoda open este definită în clasa **`fstreambase`** şi este supraîncărcată în funcţiile derivate din aceasta. Ea are aceeaşi parametri ca şi constructorul clasei şi prototipul:

  ```
  void open (char *nume_fişier, int mod_deschidere, int protecţie);
  ```

- Metoda **`close`** realizează închiderea unui fişier:

  ```
  void close();
  ```

Exerciţiu: În exemplul următor se asociază fluxului de ieşire (obiectului `fis_ies`) fişierul numit `text.txt`. Se testează cuvântul de stare de eroare, iar dacă nu au fost probleme la deschidere, în fişier se scrie un text (2 linii), apoi, o serie de constante numerice, formatate. Se închide fişierul. Acelaşi fişier este asociat unui flux de intrare (pentru citire). Textul (scris anterior) este citit în variabila *şir*, apoi este afişat. Se citesc din fişier şi apoi sunt afişate constantele numerice. Se închide fişierul.

```
#include <fstream.h>
#include <iomanip.h>
int main()
{ofstream fis_ies("text.txt");       //deschidere fişier text.txt
if (!fis_ies){    cout<<"Eroare la dechiderea fişierului!\n"; return 1;}
fis_ies<<"Scrierea unui text \n în fişier\n";
fis_ies<<100<<hex<<setw(10)<<100<<setw(20)<<setprecision(12);
cout<<123.456789012356789<<endl;
fis_ies.close();                     //închiderea fişierului
ifstream fis_intr("text.txt");       //deschiderea fis. ptr. citire
if (!fis_intr){cout<<"Eroare la deschiderea fis. ptr. citire!\n";return 1;}
char sir[100];
for (int k=0; k<9; k++) {fis_intr>>sir; cout<<sir<<'\n';}
cout<<endl; int i, j; double u; fis_intr>>i>>hex>>j>>u;
cout<<i<<' '<<j<<' '<<u<<endl; fis_intr.close(); return 0; }
```

Observaţii:
Deschiderea şi închiderea fişierului s-ar fi putut realiza şi astfel:

```
fstream fis_ies;fis_ies.open("text.txt", ios::out);fis_ies.close();
fstream fis_in;fis_in.open("text.txt", ios::in);fis_in.close();
```

Exemplu: Să se scrie un program care concatenează două fişiere, depunând rezultatul în al treilea fişier. Se crează trei fluxuri, `f1`, `f2` şi `dest`, care sunt ataşate fişierelor corespunzătoare (prin metoda `open`). Copierea în fişierul destinaţie se realizează prin folosirea metodelor `get`, `put` şi funcţia `copy`. Metoda `close` întrerupe legătura logică dintre fluxuri şi fişiere.

```
#include <iostream.h>
#include <process.h>
#include <fstream.h>
void semn_er(char *c)
      {  cerr<<"\n\n Eroare la deschiderea fisierului "<<c<<endl;exit(1);     }
void copiere(ofstream &dest, ifstream &sursa)
      {char C;
      while (dest && sursa.get(C))
            dest.put(C); }
void main()
      {char nume_sursa1[14], nume_sursa2[14], destinatie[14];
      ifstream f1, f2; ofstream dest;
      cout<<"Numele celor 3 fisiere:"<<"Destinatie Sursa1 Sursa2\n";
      cin.read(destinatie,13);cin.read(nume_sursa1,13);cin.read(nume_sursa2,13);
      destinatie[13]=nume_sursa1[13]=nume_sursa2[13]='\0';
      f1.open(nume_sursa1, ios::nocreate);
      if (!f1)    semn_er(nume_sursa1);    //utiliz operator ! redefinit
      f2.open(nume_sursa2, ios::nocreate);
      if (!f2)    semn_er(nume_sursa2);    //utiliz operator ! redefinit
      dest.open(destinatie);
      if (!dest)  semn_er(destinatie);     //utiliz operator ! redefinit
      copiere (dest, f1); copiere(dest, f2);
      f1.close();f2.close();dest.close();
      }
```

Exerciţiu: Se ilustrează folosirea fluxurilor pentru operaţii de citire/scriere în fişiere text.

```
#include <fstream.h>
#include <iostream.h>
void main()
{ ofstream fisierout("nou.txt");       // Deschide fisierul text nou.txt..
 fisierout << " Teste fisiere "; //Scrie un text in fisier
 fisierout.close(); // Inchide fisierul.
  // Deschide un alt fisier text si scrie in acesta niste numere (numerele sunt separate prin spatii)
fisierout.open("numere.txt"); fisierout <<15<<" "<<42<<" "<<1;fisierout.close();
  // Deschide fisierul nou.txt pentru citire.
ifstream fisierin;  fisierin.open("nou.txt");
char p[50]; // Stabileste o zona de memorie folosita pentru citirea textului.
fisierin >>p;       // Citeste si afiseaza primele doua cuvinte din fisier.
cout <<p<<" "; fisierin >>p;  cout <<p<<endl;
fisierin.close(); // Inchide fisierul.
int numar;  fisierin.open("numere.c");
        // Deschide fisierul text numere.c pentru citirea numerelor intregi
        // Citeste numere din fisier pana cand ajunge la sfarsitul fisierului.
        while (!fisierin.eof())
           {fisierin >> numar;                    // Citeste un numar intreg.
              if (!fisierin.eof())      cout<<numar<<endl;
        // Daca nu a ajuns la sfarsitul fisierului afiseaza numarul.
    }
 fisierin.close(); // Inchide fisierul.
  // se citeste textul din fisierul nou.txt cuvant cu cuvant.
fisierin.open("nou.txt");
while (!fisierin.eof()) // Citeste cuvinte din fisier pana ajunge la sfarsitul fisierului.
    {    // Citeste cuvant cu cuvant.
fisierin >>p;      cout <<p<<endl;
    }
}
```

