Operatii de intrare/iesire. Fluxuri.

- Fluxuri (stream-uri) si operatiile de intrare/iesire (I/O)
- Manipulatori de fluxuri
- Citirea si scrierea fisierelor text

Fluxuri.

- C++ utilizeaza fluxurile ca o interfata intre program si dispozitivele de intrare/iesire (citire/afisare).
- Fluxurile sunt independente de dispozitivul actual folosit.
- **Fluxurile (streams)** reprezintă o modalitate de a gestiona transferul de date (sub formă de octeți sau obiecte) între diferite surse și destinații. Mai concret:

1. Abstracție de nivel înalt: Un flux este o abstracție care ascunde detaliile tehnice ale dispozitivului cu care lucrăm (fie el fișier pe disc, consolă, rețea sau altă sursă/destinație de date)
2. Secvență de date: Datele sunt tratate ca o succesiune (șir) de octeți sau caractere, care pot fi citite (input stream) sau scrise (output stream).
3.Gestionare simplificată a I/O: Prin utilizarea fluxurilor, programul nostru nu trebuie să știe exact cum sunt stocate datele în dispozitivul fizic și nici detalii legate de protocoalele de transfer; interacțiunea devine mai „universală”.

Cele mai frecvente fisiere header

- **iostream** → asigura definitiile pentru operatiile de intrare/iesire (citire/afisare) ale fluxurilor (in consola);
- **fstream** → asigura definitiile pentru operatiile de intrare/iesire ale fluxurilor de fisiere;
- **iomanip** → asigura defintiile manipulatorilor folositi pentru operatiile de intrare/iesire

- Ele permit lucrul cu fluxuri si operatiile de intrare/iesire

Cele mai frecvente clase de fluxuri

Clasa	Descriere
ios	Ofera suport de baza pentru operatiile I/O formatate si neformatate
Ifstream	Ofera operatii de **intrare** la nivel inalt pentru fluxuri de fisiere
ofstream	Ofera operatii de **iesire** la nivel inalt pentru fluxuri de fisiere
fstream	Ofera operatii **I/O** la nivel inalt pentru fluxuri de fisiere
stringstream	Ofera operatii de I/O la nivel inalt pentru pentru striguri stocate in memorie

Obiecte de fluxuri globale

Obiect	Descriere
cin	Flux de intrare standard – “conectat” implicit la dispozitivul de intrare standard (tastatura);
cout	Flux de iesire standard – “conectat” implicit la dispozitivul de iesire standard (consola);
cerr	Flux de erori standard – “conectat” implicit la dispozitivul de standard de afisare a erorii (consola);
clog	Flux de tip log standard – “conectat” implicit la dispozitivul de afisare standard (consola); (avertismente, mesaje de stare, de performanta s.a.)

- Obiecte globale – initializate inainte de executia functiei main();
- Sunt incluse in fisierul header **iostream,** in clasa **ios**

Manipulatori de fluxuri

- Fluxurile au multe proprietati care ajuta la formatarea textului
- Manipulatorii de fluxuri pot fi folositi pentru fluxurile I/O
- Timpul efectului manipulatorilor de fluxuri variaza (poate afecta doar urmatorul flux sau toate fluxurile ce urmeaza in program)
- Putem folosi proprietatile de formatare a textului fie ca functii, fie ca manipulatori (noi ne vom axa pe folosirea manipulatorilor)

Exemplu:

```
std::cout.width(10); //functie
std::cout<<std::setw(10); //manipulator
```

Cei mai frecventi manipulatori de fluxuri

- **Pentru tipul de date bool** – boolapha,noboolapha (in loc de 1/0 se afiseaza true/false)
- **Pentru tipul de date int** – dec, hex, showbase,noshowbase, showpos, noshowpos, uppercase,nouppercase
- **Pentru tipul de date float/double –** fixed, scientific, setprecision, showpoint,noshowpoint, showpos,noshowpos
- **Pentru latime, aliniere si umplerea campurilor –** setw,left,right,internal,setfill

