Laborator 12

```
/*  Lab 12  Popa Larisa-Ancuta  Prob 1

-scrieți un program C++ în care afișați diferite valori în zecimal, octal și hexazecimal
-afișați valorile aliniate la dreapta, respectiv la stânga într-un câmp de afișare cu
dimensiunea 15
-utilizați manipulatorul setfill( ) pentru stabilirea caracterului de umplere și metodele
width( ) și precision( ) pentru stabilirea dimensiunii
   câmpului de afișare și a preciziei

*/

#define _CRT_SECURE_NO_WARNINGS
#include<iostream>

using namespace std;

int main()
{
	cout << "\n(aliniere dreapta)Afisare zecimal: ";
	cout.width(15);
	cout.fill('*');
	cout.precision(5);
	cout.setf(ios_base::dec | ios::right);
	cout << 100.3625;

	cout << "\n(aliniere stanga)Afisare octal: ";
	cout.width(15);
	cout.fill('@');
	cout.precision(10);
	cout.unsetf(ios::dec);
	cout.setf(ios_base::oct | ios::left);
	cout << 100;

	cout << "\nAfisare hexazecimal: ";
	cout.width(15);
	cout.fill('#');
	cout.precision(7);
	cout.unsetf(ios::oct);
	cout.setf(ios_base::hex);
	cout << 100;

	return 0;
}
```

```
/*  Lab 12  Popa Larisa-Ancuta  Prob 2

-scrieti o aplicatie C++ in care se citesc de la tastatura date de diferite tipuri
-afisati pe ecran utilizand manipulatorii standard

*/

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <iomanip>
```

```
using namespace std;

int main()
{
	int nr;
	double nr_d;
	char c[25];

	cout << "Introduceti o valoare intreaga: ";
	cin >> nr;
	cout << "Octal: " << oct << nr;
	cout << "\nHexazecimal: ";
	cout << setfill('*') << setw(5);
	cout << hex << nr;

	cout << "\n\nIntroduceti o valoare reala: ";
	cin >> nr_d;
	cout << "Precizie de 5 digiti:" << setprecision(5) << nr_d;

	cout << "\n\nIntroduceti un sir de caractere: ";
	cin >> c;
	cout << "Precizie de 10 digiti:" << fixed << setprecision(10) << c << endl <<
endl;

	return 0;
}
```

```
/* Lab 12  Popa Larisa-Ancuta  Prob 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

*/

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <iomanip>

using namespace std;

#define DIM 10

int main()
{
	int i;
	float tab[DIM], med = 0, med_prag;

	cout << "\nIntroduceti datele: \n";
	for (i = 0;i < DIM;i++)
	{
		cout << "tab[" << i << "]=";
```

```
			cin >> tab[i];
			med += tab[i];
		}
		med /= DIM;

		cout.setf(ios::dec);
		cout.precision(0);

		cout << "\nAfisare parte intreaga: \n";
		for (i = 0; i < DIM; i++)
			cout << fixed << setprecision(0) << tab[i] << " ";

		cout << "\n\nIntroduceti media de prag: ";
		cin >> med_prag;

		if (med > med_prag)
		{
			cout << "\nMedia de prag a fost depasita!\n";

			cout.unsetf(ios::dec);
			cout.precision(8);

			cout << "\nAfisare cu precizie de 8 digiti: \n";
			for (i = 0; i < DIM; i++)
				cout << tab[i] << " ";

			cout << endl << endl;
		}
		else
			cout << "\nMedia de prag nu a fost depasita!\n\n";

		return 0;
}
```

```
/*  Lab 12  Popa Larisa-Ancuta  Prob 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

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <iomanip>

using namespace std;
```

```cpp
#include "Header.h"

int main()
{
	float t;
	MiscareAccelerata o;

	cout << "    Introduceti datele \n";
	cin >> o;

	cout << "\nIntroduceti timpul de parcurgere: ";
	cin >> t;

	o.determinaMiscarea(t);

	cout << o << endl << endl;

	return 0;
}

//header
//Header.h
#pragma once

class MiscareAccelerata
{
private:
	float dc;
	float vc;
	float a;
public:
	MiscareAccelerata()
	{
		dc = 0;
		vc = 0;
		a = 0;
	}
	MiscareAccelerata(float dc0, float vc0, float a0)
