Streamuri de intrare/ ieşire

- Operaţiile de intrare-ieşire în C++ sunt implementate prin intermediul unor obiecte ce sunt instanţieri ale unei ierarhii de clase prezentată în Figura 1.

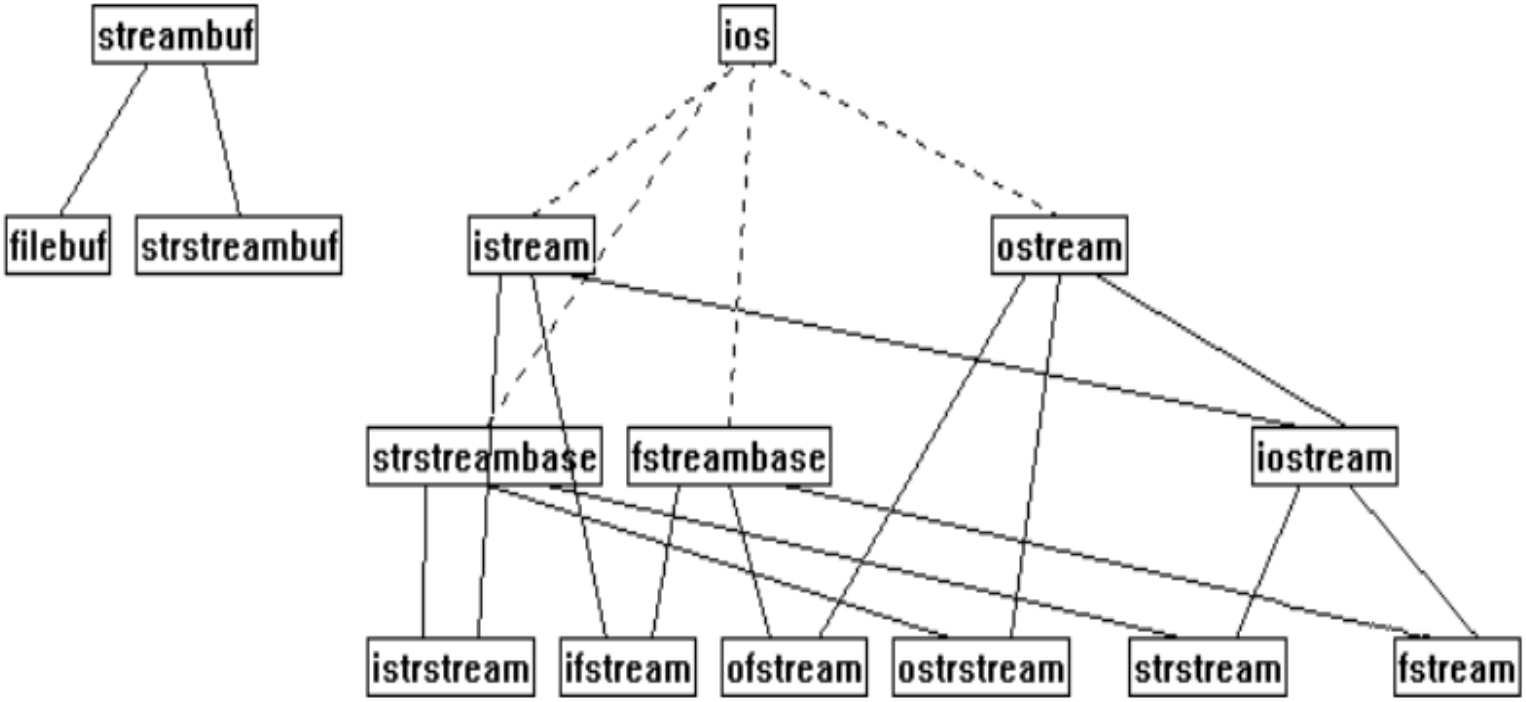

Figura 1. Ierarhia de clase ce formează biblioteca I/O în C++

- Obiectele care sunt instanţieri ale claselor ierarhiei ce are drept clasă de bază clasa ***ios*** se numesc **streamuri**
- Un stream poate fi privit, la modul abstract, ca un flux de date, mai precis, ca o succesiune de biţi de lungime infinită folosită ca buffer pentru a stoca date ce urmează a fi procesate. Există astfel streamuri de intrare, streamuri de ieşire şi streamuri de intrare/ieşire, acestea din urmă asociate unor dispozitive capabile să fie simultan surse de intrare şi de ieşire (ca de exemplu fişierele)

