10. SISTEMUL "STREAM" DE I/E DIN C++

Deși limbajul C++ conține toate rutinele din biblioteca de I/E a limbajului C, acesta ne pune la dispoziție și un sistem propriu de I/E orientat pe obiecte, care ne ajută să citim/scriem obiecte de diferite tipuri. Principalul avantaj al sistemului de I/E din C++ constă în faptul că el poate fi redefinit în cadrul propriilor clase. De asemenea, el introduce unele tehnici mai avansate cum ar fi: "redefinirea operatorilor" și "funcțiile virtuale". Ca și limbajul C, sistemul de I/E orientat pe obiecte din C++ nu face deosebire între operațiile de I/E care folosesc consola și cele care utilizează fișiere. Problemele tratate în acest capitol se referă la: principiile de bază ale sistemului de I/E din C++, operațiile de I/E cu format, funcțiile de I/E de tip "manipulator", crearea propriilor operatori *stream* de tip "inserter" și "extractor" și alte caracteristici ale sistemului *stream* din C++.

10.1. Principiile de bază ale sistemului de I/E din C++

Sistemul de I/E din C++, ca și sistemul analog al limbajului C, operează prin *streams*. Un *stream* este un dispozitiv logic care fie produce, fie consumă informație; el este cuplat la un dispozitiv fizic prin intermediul sistemului de I/E din C++. Toate *stream*-urile se comportă în aceeași manieră, chiar dacă diferă dispozitivele fizice aferente; din acest motiv, sistemul de I/E poate funcționa pe orice tip virtual de dispozitiv fizic. De exemplu, se poate folosi aceeași funcție operator atât pentru afișarea unui text pe ecran, cât și pentru scrierea sa într-un fișier sau listarea sa la imprimantă.
După cum se știe, la executarea unui program C se deschid, în mod automat, trei *streams* predefinite: **stdin**, **stdout** și **stderr**. Similar, la execuția unui program scris în C++, se deschid, în mod automat, următoarele patru *streams* predefinite:

Stream		Semnificația	Dispozitivul implicit
în C++	echivalentul în C		
cin	**stdin**	Intrarea standard	Tastatura
cout	**stdout**	Ieșirea standard	Ecranul
cerr	**stderr**	Eroarea standard	Ecranul
clog		Versiunea "tampon" a lui cerr	Ecranul

Streams comunică, în mod implicit cu consola. Totuși, în mediile care asigură gestionarea operațiilor de I/E, acestea pot fi redirecționate către alte dispozitive. C++ definește sistemul său de I/E în fișierul antet **iostream.h**. În acest fișier este definită o ierarhie complexă de clase, construită cu ajutorul moștenirii multiple, care asigură operațiile de I/E.

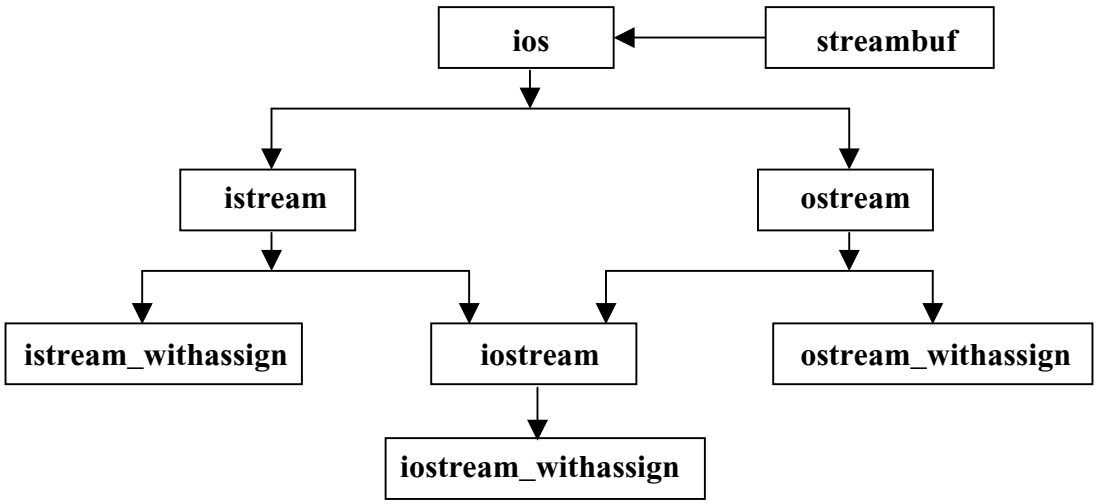

Fig. 10.1. Ierahria claselor de I/E din C++

Clasa **streambuf** furnizează buffer-ul folosit de către *stream*-uri. Toate clasele de *stream*-uri sunt derivate din clasa de bază **ios** care stochează starea *stream*-ului, asigură operațiile cu format și tratează erorile. Clasa **ios** are un obiect asociat de tip **streambuf**, cu rol de buffer al *stream*-ului. Clasele **istream** și **ostream**, derivate din **ios**, sunt destinate operațiilor de intrare, respectiv de ieșire. Clasa **iostream** este constituită prin moștenire multiplă din clasele **istream** și **ostream**, oferind astfel un suport atât pentru intrări, cât și pentru ieșiri. Clasele **istream_withassign**, **ostream_withassign** și **iostream_withassign** sunt derivate din **istream**, **ostream** și respectiv **iostream**, prin adăugarea unei definiții a operatorului de asignare "=", pentru a putea redirecta intrările și ieșirile prin asignarea unui *stream* către un altul. *Stream*-urile predefinite **cout**, **cerr** și **clog** sunt din clasa **ostream_withassign**, iar **cin** este o instanță a clasei **istream_withassign**.