	{
		dc = dc0;
		vc = vc0;
		a = a0;
	}

	friend istream& operator >> (istream& stream, MiscareAccelerata& ob);
	friend ostream& operator<<(ostream& stream, MiscareAccelerata& ob);

	void determinaMiscarea(float t);
};

istream& operator >> (istream& stream, MiscareAccelerata& o)
{
	cout << "Distanta initiala: ";
	stream >> o.dc;

	cout << "Viteza initiala: ";
	stream >> o.vc;
```

```
	cout << "Acceleratia : ";
	stream >> o.a;

	return stream;
}

ostream& operator<<(ostream& stream, MiscareAccelerata& o)
{
	stream << "\nDistanta finala: " << o.dc;
	stream << "\nViteza finala: " << o.vc;
	stream << "\nAceleratia: " << o.a;

	return stream;

}

void MiscareAccelerata::determinaMiscarea(float t)
{
	dc = dc + vc * t + a * t * t / 2;
	vc = vc + a* t;
}
```

```
/*  Lab 12  Popa Larisa-Ancuta  Prob 5

-supraîncărcați operatorii de extracție și de inserție pentru clasa Complex, în care
părțile reale și imaginare sunt ambele protected de tip real
-derivați public o clasă Punct din clasa Complex, adăugând atributul culoare pentru
punctul de coordonate x și y corespunzător părții reale,
   respectiv imaginare a numărului complex
-supraîncărcați 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

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <iomanip>

using namespace std;

#include "Complex.h"
#include "Punct.h"

int main()
{
	Complex z;
	Punct p;

	cout << "\n   Introduceti numarul complex \n";
	cin >> z;
	cout << z;
```

```
	cout << "\n\n\n   Introduceti coordonatele punctului \n";
	cin >> p;
	cout << p;

	//upcasting
	Complex *Z = &z;
	Punct *P = &p;

	Z = P;

	cout << "\n\n\nUpcasting: " << *Z << endl << endl;

	return 0;
}



//header
//Complex.h
#pragma once

class Complex
{
protected:
	float Re, Im;
public:
	Complex()
	{
		Re = 0;
		Im = 0;
	}

	friend istream& operator >> (istream& stream, Complex &o);
	friend ostream& operator<<(ostream& stream, Complex o);
};

//supraincarcarea operatorului de intrare
istream& operator >> (istream& stream, Complex &o)
{
	cout << "Partea reala: ";
	stream >> o.Re;
	cout << "Partea imaginara: ";
	stream >> o.Im;

	return stream;
}

//supraincarcarea operatorului de iesire
ostream& operator<<(ostream& stream, Complex o)
{
	stream << "\nNumarul complex este: ";
	stream << o.Re << " + i*" << o.Im;

	return stream;
}
```

```
//Punct.h
#pragma once

const int dim = 30;

class Punct :public Complex
{
	char culoare[dim];
public:
	Punct() :Complex()
	{
		strcpy(culoare, "-");
	}

	friend istream& operator >> (istream& stream, Punct &o);
	friend ostream& operator<<(ostream& stream, Punct o);
};

//supraincarcarea operatorului de intrare
istream& operator >> (istream& stream, Punct &o)
{
	char c[dim];

	cout << "Partea reala: ";
	stream >> o.Re;

	cout << "Partea imaginara: ";
	stream >> o.Im;

	cout << "Culoarea punctului: ";
	stream >> c;
	strcpy(o.culoare, c);

	return stream;
}

//supraincarcarea operatorului de iesire
ostream& operator<<(ostream& stream, Punct o)
{
	stream << "\nNumarul complex este: ";
	stream << o.Re << " + i*" << o.Im << "\n  Culoarea este: " << o.culoare;

	return stream;
}
```

```
/* Lab 12  Popa Larisa-Ancuta  Prob 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

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <iomanip>

using namespace std;

#include "Fractie.h"

int main()
{
	Fractie f1, f2;
	int nr, nm;

	cout << "\nFractia 1: " << f1.getA() << "/" << f1.getB();
	cout << "\nFractia 2: " << f2.getA() << "/" << f2.getB();