- **Clasa *streambuf*** - Clasa abstractă ***streambuf*** este responsabilă de gestiunea buffer-ului utilizat pentru crearea unei implementări eficiente a streamurilor. Prin intermediul metodelor pur virtuale clasa streambuf furnizează cadrul de implementare a comunicării cu dispozitivele fizice asociate streamurilor (fişiere pe diverse suporturi, consolă: monitor – tastatură). Stream-urile gestionează mereu un pointer la obiectul asociat de un tip derivat din streambuf.
- Un ***streambuf*** poate fi privit ca o secvenţă de caractere ce poate creşte sau descreşte în concordanţă cu cerinţele aplicaţiei. În funcţie de tipul stream-ului se asociază unul sau doi pointeri :
  - un pointer **put** ce indică poziţia următorului caracter ce se va depune în secvenţă ca urmare a unei operaţii de inserţie;
  - un pointer **get** ce indică poziţia următorului caracter ce va fi extras din secvenţă ca urmare a unei operaţii de extracţie;
- De exemplu, ***ostream*** are doar un pointer **put**, ***istream*** are doar un pointer get în timp ce ***iostream*** are ambii pointeri.
- Atunci când se creează un stream, acestuia i se asociază un pointer la un obiect de tip streambuf. Ca urmare, clasele de tip stream furnizează constructori ce au argumente de tip streambuf *.
- Toate clasele de tip stream supraîncarcă operatorii de inserţie şi respectiv de extracţie pentru a opera asupra obiectelor de tip streambuf*.
- **Clasa *filebuf*** - este derivată din clasa ***streambuf*** şi adaugă funcţionalităţile necesare pentru comunicarea cu fişiere.
- **Clasa *strstreambuf*** - este derivată din clasa streambuf şi adaugă funcţionalităţile necesare pentru citirea şi scrierea caracterelor din, respectiv într-un buffer de tip şir de caractere. Citirea şi scrierea se pot face aleatoriu în interiorul buffer-ului. Se implementează de asemenea operaţii de căutare. Zona rezervată de un obiect strstreambuf poate fi de dimensiune fixă sau dinamică. În general stream-urile de intrare sunt de dimensiune fixă, iar cele de ieşire, putând avea dimensiune imprevizibilă, pot fi şi dinamice. Diferenţa faţă de fstreambuf este aceea că, în cazul unui obiect strstreambuf nu există o sursă sau destinaţie reală pentru operaţiile de scriere, respectiv citire, astfel încât chiar buffer-ul preia unul sau ambele roluri.
- **Clasa ios -** este o clasă abstractă utilizată pentru a grupa o serie de funcţionalităţi comune, necesare pentru clasele derivate. Nu s-a intenţionat crearea de obiecte de tipul ***ios*.** Clasa gestionează informaţii despre starea stream-ului : flag-uri de eroare, flag-uri şi valori de formatare, precum şi conexiunea cu buffer-ele utilizate la intrare/ieşire (pointerul către structura de informaţii a buffer-ului).

Operaţii de I/O standard

- Funcţionalităţile legate de operaţiile de I/O standard sunt furnizate prin intermediul bibliotecii ***iostream***, parte componentă a bibliotecii standard a limbajului C++ **.**
- Obiectele şi funcţiile bibliotecii sunt incluse în spaţiului de nume (***namespace***) ***std***. Aceasta înseamnă fie că toate obiectele şi funcţiile de I/O trebuie prefixate cu "std::", fie se foloseşte instrucţiunea "using namespace std;" , fie se include varianta mai veche a fişierului header "#include <iostream.h>".
- Ierarhia de clase responsabile cu operaţiile de intrare/ieşire standard este prezentată mai jos (a se observa folosirea moştenirii multiple):

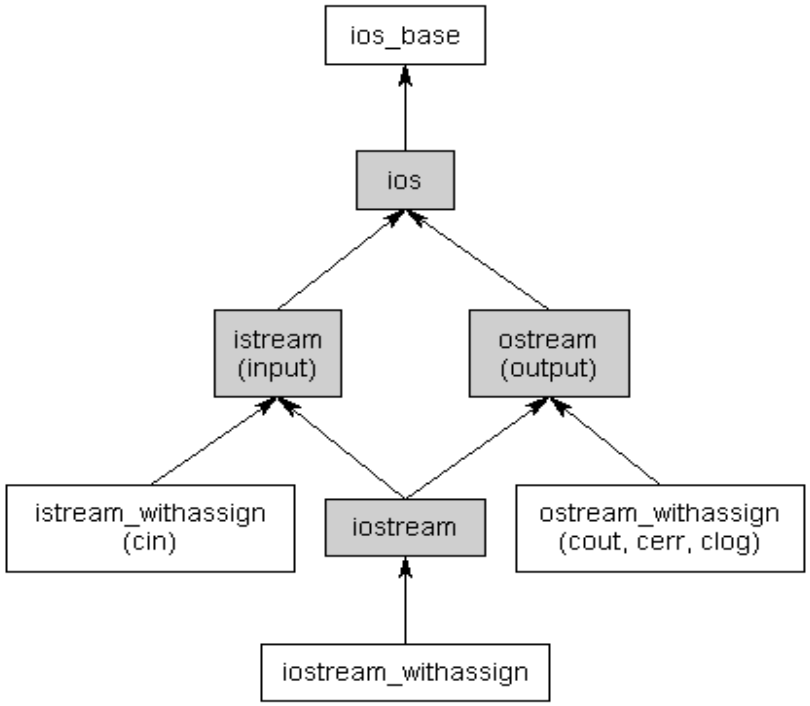

Figura 2. Ierarhia de clase pentru operaţii I/O standard

- Clasele care au terminaţia "_withassign" au definită supraîncărcarea operatorului de atribuire, permiţând astfel atribuirile între streamuri (redirectări).

