Curs 08

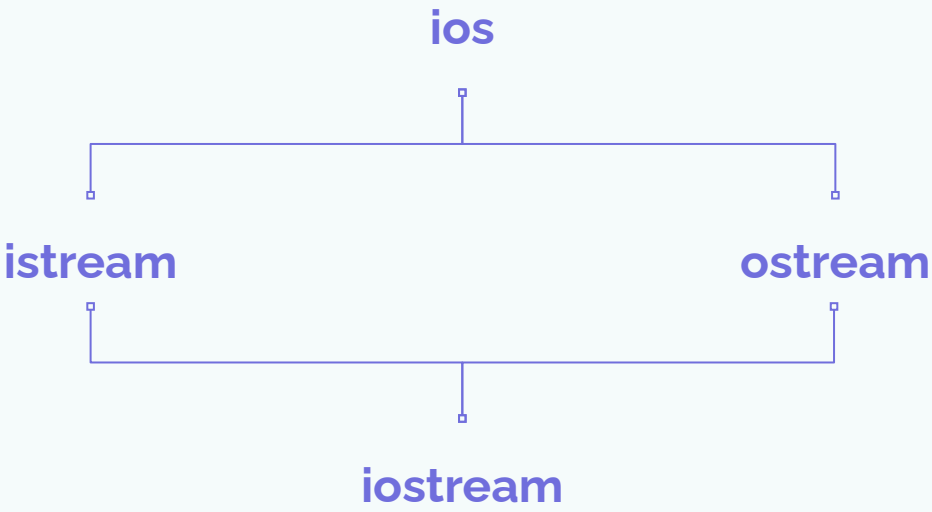
STREAM-URI

Operații de intrare-ieșire și lucrul cu fișiere

Stream

- Obiect ce permite gestionarea și/sau manipulare șirurilor de bytes
- Utilizate de obicei în C++ pentru formele supraîncărcate ale operatorilor **<<** și **>>**
- Obiecte standard existente:
  - **cout** - obiect de tip ostream (stream-ul standard de ieșire)
  - **cin** - obiect de tip istream (stream-ul standard de intrare)
  - **cerr** - obiect de tip ostream (asociat stream-ului standard de erori)

Relația dintre clasele stream standard



Exemplu

```
int main()
{
      int x = 0;
      cin >> x;
      cout << x;
      cerr << x + 1;
}
```

Datele din stream-uri pot fi formatate

- Modalități de formatare:
  - metode ale obiectelor **cin** și **cout** (ex: **width**, **fill**, etc.)
  - manipulatori (sunt prezenți în fișierul header **iomanip**)
  - flag-uri de formatare din clasa **ios** (setați prin intermediul metodei **ios::setiosflags**)
  - flag-uri de formate ale obiectelor **cin** și **cout** (setați prin metoda **setf**)

Manipulatori pe stream-uri standard

- dec
- hex
- oct
- setprecision(int)
- endl
- ends
- ws
- flush
- setbase(int)
- setfill
- setw(int)
- setiosflags(long)
- resetiosflags(long)

Exemplu

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

int main()
{
  int x = 15;
  cout << hex << x << endl;
  cout << oct << x << endl;
  float y = 5.123456;
  cout << setprecision(3) << y << endl;
  cout << setw(3)<<setfill('$') << dec << x << endl;
}
```

ios flags (folosiți cu setiosflags și resetiosflags)

- ios::left
- ios::right
- ios::internal
- ios::dec
- ios::hex
- ios::showpos
- ios::showbase
- ios::scientific
- ios::fixed
- ios::showpoint
- ios::skipws
- ios::stdio
- ios::uppercase
- ios::unitbuf

Flag-uri de formatare (folosite cu setf)

- ios::basefield
  - ios::dec
  - ios::hex
  - ios::oct
- ios::floatfield
  - ios::scientific
  - ios::fixed
- ios::adjustfield
  - ios::left
  - ios::right
  - ios::internal

Exemplu

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

int main()
{
	float x = 9.2;
	cout << hex;
	cout << setiosflags(ios::showbase | ios::showpoint);
	cout << x << endl;
	cout << resetiosflags(ios::showbase | ios::showpoint);
}
```

Citire șiruri de caractere cu cin

- La fel ca scanf și **cin** consideră în mod implicit spațiul drept terminator de șir
- Pentru a citi șiruri de caractere ce conțin spații se pot utiliza următoarele metode din clasa iostream:
  - get()
    - `istream& get (char& c);`
    - `istream& get (char* s, streamsize n);`
    - `istream& get (char* s, streamsize n, char delim);`
  - getline()
    - `istream& getline (char* s, streamsize n );`
    - `istream& getline (char* s, streamsize n, char delim );`

Exemplu

```
int main()
{
      char nume[31];
      cin.getline(nume, 30);
      char x;
      cin.get(x);
      cout << nume << endl;
      cout << x << endl;
}
```

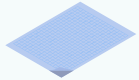
Detectare erori de citire/scriere

- În cazul în care citirea sau afișarea nu se efectuează cu succes, atunci biții de stare **failbit** și **badbit** sunt setați pe 1
- Pentru a testa valoarea biților de stare putem utiliza metodele:
  - good()
  - fail()
  - bad()