Pentru tipul de date bool

```
#include<iostream>

using namespace std;

int main() {

bool ok=true;
cout << "Valoarea lui ok este :" << ok << endl;
cout <<boolalpha;
cout << "Valoarea lui ok este :" << ok << endl;
ok = false;
cout << "Valoarea lui ok este :" << ok << endl;
cout <<noboolalpha;
cout << "Valoarea lui ok este :" << ok << endl;
return 0;
}
```

1

true

false

0

Pentru tipul de date int

- Implicit, cand afisam un tip de date intreg avem:
  - **dec** (baza 10)
  - **noshowbase** (nu se afiseaza prefixul pentru sistemul hexazecimal sau octal)
  - **nouppercase** (daca afisam prefixul pentru valorile hexazecimale ele vor fi litere mici)
  - **noshowpos** (semnul “+” nu este afisat in fata numerelor pozitive)

- Acesti manipulatori au efect pentru toate afisarile urmatoare din flux

Pentru tipul de date int

```
#include<iostream>
using namespace std;
int main() {

int num{ 255 }, num2{135};
cout << "Implicit: "<< num << endl; //255
cout << "Zecimal: "<< dec << num << endl; //255
cout << "Hexazecimal: " << hex << num << endl; //ff
cout << "Octal: " << oct << num << endl; //377
cout << num2<<endl;     //207
return 0;
}
```

Daca nu se precizeza explicit baza in care vrem sa apara numerele de tip intreg, baza folosita ultima data se va folosi in continuare la afisarea tuturor numerelor intregi din continuare programului!

Pentru tipul de date int

```
#include<iostream>
using namespace std;
int main() {

int num{ 255 }, num2{135};
cout << showbase << num << endl; //255
cout << "Zecimal: "<< dec << num << endl; //255
cout << "Hexazecimal: " << hex << num << endl; //0xff
cout << "Octal: " << oct << num << endl; //0377
cout << num2<<endl; //0207
cout << noshowbase << num2; //207
return 0;
}
```

- Prefixul pentru hexazecimal este “0x”, iar pentru octal este “0”.

Pentru tipul de date int

```
#include<iostream>
using namespace std;
int main() {

int num{ 255 }, num2{135};
cout << showbase << uppercase << num << endl; //255
cout << "Zecimal: "<< dec << num << endl; //255
cout << "Hexazecimal: " << hex << num << endl; //0XFF
cout << "Octal: " << oct << num << endl; //0377
cout << num2<<endl; //0207
cout << nouppercase << hex<<num<<endl; //0xff
return 0;
}
```

- Prefixul si numerele in baza 16 sunt scrise cu litere majuscule.

Pentru tipul de date int

```
#include<iostream>
using namespace std;
int main() {

int num1{ 255 }, num2{-255};
cout << "Implicit: " << endl;
cout << num1 << endl; //255
cout << num2 << endl; //-255
cout << showpos;
cout << num1 << endl; //+255
cout << num2 << endl; //-255
cout << noshowpos;
cout << num1 << endl;  //255
cout << num2 << endl; //-255
return 0;
}
```

Pentru tipul de date real

- Implicit, cand afisam un tip de date real (float, double) avem:
  - **setprecision** (numarul de cifre afisate 6)
  - **fixed** (nu este fixat un anumit numar de cifre dupa virgula)
  - **noshowpoint** (daca dupa virgula avem cifra/cifrele “0” aceasta/acestea nu se afiseaza)
  - **noshowpos** (semnul “+” nu este afisat in fata numerelor pozitive)

- Acesti manipulatori au efect pentru toate afisarile urmatoare din flux

Pentru tipul de date real

```
#include<iostream>
using namespace std;
int main() {
double num{ 1234.5678 };
cout << num << endl;  //1234.57
return 0;
}
```

- **Implicit**, C++ afiseaza **6 cifre** pentru un numar real, incepand cu cifra cea mai semnificativa (prima cifra), si o rotunjeste pe ultima cifra
(daca a 7 cifra este >=5 atunci a 6 a cifra o rotunjeste prin adaos, in caz contrar a 6 a cifra este rotunjita prin lipsa).