Streamuri standard în C++

- Un stream standard este un stream pre-conectat furnizat programului de către mediul de programare pentru comunicarea cu consola
- Există patru astfel de streamuri

    1. **cin** — un obiect de tip istream_withassign conectat cu intrarea standard (tipic tastatura)
    2. **cout** — un obiect de tip ostream_withassign conectat cu ieşirea standard (tipic monitorul)
    3. **cerr** — un obiect de tip ostream_withassign conectat cu ieşirea standard pentru erori (tipic monitorul), furnizând ieşire ne-bufferată
    4. **clog** — un obiect de tip ostream_withassign conectat cu ieşirea standard pentru erori (tipic monitorul), furnizând ieşire bufferată

- Operatorul de inserţie (<<) precum şi cel de extracţie (>>) sunt supraîncărcaţi pentru tipurile de date standard
- Deoarece la supraîncărcare se returnează o referinţă la stream, operatorii de inserţie şi de extracţie se pot aplica înlănţuiţi (evident, pentru fiecare tip de operaţie în parte)
- Operatorul de extracţie operează cu date "formatate", ignorând spaţiile albe (blank, tabulatori şi linie nouă).
- Un avantaj important al limbajului C++ este acela că permite supraîncărcarea operatorilor de inserţie şi extracţie pentru tipuri abstracte de date
- Este posibilă formatarea intrărilor şi/sau a ieşirilor prin intermediul indicatorilor de format (format flags)
- Indicatorii de format sunt constante aparţinând unui tip enumerate definit în clasa ***ios***. Aceştia reprezintă biţi între-un dublu cuvânt ***x_flag*** ce este dată membru a clasei ***ios***

```
class ios
{
public:
    enum io_state       { goodbit = 0, eofbit = 01, failbit = 02, badbit = 04,
                          hardfail = 08 };
    enum open_mode      { in = 01, out = 02, ate = 04, app = 010, trunc = 020,
                          nocreate = 040, noreplace = 0100 };
    enum seek_dir       { beg = 0, cur = 01, end = 02 };
    enum                { skipws = 01, left = 02, right = 04, internal = 010,
                          dec = 020, oct = 040, hex = 0100, showbase = 0200,
                          showpoint = 0400, uppercase = 01000, showpos = 02000,
                          scientific = 04000, fixed = 010000,
                          unitbuf = 020000, stdio = 040000 };

    static const long     basefield;
    static const long     adjustfield;
    static const long     floatfield;
                          ios(streambuf *);
    virtual               ~ios();
    inline int            bad() const;
    static long           bitalloc();
    inline void           clear(int state = 0);
    inline int            eof() const;
    inline int            fail() const;
    inline char           fill() const;
    char                  fill(char);
    inline long           flags() const;
    long                  flags(long);
    inline int            good() const;
    long&                 iword(int);
    inline int            operator!();
    inline                operator void *();
    inline int            precision() const;
    int                   precision(int);
    void *&               pword(int);
    inline streambuf      *rdbuf();
    inline int            rdstate() const;
    long                  setf(long setbits, long field);
    long                  setf(long);
    static void           sync_with_stdio();
    inline ostream        *tie() const;
    ostream               *tie(ostream *);
    long                  unsetf(long);
    inline int            width() const;
    int                   width(int n);
    static int            xalloc();

protected:
                          ios();
    void                  init(streambuf *);
    inline void           setstate(int state);

};
```

Figura 2. O schiţă a definiţiei clasei ***ios***

- În detaliu, semnificaţia indicatorilor de formatare este:

```
public:
    .........
      enum {
                skipws          //  skips whitspace on input
                left            //  left justification
                right           //  right justifiction
                internal        //  pads after sign or base character
                dec             //  decimal format for integers
                oct             //  octal format for integers
                hex             //  hex format for integers
                showbase        //  show the base character for octal or hex
                showpoint       //  show the decimal point for all floats
                uppercase       //  uppercase A-F for hex
                showpos         //  show + sign for numbers
                scientific      //  use exponential notation
                fixed           //  used oridnary decimal notation
                unitbuf         //  flush the buffer
                stdio           //  Flush the C stdio streams stdout and stderr after each output // operation (for programs
                                //   that mix  C and C++ output conventions).
        };
```

- Există de asemenea grupe / câmpuri de indicatori

```
ios::basefield    = ios::dec | ios::oct | ios::hex
ios::adjustfield  = ios::left | ios::right | ios::internal
ios::floatfield   = ios::scientific | ios::fixed
```

- Setările biţilor se pot face cu funcţii membru ale claselor ierarhiei
- Atunci când se setează un indicator care este parte a unui câmp (adjustfield, basefield sau floatfield), trebuie să existe garanţia faptului că ceilalţi indicatori sunt resetaţi. Numai un singur indicator din grup poate fi setat la un moment dat.
- Controlul indicatorilor de format se poate face şi cu ajutorul unor aşa-numiţi manipulatori ce sunt funcţii membru speciale ce returnează referinţă la stream şi deci se pot insera direct în stream. Pentru utilizarea manipulatorilor trebuie inclus fişierul header ***iomanip***