13.8. FIŞIERE BINARE

Fişierele binare reprezintă o succesiune de octeţi, asupra cărora la intrare sau la ieşire nu se realizează nici o conversie. Ele sunt prelucrate binar, spre deosebire de exemplul anterior, în care prelucrarea se realiza în mod text (un număr în format intern este reprezentat binar, dar în format extern el apare ca un şir de caractere. Un şir în formatul extern are terminatorul '\n', iar în format intern are terminatorul '\0'). Deoarece nu există un terminator de linie, trebuie specificată lungimea înregistrării. Metodele `write`, `read` nu realizează nici o conversie.

- Metoda **`write`** scrie în fişier un număr de n octeţi:
  `ostream &write(const char * tab, int n);`
- Metoda **`read`** citeşte din fişier n octeţi.
  `istream &read (const char * tab, int n);`

Limbajul C++ oferă (ca şi limbajul C) posibilitatea de acces direct într-un fişier conectat la un flux de date. După fiecare operaţie de citire (extragere din fluxul de intrare) sau citire (inserţie în fluxul de ieşire), pointerul care indică poziţia curentă în fluxul de date este incrementat cu un număr egal cu numărul de octeţi transferaţi. În această situaţie se realizează un acces secvenţial (ca în cazurile precedente).

Clasele `ifstream` şi `ofstream` au ca metode funcţiile **`seekg`** (membră a clasei `istream`), respectiv **`seekp`** (membră a clasei `ostream`) care permit modificarea valorii pointerului.

```
istream & seekg(long);
istream & seekg(long, seek_dir);
```

```
ostream & seekp(long);
ostream & seekp(long, seek_dir);
```

Ambele metode primesc doi parametri:
- un întreg reprezentând deplasarea pointerului în raport cu baza (referinţa) precizată de al doilea parametru;
- o constantă întreagă care precizează baza. Valorile constantei sunt definite în clasa `ios`:

```
class ios{  //. . .
public:
      enum seek_dir{
            beg=0,            deplasare faţă de începutul fişierului (implicit)
            cur=1,            deplasare faţă de poziţia curentă
            end=2             deplasare faţă de sfârşitul fişierului
      };
};
```

Pentru aflarea valorii pointerului, clasa `ifstream` are metoda **`tellg`**, iar clasa `ofstream` are metoda **`tellp`**, cu prototipurile:

```
long istream::tellg();
long ostream::tellp();
```

Exemplu: Se crează ierarhia de clase din figura 13.5., în care se implementează clasa `student` şi clasa proprie `fisier_stud`, clasă derivată din `fstream`. În acelaşi fişier, binar, se realizează şi scrierea şi citirea. Datorită posibilităţii de acces direct, se pot modifica informaţiile referitoare la studentul înregistrat în fişier pe poziţia k.