10.2. Operații de I/E cu format

10.2.1. Setarea și resetarea indicatorilor de format

Pentru formatarea datelor în scopul afișării sau citirii acestora, fiecare *stream* din C++ are asociat un număr de "indicatori de format" (*flags*) codificați într-o variabilă de tip **long int**. Constantele corespunzătoare acestor "indicatoare" sunt conținute în următoarea enumerare definită în clasa **ios**:

```
// Indicatoarele de format
enum {
        skipws      = 0x0001,
        left        = 0x0002,
        right       = 0x0004,
        internal    = 0x0008,
        dec         = 0x0010,
        oct         = 0x0020,
        hex         = 0x0040,
        showbase    = 0x0080,
        showpoint   = 0x0100,
        uppercase   = 0x0200,
        showpos     = 0x0400,
        scientific  = 0x0800,
        fixed       = 0x1000,
        unitbuf     = 0x2000,
        stdio       = 0x4000
};
```

Descrierea acestor flaguri este prezentată în Talelul 10.1.

Tabelul 10.1. *Flag-urile de formatare din C++*

Flag	Semnificație
ios :: skipws	Ignoră caracterele de tip *whitespace* (spațiu, tab, *newline*) din intrare
ios :: left	Aliniază ieșirea la stânga, în cadrul lungimii specificate a câmpului
ios :: right	Aliniază ieșirea la dreapta (implicit)
ios :: internal	Valoarea numerică se extinde astfel încât să completeze tot câmpul
ios :: dec	Se folosește (revine la) notația zecimală (implicită) a întregilor
ios :: oct	Se folosește notația octală pentru întregi
ios :: hex	Se folosește notația hexazecimală pentru întregi
ios :: showbase	Stabilește baza de numerație a ieșirii (de exemplu, prefixul 0x pentru hexazecimal și prefixul 0 pentru octal)
ios :: showpoint	Include un punct zecimal pentru valorile în virgulă mobilă
ios :: uppercase	Determină utilizarea literelor majuscule în rezultatele prezentate la ieșire
ios :: showpos	Afișează semnul plus la afișarea valorilor pozitive
ios :: scientific	Determină folosirea notației științifice pentru numerele în virgulă mobilă (de exemplu, -1.23e+02)

ios :: fixed	Determină folosirea notației zecimale normale pentru numere în virgulă mobilă (de exemplu, -123.45)
ios :: unitbuf	Eliberează *stream*-ul după fiecare operație de I/E
ios :: stdio	Determină eliberarea fiecărui stream după o operație de ieșire

Când un *flag* de format este poziționat (setat), este activată caracteristica respectivă; când el este șters (resetat), se utilizează formatul implicit.
Pentru a seta un *flag* de format, se utilizează funcția *setf()*, membră a clasei **ios**. Sintaxa sa este:

long setf (**long** flags);

Această funcție activează flag-urile specificate de argumentul flags (toți ceilalți indicatori rămânând neschimbați). De exemplu, pentru a activa *flag*-ul **showpos**, se poate folosi instrucțiunea:

STREAM**.**setf (**ios** :: showpos);

unde parametrul STREAM se referă la *stream*-ul aferent și unde se remarcă utilizarea operatorului *scope resolution* (::) în scopul recunoașterii constantei **showpos**.
Observație. Funcția *setf()* este o funcție membră a clasei **ios** și afectează *stream*-ul creat de aceasta. Prin urmare, orice apel la funcția *setf()* se referă la *stream*-ul corespunzător. Fiecare *stream* conține (în mod individual) informațiile de stare a propriului format.
Pentru poziționarea mai multor indicatori de stare, se folosește operatorul "|" (OR). De exemplu, apelul de mai jos, poziționează *flag*-urile **showbase** și **hex**:

STREAM**.**setf (**ios** :: showbase | **ios** :: hex);

Funcția *unsetf()*, membră a clasei **ios**, este complementul funcției *setf()*. Prototipul acesteia este:

long unsetf (**long** flags);

Această funcție șterge unul sau mai mulți indicatori de format specificați în parametrul *flags* și restaurează starea anterioară a flagurilor. Toți ceilalți indicatori rămân neschimbați.
Pentru a afla stările curente ale *flag*-urilor se folosește funcția *flags()*, membră a clasei **ios**. Prototipul ei este:

long flags();

Funcția întoarce starea curentă a fiecărui *flag* de format codificat într-o variabilă de tip **long int**.
Funcția *flags()* are și o a doua versiune, care ne permite să poziționăm toți indicatorii asociați unui *stream*, la valorile specificate de argumentul ei. Prototipul acestei funcții este:

long flags (**long** f);

unde f este un șablon de biți care se copiază în variabila care conține indicatorii de format asociați *stream*-ului, suprascriind deci stările anterioare.