- Un stream are asociată o stare de eroare care se defineşte cu ajutorul unor constante ale tipului enumerat ***io_state*** definit în clasa ***ios***.
- ***io_state*** reprezintă o colecţie de biţi (flag-uri) ce descriu starea internă a unui obiect de tip stream
- În detaliu, semnificaţia flag-urilor este:

```
class ios  {
.....
  public :
  enum io_state {
            goodbit ,    // Semnalează  faptul că nu s-au produs erori.
            eofbit ,     // Semnalează că s-a întâlnit end-of-file pe parcursul operaţiei de extracţie
            failbit ,    // Semnalează că extracţia sau conversia au eşuat, dar stream-ul este încă utilizabil
            badbit ,     // Semnalează că s-a produs o eroare severă, uzual într-o operaţie asupra obiectului
                         // asociat de tip streambuf, recuperarea din eroare fiind puţin probabilă
            hardfail     // Semnalează producerea unei erori din care streamul nu se poate recupera
            };
```

- Accesul la biţii de eroare se face prin intermediul unor funcţii membru

eof()	true dacă ***eofbit*** e setat, false altfel
fail()	true dacă ***failbit*** sau ***hardfail*** sunt setate, false altfel
bad()	true dacă ***badbit*** este setat, false altfel
good()	true dacă ***goodbit*** estet setat, false altfel
clear()	şterge toate flagurile (fără argumente) sau şterge un flag specificat

- Un stream intrat într-o stare de eroare nu mai permite operaţii de intrare/ieşire până când condiţia de eroare este înlăturată şi biţii de eroare sunt şterşi
- Un stream poate fi testat pentru a stabili dacă se află sau nu într-o stare de eroare într-o manieră similară testării unei expresii logice.
- Acest lucru este posibil datorită faptului că în clasa ***ios*** este supraîncărcat operatorul de negare **!** (! *stream* este diferit de zero dacă cel puţin unul din biţii de eroare asociaţi stării streamului *stream* este setat.
- De asemenea este definită conversia unui stream într-un pointer spre ***void*** . Pointerul rezultat este nul dacă cel puţin unul din biţii de eroare ai stării este setat. Pointerul respective se poate folosi numai pentru testarea stării streamului.

Indicatori de format ios (formatting flags)

Indicator	Semnificaţie	Grupa
skipws	ignoră spaţiile albe (whitespace) în intrare.	
left	cadrare stânga	\
right	cadrare dreapta	- adjustfield
internal	utilizează spaţii între semn sau bază şi numar	/
dec	zecimal.	\
oct	octal.	- basefield
hex	hexazecimal.	/
showbase	indicator de bază în ieşire (0 octal, 0x hexa).	
showpoint	afişează punctul zecimal.	
uppercase	utilizează majuscule X, E, şi cifre hexa ABCDEF (implicit, litere mici).	
showpos	'+' înaintea nr. pozitive	
scientific	format exponential	- floatfield
fixed	format virgulă fixă	/
unitbuf	goleşte toate streamurile după inserţie	
stdio	goleşte stdout, stderror după inserţie	

Functii membru ios

Functie	Semnificatie
fill()	returnează caracterul de umplere
fill(ch)	setează caracterul de umplere (returnează vechiul caracter)
precision()	returnează precizia (numarul de cifre afişate la partea fractionara).
precision(p)	setează precizia
width()	returnează lungimea campului (în caractere).
width(w)	setează lungimea câmpului (în caractere).
setf(ind)	setează indicatorii specificaţi.
unsetf(ind)	resetează indicatorii specificaţi.
setf(ind, grupa)	şterge câmpul specificat (grupa) şi apoi setează indicatorii

Manipulatori ios

Manipulator	Semnificatie
ws	setează ignorarea spaţiilor albe (whitespace) în intrare.
dec	zecimal
oct	octal
hex	hexazecimal.
endl	inserează linie noua şi goleşte stream-ul (de ieşire)
ends	inserează caracterul nul pentru a încheia un şir de ieşire
flush	goleşte stream-ul de ieşire

Manipulator	Argument	Semnificaţie
setw	lăţimea câmpului (int)	Setează lungimea câmpului
setfill	caracter de umplere (int)	Setează caracterul de umplere (implicit este spaţiu).
setprecision	precizie (int)	Setează precizia (nr. de cifre afişate la partea fracţionară).
setiosflags	indicatori de format(long)	Setează indicatorii specificaţi.
resetiosflags	indicatori de format (long)	Resetează indicatorii specificaţi