	cout << "\n\nIntroduceti valori noi: \nFractia 1:\n  a=";
	cin >> nr;
	cout << "  b=";
	cin >> nm;

	f1.setA(nr);
	f1.setB(nm);

	cout << "\n\nFractia 2:\n  a=";
	cin >> nr;
	cout << "  b=";
	cin >> nm;

	f2.setA(nr);
	f2.setB(nm);

	cout << "\nFractia 1:\n";
	cout.width(10);
	cout.fill('*');
	cout.setf(ios::left);
	cout << f1.getA() << "/";
```

```
	cout.width(10);
	cout.fill('*');
	cout.setf(ios::right);
	cout << f1.getB();

	cout << "\nFractia 2:\n";
	cout.width(10);
	cout.fill('*');
	cout.setf(ios::left);
	cout << f2.getA() << "/";
	cout.width(10);
	cout.fill('*');
	cout.setf(ios::right);
	cout << f2.getB();

	f1.AfisareIcount1();

	Fractie f3 = f1 + f2, f4 = f1 - f2, f5 = f1*f2, f6 = f1 / f2;

	cout << "\n\nf1+f2=" << f3 << "\nf1-f2=" << f4 << "\nf1*f2=" << f5 << "\nf1/f2="
<< f6;

	f2.AfisareIcount2();

	cout << "\n\nRezultatele simplificate: ";
	cout << "\nf1+f2=" << fixed << setprecision(4) << f3.simplificare(f3);
	cout << "\nf1-f2=" << fixed << setprecision(4) << f4.simplificare(f4);
	cout << "\nf1*f2=" << fixed << setprecision(4) << f5.simplificare(f5);
	cout << "\nf1/f2=" << fixed << setprecision(4) << f6.simplificare(f6);

	cout << endl << endl;

	return 0;
}

//header
//Fractie.h
#pragma once

class Fractie
{
	int a, b;
	static int icount;
public:
	Fractie()
	{
		a = 0;
		b = 1;
		icount++;
	}
	Fractie(int nr, int nm)
	{
		if (b != 0)
		{
			a = nr;
			b = nm;
		}
		else
```

```
                        cout << "\nNumitorul trebuie sa fie diferit de zero!\n\n";

                icount++;
        }

        void setA(int val) { a = val; }
        void setB(int val) { b = val; }

        int getA() { return a; }
        int getB() { return b; }

        float simplificare(Fractie &f);

        friend Fractie operator+(Fractie f1, Fractie f2);
        friend Fractie operator-(Fractie f1, Fractie f2);
        friend Fractie operator*(Fractie f1, Fractie f2);
        friend Fractie operator/(Fractie f1, Fractie f2);

        static void AfisareIcount1();

        friend ostream& operator<<(ostream& stream, Fractie o);

        static void AfisareIcount2();

};

float Fractie::simplificare(Fractie &f)
{
        int nr = f.a, nm = f.b;

        if (nr < 0)
                nr *= (-1);
        if (nm < 0)
                nm *= (-1);

        while (nr != nm)
        {
                if (nr > nm)
                        nr -= nm;
                else
                        nm -= nr;
        }

        f.a /= nr;
        f.b /= nr;

        return (float)f.a / f.b;
}

Fractie operator+(Fractie f1, Fractie f2)
{
        Fractie sum;

        sum.a = f1.a*f2.b + f2.a * f1.b;
        sum.b = f1.b*f2.b;

        return sum;
}
```

```
Fractie operator-(Fractie f1, Fractie f2)
{
	Fractie dif;

	dif.a = f2.a * f1.b - f1.a*f2.b;
	dif.b = f1.b*f2.b;

	return dif;
}

Fractie operator*(Fractie f1, Fractie f2)
{
	Fractie prod;

	prod.a = f1.a*f2.a;
	prod.b = f1.b*f2.b;

	return prod;
}

Fractie operator/(Fractie f1, Fractie f2)
{
	Fractie div;

	div.a = f1.b*f2.a;
	div.b = f1.a*f2.b;

	return div;
}

void Fractie::AfisareIcount1()
{
	cout << "\n\nIcount\n";
	cout.width(15);
	cout.fill('$');
	cout.setf(ios::left);
	cout << icount;

}

ostream& operator<<(ostream& stream, Fractie o)
{
	stream << o.a << "/" << o.b;
	return stream;
}

void Fractie::AfisareIcount2()
{
	cout << "\n\nIcount\n";
	cout.width(20);
	cout.fill('#');
	cout.setf(ios::right);
	cout << icount;
}

int Fractie::icount = 0;
```