Exemplu

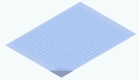
```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
  int x = 0;
  cout << "x = ";
  cin >> x;
  if(cin.fail())
  {
    cout << "The value is not an integer" << endl;
  }
}
```

Lucrul cu fișiere în C++



- Stream-uri prezente în fișierul header **fstream**
- Denumite și stream-uri/fișiere non-standard
- Lucrul în mod text este asemănător cu cel pe fișiere standard
- Lucrul în mod binar presupune serializarea bit-cu-bit a atributelor clasei

Clase utilizate

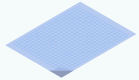


- ifstream - fișiere de intrare (mod citire)
- ofstream - fișiere de ieșire (mod scriere)
- fstream - fișiere de intrare-ieșire (mod citire-scriere)

Exemplu: deschidere fișier

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

int main()
{
  ifstream f("fisier1.txt");
  //sau
  ofstream g;
  g.open("fisier2.txt");
}
```

Modalități deschidere fișier

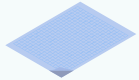


- ios::in - mod citire
- ios::out - mod scriere
- ios::ate - deschidere și poziționare la sfârșit
- ios::app - deschidere pentru adăugare
- ios::trunc - deschidere cu ștergere a conținutului existent
- ios::nocreate - deschide fișierul doar dacă există deja
- ios::noreplace - deschide fișierul doar dacă nu există deja
- ios::binary - deschide în mod binar

Exemplu: deschidere fișier

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

int main()
{
  ifstream f;
  f.open("fisier.bin", ios::in | ios::binary);
  ofstream g("fisier.txt", ios::out | ios::app);
}
```

Citire și scriere în mod text



- Folosind operatorii **<<** și **>>**
- Folosind metodele **get** și **put** pentru un singur caracter/șiruri de caractere
- Folosind metodele **getline** și **write** pentru șiruri de caractere ce conțin spații sau de lungime fixă
- Deoarece clasa **ifstream** este derivată din clasa **istream**, supraîncărcarea operatorului de citire standard pot fi refolosită pentru citire din fișiere text
- Deoarece clasa **ofstream** este derivată din clasa **ostream**, supraîncărcarea operatorului de scriere standard pot fi refolosită pentru scriere în fișiere text

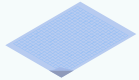
Exemplu: scriere în fișier text

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

int main()
{
  ofstream f;
  f.open("fisier.txt", ios::out | ios::trunc);
  int x = 5;
  float y = 3.5;
  f << x << endl;
  f << y << endl;
  string s = "ionel popescu";
  f << s << endl;
  f.close();
}
```

Exemplu: citire din fișier text

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;
int main()
{
  ifstream f;
  f.open("fisier.txt", ios::in);
  int x = 0;
  float y = 0;
  string s;
  f >> x;
  f >> y;
  getline(f, s);
  cout << x << y << s << endl;
  f.close();
}
```

Citire și scriere în mod binar



- Nu există funcții specifice în C++ pentru a serializa tipuri de date complexe
- Serializarea se face prin utilizarea metodei **write()** ce poate scrie un vector de caractere de lungime fixă
- Deserializarea se face prin utilizarea metodei **read()** ce poate citi un vector de caractere de lungime fixă
- Toate celelalte tipuri de date (cu excepția char*) vor fi convertite prin intermediul operatorului de cast explicit la char* pentru a putea fi scrise/citite binar

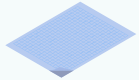
Exemplu: scriere binară în fișier

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  ofstream f("fisier.bin", ios::out | ios::binary);
  int x = 5;
  float y = 3.5;
  string s = "Ion Popescu";
  f.write((char*)&x, sizeof(x));
  f.write((char*)&y, sizeof(y));
  int length = s.length();
  f.write((char*)&length, sizeof(length));
  f.write(s.c_str(), length + 1);
  f.close();
}
```

Exemplu: citire binară din fișier

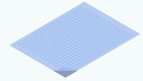
```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  ifstream g("fisier.bin", ios::in | ios::binary);
  int x = 0;
  float y = 0;
  string s;
  g.read((char*)&x, sizeof(x));
  g.read((char*)&y, sizeof(y));
  int length = 0;
  g.read((char*)&length, sizeof(length));
  char* sir = new char[length + 1];
  g.read(sir, length + 1);
  s = sir;
  delete[] sir;
  g.close();
  cout << x << y << s << endl;
}
```

Alte metode utile



- eof() - returnează true dacă s-a ajuns la sfârșitul fișierului
- seekg() - poziționare relativă/absolută în fișiere de intrare
- seekp() - poziționare relativă/asbolută în fișiere de ieșire
- Poziționarea relativă se face față de ios::beg, ios::cur sau ios::end
- tellg() - poziționarea cursorului pentru fișiere de intrare
- tellp() - poziționarea cursorului pentru fișiere de ieșire

Tipuri de organizare a informației în fișiere



- Fișiere secvențiale
- Fișiere cu acces direct
- Fișiere indexate
- Fișiere de tip invers