Functii istream

Functie	Semnificaţie
>>	extracţie pentru tipurile de bază şi pentru cele ce l-au supraîncărcat
get()	returnează caracterul extras din stream sau EOF
get(ch);	extrage un caracter în ch, returnează referinţă la stream .
get(str)	extrage caractere în vectorul str, până la '\0', returnează referinţă la stream.
get(str, MAX)	extrage până la MAX caractere în vectorul str, returnează referinţă la stream.
get(str, DELIM)	extrage caractere în str până la delimitatorul specificat (implicit '\n'); lasă caracterul DELIM în stream, returnează referinţă la stream.
get(str, MAX, DELIM)	extrage caractere până la MAX caractere în str sau până la characterul DELIM ; lasa caracterul DELIM în stream, returnează referinţă la stream.
getline(str, MAX, DELIM)	extrage până la MAX caractere din str sau până la caracterul DELIM (elimina caracterul DELIM din stream), returnează referinta la stream.
putback(ch)	inserează ultimul caracter citit inapoi în streamul de intrare
ignore(MAX, DELIM)	extrage şi ignora până la MAX caractere sau până la delimitatorul specificat (inclusiv), implicit '\n')
peek(ch)	citeste un caracter, lasandu-l în stream.
gcount()	returnează numarul de caractere citit de un apel (imediat precedent) de get(), getline(), read().
read(str, MAX)	pentru fişiere; extrage până la MAX caractere în str până la EOF.
seekg(positie)	setează distanţa (in bytes) a pointerului de fişier fata de inceputul fişierului
seekg(positie, seek_dir)	setează distanţa (in bytes) a pointerului de fişier fata de o pozitie specificata seek_dir, ce poate fi ios::beg, ios::cur, ios::end.
tellg(pos)	returnează poziţia (in bytes) a pointerului de fişier faţă de începutul fişierului

Functii ostream

Functie	Semnificaţie
<<	inserţie formatată pentru toate tipurile standard şi pentru cele ce l-au supraîncărcat
put(ch)	inserează un caracter ch în stream.
flush()	goleşte conţinutul buffer-ului şi inserează linie nouă
write(str, SIZE)	inserează SIZE caractere din vectorul str în stream.
seekp(pozitie)	setează distanţa în bytes a pointerului de fişier fata de începutul fişierului
seekp(pozitie, seek_dir)	setează distanţa (in bytes) a pointerului de fişier fata de o poziţie specificată seek_dir (poate fi ios::beg, ios::cur, or ios::end).
tellp()	returnează poziţia pointerului de fişier în bytes.

Biţi de eroare (Error-status bits)

Nume	Semnificatie		
goodbit	fără erori (nici un bit setat, valoare= 0).	\	
eofbit	s-a atins sfârşit de fişier	-	
failbit	operaţie eşuată (eroare utilizator, EOF prematur).	-	date membru io_state
badbit	operaţie invalid a(streambuf neasociat).	-	
hardfail	eroare nerecuperabilă.	/	

Functie	Semnificatie
eof()	returnează true dacă bitul EOF e setat.
fail()	returnează true dacă unul din biţii fail bit, sau bad sau hard-fail sunt setaţi
bad()	returnează true dacă unul din biţii bad sau hard-fail sunt setaţi
good()	returnează true dacă totul e OK; nici un bit nu e setat
clear(int=0)	fără argument, şterge toti biţii de eroare; altfel setează biţii specificati
rdstate	fără argument, returnează data membru state

Biţi de mod pentru functia open()

Bit	Rezultat
in	deschis pentru citire (implicit pentru ifstream).
out	deschis pentru scriere (implicit pentru ofstream).
ate	începe citirea sau scrierea la sfârşitul fişierului(AT End).
app	începe scrierea la sfârşitul fişierului(APPend).
trunc	dacă fişierul exista, il trunchiază la lungime 0(TRUNCate).
nocreate	semnalează eroare dacă la deschiderea fişierului acesta nu exista deja
noreplace	semnalează eroare dacă la deschiderea fişierului în ieşire acesta exista deja, cu exceptia situatiilor în care ate sau app sunt setate.
binary	deschide fişierul în modul binar (nu text)

Observaţii :

- În cazul manipulatorilor fără argument, setările rămân valabile până la distrugerea streamului, în timp ce manipulatorii ce au argumente afectează doar următorul articol din stream
- Indicatorii de format se pot seta simultan folosind operatorii pe biţi:

  Exemplu: `cout.setf(ios::showpos | ios::uppercase);`

- Manipulatorii setiosflags(flag) şi setf(flag) setează doar indicatorii precizaţi. Dacă un indicator este parte a unui grup, nu se şterg ceilalţi indicatori ai grupului.
- Funcţia membru setf(flag, field) este utilă pentru setarea indicatorilor ce fac parte dintr-un grup. De asemenea manipulatorii hex, oct şi dec resetează în mod corespunzător ceilalţi indicatori ai grupului basefield.

Exemple:
```
cout.setf(ios::oct, ios::dec | ios::oct | ios::hex);
cout.setf(ios::right, ios::adjustfield);
```

Exemple :

1. Operarea cu funcţii membru

```
cout.setf(ios::showpos); // setează flag-ul ios::showpos
cout << 27 << endl;
```

Efectul execuţiei este afişarea valorii ca +27

2. În cazul citirii unui şir de caractere, se poate limita numărul caracterelor citite din stream

```
#include <iomanip>
.........
char buf[10];
cin >> setw(10) >> buf;  // echivalent cin.width(10);
```

Drept rezultat, se vor citi primele 9 caractere din stream (+ '\0'). Celelalte caractere rămân in stream urmând a extrase la o următoare operaţie de citire. Este afectată doar prima citire!