Pentru tipul de date real

```
#include<iostream>
using namespace std;
int main() {
double num{ 123456789.987654321 };
cout << num << endl; // 1.23457e+08
return 0;
}
```

- C++ incearca sa faca afisarea obisnuita cu 6 cifre, dar pentru ca numarul dat si numarul afisat nu au cum sa fie “aproximativ” egale, atunci C++ foloseste **scrierea stiintifica** (se pastreaza precizia de 6, adica primele 6 cifre sunt afisate cu rotunjirea ultimei cifre).

Pentru tipul de date real

```
#include<iostream>
#include<iomanip>

using namespace std;
int main() {
double num{ 123456789.987654321 };
cout << setprecision(9); //123456790
cout << num << endl;
return 0;
}
```

- Atentie! Se afiseaza primele 9 cifre si se rotunjeste rezultatul

Pentru tipul de date real

```
#include<iostream>
#include<iomanip>

using namespace std;
int main() {
double num{ 123456789.987654321 };
cout << fixed;
cout << num << endl;//123456789.987654
return 0;
}
```

- Atentie! Se afiseaza cu **6 cifre zecimale** si se rotunjeste ultima zecimala

Pentru tipul de date real

```
#include<iostream>
#include<iomanip>

using namespace std;
int main() {
double num{ 123456789.987654321 };
cout << setprecision(3) << fixed;
cout << num << endl; //123456789.988
double num2{ 987654321.123456789 };
cout << num2 << endl; //987654321.123
return 0;
}
```

- Se afiseaza cu **3 cifre zecimale** si se rotunjeste ultima zecimala

Pentru tipul de date real

```
#include<iostream>
#include<iomanip>

using namespace std;
int main() {
double num{123456789.987654321 };
cout << scientific<<setprecision(4);
cout << num << endl; // 1.235e+08
return 0;
}
```

- Se afiseaza in forma stiintifica, pastrand primele **3 cifre** si rotunjind-o pe a 4a cifra

Pentru tipul de date real

```
#include<iostream>
#include<iomanip>

using namespace std;
int main() {
double num{123456789.987654321 };
cout <<scientific<<setprecision(4) <<uppercase;
cout << num << endl;                                    // 1.235E+08
return 0;
}
```

- Se afiseaza in forma stiintifica, pastrand primele **3 cifre** si rotunjind-o pe a 4a cifra, litera “e” devine “E”

Pentru tipul de date real

```
#include<iostream>
#include<iomanip>

using namespace std;
int main() {
double num{ 123456789.987654321 };
cout <<fixed<<setprecision(3)<<showpos;
cout << num << endl;                                    //+123456789.988
return 0;
}
```

- Se afiseaza semnul “+” pentru numerele pozitive

Pentru tipul de date real

```
#include<iostream>
#include<iomanip>

using namespace std;
int main() {
double num{ 12.34 };
cout << num << endl;   //12.34
cout << showpoint;
cout << num << endl; //12.3400
num = 12.3;
cout << num << endl; //12.3000
cout << setprecision(8);
cout << num << endl; //12.300000
return 0;
}
```

- Afiseaza zerouri pentru zecimale pana se ajunge la precizia stabilita; implicit, precizia este 6, ulterior am modificat-o in 8.

Pentru latime, aliniere si umplerea campurilor

- Implicit, atunci cand se afiseaza un anumit tip de date, avem:
  - **setw** (latimea campului alocat tipului de date nu este setata implicit)
  - **left** (cand nu este precizata latimea campului), **right** (cand este precizata latimea campului)
  - **fill** (nu este setata implicit), este folosit spatiul gol.

