RO: Fişiere în C++
EN. Files in C++

Obiective:

- Abilitatea de a utiliza clasele, metodele şi operatorii pentru intrare/ieşire, pentru intrările şi ieşirile standard şi pentru lucrul cu fişiere

Objectives:

- The ability of using classes, methods and I/O operators for standard input and output and for working with files

Rezumat:

Pentru a realiza operaţii de I/E cu fişiere, trebuie să includem în program fişierul de tip antet (spaţiul de nume) *fstream*, care defineşte mai multe clase, incluzând: *ifstream*, *ofstream* şi *fstream*. Aceste clase sunt derivate din clasele *istream* şi *ostream*. În C++, un fişier este deschis prin cuplarea lui la un *stream*. Înainte de deschiderea fişierului trebuie să obţinem mai întâi un stream. Pentru a crea un stream de intrare, el trebuie declarat ca fiind de tip *ifstream*, iar pentru a crea unul de ieşire, streamul trebuie declarat de tip *ofstream*. Streamurile care realizează ambele tipuri de operaţii trebuie declarate de tip *fstream*.

Metoda *open( )* este folosită pentru a asocia un fişier unui stream. Această metodă este membră a tuturor celor trei clase de tip stream şi are prototipul:

*void open(char *nume_fişier, int mod, int acces);* unde primul parametru reprezintă un pointer la numele fişierului, al doilea este o valoare care specifică modul de deschidere a fişierului, al treilea e implicit setat pe mecanismul de acces din SO DOS.

Se poate utiliza un mecanism implicit simplificat de deschidere a fisierelor de forma:

```
ifstream fin ("test");          // in file
ofstream fout ("test");         // out file
fstream finout ("test");        // in-out file
```

Pentru închiderea unui fişier se foloseşte metoda membră *close( )*. Metoda *close( )* nu are parametri şi nu returneaza nici o valoare. Pentru a închide un fişier cuplat la un stream denumit stream_propriu, se foloseşte instrucţiunea:

stream_propriu.close( );

Limbajul C++ asigură numeroase metode de I/E de tip binar; *get( )* şi *put( )* sunt metode binare de I/E de nivel fizic. Cu ajutorul metodei membre *put( )* scriem un octet, iar cu metoda *get( )* citim un octet. Prototipul metodei *get( )* este următorul:

istream& get(char& car);

Metoda citeşte din stream-ul asociat un singur caracter şi plasează valoarea sa în variabila *car*, returnează o referinţă a stream-ului de intrare.

Metoda *put( )* are prototipul:

ostream& put(char car);

Metoda scrie caracterul *car* în stream şi returnează o referinţă a stream-ului de ieşire. Există si alte variante ale metodelor.

In cadrul operațiilor de I/E se mai folosește metoda *ignore( )* care extrage si sare peste cel mult *n* caractere dar se oprește la întâlnirea delimitatorului care e extras din stream (al doilea parametru). Formatul este:

istream& ignore (int n = 1, int delim = EOF);

Metoda o găsim deseori legata de metoda *get( )* astfel:

cin.ignore( );
cin.get( );

E de remarcat ca funcția are toți parametrii impliciți si poate fi apelata fără parametrii.

Pentru a scrie blocuri de date binare, se utilizează metodele *read( )/write( )*, care au următoarele prototipuri:

*istream& read(unsigned char *buf, int numar);*
*ostream& write(const unsigned char *buf, int numar);*

Metoda *read( )* citeşte *numar* de octeţi din stream-ul asociat şi îi plasează în bufferul pe care il indica pointerul *buf.*

Metoda *write( )* scrie din zona tampon, pe care o indica *buf*, *numar* octeţi în stream-ul asociat. În sistemul de I/E din C++, putem avea acces de tip *aleator* la fişiere, prin utilizarea metodelor *seekg( )* şi *seekp( ).* Formele cele mai uzuale pentru aceste metode sunt:

istream& seekg (streamoff offset, seek_dir origine) ;

ostream& seekp (streamoff offset, seek_dir origine) ;

Tipul *streamoff* este definit în fişierul antet *iostream* şi el poate conţine valoarea maximă a offset-ului, iar *seek_dir* este un tip de enumerare care are trei valori:

ios::beg, ios::cur, ios::end.