1. Utilizarea funcției *setf()* și *unsetf*() este prezentată în exemplul următor:

```
// Program P10_1.CPP  Utilizarea functiei setf()
# include <iostream.h>
void  main (void)
{       // Afișarea valorilor utilizând poziționările implicite
        cout << 175.66 << " Salut ! " << 100 << '\n';
        cout << 10 << ' ' << -10 << '\n';
        cout << 100.0 << '\n' << '\n';
        // Se schimba formatul de afișare
        cout.setf (ios :: hex | ios :: scientific);          // Se setează indicatorii hex și scientific
        cout << 175.66 << " Salut ! " << 100 << '\n' << '\n';
        cout.setf (ios :: showpos);                          // Se setează indicatorul showpos
        cout << 10 << ' ' << -10 << '\n' << '\n';
        cout.setf (ios :: showpoint);                        // Se setează indicatorul showpoint
```

```
        cout.unsetf (ios :: scientific);                    // Se dezactivează indicatorul scientific
        cout << 100.0 << '\n';
        cout.setf (ios :: uppercase | ios :: showbase); // Se setază indicatorii uppercase și showbase
        cout << 66 << " Salut ! " << '\n';
        cout.unsetf (ios :: uppercase);                     // Se dezactivează indicatorul uppercase
        cout << 66 << '\n' << '\n';
        cout.unsetf (ios :: hex);                           // Se dezactivează indicatorul hex
        cout.setf (ios :: oct);                             // Se activează indicatorul oct
        cout << 66 << '\n' << '\n';
}
```

La execuția programului, pe ecran apar afișate:

```
        175.66  Salut !  100
        10  -10
        100
        1.7566e+02  Salut !  64
        a  fffffff6
        +100.000000
        0X42  Salut !
        0x42
        0102
```

Se observă că indicatorul **showpos** afectează doar afișajul în zecimal, nu și pe cel în hexazecimal. Funcția *unsetf()* are într-adevăr efect complementar funcției *setf()*.

2. Programul următor utilizează funcția *flags()* pentru a afișa poziționările indicatorilor de format, asociați dispozitivului standard **cout**:

```
// Program P10_2a.CPP   Utilizarea funcției flags()
# include <iostream.h>
void afiseaza_indicatori();             // Prototipul funcției afiseaza_indicatori()
void  main (void)
{       afiseaza_indicatori();          // Se afișează valorile implicite ale indicatorilor de format
        cout.setf (ios :: oct | ios :: showbase | ios :: fixed);    // Se setează câțiva indicatori
        afiseaza_indicatori();
}
void afiseaza_indicatori()              // Această funcție afișează starea indicatorilor de format
{       long f, i;
        int j;
        char flgs [15][12] = { "skipws", "left", "right", "internal", "dec", "oct", "hex", "showbase",
                "showpoint", "uppercase", "showpos", "scientific", "fixed", "unitbuf", "stdio" };
        f = cout.flags();                                   // Se obțin setările curente ale indicatorilor
        for ( i = 1, j = 0; i <= 0x4000; i = i << 1, j++)       // Se verifică fiecare indicator
                if (i & f) cout << flgs[j] << " este activat\n";
                else cout << flgs[j] << " este dezactivat\n";
      cout << "\n";
}
```

Pe ecran se vor afișa mai întâi valorile implicite ale indicatorilor de format precizați în tabloul flgs[15][12], iar apoi starea indicatorilor de format după activarea câtorva dintre ei. Se va observa că anumiți indicatori (**skipws**, **unitbuf**) sunt implicit activați (Pentru dezactivarea acestora se va folosi *unsetf()*).

3. Programul următor utilizează a doua versiune a funcției *flags()*. Se dorește activarea indicatorilor **showpos**, **showbase**, **oct** și **right** care au valorile 0x0400, 0x0080, 0x0020, respectiv 0x0004. Reunind (efectuând funcția OR între) acești indicatori, se obține valoarea 0x04A4, care va fi utilizată ca argument al funcției *flags()*. Precizăm, că *toți ceilalți indicatori vor fi dezactivați*. Aceasta se poate vedea prin rularea programului:

```
// Program P10_2b.CPP   Utilizarea funcției flags()
# include <iostream.h>
void afiseaza_indicatori();              // Prototipul funcției afiseaza_indicatori()
void  main (void)
{       afiseaza_indicatori();           // Se afișează valorile implicite ale indicatorilor de format
        long f = 0x04A4;
        cout.flags (f);                  // Poziționează toți indicatorii
        afiseaza_indicatori();
}
void afiseaza_indicatori()               // Această funcție afișează starea indicatorilor de format
{       long f, i;   int j;
        char flgs [15][12] = { "skipws", "left", "right", "internal", "dec", "oct", "hex", "showbase",
                "showpoint", "uppercase", "showpos", "scientific", "fixed", "unitbuf", "stdio" };
        f = cout.flags();                                  // Se obțin setările curente ale indicatorilor
        for ( i = 1, j = 0; i <= 0x4000; i = i << 1, j++)          // Se verifică fiecare indicator
                if (i & f) cout << flgs[j] << " este activat\n";
                else cout << flgs[j] << " este dezactivat\n";
        cout << "\n";
}
```

10.2.2. Utilizarea funcțiilor width(), precision() și fill()

Toate cele trei funcții sunt membre ale clasei **ios** și poziționează următorii parametri de format: lățimea câmpului de afișat, precizia și caracterul de inserat.