- Unii din acesti manipulatori au efect doar pentru urmatorul element ce apare in flux

Pentru latime, aliniere si umplerea campurilor

- Implicit,

```
#include<iostream>
#include<string>

using namespace std;
int main() {
double num{ 1234.5678 };
string text{ "Buna" };
cout << num << text << endl;   //1234.57Buna
return 0;
}
```

Pentru latime, aliniere si umplerea campurilor

```
#include<iostream>
#include<string>
#include<iomanip>

using namespace std;
int main() {
double num{ 1234.5678 };
string text{ "Buna" };
cout << "12345678901234567890123456789" << endl;
cout << setw(10)<<num << text << endl;
return 0;
}
```

```
//12345678901234567890123456789
    1234.57Buna
```

- Se seteaza latimea campului de (cel putin) 10 caractere (alinierea implicita acum este la **dreapta**, observam ca “7” este sub “0”); Dacă textul/numărul are mai puțin de 10 caractere, se vor folosi spații libere în stânga pentru a completa. Dacă are mai mult, câmpul se va ajusta, afișând întregul conținut, dar pierzând eventualul efectul de aliniere.
- Urmatoroarele elemente afisate vor fi aliniate, implicit, la stanga, deoarece **latimea campului nu este stabilit si pentru acestea.**

Pentru latime, aliniere si umplerea campurilor

```
#include<iostream>
#include<string>
#include<iomanip>                    //12345678901234567890123456789012345678901234567890
                                          1234.57      Buna      Buna


using namespace std;
int main() {
double num{ 1234.5678 };
string text{ "Buna" };
cout << "12345678901234567890123456789012345678901234567890" << endl;
cout << setw(10) << num << setw(10) << text << setw(10)<<text << endl;
return 0;
}
```

Pentru latime, aliniere si umplerea campurilor

```
#include<iostream>
#include<string>
#include<iomanip>                    //12345678901234567890123456789012345678901234567890
                                       1234.57    Buna


using namespace std;
int main() {
double num{ 1234.5678 };
string text{ "Buna" };
cout << "12345678901234567890123456789012345678901234567890" << endl;
cout << setw(10)<<left << num<<setw(10) << text << endl;
return 0;
}
```

- Alinierea se pastreaza si pentru celelalte cantitati afisate!

Pentru latime, aliniere si umplerea campurilor

```
#include<iostream>
#include<string>
#include<iomanip>                    //12345678901234567890123456789012345678901234567890
                                       ---1234.57------Buna


using namespace std;
int main() {
double num{ 1234.5678 };
string text{ "Buna" };
cout << "12345678901234567890123456789012345678901234567890" << endl;
cout << setfill('-');
cout << setw(10) << num<< setw(10)<<text << endl;
return 0;
}
```

- Setarea de umplere a campului liber cu un anumit caracter ramane valabila si pentru urmatoarele fluxuri.

Pentru latime, aliniere si umplerea campurilor

```
#include<iostream>
#include<string>
#include<iomanip>                    //1234567890123456789012345678901234567890
                                       ---1234.57******Buna


using namespace std;
int main() {
double num{ 1234.5678 };
string text{ "Buna" };
cout << "1234567890123456789012345678901234567890" << endl;
cout << setfill('-');
cout << setw(10) << num<< setfill('*')<<setw(10) << text << endl;
return 0;
}
```

- Putem schimba pe parcursul programului caracterul cu care umplem campul liber.

Challenge Problem

- Realizati un program in care sa folositi urmatoarele structuri si folositi manipulatorii ca afisarea sa fie urmatoarea:

```
#include<iostream>
#include<string>
#include<iomanip>
#include<vector>

struct Oras {
string nume;
long popoulatie;
double cost;
};

struct Tara {
string nume;
vector<Oras> orase;
};

struct Tur {
string titlu;
vector<Tara> tari;
};
```

```
Tara        Oras              Populatie    Pret
-------------------------------------------------
Colombia    Bogota             8773000   400.98
            Cali               2491000   424.12
            Medellin           2464000   350.98
            Cartagena           972000   345.34

Brazilia    Rio De Janeiro    13500000   567.45
            Sao Paulo         11310000   975.45
            Salvador          18324000   855.99

Chile       Valdivia            260000   569.12
            Santiago           7840000   520.00

Argentina   Buenos Aires       3010000   723.77
```