Aflarea poziţiei curente de citire/scriere se poate face cu metodele *tellg( ) sau tellp( ),* care au următoarele sintaxe :

```
long tellg( );
long tellp( );
```

Sistemul de I/E al limbajului C++ conţine informaţii de stare cu privire la operaţiile de I/E. Starea curentă a sistemului de I/E este conţinută într-o variabilă de tip întreg, în care se codifică următorii indicatori: *goodbit, eofbit, failbit, badbit.* Acești indicatori sunt obținuți prin apelul metodei *std::ios::rdstate( ).*

O a doua metoda de obținere a stării sistemului de I/E este prin utilizarea metodelor specializate: *int bad( ), int eof( ), int fail( ), int good( ).*

Metodele *bad( )* si *fail( )* returnează o valoare de tip *bool*:
true daca *badbit* respectiv *failbit* sunt activate, si metoda *good( )* returnează *true* daca nici unul din flagurile de stare (*eofbit, failbit* si *badbit*) este setat.

Metoda *eof( )* returnează o valoare diferita de zero daca se atinge sfârșitul fișierului (marcat de EOF=-1). Flag-ul *eofbit* va fi verificat doar după citirea din fișier si când nu mai sunt date, deci este EOF!!!

```
while (!IN.eof( )){
        IN.get (&car) ;
        // verifica eroarea dupa citire si eof
        if (! IN.good( ) && !IN.eof( ) ) {
    cout << "Eroare de I/E .....se termina programul\n" ;
        return 1;           }
}
```

Metoda *getline( )* permite citirea unui șir de caractere până la întâlnirea delimitatorului sau până la citirea a *nr_max*-1 caractere. Este adecvată pentru citirea liniilor de fișiere text. Sintaxa ei este:

istream& getline(char sir,int nr_max, char delim = '\n');*

Metoda *gcount( )* numără câte caractere s-au citit de la ultima citire. Se apelează după utilizarea metodelor *get( ), getline( )* şi *read( ).* Prototipul metodei este:

```
int gcount( );
```

In cazul metodei *peek( )* aceasta returnează următorul caracter fără a-l extrage din stream.

```
int peek( );
```

Mai avem:

```
istream& putback(char c);// put back character c in the stream
```

Exemple:

Exemplul 1:

```
//  program ce utilizeaza metoda put( ) pentru a scrie caractere intr-un fisier, pana cand se
//introduce   caracterul '$'

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

int main(int argc, char*argv[])
{
        char car;
```

```
        if (argc!=2)  {
                cout <<"Specificati numele fisierului! \n" ;
                exit(1);
        }

        ofstream out;
        out.open(argv[1]);
        if (!out)  {
                cout  <<"Nu poate deschide fisierul";
                exit(1);
        }

        cout << "Introduceti un $ pentru a opri programul\n";
        do {
                cin.get(car);
                out.put(car);
        } while (car! ='$' );
        out.close( );
        return 0;
}
```

Exemplul 2:

```
// program ce utilizeaza metoda read( ) pentru a citi date dintr-un fisier

#include <iostream>
#include <fstream>
using namespace std;
const int dim =11;

int main( )
{
        char sir[dim]=" ";
        ifstream in ;
        in.open("test.txt");
        if (!in) {
                cout<< "Nu poate deschide fisierul de intrare\n" ;
                exit(1);
        }

        while(1){
                in.read(sir,dim-1) ;//citire la multiplu de 10 caractere- ce e in plus?
                        if (in.eof( ))break;
                cout<<sir ;
        }
        in.close( ) ;
        return 0;
}
```