3. Formatul pentru afişarea numerelor reale se pot utiliza următoarele setări:

```
cout.setf(ios::scientific, ios::floatfield);
cout.precision(2);
// echivalent  cout << setprecision(2);
```

4. Operaţii I/O standard

```
#include <iostream>
#include<iomanip>
using namespace std;

int main()
{
    int i=123;
    cout.setf(ios::showpos);
    cout.width(7);
    cout.setf(ios::showbase);
    cout.setf(ios::right,ios::adjustfield);
    //cout.setf(ios::internal,ios::adjustfield);
    cout<<i<<endl;

    cout.width(5);//AFECTEAZA DOAR PRIMUL cout!
    cout.setf(ios::showpos);
    cout.setf(ios::left,ios::adjustfield);
    cout.fill('0');
    cout<<i<<endl; cin.get();
    // cout<<"i="<<i<<endl; //i=000+123

    cout.width(5);
    cout.setf(ios::right,ios::adjustfield);
    cout.fill(' ');
    cout.setf(ios::showpos);
    cout<<i<<endl;;    cin.get();
    cout.setf(ios::oct,ios::basefield);
    cout<<i<<endl;cin.get();// ramane setat

    cout.setf(ios::showbase|ios::hex,ios::basefield);
    cout<<i<<endl;    cin.get();

    double pi=3.14159265;
    cout.setf(ios::fixed,ios::floatfield);
    cout<<pi<<endl;    cin.get();
    cout.setf(ios::scientific,ios::floatfield);
    cout<<pi<<endl;    cin.get();

    cout<<setw(6) <<
    resetiosflags(ios::internal|ios::right)<<
    setiosflags(ios::left)<<setfill('0')<<dec<<i<<endl;
    cin.get();

    cout<<setw(6) <<
    resetiosflags(ios::internal|ios::left)<<
    setiosflags(ios::right)<<setfill(' ')
    <<setiosflags(ios::showpos)<<i<<endl;
    cin.get();

    cout<<hex<<i<<endl;
    cout<<oct<<i<<endl;
    cin.get();
    cout<<setprecision(7)<<pi<<endl;
    cin.get();
    return 0;
}
```

```
   +123

+1230

 +123

0173

0x7b

+3.141593

+3.141593e+000

+12300

  +123

0x7b
0173

+3.1415927e+000
```

5. Testarea stării unui stream

```
// valori numerice

#include <iostream>
using namespace std;

int main()
{
    double d;
    while (cin >> d)
        cout << d;
    return 0;
}
/*
12.3
12.366
66
-4
-4
@

//caractere

#include <iostream>
using namespace std;

int main()
{
    char ch;
    while (cin >> ch)
        cout << ch;
    return 0;
}

abcdef
abcdef
123456
123456
^Z
```

```
// recuperare din eroare

#include <iostream>
using namespace std;

int main()
{
    double d;
    while (cin >> d)
        cout << d;

    cin.clear(); //OBLIGATORIU

    char ch;
    while (cin >> ch)
        cout << ch;

    return 0;
}


1
12
23
34
45
56
6a
ax
x^Z
```

Operarea cu fişiere în C++

- Operarea cu fişiere în C++ este în mare măsură similară operării cu streamurile standard
- Există trei clase dedicate exploatării fişierelor, destinate operaţiilor de intrare, ieşire şi respective intrare/ieşire:
  - ifstream (derivată din ***istream***)
  - ofstream (derivată din ***ostream***)
  - fstream (derivată din ***iostream***).
- Pentru a utiliza aceste clase trebuie inclus fişierul header ***fstream***.
- Spre deosebire de streamurile ***cout, cin, cerr*** şi ***clog***, care sunt deja pregătite pentru utilizare, streamurile de tip fişier trebuie explicit construite de către programator.
- Pentru a deschide un fişier pentru citire şi/sau scriere se instanţiază un obiect aparţinând clasei corespunzătoare de I/O (eventual cu numele/specificatorul fişierului ca parametru)
- Pentru a efectua operaţiile propriu-zise se utilizează operatorii de inserţie (<<) sau extracţie (>>)
- Pentru a închide un fişier se apelează explicit funcţia membru ***close()*** (sau, cea ce nu este recomandat, se poate miza pe încheierea domeniului de valabilitate a obiectului şi apelarea implicită a destructorului clasei respective).