Operatii de citire

- Pentru a putea citi din fisiere, cel mai des se folosesc fisierele header **fstream** si **ifstream**

1. Adaugam directiva #include <fstream>
2. Declaram un obiect de tipul fstream sau ifstream
3. Conectam fisierul cu stream-ul creat
4. Citim datele din fisier cu ajutorul stream-ului
5. Inchidem stream-ul

Citirea fisierelor cu fstream

- **Declaram un obiect de tipul fstream**
  - std::fstream in_file {“test.txt”, std::ios::in};
    - Pe langa declararea obiectului (cu rosu) am facut si initializarea lui care contine 2 parametrii (cu albastru); prin intermediul primului parametru dam calea catre fisier
      - Poate fi foarte specifica in functie de sistemul de operare si chiar de programul folosit
      - Cu “..” urcam un nivel mai sus

      Al doilea parametru specifica modul de deschidere al fisierului. Prin argumentul dat, specificam ca deschidem fisierul in “**input mode**” (putem sa citim fisierul, dar nu putem scrie in el).

      Modul default de deschidere a unui obiect de tip fstream este std::ios::in | std:ios::out, deci si **pentru citire si pentru scriere (**“|” operatorul bitwise OR pe biti**)**; Daca dorim acest lucru, atunci putem face simplu declararea std::fstream in_file {“test.txt”};

      Pentru a citi un document care nu este text (**imagine,** de exemplu), al doilea argument ar trebui sa fie in modul input si binar `std::ios:in | std::ios::binary`

Citirea fisierelor cu fstream

Pentru a deschide un fisier din care dorim sa citim putem proceda in urmatorul mod

```
#include<iostream>
#include<fstream>
#include<string>

using namespace std;

int main() {
//fstream in_file("test.txt", ios::in);
fstream in_file;
string file_name;
cout << "Dati numele fisierului din care vreti sa cititi:";
cin >> file_name;
in_file.open(file_name, ios::in);
if (in_file.is_open()) {
cout << "Fisierul este deschis cu succes" << endl;
}
else {
cerr << "Eroare la deschidere" << endl;
}

return 0;
}
```

Citirea fisierelor cu ifstream

- **Declaram un obiect de tipul ifstream**
  - std::ifstream in_file {“test.txt”, std::ios::in};
    - ifstream poate fi folosit doar pentru **modul de citire**
    - Din moment ce std::ios::in este modul implicit, acest argument este optional, prin urmare putem face declararea cu:
      - std::ifstream in_file {“test.txt”}

Pentru a citi un document care nu este text (**imagine** de exemplu), al doilea argument ar trebui sa fie in modul input si binar; cum modul input este implicit, ar trebui doar mentionat modul binar `std::ios::binary`

Citirea fisierelor cu ifstream

Pentru a deschide un fisier din care dorim sa citim putem proceda in urmatorul mod

```
#include<iostream>
#include<fstream>
#include<string>

using namespace std;

int main() {
//ifstream in_file("test.txt");
ifstream in_file;
string file_name;
cout << "Dati numele fisierului din care vreti sa cititi:";
cin >> file_name;
in_file.open(file_name);
if (in_file.is_open()) {
cout << "Fisierul este deschis cu succes" << endl;
}
else {
cerr << "Eroare la deschidere" << endl;
}

return 0;
}
```

Citirea fisierelor

O alta modalitate de verifica daca un fisier s-a deschis cu succes este

```
#include<iostream>
#include<fstream>
#include<string>

using namespace std;

int main() {
//ifstream in_file("test.txt");
ifstream in_file;
string file_name;
cout << "Dati numele fisierului din care vreti sa cititi:";
cin >> file_name;
in_file.open(file_name);
if (in_file) {
cout << "Fisierul este deschis cu succes" << endl;
}
else {
cerr << "Eroare la deschidere" << endl;
}

return 0;
}
```