Exemplul 3:

```
//scriere string, double si int  separate prin spatiu cu citirea  pana la EOF =-1
#include <iostream>
#include <fstream>
using namespace std;
const int dim =20;

int main( )
{
        ofstream out("test.txt"); // output, normal file

        if (!out) {
```

```
		cout << "Cannot open test.txt file.\n";
		exit(1);
	}

	out << "R " << 9.9 << " " << 10 << endl;
	out << "T " << 9.9 << " " << 9 << endl;
	out << "M " << 4.8 << " " << 4 << endl;
	out<<std::ifstream::traits_type::eof();// write EOF
	out.close( );
	ifstream in("test.txt"); // input
	if (!in) {
		cout << "Cannot open test.txt file.\n";
		exit(1);
	}
	char item[dim];
	double cost;
	int mark;
		/*
	int eof_m;
	while (1){
		in >> item >> cost >> mark;
			if (eof_m = in.peek( ), eof_m == -1)break;
		cout << item << " " << cost << " " << mark << "\n";
	};
		*/
		while (1) {
		in >> item >> cost >> mark;
			if (in.eof( )) break;
		cout << item << " " << cost << " " << mark << "\n";
			}
	in.close( );
	return 0;
}
```

Exemplul 4:

```
// program ce utilizeaza accesul aleator la fisiere, modifica un anumit octet din fisier
#include<iostream>
#include<fstream>
using namespace std;

int main(int argc, char *argv[ ]){
	if(argc!=4){
		cout<<"Utilizare: <nume fisier exe> <nume_fisier> <pozitie> <car>";
		exit(1);
	}
	fstream out;
	out.open(argv[1],ios::in|ios::out|ios::binary);
	if(!out){
		cout<<"Nu poate deschide fisierul";
		exit(1);
	}
	out.seekp(atoi(argv[2]),ios::beg);
	out.put(*argv[3]);
	out.close( );
	return 0;
}
```

Exemplul 5:

```
/* a)Program care utilizeaza o functie de tip extractor si o functie de tip inserter
pt. a citi de la tastatura si a scrie obiecte din clasa Coord pe ecran sau intr-un fisier text*/
```

```
//Coord.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 &, Coord ob);
friend istream& operator>> (istream &, Coord &ob);
};

// inserter pentru clasa Coord
ostream& operator<< (ostream &stream, Coord ob)
{
stream<<"Coordonatele sunt:";
stream << ob.x <<", "<< ob.y << '\n';
return stream;
}

// extractor pentru clasa Coord
istream& operator>> (istream &stream, Coord &ob)
{
cout<< "Introduceti coordonatele: ";
stream>> ob.x >> ob.y;
return stream;
}

// programul principal
#include <iostream>
using namespace std;
#include "Coord.h"

int main ( )
{
Coord A(2,2), B(10,20);
cout<< A << B;
cin >> A;
cout << A;

ofstream fout;
fout.open("test.txt", ios::out);
fout << A;
fout.close( );
return 0;
}

//b) varianta cu supraincarcare diferita consola-fisier

//Coord.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 &, Coord ob);
```

```
        friend istream& operator>> (istream &, Coord &ob);
        friend ofstream& operator<< (ofstream &, Coord ob);
        friend ifstream& operator>> (ifstream &, Coord &ob);
};

// inserter pentru clasa Coord
ostream& operator<< (ostream &stream, Coord ob)
{
        stream << "Coordonatele sunt:";
        stream << ob.x << ", " << ob.y << '\n';
        return stream;
}

// extractor pentru clasa Coord
istream& operator>> (istream &stream, Coord &ob)
{
        cout << "Introduceti coordonatele noului punct: ";
        stream >> ob.x >> ob.y;
        return stream;
}

// file inserter pentru clasa Coord
ofstream& operator<< (ofstream &stream, Coord ob)
{
        stream << ob.x << " " << ob.y << '\n';
        return stream;
}

// file extractor pentru clasa Coord
ifstream& operator>> (ifstream &stream, Coord &ob)
{
        stream >> ob.x >> ob.y;
        return stream;
}

// programul principal
#include <iostream>
#include <fstream>
using namespace std;
#include "Coord.h"

int main( )
{
        ofstream fout;
        ifstream fin;
        fout.open("test.txt", ios::out | ios::trunc);
        if (!fout) {
                perror("Cannot open test.txt file.\n");
                exit(1);
        }
        Coord A(7, 2), B(17, 20);
        cout << A << B;
        //cout << "\nScrie A (7,2) si B(17,20) in fisier\n";
        fout << A << B;
        cin >> A;
        //cout << "\nValorile noi introduse sunt: \n";
        cout << A;
        fout << A;
        //fout << std::ifstream::traits_type::eof();// write EOF
        fout.close( );
```

```
	fin.open("test.txt");
	if (!fin) {
		perror ( "Cannot open test.txt file.\n");
		exit(1);
	}

	while (1) {
		fin >> A; cout << A;
		if (fin.eof( )) break;
		fin >> A; cout << A;
	}
/*
	while (!fin.eof( )) {//read twice the last values - past the end
		fin >> A; cout << A;
	}
*/
	fin.close( );
	return 0;
}
```

