RO: Intrări/ieşiri C++. Supraîncărcarea operatorilor de I/E.

EN: Input/Output in C++. The I/O operators overloading.

Obiective:

- Înţelegerea noţiunilor teoretice legate de sistemul de intrare/ieşire prin aplicarea lor în practică, dezvoltarea de programe C++ cu intrari/ieşiri.

Objectives:

- Understanding the theoretical aspects of the I/O system by applying them into the developed programs.

Rezumat:

Sistemul de Intrare/Ieşire (I/E) din C++ operează prin *stream*-uri.

Un *stream* este un dispozitiv logic, care fie produce, fie consumă informaţie şi este cuplat la un dispozitiv fizic prin intermediul sistemului de I/E din C++. La execuţia unui program în C++ se deschid în mod automat următoarele patru stream-uri predefinite: *cin, cout, cerr, clog*.

Sistemul de I/E din C++ ne permite să formatăm datele. Fiecare stream din C++ are asociat un număr de "indicatori de format" (flag-uri), care determină modul de afişare. În clasa *ios_base* este definit tipul ***fmtflags***, pentru flagurile de formatare ale fluxului. În această clasă există mai multe constante publice de acest tip, care pot fi folosite pentru formatarea fluxurilor şi care sunt enumerate în continuare:

skipws – ignorarea caracterelor de tip whitespace
left – aliniere la stanga
right- aliniare la dreapta
internal- o valoare numerică se extinde pentru completarea câmpului
dec- afisare în zecimal
oct – afisarea în octal
hex- afisare în hexazecimal
showbase - afişarea simbolului bazei de numeraţie în care se face afişarea
boolalpha- citeşte/scrie elementele booleene ca stringuri (true şi false).
showpoint- afişarea tuturor zerourilor şi a punctului zecimal pentru o valoare în virgulă mobilă
showpos - determină afişarea semnului înaintea valorilor numerice pozitive
scientific- afişarea numerelor în virgulă mobilă în format ştiinţific (cu exponent)
fixed – afişarea în forma obişnuită a numerelor în virgulă mobilă parte întreagă/ fracţionară
unitbuf- streamul este eliberat după fiecare operaţie de ieşire
uppercase- afişarea cu majusculă a caracterelor generate de stream

Formatarea datelor poate fi realizata prin intermediul flag-urilor, manipulatorilor standard si a manipulatorilor utilizator.

Stabilirea formatului prin intermediul flagurilor se poate realiza prin utilizarea metodei:

long setf (long f);
unde flagurile sunt codificate în parametrul de tip *long* al metodei.

fmtflags setf (fmtflags mask);
fmtflags setf (fmtflags mask, fmtflags unset);

Resetarea flagurilor se realizează cu metoda:
long unsetf (long flags);
void unsetf (fmtflags flags);

Starea curenta a indicatorilor de format poate fi obținută cu metoda *flags():*
fmtflags flags( );
fmtflags flags(fmtflags f);

Există trei metode membre ale clasei *ios*, care stabilesc lățimea câmpului, precizia şi caracterul de inserat. Acestea sunt *width( ), precision( ) şi fill( )*. Prototipul metodei *width( )* este următorul:

int width(int w);

unde *w* – este lăţimea câmpului, iar valoarea returnată este valoarea anterioară a câmpului.

La afișarea unei valori în virgulă mobilă, se folosesc în mod implicit (default) 6 cifre pentru întregul număr (parte întreagă si fracționară). Totuși doar partea fracționară o putem fixa daca alegem un tip real specific (*fixed* sau *scientific*) si apelăm metoda *precision( )*, altfel se va referi la precizia specifica părții întregi si fracționare:

int precision(int p); unde *p* – stabileşte precizia , metoda returnează vechea valoare.

Când un câmp trebuie completat, se folosește în mod implicit caracterul "spațiu". Se poate specifica un caracter alternativ cu ajutorul metodei *fill( )*, cu prototipul:

char fill(char ch); unde *ch* – noul caracter cu care se completează câmpul, metoda returnează vechiul caracter.

Metodele de tip "manipulator" sau "manipulatorii standard" permit formatarea operaţiilor de I/E. În limbajul C++ sunt : *dec, endl, ends, flush, hex, oct, resetiosflags(long f), setbase (int base), setfill (int ch), setioflags(long f), setprecision(int p), ws, setw(int w).*
Manipulatorii standard cu parametrii pentru a putea fi utilizati necesita a folosi includerea,
#include <iomanip> din spatiul de nume standard.