Exemplu :

```
#include <ofstream>
#include <iostream>
using namespace std;

.....
    ofstream outf("Exemplu.txt"); // în directorul curent, implicit text
    if (!outf)
    {
        cerr << "Exeplu.txt nu s-a putut deschide pentru scriere!" << endl;
        exit(1);
   }
   outf << "Linia 1" << endl;
   outf << "Linia 2" << endl;
.....
```

Moduri de operare cu fişiere

Moduri de deschidere a fişierelor

- Modalităţile de deschidere a unui fişier sunt descrise prin intermediul unei colecţii de biţi (flags) care specifică modul de operare atunci când se apelează funcţia ***open()***

Mod	Semnificaţie
app	Deschide un fişier în modul adăugare la sfârşit
ate	Caută sfârşitul fişierului înainte de a citi/scrie
binary	Deschide un fişier în mod binar (în loc de text)
in	Deschide un fişier în modul citire (implicit pentru ifstream)
nocreate	Deschide un fişier numai dacă acesta există deja
noreplace	Deschide un fişier numai dacă acesta nu există deja
out	Deschide un fişier modul scriere (implicit pentru ofstream)
trunk	Şterge fişierul dacă acesta există deja

- Este posibil să se specifice mai multe moduri utilizând operatorul pe biţi | (sau)
- Observaţie ios::in şi ios::out sunt implicite pentru clasele ifstream şi ofstream respectiv.
- Dacă se optează pentru utilizarea clasei ***fstream*** (ce poate opera atât pentru citiri cât şi pentru scrieri), trebuie specificate explicit modurile ***ios::in*** şi/sau ***ios::out***, în funcţie de modul dorit de exploatare a fişierului.

Exemplu : `fstream iofile("Sample.dat", ios::in | ios::out);`

Moduri de acces în fişiere

- Fiecare clasă ce operează cu fişiere conţine un pointer pentru controlul poziţiei curente din/în care se citeşte/scrie
- Implicit, la deschidere, pointerul este setat la începutul fişierului, cu excepţia modului de deschidere pentru adăugare (append) , când este plasat la sfârşitul fişierului
- Există două tipuri de acces la un fişier:
  - modul secvenţial - informaţia se citeşte în ordinea în care a fost scrisă
  - modul direct - informaţia se poate citi/scrie în locaţii specificate
- Accesul direct se face prin manipularea pointerului către fişier folosind funcţii specifice: ***seekg(),*** pentru operaţiile de intrare and ***seekp(),*** pentru operaţiile de ieşire. Acestea au doi parametrii: primul reprezintă un offset ce determină numărul de octeţi cu care se face deplasamentul faţă de poziţia pointerului fişierului, iar al doilea, este un indicator ios ce specifică poziţia pointerului faţă de care se aplică parametrul offset.
- Un offset pozitiv înseamnă deplasarea pointerului fişierului către sfârşitul fişierului, un offset negativ înseamnă deplasarea pointerului fişierului către începutul fişierului

Indicator	Semnificaţie
beg	Offset-ul este relativ la începutul fişierului (implicit)
cur	Offset-ul este relativ la poziţia curentă a pointerului fişierului
end	Offset-ul este relativ la sfârşitul fişierului

Exemple:

```
inf.seekg(14, ios::cur);     // deplasare înainte cu 14 bytes
inf.seekg(-18, ios::cur);    // deplasare înapoi cu 18 bytes
inf.seekg(22, ios::beg);     // deplasare la cel de al 22-lea byte din fisier
inf.seekg(-28, ios::end);    // deplasare la cel de al 28-lea byte
                             // înainte de sfârşitul fişierului
inf.seekg(0, ios::beg);      // deplasare la începutul fişierului
inf.seekg(0, ios::end);      // deplasare la sfârşitul fişierului
```

- Alte două funcţii utile sunt ***tellg()*** şi ***tellp(),*** ce returnează valoarea absolută a poziţiei pointerului fişierului, în felul acesta ele pot fi folosite şi pentru a determina dimensiunea fişierului

Exemplu :

```
ifstream inf("Test.txt");
inf.seekg(0, ios::end);     // deplasare la sfârşitul fişierului
cout << inf.tellg();        // se afişează numărul de octeţi ai
                            // fişierului
```

- Clasa ***fstream*** este capabilă să ofere posibilitatea efectuării operaţiilor de citire şi scriere în acelaşi timp (aproape!)
- Nu este insă posibilă comutarea între cele două operaţii în mod arbitrar, sigurul mod posibil este acela de a efectua o operaţie de căutare a pointerului fişierului (dacă nu se doreşte deplasare pointerului, se caută poziţia curentă a acestuia)

Exemplu :

```
iofile.seekg(iofile.tellg(), ios::beg); // se caută poziţia curentă
```

Exemplul 1. Controlul poziţiei pointerilor put şi get

```
#include <fstream>
#include <iostream>
#include<stdlib.h>

using namespace std;

int main ()
{
  long pos;

//  fstream fis("test.txt",ios::out | ios::in | ios::trunc);
  fstream fis("test.txt");
  if(!fis)
  {
      cout<<"ERR\n"; exit(1);
  }

  fis.write ("This is an apple",17);
  pos=fis.tellp();
  fis.seekp (pos-8);
  fis.write (" sam",4);

  char temp[256];
  fis.seekg(0, ios::beg);

  fis.read(temp, 17);
  cout<<temp<<endl;

  return 0;
}
```

