


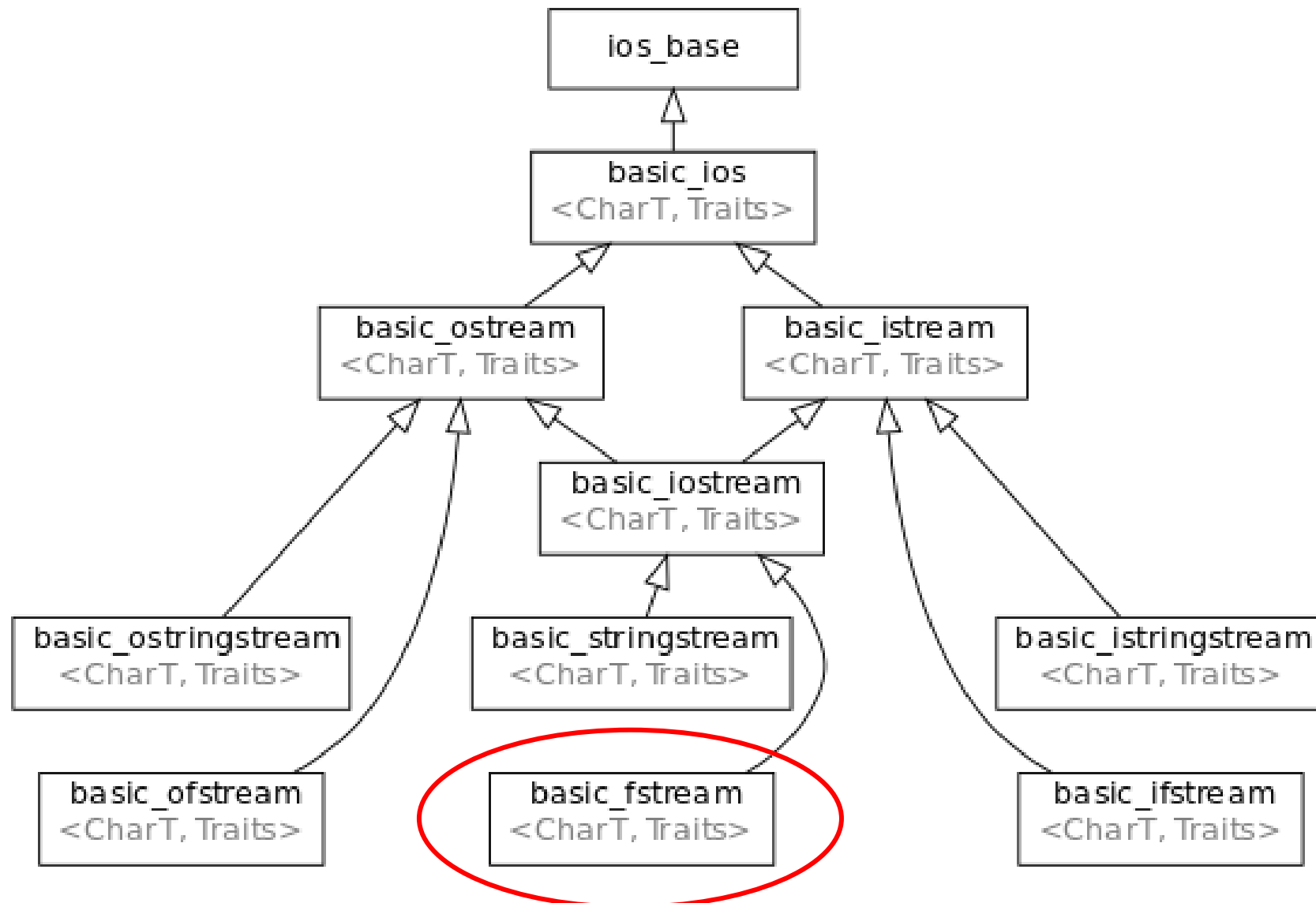
# Capitolul 7. Lucrul cu fișiere

# Fluxurile de intrare/ieșire

 Flux de intrare/ieșire = un debit de date care „curge” spre/dinspre program; o punte de legătură între diverse medii (program – consolă (afișare); consolă (citire) – program, program – fișier, etc.)

De obicei, fluxurile de date sunt secvențe de caractere, însoțite de funcții specifice.

# Fluxurile de intrare/ieșire



# std::basic\_fstream



# std::basic\_fstream

 **def** std::basic\_fstream = clasă template care implementează operații de nivel înalt pentru fluxuri de date bazate pe fișiere.

În practică, se folosește un alias al ei:

```
typedef basic_fstream<char, char_traits<char>> fstream;
```

# `fstream` – funcții de bază

## 1. Deschiderea fișierelor

- folosind funcția `open()` - funcție ce deschide un fișier și îl asociază `fstream`-ului din care este apelată.

```
void open(const char *filename,  
         ios_base::openmode mode = ios_base::in|ios_base::out);
```

- folosind constructorul `fstream` cu cel puțin un argument;

În cazul în care nu se reușește deschiderea fișierului, se setează fanionul `failbit` al `fstream`-ului.