Manipulatorii utilizator sunt de două tipuri:

- cei care operează cu stream-uri de intrare;
- cei care sunt asociaţi stream-urilor de ieşire.

Forma generală a manipulatorilor utilizator de ieşire este următoarea:

```
ostream& nume-manipulator (ostream& stream)
{
        // cod
        return stream;
}
```

Sintaxa metodelor manipulator utilizator de intrare este următoarea:

```
istream& nume manipulator (istream& stream)
{
        // cod
        return stream;
}
```

Noile versiuni C++ 1y/2z au introdus noi elemente legate de formatarea datelor cu manipulatori:https://www.tutorialspoint.com/cpp_standard_library/iomanip.htm
https://en.cppreference.com/w/cpp/io/manip printre care amintim:
-*hexfloat, defaultfloat* (C++11), pentru numerele reale,
-*emit_on_flush, no_emit_on_flush* (C++20), controlează dacă stream-ul *baza_syncbuf* al unui flux se emite în flux
-*get_money* (C++11), parsare valoare monetara
-*put_money* (C++11), formatează și produce o valoare monetară
-*get_time* (C++11), parsare data/timp la format specificat
-*put_time* (C++11), formatează și scoate o valoare de tip dată/timp în funcție de formatul specificat
-*quoted* (C++14), inserează și extrage șiruri intre ghilimele cu spații încorporate

Supraîncărcarea operatorilor de intrare-ieșire.

Operatorii de intrare/ieșire pot fi supraîncărcați într-o anumită clasa prin funcții *friend*. In cazul accesului direct la atribute (ele fiind publice) sau prin metode membre publice la atribute *private* sau *protected* se pot folosi si functii globale. Supraîncărcarea operatorului de ieșire se realizează printr-o funcție *inserter*, care are următoarea forma generală:

```
ostream& operator <<(ostream& stream, Nume_clasa ob)
{
        // corp inserter
```

```
        return stream;
}
```

Supraîncărcarea operatorului de intrare se realizează printr-o funcție *extractor*, care are următoarea formă generală:

```
istream& operator >>(istream& stream, Nume_clasa& ob)
{
        // corp extractor
        return stream;
}
```

Exemple:

Exemplul 1:
//Ilustreaza utilizarea metodei *setf( ) si unsetf( )***.**

```
#include <iostream>
using namespace std;
const int dim =30;

int main( ){
// afisarea valorilor folosind poziţionarile implicite
cout<< 123.33 <<" salut! " << 100 <<'\n';
cout<< 10 << ' ' << -10 <<'\n';
cout << 100.01 <<'\n';
cout << 100.0 <<'\n';
//  schimbam formatul
cout.unsetf(ios::dec);
cout.setf( ios_base::hex);
cout << 123 << " salut! " << 100 <<'\n';
cout.setf (ios::showpos|ios::showbase);
cout <<10 << ' '<< -10 << '\n';
cout.setf(ios::scientific);
cout<<100.1<<'\n';
cout.unsetf(ios::scientific);
cout.setf(ios::dec|ios::showpoint);
cout << 100.0<<'\n';

//alinieri
cout.width(dim);
cout.fill('*');
cout.setf(ios::right);
cout<<"Aliniere dreapta"<<'\n';

cout.unsetf(ios::right);
cout.width(dim);
cout.fill('*');
cout.setf(ios::left);
cout<<"Aliniere stanga"<<'\n';
return 0;
}
```

Exemplul 2 :
//Folosirea manipulatorilor standard de I/E.

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

```
using namespace std;

int main( ){
cout << hex << 100 << endl;
cout << oct << 100 << endl;
cout << setfill( 'Y') << setw (8);
cout << 10 << endl;
cout << setprecision(8);
cout<<100.120<<endl;
cout << setprecision(4);
cout<<-100.120<<endl;

cout.setf(ios::fixed);
cout<<setprecision(8);
cout<<100.120<<endl;
cout<<setprecision(4);
cout<<-100.120<<endl;
return 0 ;
}
```

Exemplul 3:

```
//Manipulatori proprii - utilizator

#include<iostream>
using namespace std;

ostream& init(ostream& stream);


int main()
{
        cout.setf(ios::uppercase);
        cout << "valoarea de afisat: ";
        cout << init << 111.123456;
        return 0;
}

ostream& init(ostream& stream)
{
        stream.width(20);
        stream.precision(5);
        stream.fill('$');
        stream.setf(ios::showpos | ios::scientific | ios::uppercase);
        return stream;
}
```

