Operații de intrare/ieșire în C++

Mihai Gabroveanu

Operații I/O în limbajul C

- Biblioteca standard stdio.h și conio.h
- Fișiere text
  - printf/fprintf
  - scantf/fscanf
- Fișiere binare
  - fread
  - fwrite

Operații I/O în limbajul C++

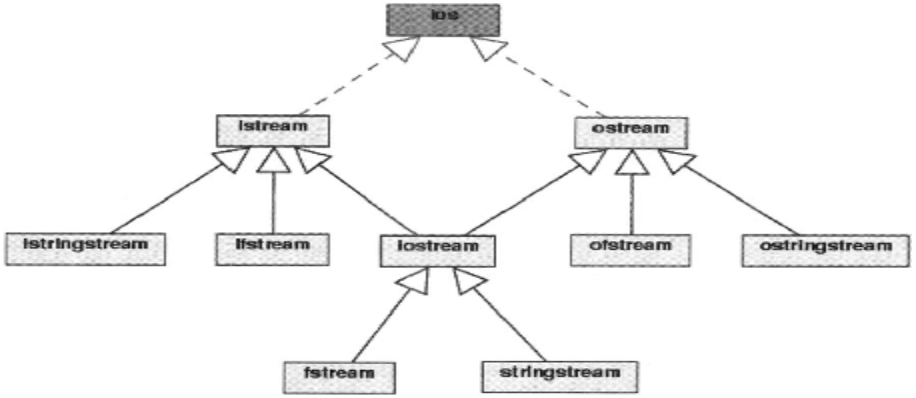
- Limbajul C++ moștenește de la C funcțiile de I/O
  - Dezavantaj: permit manipularea doar a tipurilor de bază
- Limbajul C++ introduce o ierarhie de clase pentru a realiza operațiile de intrare/ieșire (I/O)
- Această ierarhie are la bază conceptul de **stream** (flux)

Ce este un stream?

- **Stream**-ul este un concept abstract care înglobează orice ***flux*** de date de la o sursă (canal de intrare) la o destinație (canal de ieșire)
- Biblioteca standard ce înglobează ierarhia de clase pentru opertații de I/O

```
#include <iostream.h>  // compilatoare vechi
sau
#include <iostream> // compilatoare noi
using namespace std;
```

Ierarhia de clase I/O



Stream-uri standard

- Stream-uri standard create automat la rularea unui program:
  - *cin* – obiect de tip *istream* (*stream*-ul standard de intrare, de regula tastatura)
  - *cout* – obiect de tip o*stream* (*stream*-ul standard de ieşire, de regulă ecranul)
  - *cerr* – obiect de tip o*stream* (*stream*-ul standard de eroare, , de regulă ecranul)
- Operatori
  - Inserare <<
  - Extragere >>

Formatarea ieșirii

- Funcții pentru formatarea ieșirii

long setf(long indicator);//activarea indicatorilor de format
long unsetf(long indicator);//dezactivarea indicatorilor de format
int width(int w);//specifica latimea campului de formatat
int precision(int p);//specifica numarul de cifre de dupa virgula
char fill(char ch);//specifica caracterul de umplere, implicit spatiu

Exemplu

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

void main(){
      cout << 128 << endl ;// afişarea se face pe dimensiune implicita
      cout.unsetf(ios::dec);//dezactivez afisarea in zecimal
      cout.setf(ios::hex);// activez afisare in hexazecimal
      cout.setf (ios::showbase);//se afiseaza baza
      cout << 255 << endl ;
      cout.setf(ios::scientific);
      cout << 123.456;
}
```

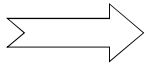
```
128
0xff
1.234560e+002
```

Exemplu

```
#include <iostream>

using namespace std;

int main(){
      cout << 123 << endl ;// afişarea se face pe dimensiune implicita
      cout.width(10);// se modifica câmpul la dimensiunea 10
      cout << 123 << endl ;
      cout.width(10);// se modifica câmpul la dimensiunea 10
      cout.fill('*');// se stabileşte caracterul de umplere '*'
      cout << 123 << endl ;
      cout.width(10);// se modifica câmpul la dimensiunea 10
      cout.precision(2);// se specifica afişarea cu precizie 2
      cout.setf(ios::fixed);// fixed este activat
      cout << 123.456 << endl;
      return 0;
}
```