Citirea fisierelor

Dupa ce terminam de citit din fisier, trebuie sa **inchidem** fisierul. Daca in cazul fisierelor din care citim, inchiderea lor dupa ce terminam de citit din ele nu este obligatorie, in cazul fisierelor in care scriem, inchiderea lor este esentiala; omitand acest lucru se poate ajunge la pierderea informatiilor.

```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;
int main() {
//ifstream in_file("test.txt");
ifstream in_file;
string file_name;
cout << "Dati numele fisierului din care vreti sa
cititi:";
cin >> file_name;
in_file.open(file_name);
if (in_file) {
cout << "Fisierul este deschis cu succes" << endl;
}
else {
cerr << "Eroare la deschidere" << endl;
}
in_file.close();
return 0;
}
```

Modalitati de citire din fisiere txt

- Cu ajutorul operatorului de extractie “>>” (asa cum se foloseste la cin)

```
int num{};
double total{};
string name{};

in_file >> num;
in_file >> total >> name;
```

```
100
255.67
Simona
```

Atentie! In fisier trebuie sa fie scrise in ordine: un numar intreg, un numar real, apoi un cuvant. Pot fi scrise pe linii diferite sau pe aceesi linii, separate de un spatiu sau mai multe spatii libere.

Modalitati de citire din fisiere txt

- Cu ajutorul lui `getline` (cand dorim sa citim cate o linie o data)

```
string line{};
getline(in_file, line);
```

100 255.67
Simona

Daca scriem apoi
`cout<<line;`
se va afisa “100 255.67”

Modalitati de citire din fisiere txt

- Cu ajutorul lui `getline` (cand dorim sa citim cate o linie o data) putem afisa tot continutul fisierului – **Metoda 1**

```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;
int main() {
//ifstream in_file("test.txt");
ifstream in_file;
string file_name;
cout << "Dati numele fisierului din care vreti sa cititi:";
cin >> file_name;
in_file.open(file_name);
if(!in_file) {
cerr << "Eroare la deschidere" << endl;
return 1;
}
while (in_file) {
string line{};
getline(in_file, line);
cout <<line << endl;
}
in_file.close();
return 0;
}
```

100 255.67
Simona

Modalitati de citire din fisiere txt

- Cu ajutorul lui `getline` (cand dorim sa citim cate o linie o data) putem afisa tot continutul fisierului – **Metoda 2**

```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;
int main() {
//ifstream in_file("test.txt");
ifstream in_file;
string file_name;
cout << "Dati numele fisierului din care vreti sa cititi:";
cin >> file_name;
in_file.open(file_name);
if(!in_file) {
cerr << "Eroare la deschidere" << endl;
return 1;
}
string line;
while (getline(in_file, line)) {
cout <<line << endl;
}
in_file.close();
return 0;
}
```

100 255.67
Simona

Modalitati de citire din fisiere txt

- Cu ajutorul lui get(cand dorim sa citim cate un caracter o data) putem afisa tot continutul fisierului

```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;
int main() {
//ifstream in_file("test.txt");
ifstream in_file;
string file_name;
cout << "Dati numele fisierului din care vreti sa cititi:";
cin >> file_name;
in_file.open(file_name);
if(!in_file) {
cerr << "Eroare la deschidere" << endl;
return 1;
}
char c;
while (in_file.get(c)) {
cout <<c;
}
in_file.close();
return 0;
}
```

100 255.67
Simona

Scrierea in fisiere

- Cele mai frecvente fisiere header folosite sunt fstream si ofstream

1. Adaugam directiva `#include <fstream>`
2. Declaram un obiect de tipul fstream sau ofstream
3. Conectam fisierul cu stream-ul creat (daca fisierul nu exista, va fi creat. Daca exista, continutul lui va fi sters in cazul in care nu specificam altceva)
4. Scriem datele din fisier cu ajutorul stream-ului (in mod text sau binar)
5. Inchidem stream-ul (foarte important deoarece in acest moment fisierul este salvat)