Funcția *width()*, al cărei prototip este următorul:

int width (**int** w);

stabilește lățimea minimă w a câmpului pe care va fi afișat un rezultat. Dacă valoarea de afișat utilizează un spațiu mai mic decât cel specificat, câmpul este completat cu caracterul curent de inserat (implicit acesta este "spațiu"). Dacă, valoarea de afișat depășește lățimea câmpului, câmpul va fi suprascris, și valorile nu vor fi truncheate.

Se știe că la afișarea unei valori în virgulă mobilă, se folosesc, implicit, 6 cifre după punct. Funcția *precision()*,al cărei prototip este următorul:

int precision (**int** p);

stabilește, prin valoarea parametrului p, precizia de reprezentare după punctul zecimal (cu aproximările corespunzătoare).

Când un câmp de reprezentare trebuie completat, se folosește (în mod implicit) caracterul "spațiu". Totuși, se poate specifica un alt caracter de inserat apelând funcția *fill()*, al cărei prototip este:

char fill (**char** ch);

unde ch este noul caracter de inserat. Funcția înapoiază vechiul caracter.

Modul de utilizare a acestor funcții este prezentat în exemplele următoare:

```
4. // Program P10_3a.CPP   Utilizarea functiilor width(), precision(), fill()
# include <iostream.h>
void  main (void)
```

```
{       cout.width (10);                // Se stabilește lățimea minimă a câmpului
        cout << "Salut !" << '\n';      // Implicit, se adoptă alinierea la dreapta
        cout.fill ('%');                // Se stabilește caracterul de completare
        cout.width (10);                // Se stabilește lățimea minimă a câmpului
        cout << "Salut !" << '\n';      // Implicit, se adoptă alinierea la dreapta
        cout.setf (ios :: left);        // Alinierea se face acum la stânga
        cout.width (10);                // Se stabilește lățimea minimă a câmpului
        cout << "Salut !" << '\n';      // Ieșirea este aliniată la stânga
        cout.width (10);                // Se stabilește lățimea minimă a câmpului
        cout << 123.456789 << '\n';     // Formatul de precizie este cel implicit
        cout.width (10);                // Se stabilește lățimea minimă a câmpului
        cout.precision (3);             // Se stabilește precizia de afișare
        cout << 123.456789 << '\n';     // Precizia este de trei cifre
        cout.fill (` `);
cout << 123.456289 << '\n';
}
```

Pe ecran se va afișa:

```
          Salut !
%%%Salut !
Salut !%%%
123.456789
123.457%%%
123.456
```

Se observă că lățimea câmpului trebuie stabilită înainte de fiecare instrucțiune de afișare.

5. Programul următor creează un tabel aliniat de numere, prin apelarea funcțiilor de I/E cu format:

```
// Program P10_3b.CPP  Utilizarea functiilor width() și precision()
# include <iostream.h>
# include <math.h>
void  main (void)
{       double x;
        cout.precision (3);                             // Se stabilește precizia de reprezentare
        cout << "     x      sqrt(x)      x^2\n\n";     // Capul de tabel
        for (x = 2.0; x <= 20.0; x++) {
                cout.width (7);
                cout << x << "       ";
                cout.width (7);
                cout.setf (ios :: left);
                cout << sqrt (x) << "  ";
                cout.unsetf (ios :: left);
                cout.width (7);
                cout << x * x << '\n'; }
}
```

Acest program creează tabelul de mai jos:

```
        x       sqrt(x)         x^2
        2       1.414             4
        3       1.732             9
      . . . . . . . . . . . . . . . . . . . .
       20       4.472           400
```

10.3. Utilizarea funcțiilor de I/E de tip manipulator

Manipulatorii sunt funcții speciale de formatare a informațiilor în operațiilor de I/E. Aceștia sunt asociați unui *stream* și pot să apară în unele instrucțiuni de I/E schimbând starea *sream*-ului respectiv.

Manipulatorii standard din sistemul de I/E din C++ sunt prezentați în Tabelul 10.2.