Exemplul 6:

```
/*Program care gestioneaza sfarsitul de fisier folosind metoda good( )*/

#include <iostream>     // std::cin, std::cout
#include <fstream>      // std::ifstream
//using namespace std;
#include <stdlib.h>
const int dim1 =256;
const int dim2 =25;

int main( ) {
	char str[dim1];
	std::fstream io;

	std::cout << " File to generate \n";
	io.open("test.dat", std::ios::out);
	if (!io) {
		std::cout << "Nu poate deschide fisierul de scriere\n";
		exit(1);
	}
	io.write("Sir de test de verificat", dim2);
	io<<"\njhgfs jdgahjs";
	io.close( );
	std::cout << "Enter the name of an existing test.dat file: ";
	std::cin.get(str, dim1);    // get c-string
	std::ifstream is(str);     // open file
	if (!is) {
	std::cout << "Nu poate deschide fisierul de citire\n";
		exit(1);
	}
	while (is.good( )) // loop while extraction from file is possible
	{
		char c = is.get( );       // get character from file
		if (is.good( ))
			std::cout << c;
	}
	is.close( );  // close file
	return 0;
}
```

Exemplul 7:

```
/*Program care gestioneaza flag-urile de stare*/
// error state flags
#include <iostream>     // std::cout, std::ios
#include <sstream>      // std::stringstream

void print_state(const std::ios& stream);

int main( ) {
   std::stringstream stream;
   stream.clear(stream.goodbit);
   std::cout << "goodbit:"; print_state(stream); std::cout << '\n';
   stream.clear(stream.eofbit);
   std::cout << " eofbit:"; print_state(stream); std::cout << '\n';
   stream.clear(stream.failbit);
   std::cout << "failbit:"; print_state(stream); std::cout << '\n';
   stream.clear(stream.badbit);
   std::cout << " badbit:"; print_state(stream); std::cout << '\n';
}
void print_state(const std::ios& stream) {
   std::cout << " good( )=" << stream.good( );
   std::cout << " eof( )=" << stream.eof( );
   std::cout << " fail( )=" << stream.fail( );
   std::cout << " bad( )=" << stream.bad( );
}
```

Teme:

1. Să se scrie un program care foloseşte metoda *seekg( )* pentru poziţionare la mijlocul fişierului si apoi afişează conţinutul fişierului începând cu această poziţie. Numele fişierului se citeste din linia de comandă.
2. Scrieţi un program care utilizează metoda *write( )* pentru a scrie într-un fişier şiruri de caractere. Afişaţi apoi conţinutul fişierului folosind metoda *get( )*. Numele fişierului se va citi de la tastatură.
3. Scrieţi o aplicaţie C++ care citeşte un fişier utilizând metoda *read( )*. Verificaţi starea sistemului după fiecare operaţie de citire. Numele fişierului se va citi din linia de comandă. Afişaţi pe ecran conţinutul fişierului.
4. Scrieți o aplicație C++ în care deschideți un fișier în mod binar pentru citire. Afișați un mesaj corespunzator dacă fișierul nu a fost creat în prealabil și cereți reintroducerea numelui fișierului. Presupunând că în fișierul deschis există înregistrări de tip agendă (*nume, localitate, număr de telefon*), utilizați supraîncărcarea operatorilor de inserție și extracție pentru afisarea pe ecran a conținutului fișierului.
5. Considerati clasa *Fractie* care are doua atribute intregi private *a* si *b* pentru numarator si numitor, doua metode de tip *set( )* respectiv *get( )* pentru atributele clasei. Declarati o metoda *simplifica( )* care simplifica un obiect *Fractie*. Definiti un constructor explicit fara parametri care initializeaza *a* cu 0 si *b* cu 1, si un constructor explicit cu doi parametri care va verifica posibilitatea definirii unei fractii (*b*!=0). Definiti un destructor explicit care afiseaza un mesaj. Supraincarcati operatorii de adunare, scadere, inmultire si impartire (+,-,*,/) a fractiilor folosind metode membre care si simplifica daca e cazul rezultatele obtinute.
   Supraincarcati operatorii de intrare (>>, extractie) si iesire (<<, insertie) cu functii friend care permit citirea si scrierea obiectelor. Instantiati doua obiecte de tip *Fractie* cu date citite de la tastatura. Afisati atributele initiale ale obiectelor folosind supraincarcarea operatorului de iesire. Cititi alte 4 obiecte de tip *Fractie* folosind supraincarcarea operatorului de intrare. Efectuati operatiile implementate prin metodele membre (adunarea si scaderea primelor doua, respectiv inmultirea si impartirea ultimelor doua), stocand rezultatele in alte 4 obiecte. Afisati rezultatele. Scrieti intr-un fisier cu numele introdus de la tastatura cele 4 obiecte initiale precum si rezultatele obtinute, pe randuri diferite.

Homework:

1. Write a program that uses the *seekg( )* method for mid-file positioning and then displays the file's content, starting with this position. The filename is read from the command line.
2. Write a program that uses the *write( )* method for writing some character arrays into a file. Display the file's content using the g*et( )* method. The filename is read from the keyboard.
3. Write a C++ application that reads a file's content using the *read( )* method. The obtained data is displayed on the screen. Check the system's state after each reading operation. The filename is read from the command line.
4. Write a C++ application that opens a binary file for reading. The filename is read from the keyboard. Display a message if the file doesn't exist and ask the user to re-enter the filename. Assuming that the file contains some agenda records (*name, city, phone number*) overload the insertion and extraction operators for reading the file's content and for displaying it on the screen.
5. Consider the *Fraction* class that has two private attributes *a* and *b* for the nominator and denominator and two corresponding setter and getter methods. Declare a method named *simplify( )* that simplifies a fraction. Define an explicit constructor without parameters that initializes *a* with 0 and *b* with 1 and another explicit constructor that receives two parameters representing the values of the nominator and denominator. This constructor verifies if the fraction can be defined (*b* != 0). Overload the addition, subtraction, multiplication and division operators (+, -, *, /) using member methods that simplify (if necessary) the obtained results.
   Overload the input (>>, extraction) and output (<<, insertion) operators using friend functions that allow reading and writing the data related to an entire object. Instantiate two *Fraction* objects with data read from the keyboard. Display the initial attributes of the objects by usign the insertion operator. Read another four objects using the extraction operator. Perform the operations implemented with member methods (the addition and subtraction of the first two objects, the multiplication and division of the last ones) and store the results into another four objects. Display the results. Write into a file the original values and the obtained results, on different rows.