```
123
       123
*******123
****123.46
```


Manipulatori

- Manipulatori sunt funcţii speciale ce pot fi incluse (inserate) în operaţiile de I/O cu scopul stabilirii caracteristicilor de formatare
- Pentru utilizarea acestora trebuie inclusă biblioteca

  #include <iomanip.h> // compilatoare vechi

  sau

  #include <iomanip>

  using namespace std;

-

Manipulatori

- Manipulatorii definiţi sunt:
  - dec - activează bitul dec (la intrare/ieşire)
  - hex - activează bitul hex (la intrare/ieşire)
  - oct - activează bitul oct (la intrare/ieşire)
  - showbase - afiseaza baza de numeratie
  - noshowbase – dezactiveaza afisarea bazei de numeratie
  - endl - inserează caracterul '\n' şi goleşte tamponul (la ieşire)
  - flush - goleşte tamponul (la ieşire)
  - setfill(char) - stabileşte caracterul de umplere;
  - setw(int) - stabileşte dimensiune câmpului pentru operaţia următoare

Manipulatori (cont)

- Manipulatorii definiţi sunt:
  - showpos - adaugă simbolul + in faţa valorilor pozitive
  - noshowpos - elimina simbolul + in faţa valorilor pozitive
  - left - aliniaza la stanga valoarea afisata
  - right - aliniaza la dreapta valoarea afisata
  - internal - aliniere implicita
  - setprecision(int) - stabileşte precizia pentru numere în virgulă mobilă;
  - fixed – afisare numere reale cu numar fix de zecimale
  - scientific – afisare numere reale in format stiintific

Exemplu

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

int main(){
  cout << 123 << endl ;
  cout << hex << showbase << 123 << endl ;
  cout << oct << showbase << 123 << endl ;
  cout << 1234.56 <<endl;
  cout << setprecision(2) << setw(10) <<setfill('*') <<
setiosflags(ios::fixed);
  cout << 1234.56 <<endl;
  return 0;
}
```

```
123
0x7b
0173
1234.56
***1234.56
```


Definirea de manipulatori

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

ostream& formatReal(ostream& out){
   out.width(10);
   out.precision(2);
   out.setf(ios::fixed);
   out.fill('*');
   return out;
}

int main(){
   cout << formatReal<< 1234.56 <<endl;
   return 0;
}
```

```
***1234.56
```


Supraîncărcarea operatorilor << si >>

- Se face numai cu funcţii friend
- Sintaxa

```
class IdClasa{
 ...
 friend istream& operator>>(istream& in, IdClasa& ob);
 friend ostream& operator<<(ostream& out, const IdClasa& ob);
};
```


Supraîncărcarea operatorilor << si >>

- Implementare

```
istream& operator>>(istream& in, IdClasa& ob){
  //citire date membre ale obiectului ob din input stream-ul in
  ...
  return in;
}

ostream& operator<<(ostream& out, const IdClasa& ob){
   //afisare date membru ale obiectului ob in output stream-ul out
  ...
  return out;
}
```

Exemplu

```
#include <iostream>

using namespace std;
class Complex{
  private:
   float re;
   float im;
  public:
   Complex(){
     re = 0;
     im = 0;
   }

  friend istream& operator>>(istream& in, Complex& z);
  friend ostream& operator<<(ostream& out, const Complex& z);
};
```


Exemplu (cont.)