Tabelul 10.2. *Manipulatorii de I/E din C++*

Manipulator	Acțiune	Tip	Exemplu
dec	Formatează datele numerice în zecimal (similar cu formatul %d din C)	I / E	cout << dec << variabila_int; cin >> dec >> variabila_int;
hex	Formatează datele numerice în hexazecimal (similar cu formatul %x din C)	I / E	cout << hex << variabila_int; cin >> hex >> variabila_int;
oct	Formatează datele numerice în octal (similar cu formatul %o din C)	I / E	cout << oct << variabila_int; cin >> oct >> variabila_int;
ws	Ignoră caracterele *whitespace* din *stream*-ul de intrare	I	cin >> ws;
endl	Inserează caracterul *newline* `\n` și golește buffer-ul de ieșire	E	cout << endl;
ends	Inserează caracterul *null* `\0`într-un șir	E	cout << ends;
flush	Golește (eliberează) buffer-ul lui **ostream**	E	cout << flush;
resetiosflags (**long** f)	Resetează indicatorii de format specificați în parametrul "f"	I / E	cout << resetiosflags(**ios**::dec); cin >> resetiosflags(**ios**::hex);
setbase (**int** base)	Stabilește baza de numerație la "base": 0, 8, 10, 16. (0 este setarea implicită)	I / E	cout << setbase(10); cin >> setbase(8);
setfill (**int** ch)	Stabilește caracterul de inserat dat de "ch"	I / E	cout << setfill(`.`); cin >> setfill(` `);
setiosflags (**long** f)	Setează (activează) indicatorii specificați în parametrul "f"	I / E	cout << setiosflags(**ios**::dec); cin >> setiosflags(**ios**::hex);
setprecision (**int** p)	Stabilește precizia (numărul de cifre de după punctul zecimal) la valoarea "p"	I / E	cout << setprecision(6); cin >> setprecision(8);
setw (**int** w)	Stabilește lățimea câmpului la numărul de caractere specificat în parametrul "w"	I / E	cout << setw(6) << var; cin >> setw(24) >> buf;

Există două tipuri de manipulatori: cu parametri și fără parametri. *Manipulatorii fără parametri* sunt declarați în fișierul antet **iostream.h** și pot fi folosiți în lanțul de operații de I/E cu sintaxa:

output << manipulator;
input >> manipulator;

Exemplu: cout << oct << 100 << hex << 100;

Manipulatorii cu parametri sunt declarați în fișierul antet **iomanip.h** și pot fi folosiți în lanțul de operații de I/E cu sintaxa:

output << manipulator (parametru);
input >> manipulator(parametru);

Exemplu: cout << setw(10) << setfill(`*`) << setiosflags(ios::left) << 100;

Un manipulator de I/E afectează doar *stream*-ul asociat; în schimb el nu afectează *stream*-urile dechise în mod curent. Principalul avantaj al utilizării manipulatorilor constă în faptul că sunt mai ușor de utilizat și permit scrierea unui cod mai compact.

Modul de utilizare al manipulatorilor poate fi urmărit și în exemplele următoare:

1. // Program P10_4a.CPP *Utilizarea funcțiilor manipulator fară si cu parametri*

 # include <iostream.h>
 # include <iomanip.h>

```
void  main (void)
{       cout << hex << 100 << endl;             // Se afișează în hexazecimal
        cout << oct << 10 << endl;              // Se afișează în octal
        cout << setfill ('X') << setw (10) << setbase(10);     // Caracterul de inserat este X, lățimea
                                                // câmpului este 10, iar baza de numerație este 10
        cout << 100 << " Pa " << endl;
        long f = 0x0060;                        // Se dezactivează manipulatorii hex și oct
        cout << resetiosflags(f);
        cout << 100 << endl;
        cout << 10 << endl;
        cout << setfill (' ') << setw (10);     // Caracterul de inserat este ` `, lățimea câmpului este 10
        cout << 100 << " Pa " << endl;
        f = 0x04A4;                             // Seteaza indicatorii showpos, showbase, oct și right
        cout << setiosflags (f);
        cout << 100 << "  " << 123.456789 << endl;
        cout << -10 << endl;
        cout << setfill ('*') << setw (10);   // Caracterul de inserat este `*`, lățimea câmpului este 10
        cout << 100 << " Pa " << endl;
        cout << resetiosflags (f);              // Resetează indicatorii showpos, showbase, oct și right
        cout << 100 << "  " << 123.456789 << endl;
        cout << -10 << endl;
        cout << setfill ('*') << setw (10) << setbase(16);
        f = 0x0082;                             // Setează indicatorii showbase și left
        cout << setiosflags (f);
        // cout << setiosflags(ios::showbase |ios::left);         // Setează indicatorii showbase și left
        cout << 100 << "  Pa " << endl;
}
```

Rezultatele afișate de acest program sunt:

```
        64
        12
        XXXXXXX100  Pa
        100
        10
                    100  Pa
        0144   +123.456789
        037777777766
        ******0144  Pa
        100  123.456789
        -10
        0x64******  Pa
```

2. // Program P10_4b.CPP *Utilizarea funcțiilor manipulator fară și cu parametri*

```
# include <iostream.h>
# include <iomanip.h>
void  main (void)
{       int i = 100;              // Întreg inițializat cu 100
        cout << setfill('.');     // Caracterul de inserare este punctul
        // Etichetele sunt aliniate la stânga iar valorile sunt aliniate la dreapta
        cout << setiosflags(ios :: left);
```

```
        cout << setw(20) << "Zecimal";
        cout << resetiosflags(ios :: left);
        cout << setw(6) << dec << i << endl;
        cout << setiosflags(ios :: left);
        cout << setw(20) << "Hexazecimal";
        cout << resetiosflags(ios :: left);
        cout << setw(6) << hex << i << endl;
        cout << setiosflags(ios :: left);
        cout << setw(20) << "Octal";
        cout << resetiosflags(ios :: left);
        cout << setw(6) << oct << i << endl;
}
```