Exemplul 2. Copiere de fisiere text

```
#include<iostream>
#include<stdlib>
#include <fstream>

using namespace std;

int main(int argc, char **argv)
{
    if(argc!=3)
    {
           cout<<"Nr. Incorect de arg !\n";      exit(1);
    }
    ifstream sursa(argv[1]);
    char linie[256];

    if(sursa.fail())
       cerr<<"Eroare la desch fis"   <<argv[1]<<endl;
    else
    {
        ofstream dest(argv[2]);
        if(dest.fail())
            cerr<<"Eroare la desch fis"   <<argv[2]<<endl;
        else
        {
            while(!sursa.eof())
            {
                sursa.getline(linie, sizeof(linie));
                if(sursa)       //.good())
                {
                    dest<<linie<<endl;
                    if(dest.fail())
                    {
                        cerr<<"Eroare la scriere fis"   <<argv[2]<<endl;
                        cin.get();
                     break;
                    }
                }
             }
           sursa.close();
             dest.close();
        }
    }
   cout<< "Succes!"<<endl;
   cin.get();
   return 0;
}
```

Exemplul 3. Editarea unei agende telefonice

```
#include<iostream>
#include <fstream>
#include<stdlib.h>
#include<string.h>
using namespace std;

class Agenda
{
        char nume[80];
        char  numar[6];
     public:
        Agenda(){};
        Agenda(char *n,  char *nr)
        {
           strcpy(nume,n);
           strcpy(numar,nr);
        }
        friend ostream& operator<<
 (ostream& stream, const Agenda& a);
        friend istream& operator>>
 (istream& stream, Agenda& a);
};

ostream& operator<<
 (ostream& stream, const Agenda& a)
{
     stream<<a.nume<<" ";
     stream<<a.numar<<"\n";
     return stream;
};

istream& operator>>
        (istream& stream, Agenda& a)
{
    cout<<"Nume ";
    stream>>a.nume;
    cout<<"Numar ";
    stream>>a.numar;
    cout<<"\n";
    return stream;
};

int main()
{
  Agenda a;
  char c;
fstream  at("tel.txt", ios::app
 |ios::in | ios::out);
if(!at)
{
      cout<<"Nu pot deschide!!\n";
      return 1;
}
for(;;)
{
      do
      {
          cout<<"1. Introducere\n";
          cout<<"2. Afisare\n";
          cout<<"3. Incheiere\n";
          cout<<"Introduceti o
optiune: ";
          cin>>c;
      }while(c<'1' || c>'3');

      switch(c)
      {
          case '1':
            cin>>a;
            cout<<"Intrarea este ";
            cout<<a<<endl;
            at<<a;
            break;
           case'2':
             char linie[255];
             at.seekg(0,ios::beg);
             while(!at.eof())
             {
  at.getline(linie,sizeof(linie),'\n');
  cout<<linie<<endl;
             }
             at.clear();//reset eof;
             cout<<endl;
             break;
          case '3':
             at.close();
             return 0;
      }
  }
}
```

Exemplul 4. Prelucrare de date prin istrstream

```
#include<iostream>
#include<iomanip>
#include <fstream>
#include<string.h>
#include<strstream>

using namespace std;

int main(void)
{
    char *numbers=
       "ABC \n 1001\n 1.2345\n
123456789L";

    int count;
    float rate;
    long distance;

    istrstream buff(numbers);

    buff>>count;
    if(buff.fail())
    {
        cerr<<"Err cit int\n";
        buff.clear();
        buff.ignore(10,'\n');
    }

    buff>>rate;
    if(buff.fail())
    {
        cerr<<"Err cit float\n";
        buff.clear();
    }

    buff>>distance;
    if(buff.fail())
    {
        cerr<<"Err cit long\n";
        buff.clear();
    }
    cout<<"distance 1=" << distance
 << endl;

    buff>>distance;
    if(buff.fail())
    {
        cerr<<"Err cit long\n";
        buff.clear();    }

    cout<<"count="<<count<<endl;
    cout<<"rate="<<rate<<endl;
    cout<<"distance 2=" <<distance
        <<endl;
    cin.get();
    return 0;
}
```

Rezultate :

Err cit int
distance 1=1
Err cit long
count=0
rate=1001
distance 2=0

Exemplul 5. Prelucrare date prin ostrstream

```
#include<iostream>
#include<strstream>
using namespace std;

int main(void)
{
    ostrstream buff;
    int i=99;
    char ch1='O',ch2='K';
    buff<<i<<"  "<< ch1 << ch2 <<
          endl<<'\0';
    char *p=buff.str();
// str returneaza un pointer la
// sirul controlat de
// obiectul de tip ostrstream
    cout<<"Continut :\n"<<p<<endl;
    cin.get();
    return 0;
}
```

Rezultate :

Continut :
99 OK

- Observaţie:

Clasele strstreambuf, istrstream, ostrstream, strstream sunt actualmente depreciate.

O listă cu substituiri valide este :

In loc de:	Se foloseşte:
`<strstream>, strstreambuf`	`<sstream>, basic_stringbuf`
`<strstream>, istrstream`	`<sstream>, basic_istringstream`
`<strstream>, ostrstream`	`<sstream>, basic_ostringstream`
`<strstream>, strstream`	`<sstream>, basic_stringstream`