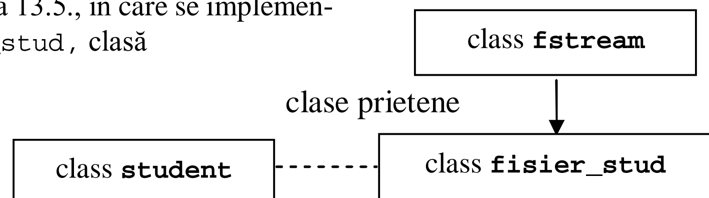

Figura 13.5. Ierarhia de clase

```
#include <fstream.h>
#include <iomanip.h>
class student
{     char nume[20];
      int grupa, note[3];
public:
      void citire_date();
      friend ostream &operator<<(ostream &, const student &);
      friend class fisier_stud;
};

class fisier_stud:public fstream              //fişier binar cu obiecte din cls student
{public:
fisier_stud(){;};
fisier_stud(const char*num_f,int mod,int
protecţie=filebuf::openprot):fstream(num_f,mod |ios::binary,protecţie){}
void open(const char *num_f, int mod, int protectie=filebuf::openprot)
      {fstream::open(num_f,mod|ios::binary,protectie);}//apelul met. open din cls. fstream
int citeste_f(student &);
int scrie_f (const student &);
};

void student::citire_date()
{ int gr, OK;
for (int k=0; k<21; k++)      nume[k]=0;
cin.ignore(1000, '\n'); cout<<"Numele studentului:";
cin.get(nume, 21);cin.ignore(1000, '\n');
do{
```

```
      cout<<"Grupa:"; cin>>gr;      OK=(cin && gr>0 && gr<8000);
      if (!OK){
            cout<<"Eroare. Repetaţi introducerea!\n"; cin.clear();      }
      else {grupa=gr; cin.ignore(1000, '\n');}
}while (!OK);
for (int k=0; k<3; k++){
int n;
do{
      cout<<"nota "<<k+1<<":";      cin>>n;     OK=cin && n>0 && n<=10;
      if (!OK){
            cout<<"Nota gresită.Repetati!\n";cin.clear();cin.ignore(1000, '\n');
      }
}while (!OK);
note[k]=n;cin.ignore(1000, '\n');
}
}
ostream &operator << (ostream &ies, const student &stud)
{ies<<"Student:"<<stud.nume<<" Grupa:"<<stud.grupa<<" Note:";
for (int k=0; k<3; k++)       ies<<stud.note[k]<<' ';
ies<<endl;return ies;}

typedef union {student s; char sbin[sizeof(student)];} stud;

int fisier_stud::scrie_f(const student &s)
{stud s1;   s1.s=s;     write(s1.sbin, sizeof(student));    return bad();     }

int fisier_stud::citeste_f(student &s)
      {stud s1;   read(s1.sbin, sizeof(student));     s=s1.s;     return bad();}

main()
{char Nume_Fis[]="Stud_bin.bin"; student s; fisier_stud fs; int gata;
//deschidere fişier ptr. scriere
fs.open (Nume_Fis, ios::out);
if (!fs){
      cout<<"eroare la deschiderea fişierului "<<Nume_Fis<<" ptr. scriere\n";
      return 1;}
cout<<"Poz. în fiş la deschidere:"<<fs.tellp()<<endl; gata=0;
while(!gata)
{char rasp;
cout<<"Datele studentului:\n";s.citire_date();cout<<"Date introduse:"<<s<<endl;
do{
    cout<<"Scrieţi date în fişier [D|N] ?";
      rasp=toupper(cin.get());cin.ignore(1000, '\n');
} while (răsp!='D' && răsp!='N');
if (răsp == 'D'){
      fs.scrie_f(s); cout<<"Poz. în fişier: "<<fs.tellp()<<endl;}
do{
        cout<<"Continuaţi introducerea [D|N]?";răsp=toupper(cin.get());
      cin.ignore(1000, '\n');
} while (răsp!='D' && răsp!='N');
if (răsp=='N')          gata=1;
}
fs.close();
//citire
fs.open(Nume_Fis, ios::in);
if (!fs){
      cout<<"Eroare la deschiderea fiş-lui "<<Nume_Fis<<" ptr. citire\n";
      return 1; }
cout<<"Poz. în fiş la deschidere:"<<fs.tellg()<<endl;
cout<<"Date citite din fişier:\n";  int k=0;
while (!fs.eof())
{     fs.citeşte_f(s);
```

```
      if (!fs && !fs.eof()){  cout<<"Eroare la citire\n";return 1; }
      if (!fs.eof()){cout<<s<<endl;cout<<"Poz în fişier:"<<fs.tellg();
      cout<<endl;k++;}
}
cout<<"S-au citit din fişier:"<<k<<" studenţi\n";
fs.close();

fs.open(Nume_Fis, ios::in | ios::out); //deschidere fisier actualizare (intr/ies)
if (!fs){
      cout<<"Eroare la deschidere fis. "<<Nume_Fis<<" ptr. citire/ scriere\n";
      return 1;
}
cout<<"Daţi numărul stud-lui pentru care vreţi să inlocuiţi datele:";cin>>k;
cin.ignore(1000, '\n'); fs.seekg(k*sizeof(stud));s.citire_date();fs.scrie_f(s);
fs.seekg(0); k=0;
while(!fs.eof()){
      fs.citire_f(s);
      if (!fs && !fs.eof()){cout<<"Eroare la citirea din fiş:"<<Nume_Fis<<endl;
            return 1;   }
if (!fs.eof()){
      cout<<s<<endl;cout<<"Poz în fişier:"<<fs.tellg()<<endl;k++;}
}
cout<<"S-au gasit în fişier:"<<k<<" studenţi\n";
fs.close();
return 0;
}
```

ÎNTREBĂRI ŞI EXERCIŢII

Chestiuni teoretice

1. Ce înţelegeţi prin conceptul de stream?
2. Care consideraţi că ar fi avantajele utilizării abordării orientate pe obiecte a sistemului de I/O?
3. Ce sunt manipulatorii?
4. Cum îşi poate crea utilizatorul manipulatorii proprii?

Chestiuni practice

Rezolvaţi problemele din capitolul 8, folosind abordarea orientată pe obiecte.