Acest program generează următoarea ieșire:

```
        Zecimal . . . . . 100
        Hexazecimal . . . . . 64
        Octal . . . . . 144
```

3. Programul următor reia exemplul din programul P10_3b.CPP pentru a afișa un tabel cu radicali și pătrate, dar utilizează manipulatori de I/E.

```
// Program P10_4c.CPP
# include <iostream.h>
# include <iomanip.h>
# include <math.h>
void  main (void)
{  double x;
   cout << setprecision (3);
   cout << "      x    sqrt(x)       x^2\n\n";
   for ( x = 2.0; x <= 20.0; x++) {
        cout << setw (7) << x << "     ";
        cout <<setiosflags (ios :: left);
        cout << setw (7) << sqrt(x) << "  ";
        cout << resetiosflags (ios :: left);
        cout << setw (7) << x * x << '\n';
   }
}
```

10.4. Redefinirea operatorilor << și >>

Motivul principal pentru care utilizăm sistemul de I/E din C++, în locul celui existent în limbajul C, îl constituie posibilitatea redefinirii operatorilor de I/E de tip *stream* cu scopul de a face mai ușoară citirea, respectiv tipărirea obiectelor.

*10.4.1. Crearea propriilor funcții de tip **inserter***

În acest paragraf se va prezenta modul de redefinire a operatorului de ieșire <<. Limbajul C++ definește operația de ieșire *insertion*, iar operatorul << este denumit *insertion operator*. Când se redefinește operatorul de ieșire, se ceează o *funcție inserter*, sau pe scurt, un ***inserter***. Această denumire provine de la faptul că un operator de ieșire *inserează* informația în *stream*.

Toate funcțiile ***inserter*** au forma generală:

```
        ostream & operator << (ostream & STREAM, nume_clasa OBIECT)
        {       // corp inserter
```

return STREAM; }

Primul parametru reprezintă o referință a unui obiect de tip **ostream**; acest lucru înseamnă că STREAM trebuie să fie un *stream* de ieșire. Al doilea parametru primește obiectul care va fi afișat. ***Inserter***-ul înapoiază o referință a unui *stream* de tip **ostream** sau de tipul uneia din clasele derivate din aceasta.

Observație. Un ***inserter*** nu trebuie să fie o funcție membră a clasei în care operează (aceasta pentru că parametrul din stânga care este un *stream* nu poate fi membru al clasei), dar poate fi o funcție de tip **friend**. De fapt, în majoritatea aplicațiilor funcțiile ***inserter*** sunt declarate ca funcții **friend** ale clasei în care au fost create.

Exemple de definire și utilizare a funcțiilor de tip ***inserter***:

1. Utilizarea unui ***inserter*** de tip **friend** pentru obiecte de tip COORD

```
// Program P10_5a.CPP
# include <iostream.h>
class COORD {
        int x, y;                                   // Variabilele x și y sunt de tip private
public:
        COORD () { x = 0; y = 0; }
        COORD (int i, int j) { x = i; y = j; }
        friend ostream & operator << ( ostream &, COORD );    // Prototipul inserter-ului
};
ostream & operator << ( ostream & STREAM, COORD  OB )    // Definiția inserter-ului
{       STREAM << OB.x << ", " << OB.y << '\n';
        return STREAM;   }
void  main (void)
{   COORD A(1, 2), B(10, 20);        // Se creează două obiecte de tip COORD
    cout << "Obiectul A are coordonatele: " << A;   // Se afișează coordonatele obiectului A
    cout << "Obiectul B are coordonatele: " << B;   // Se afișează coordonatele obiectului B
}
```

Programul generează rezultatele:

```
        Obiectul A are coordonatele: 1, 2
        Obiectul B are coordonatele: 10, 20
```

Inserter-ul din acest program ilustrează genericitatea celor creați. În acest caz, instrucțiunea de ieșire din interiorul ***inserter***-ului trimite valorile x și y în STREAM, indiferent de tipul *stream*-ului pasat funcției. Dacă funcția ***inserter*** ar fi fost scrisă sub forma:

```
ostream & operator << ( ostream & STREAM, COORD  OB )    // Definiția inserter-ului
{       cout << OB.x << ", " << OB.y << '\n';
        return STREAM;   }
```

informația ar fi fost afișată pe dispozitivul standard de ieșire, care este cuplat la **cout** și acest lucru nu mai permite utilizarea funcției ***inserter*** cu alte *stream*-uri.