Exemplul 4:

```
//supraincarcarea operatorilor de insertie si extractie cu functii friend
//MyClass.h
#define _CRT_SECURE_NO_WARNINGS
const int dim =10;

class MyClass{
private:
        int x;
        char text[dim];

public:
        MyClass( ){ }
```

```
        MyClass(int x, const char *text){
                this->x = x;
                strcpy(this->text, text);
        }

        friend istream& operator>>(istream& stream, MyClass &obj);
        friend ostream& operator<<(ostream& stream, MyClass obj);

};

//supraincarcarea operatorului de extractie (intrare)
istream& operator>>(istream& stream, MyClass &obj){
        cout<<"\nIntroduceti valoarea variabilei x: ";
        stream>>obj.x;
        cout<<"\nIntroduceti valoarea variabilei text: ";
        stream>>obj.text;
        return stream;
}

//supraincarcarea operatorului de insertie (iesire)
ostream& operator<<(ostream& stream, MyClass obj){
        stream<<"\nVariabilele din clasa au valorile: ";
        stream<<obj.x<<", ";
        stream<<obj.text;
        return stream;
 }

//main
#include <iostream>
using namespace std;
#include "MyClass.h"

int main( ){
        MyClass test_obj1(7, "aaa");
        cout<<test_obj1;

        MyClass test_obj2;

        cin>>test_obj2;
        cout<<test_obj2;
        return 0;
}
```