- Fisierele vor fi suprascrise daca se foloseste modul default
- Fisierele pot fi deschise pentru a adauga continut la continutul existent
- Fisierele pot fi deschise in mod text si binar

Deschiderea unui fisier

- std::**fstream** out_file {“test..txt”, **std::ios::out**} //mod text

- std::**fstream** out_file {“test.jpg”, **std::ios::out | std::ios::binary**} //mod binar
- std::**ofstream** out_file {“test.txt”, **std::ios::out**}

Din moment ce ofstream foloseste modul **out** ca implicit putem sa nu mai adaugam ultimul parametru:

std::**ofstream** out_file {“test.txt”}
- std::**ofstream** out_file {“test.jpg”, **std::ios::binary**} //mod binar

Tipuri de deschidere a fisierelor pentru scriere

// stergem continutul fisierului in momentul deschiderii
std::ofstream out_file{“test.txt”, **std::ios::trunc**}

// adaugam text la continutul existent
std::ofstream out_file {“test.txt”, **std::ios::app**}

// deschidem fisierul si setam cursorul la finalul fisierului
std::ofstream out_file {“test.txt”, **std:ios::ate**};

Scrierea unui text de la tastatura in fisier

```
std::ofstream out_file;
std::string file_name;
std::cin >> file_name; //user-ul va tasta calea fisierului
out_file.open (file_name);
```

Verificam daca fisierul a fost deschis cu succes

```
if (out_file.is_open() ) {
     // scriem in fisier
} else {
     // deschiderea nu a fost realizata cu succes
}
```

Alta metoda de verificare daca fisierul a fost deschis cu succes

```
if (out_file) { //stream-ul va returna true daca fisierul a fost deschis
     // scriem in fisier
} else {
     // fisierul nu a fost deschis cu succes
}
```

Inchiderea fisierului

- Pentru modul output este critic deoarece atunci continutul fisierului este salvat.

```
out_file.close();
```

Scrierea fisierelor

Dorim sa obtinem fisierul ce contine un integer, un double si un string:
100
255.67
Simona

- Folosim operatorul de insertie "<<" (se foloseste ca si la cout);

```
#include<iostream>
#include<fstream>
#include<string>

using namespace std;

int main() {
ofstream out_file;
string file_name;
cout << "Dati numele fisierului in care vreti sa scrieti:";
cin >> file_name;
out_file.open(file_name);
if(!out_file) {
cerr << "Eroare la deschidere" << endl;
return 1;
}
else{
int num{ 100 };
double total{ 255.67 };
string name{"Simona"};
out_file << num << "\n" << total << "\n" << name << endl;
}
out_file.close();
return 0;
}
```

Copierea continutului unui fisier in alt fisier

```
#include<iostream>
#include<fstream>
#include<string>

using namespace std;

int main() {
ifstream in_file("test.txt");
ofstream out_file("copie_test.txt");
if(!in_file) {
cerr << "Eroare la deschiderea fisierului test.txt" << endl;
return 1;
}
if(!out_file) {
cerr << "Eroare la crearea fisierului copie_test.txt" << endl;
return 1;
}
string line{};
while (getline(in_file, line))
out_file << line << endl;
in_file.close();
out_file.close();
return 0;
}
```

Copierea continutului unui fisier in alt fisier caracter cu caracter (get/put)

```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;
int main() {
ifstream in_file("test.txt");
ofstream out_file("copie_test.txt");
if(!in_file) {
cerr << "Eroare la deschiderea fisierului test.txt" << endl;
return 1;
}
if(!out_file) {
cerr << "Eroare la crearea fisierului copie_test.txt" << endl;
return 1;
}
char c;
while (in_file.get(c))
out_file.put(c);
in_file.close();
out_file.close();
return 0;
}
```