2. Rescriem programul de mai sus, omițând cuvântul cheie **friend**. Consecința acestei omiteri o reprezintă faptul că variabilele x și y trebuie declarate publice, întrucât funcția ***inserter*** nu are acces la variabile de tip **private** ale clasei.

```
// Program P10_5b.CPP
# include <iostream.h>
class COORD {
```

```
public:
        int x, y;                                       // Variabilele x și y sunt de tip public
        COORD () { x = 0; y = 0; }
        COORD (int i, int j) { x = i; y = j; }
};
ostream & operator << ( ostream & STREAM, COORD  OB )   // Definiția inserter-ului
{       STREAM << OB.x << ", " << OB.y << '\n';
        return STREAM;   }
void  main (void)
{   COORD A(1, 2), B(10, 20);        // Se creează două obiecte de tip COORD
    cout << "Obiectul A are coordonatele: " << A;   // Se afișează coordonatele obiectului A
    cout << "Obiectul B are coordonatele: " << B;   // Se afișează coordonatele obiectului B
}
```

3. Un ***inserter*** poate afișa și informație grafică, nu numai text. Programul următor creează o clasă numită TRIUNGHI, care memorează baza și înălțimea unui triunghi dreptunghic. ***Inserter***-ul clasei asigură afișarea triunghiului pe ecranul terminalului:

```
// Program P10_6.CPP   Utilizarea unui inserter de tip friend pentru obiecte de tip TRIUNGHI
# include <iostream.h>
class TRIUNGHI {
        int  inaltimea, baza;             // Baza și înălțimea triunghiului sunt de tip întreg
public:
        TRIUNGHI (int i, int b) { inaltimea = i; baza = b; }
        friend ostream & operator << ( ostream &, TRIUNGHI );   // Prototipul inserter-ului
};

ostream & operator << ( ostream & STREAM, TRIUNGHI OB )   // Definiția inserter-ului
{       int i, j, h, k;
        i = j = OB.baza - 1;
        for (h = OB.inaltimea - 1; h; h--) {
                for (k = i; k; k--)
                        STREAM << ' ';
                STREAM << '*';
                if (j != i) {
                        for (k = j-i-1; k; k--)
                                STREAM << ' ';
                        STREAM << '*';
                        }
                i--;
                STREAM << '\n';
        }
        for (k = 0; k < OB.baza; k++)
                STREAM << '*';
        STREAM << '\n';
        return STREAM;
}
void  main (void)
{  TRIUNGHI T1(5, 5), T2(10,10), T3(12, 12);       // Se creează trei triunghiuri dreptunghice
```

```
   cout << T1 << endl << T2 << endl << T3;          // care se afișează pe ecran
}
```

10.4.2. Crearea propriilor funcții de tip ***extractor***

Operatorul de tip *stream* de intrare >> poate fi redefinit într-o manieră asemănătoare celui de ieșire. El este denumit *extractor operator*, iar funcția operator care îl redefinește se numește ***extractor***. Această denumire provine de la faptul că un operator de intrare *extrage* informația dintr-un *stream*.
Sintaxa generală a funcțiilor ***extractor*** este:

```
istream & operator >> (istream & STREAM, nume_clasa  & OBIECT)
{       // corp inserter
        return STREAM;  }
```

Primul parametru reprezintă o referință a unui obiect de tip **istream**, care este un *stream* de intrare. Al doilea parametru este o referință a obiectului care primește datele de intrare. ***Extractor***-ul înapoiază o referință a unui *stream* de tip **istream**, care este un *stream* de intrare**.**
Observație. Așa cum un ***inserter*** nu poate fi o funcție membră a clasei în care operează, nici un ***extractor*** nu poate fi, dar poate fi o funcție de tip **friend**.

Exemple de definire și utilizare a funcțiilor de tip ***inserter*** și **extractor**:

1. Programul următor adaugă un ***extractor*** clasei COORD:

```
// Program P10_7.CPP   Utilizarea unui inserter și a unui extractor de tip friend
# include <iostream.h>
class COORD {
        int x, y;
public:
        COORD () { x = 0; y = 0; }
        COORD (int i, int j) { x = i; y = j; }
        friend ostream & operator << ( ostream & STREAM, COORD  OB );
        friend istream & operator >> ( istream & STREAM, COORD &  OB );
};
// Un inserter pentru clasa COORD
ostream & operator << ( ostream & STREAM, COORD  OB )
{       STREAM << OB.x << ", " << OB.y;
        return STREAM;
}
// Un extractor pentru clasa COORD
istream & operator >> ( istream & STREAM, COORD & OB )
{       cout << "Introduceti coordonatele: ";
        STREAM >> OB.x >> OB.y;
        return STREAM;
}
void  main (void)
{  COORD A(1, 2), B(10, 20);                    // Se creează obiectele A și B
   cout << "A = " << '(' << A << ')' << endl;   // Se afișează A
   cout << "B = " << '(' << B << ')' << endl;   // Se afișează B
   cin >> A;                                   // Se creează un nou nou obiect de tip COORD
   cout << "Noul A = " << '(' << A << ')' << endl;  // Se afișează noul obiect
}
```

Pe ecran se va afișa: A = (1, 2)

```
                    B = (10, 20)
                    Introduceți coordonatele: x y
                    Noul A = (x, y)
```
unde x și y sunt valorile introduse de la tastatură.