```
istream& operator>>(istream& in, Complex& z){
  in>>z.re;
  in>>z.im;
  return in;
}

ostream& operator<<(ostream& out, const Complex& z){
   out<<z.re << std::showpos << z.im<<"*i";
   out<<std::noshowpos;
   return out;
}

int main(){
    Complex z;
    cin>>z;
    cout<<z;
}
```


Operații de intrare/ieșire cu fișiere

- Operațiile de intrare/ieșire cu fișiere se fac prin intermediul claselor
  - ifstream
  - ofstream
  - fstream

Operații de intrare/ieșire cu fișiere

- Constructori

ofstream::ofstream(const char* name,ios::openmode = ios::out | ios::trunc);
ifstream::ifstream(const char* name, ios::openmode = ios::in);
fstream::fstream(const char* name, ios::openmode = ios::in | ios::out);

Operații de intrare/ieșire cu fișiere

- Moduri de deschidere a fişierelor
  - *ios::append* - adauga la sfarsitul fişierului;
  - *ios::atend* - poziţioneaza pointer-ul la sfârşitul fişierului, însă informaţiile pot fi scrise oriunde în cadrul fişierului;
  - *ios::truncate* - este modul de deschide implicit: vechiul continut al fişierului este pierdut;
  - *ios::nocreate* - dacă fişierul nu exista, atunci operaţia eşuează;
  - *ios::noreplace* - dacă fişierul deja există, atunci operaţia eşuează;
  - *ios::binary* – deschide fișierul in mod binar.

Operații de intrare/ieșire cu fișiere

- Rezultatul operaţiilor de intrare/ieşire poate fi testat prin intermediul a patru functii membre:
  - *eof()* verifică dacă s-a ajuns la sfarsitul fişierului;
  - *bad()* verifică dacă s-a executat o operaţie invalidă;
  - *fail()* verifică dacă ultima operaţie a eşuat;
  - *good()* verifică dacă toate cele trei rezultate precedente sunt false.
- Închiderea unui fișier se face apelând funcția
  - *close()*

Exemplu

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

using namespace std;

int main(){
    int n=10, t;
    ofstream out("output.txt");
    for(int i=0;i<n;i++)
            out<<i*i<<endl;
    out.close();
    ifstream in("output.txt");
    if (!in){
      cerr<<"Eroare deschidere";
      exit(0);
    }
    while(!in.eof()){
     in>>t;
     if (in.good()){
        cout<<t<<endl;
     }
    }
    in.close();
    getch();
}
```


Operații de citire/scriere în mod binar

- Scrierea într-un fișier binar se face cu funcția

  ostream& write (const char* s , streamsize n);
- Citirea dintr-un fișier binar se face cu funcția

  istream& read (const char* s , streamsize n);

Exemplu

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

using namespace std;

int main(){
 int n=10, t;
 ofstream out("output.bin",
                  ios::binary);
 for(int i=0;i<n;i++)
     out.write(reinterpret_cast<char*>(&i),
                          sizeof(i));

out.close();
```

```
ifstream in("output.bin",ios::binary);
    if (!in){
      cerr<<"Eroare deschidere fisier";
      exit(0);
    }
    while(!in.eof()){
     in.read(reinterpret_cast<char*>(&t),
                                sizeof(t));
     if (in.good()){
        cout<<t<<endl;
     }
    }
    in.close();
    getch();
}
```

reinterpret_cast *convertește orice tip de pointer către orice alt tip de pointer, chiar și pentru pointer către clase care nu au legătură. Rezultatul conversiei este o copie binară simplă a valorii de la un pointer la altul. Sunt permise toate conversiile de pointer: nu este verificat nici conținutul indicat, nici tipul de pointer în sine.*

Poziționarea într-un fișier

- Obținerea poziției într-un stream asociat unui fișier se face cu:
  - pos_type ostream::tellp(); // pentru stream-urile de iesire
  - pos_type istream::tellg(); //pentru stream-urile de intrare
- Poziționarea într-un stream se face cu:
  - ostream& ostream::seekp(off_type offset,ios::seekdir pos); //pentru stream-urile de iesire
  - istream& istream::seekg(off_type offset,ios::seekdir pos); //pentru stream-urile de intrare

unde pos poate avea una din următoarele valori

- ios::beg – specifică poziţia de la începutul stream-ului;
- ios::end – specifică poziţia de la sfârşitul stream-ului;
- ios::cur – specifică poziţia relativ la locatia curentă în stream;