Exemplul 5:

```
a)
//supraincarcarea operatorilor de insertie si extractie cu functii friend
//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);
};// Coord class

// inserter pentru clasa Coord
ostream& operator<< (ostream& stream, Coord ob) {
```

```
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;
}

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

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

b)
//supraincarcarea operatorilor de insertie si extractie cu functii globale
//Coord.h

class Coord {
        int x, y;
public:
        Coord ( ) { x = 0; y = 0; }
        Coord (int i, int j) { x = i; y = j; }
        int getX( ) {
                return x; }
        int getY( ) {
                return y; }
        void setX(int a) {
                x = a;    }
        void setY(int b) {
                y = b;    }
};// Coord class

// global inserter pentru clasa Coord
ostream& operator<< (ostream& stream, Coord ob) {
        stream << ob.getX() << " " << ob.getY() << '\n';
        return stream;
}

// global extractor pentru clasa Coord
istream& operator>> (istream& stream, Coord& ob) {
        int a, b;
        cout << "Introduceti coordonatele: ";
        stream >> a >> b;
        ob.setX(a);
        ob.setY(b);
        return stream;
}

//main
#include <iostream>
```

```
using namespace std;

#include "Coord.h"

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

Teme:

1. Scrieţi un program C++ în care afişaţi diferite valori în zecimal, octal şi hexazecimal. Afişaţi valorile aliniate la dreapta, respectiv la stânga într-un câmp de afişare cu dimensiunea 15. Utilizaţi manipulatorul *setfill( )* pentru stabilirea caracterului de umplere şi metodele *width( )* şi *precision( )* pentru stabilirea dimensiunii câmpului de afişare şi a preciziei.
2. Scrieti o aplicatie C++ in care se citesc de la tastatura date de diferite tipuri, urmand a fi afisate pe ecran utilizand manipulatorii standard.
3. Considerati achizitia de date de la un dispozitiv electronic (10 date). Afisati folosind un mesaj adecvat datele primite considerand un format minimal (partea intreaga). Determinati media acestor valori, iar daca depaseste un prag stabilit anterior, afisati aceste date in format detaliat considerand ca avem date de tip real, cu o precizie de 8 digiti.
4. Definiti o clasa numita *MiscareAccelerata* care contine atributele private *dc* (distanta curenta, *vc* (viteza curenta) si *a* (acceleratia). atributele *dc, vc* si *a* sunt initializate in constructor iar valoarea lor este cea data de *d0* si *v0,* si *a0* ca si parametri. In clasa sunt supraincarcati operatorii de extractie si de inserție pentru a se putea initializa si afisa caracteristicile unei instante. Implementati metoda *determinaMiscarea* care re-calculeaza variabilele *dc* si *vc,* pe baza unui numar de secunde primit ca parametru si avand in vedere legea miscarii rectilinii uniform accelerate cu acceleratie *a0*.
   Instantiati clasa si apoi folositi membrii definiti.
5. Supraîncărcaţi operatorii de extracţie şi de inserţie pentru clasa *Complex*, în care părţile reale şi imaginare sunt ambele *protected* de tip real. Derivaţi public o clasă *Punct* din clasa *Complex*, adăugând atributul culoare pentru punctul de coordonate x şi y corespunzător părţii reale, respectiv imaginare a numărului complex. Supraîncărcaţi din nou aceiași operatori de intrare-ieșire. Instantiati obiecte de tip *Complex* si *Punct* si verificati functionalitatea supraincarcarii operatorilor. Asignati un obiect de tip *Punct* la unul de tip *Complex* prin upcasting si afisati atributele lui.
6. 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* returnand o valoare reala. Considerati o variabila privata statica intreaga *icount*, care va fi initializata cu *0* si incrementata in cadrul constructorilor din clasa. Definiti un constructor explicit fara parametri care initializeaza *a* cu 0 si *b* cu 1, si un constructor explicit cu doi parametri care va putea fi apelat daca se verifica posibilitatea definirii unei fracții (*b*!=0). Definiti un destructor explicit care afiseaza un mesaj. Supraincarcati operatorii de adunare, scadere, inmultire si impartire (+,-,*,/) a fractiilor folosind functii friend fara a simplifica rezultatele obtinute. Instantiati doua obiecte de tip *Fractie* cu date citite de la tastatura. Afisati atributele initiale ale obiectelor pe linii diferite iar fiecare membru al fractiei va fi afisat pe o latime de 10 digiti, caracter de umplere *, primul numar aliniat la stanga iar al doilea aliniat la dreapta. Printr-o metoda accesor, afisati contorul *icount* ca si un intreg cu semn, pe 15 pozitii, caracter de umplere $, aliniat la stanga. Efectuati operatiile implementate prin functiile *friend*, initializand alte 4 obiecte cu rezultatele obtinute. Afisati rezultatele (numarator/numitor) folosind supraincarcarea operatorul de iesire (<<, insertie) si contorul (ca si un intreg cu semn, pe 20 de pozitii, caracter de umplere #, aliniat la dreapta) dupa ultima operatie folosind o metoda accesor adecvata. Simplificati rezultatele obtinute pe care le veti afisa ca numere reale de tip *fixed* cu o precizie de 4 digiti la partea fractionara.

Homework:

1. Write a C++ program that displays some numerical values in decimal, octal and hexadecimal. Display the values left and right aligned, inside a field that can hold 15 characters. Use the *setfill( )* manipulator for setting the filling character and the *width( )* and *precision( )* methods for setting the displaying field size and the values representation precision.
2. Write a C++ application that reads from the keyboard a series of values of various types. Display the values using the standard manipulators.
3. Consider a data acquisition process from a hardware device (10 variables). Display, using an appropriate message, the data in a minimal format (the integer part). Determine the average value of the displayed numbers and if it is greater than a previously defined threshold, display the data in a detailed format (float variables, 8 digits precision).
4. Define a class called *AcceleratedMovement* that contains the private attributes *dc* (the current distance), *vc* (the current speed) and *a* (the acceleration). The values of *dc*, *vc* and *a* are initialized in the constructor and their values are equal to *d0*, *v0* and *a0* (as parameters).The class overloads the extraction and insertion operators for initializing and displaying the characteristics of a certain instance.
   Implement the method *determineMovement* that re-calculates the values of *dc* and *vc*, considering a number of seconds (received as paramenter) and the law of uniformly accelerated linear motion with *a0* acceleration. Instantiate the class and use the defined members.
5. Overload the extraction and insertion operators for the *Complex* class (both the imaginary and real parts are represented as *protected* real values). Create another class named *Point* that inherits the first class and introduces the color attribute for a point that has as coordinates the real and imaginary parts of the complex number. Overload again the extraction and insertion operators. Create some instances of the defined classes and verify the functionality of the overloaded operators. Assign an object of *Point* type to an object of *Complex* type and dispaly the attributes of the obtained object.
6. 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 and returns a real value. 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. For this constructor is verified if b!=0 before to be called. Define a destructor that displays a message. Consider a static variable *icount* that will be initialized with 0 and incremented in the class's constructors. Overload the addition, subtraction, multiplication and division operators (+, -, *, /) using friend functions, without simplifying the obtained results. Instantiate two *Fraction* objects and read the corresponding data from the keyboard. Display the initial attributes of the objects, on different lines, in 10 digits placeholders using * as filling character. The denominator will be left aligned while the nominator will pe positioned in the right part of the displaying field. Using an accesor method display the value of *icount* as a signed integer, on 15 characters, left aligned, using '$' as filling character. Perform the operations implemented with friend functions initializing another four objects with the obtained results. Display the data (denominator/nominator) by overloading the output (<<, insertion) operator and the counter (as a signed integer, on 20 characters, right aligned, using '#' as filling character) after the last operation. Simplify the results and display the resulting values as *fixed* float numbers with 4 digits precision.