2. Programul următor conține o clasă, numită INVENTAR, care stochează numele unui articol, tipul și prețul său. Programul include atât un ***inserter***, cât și un ***extractor***.

```
// Program P10_8.CPP   Crearea clasei INVENTAR și a unor obiecte de acest tip
# include <iostream.h>
# include <string.h>
class INVENTAR {
        char  articol [40];      // Numele articolului
        int  tip;                // Tipul articolului
        double  pret;            // Prețul articolului
public:
        INVENTAR (char * a, int t, double p)
        {       strcpy (articol, a);
                tip = t;
                pret = p;
        }
        friend ostream & operator << ( ostream & STREAM, INVENTAR OB );
        friend istream & operator >> ( istream & STREAM, INVENTAR & OB );
};
// Un inserter pentru clasa INVENTAR
ostream & operator << ( ostream & STREAM, INVENTAR OB )
{       STREAM << OB.articol << ": " << "tip " << OB.tip;
        STREAM << ", pret " << OB.pret << '\n';
        return STREAM;
}
// Un extractor pentru clasa INVENTAR
istream & operator >> ( istream & STREAM, INVENTAR & OB )
{       cout << "Introduceti numele articolului: ";    STREAM >> OB.articol;
        cout << "Introduceti tipul: ";   STREAM >> OB.tip;
        cout << "Introduceti pretul: "; STREAM >> OB.pret;
        return STREAM;
}
void  main (void)
{
   INVENTAR OB1("Surub", 3, 2000.0), OB2("Piulita", 3, 1000.0);
   cout << OB1 << OB2;  cout << endl;       // Se afișează obiectele OB1 și OB2
   cin >> OB1;                              // Se creează un nou obiect de tip INVENTAR
   cout << OB1;                             // Se afișează noul obiect
}
```

10.5. Crearea unor manipulatori definiți de utilizator

Manipulatorii sunt importanți din două puncte de vedere. În primul rând, un manipulator poate "compacta" o secvență de operații de I/E, care se repetă, într-una singură, simplificându-se astfel codul sursă. În al doilea rând, un manipulator ajustat poate fi util când trebuie să se efectueze operații de I/E pe dispozitive care nu sunt standard. Se știe că există două tipuri de manipulatori:

unii asociați *stream*-urilor de intrare și alții asociați *stream*-urilor de ieșire. Totuși, mai există un criteriu de departajare a acestora: manipulatori cu parametru și manipulatori fără parametru. Crearea unor manipulatori proprii fără parametru, se face după următoarea regulă generală:

- pentru ieșire:

```
ostream & nume_manipulator (ostream & STREAM)
{       // Cod
        return STREAM;  }
```

- pentru intrare:

```
istream & nume_manipulator (istream & STREAM)
{       // Cod
        return STREAM;  }
```

Se observă că funcțiile manipulator primesc o referință a *stream*-ului cu care se operează și înapoiază o refeință la acel *stream*.

Exemple:

1. Programul de mai jos creează un manipulator denumit *init()*, care inițializează lățimea câmpului la 10 caractere, stabilește precizia la 4 și caracterul de completat la `*`.

```
// Program P10_9a.CPP  Crearea propriilor funcții de tip manipulator
# include <iostream.h>
ostream & init ( ostream & STREAM )      // Definirea propriului manipulator
{       STREAM.width (10);
        STREAM.precision (4);
        STREAM.fill ('*');
        return STREAM;
}
void  main (void)
{  cout << 123.456789 << endl;
   cout << init << 123.456789 << endl;
   cout << init << 123.456729 << endl;
}
```

Se va afișa:
```
123.456789
**123.4568
**123.4567
```

Se vede ca funcția *init()* este utilizată ca un operand al unei expresii de I/E, în aceeași manieră în care sunt folosiți manipulatorii interni.

2. Funcțiile manipulator de mai jos *atn()* și *obs()* asigură un mod mai simplu de afișare a unor cuvinte sau fraze des folosite:

```
// Program P10_9b.CPP  Crearea propriilor funcții de tip manipulator
# include <iostream.h>
ostream & atn ( ostream & STREAM )      // Atenție
{       STREAM << "Atentie: ";
        return STREAM;
}
ostream & obs ( ostream & STREAM )      // Observație
{       STREAM << "Observatie: ";
        return STREAM;
}
```

```
void  main (void)
{  cout << atn << "Circuit de inalta tensiune\n";
   cout << obs << "Stingeti toate becurile\n";
}
```

3. Programul următor creează o funcție manipulator denumită *obține_acces()*, care generează emiterea unui sunet și ne cere să introducem o parolă pentru a obține accesul la calculator:

```
// Program P10_9c.CPP  Crearea propriilor funcții de tip manipulator

# include <iostream.h>
# include <string.h>

// O funcție simplă de tip manipulator de intrare
istream & obtine_acces ( istream & STREAM )
{       cout << '\a';                          // Emite un sunet
        cout << "Introduceti parola: ";
        return STREAM;
}
void  main (void)
{  char par [60];
   do {
        cin >> obtine_acces >> par;
   } while ( strcmp (par, "parola") );
   cout << "Acces reusit !\n";
}
```

Se vede că acest program se încheie când de la tastatură se introduce cuvântul "parola".