# fstream – funcții de bază

## Crearea unui `fstream v1` – fără a-l asocia unui fișier:

```
basic_fstream();
```

### Exemplu:

```
std::fstream f1;  
f1.open("new_file.txt", std::fstream::out);
```

## Crearea unui `fstream v2` – asociindu-i unui fișier:

```
explicit basic_fstream(const char* filename,  
std::ios_base::openmode mode =  
ios_base::in|ios_base::out);
```

### Exemplu:

```
std::fstream f2("new_file.txt", std::fstream::out);
```

# `fstream` – funcții de bază

`openmode` specifică modul în care va fi deschis fișierul:

- `app` – se poziționează la finalul fișierului înainte de fiecare scriere
- `binary` – deschide fișierul în modul binar
- `in` – deschide fișierul în modul citire
- `out` – deschide fișierul în mod scriere
- `trunc` – șterge conținutul fișierului la deschidere (dacă acesta există)
- `ate` – deschide fișierul și se poziționează la sfârșitul său (at end)



# `fstream` – funcții de bază

2. Închiderea fișierelor folosind funcția `close()` - funcție ce închide un fișier. Această funcție este apelată implicit de destructorul `fstream`-ului la încheierea domeniului de vizibilitate. De obicei, nu se apelează explicit.

```
void close();
```

În cazul în care nu se reușește închiderea fișierului, se setează fanionul `failbit` al `fstream`-ului.

# `fstream` – funcții de bază

3. Verificarea dacă un `fstream` are un fișier asociat.

```
bool is_open();
```

# `fstream` – funcții de bază

## 4. Verificarea stării unui `fstream`:

`bool good() const;` - verifică dacă nu s-a produs nicio eroare

`bool eof() const;` - verifică dacă s-a ajuns la finalul fișierului (End Of File)

`bool fail() const;` - verifică dacă s-a produs o eroare în stream-ul asociat. În caz afirmativ, fluxul poate fi utilizat în continuare. Este echivalentă cu `bool operator!() const;`

`bool bad() const;` - verifică dacă s-a produs o eroare fatală în stream-ul asociat. În caz afirmativ, fluxul nu mai poate fi utilizat.

# fstream – funcții de bază

## 5. Citire/scriere formatată.

```
basic_ostream& operator>> ();
```

```
basic_ostream& operator<< ();
```

Acești operatori pot fi supraîncărcați pentru clasele definite de către utilizator la fel ca în cazul operatorilor >> și << care acționează asupra consolei (`std::cin`, `std::cout`).

# fstream – funcții de bază

## 6. Citire/scriere neformatată.

```
basic_istream& get( char_type* s, std::streamsize count,  
char_type delim );
```

```
basic_istream& getline( char_type* s, std::streamsize  
count, char_type delim );
```

În cazul în care se întâlnește delimitatorul în stream-ul citit, funcția `getline()` extrage caracterul respectiv, însă nu îl stochează. Funcția `get()` nu extrage delimitatorul din stream-ul citit.

```
basic_ostream& put( char_type ch );
```

# `fstream`

Modul general de lucru:

1. declarăm un `fstream`
2. atașăm `fstream` la un fișier  $\Leftrightarrow$  deschidem fișierul
3. verificăm că fișierul a fost deschis cu succes
4. procesăm fișierul
5. închidem fișierul

# fstream - exemple

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

int main(){
    std::fstream myfile;
    myfile.open ("example.txt", std::fstream::out);
    if (!myfile.is_open()){
        std::cout << "Eroare la deschiderea fisierului.";
    } else {
        myfile << "Mesaj 1 de test.\n";
        myfile.close();
    }
    return 0;
}
```

example.txt

Mesaj 1 de test.

# fstream - exemple

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

int main(){
    std::fstream myfile;
    myfile.open ("example.txt", std::fstream::out std::fstream::app);
    if (!myfile.is_open()){
        std::cout << "Eroare la deschiderea fisierului.";
    } else {
        myfile << "Mesaj 2 de test." << std::endl;
        myfile << "Inca un mesaj de test." << std::endl;
        myfile.close();
    }
    return 0;
}
```

example.txt

Mesaj 1 de test.  
Mesaj 2 de test.  
Inca un mesaj de test.



# fstream - exemple

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

int main(){
    std::fstream myfile;
    myfile.open ("example.txt", std::fstream::out std::fstream::trunc);
    if (!myfile.is_open()){
        std::cout << "Eroare la deschiderea fisierului.";
    } else {
        myfile << "Un mesaj nou." << std::endl;
        myfile << "Inca un mesaj de test." << std::endl;
        myfile.close();
    }
    return 0;
}
```

example.txt

Un mesaj nou.  
Inca un mesaj de test.

# fstream - example

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

int main() {
    int a, b;
    std::fstream myfile;
    myfile.open ("example.txt", std::fstream::in);
    if (!myfile.is_open()) {
        std::cout << "Eroare la deschiderea fisierului.";
    } else {
        myfile >> a;
        myfile >> b;
        std::cout << a+b << std::endl;
        myfile.close();
    }
    return 0;
}
```

example.txt

3  
4

# fstream - example

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

class Complex{
private:
    float re;
    float im;
public:
    friend std::fstream& operator >> (std::fstream& in, Complex &c);
    friend std::fstream& operator << (std::fstream& out, const Complex &c);
    void display(){std::cout << re << " " << im << std::endl;}
};

std::fstream& operator >> (std::fstream& in, Complex &c){
    in >> c.re;
    in >> c.im;
    return in;
}

std::fstream& operator << (std::fstream& out, const Complex &c){
    out << c.re << "\n";
    out << c.im << "\n";
    return out;
}
```

# fstream - example

```
int main(){
    std::fstream myfile;
    Complex c;

    myfile.open("example.txt", std::fstream::in|std::fstream::out);
    if (!myfile.is_open()){
        std::cout << "Eroare la deschiderea fisierului.";
    } else {
        c.display();
        myfile >> c;
        c.display();

        myfile.close();
    }
    return 0;
}
```

# Sfârșit capitol 7