Laborator 8

```
/* Lab 8  Popa Larisa-Ancuta  Prob 1

-construiți o aplicație în care clasa OraCurenta are ca atribute private ora, minutele și
secundele
-metode publice de tip set/get pentru atributele clasei
-adaugați o functie friend clasei prin care să se poată copia conținutul unui obiect
OraCurenta dat ca si parametru
    intr-un alt obiect instanță a aceleiași clase care va fi returnat de functie, ora
fiind insa modificata la Greenwich Mean Time
-utilizati timpul curent al calculatorului

*/
//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <time.h>

using namespace std;

#include "Class.h"

int main()
{
	OraCurenta o, new_o;

	time_t my_time;
	struct tm * timeinfo;

	time(&my_time);
	timeinfo = localtime(&my_time);

	o.set_ora(timeinfo->tm_hour);
	o.set_minute(timeinfo->tm_min);
	o.set_secunde(timeinfo->tm_sec);

	cout << "\nOra curenta: " << o.get_ora() << ":" << o.get_minute() << ":" <<
o.get_secunde() << endl;

	new_o = copiere(o);
	cout << "\nOra curenta GMT: " << new_o.get_ora() << ":" << new_o.get_minute() <<
":" << new_o.get_secunde() << endl << endl;

	return 0;
}

//header Class.h
#pragma once
class OraCurenta
{
private:
	int ora, minute, secunde;
public:
	OraCurenta()
	{
		ora = minute = secunde = 0;
	}
```

```
        void set_ora(int);
        void set_minute(int);
        void set_secunde(int);

        int get_ora(void);
        int get_minute(void);
        int get_secunde(void);

        friend OraCurenta copiere(OraCurenta);
};

void OraCurenta::set_ora(int o)
{
        ora = o;
}

void OraCurenta::set_minute(int m)
{
        minute = m;
}

void OraCurenta::set_secunde(int s)
{
        secunde = s;
}

int OraCurenta::get_ora(void)
{
        return ora;
}

int OraCurenta::get_minute(void)
{
        return minute;
}

int OraCurenta::get_secunde(void)
{
        return secunde;
}

OraCurenta copiere(OraCurenta sursa)
{
        OraCurenta destinatie;

        time_t curr_time;
        curr_time = time(NULL);

        tm *tm_gmt = gmtime(&curr_time);

        destinatie.set_ora(tm_gmt->tm_hour);

        destinatie.minute = sursa.minute;
        destinatie.secunde = sursa.secunde;

        return destinatie;
}
```

```
/*  Lab 8  Popa Larisa-Ancuta  Prob 2

-scrieți o aplicație C/C++ în care clasa Calculator are un atribut privat memorie_RAM
(int)
-o funcție prietenă tehnician_service( ) care permite modificarea valorii acestui atribut
-functia friend va fi membra intr-o alta clasa, Placa_de_baza care are o componentă
denumire_procesor (sir de caractere)
-scrieți codul necesar care permite funcției prietene tehnician_service( )
-să modifice (schimbe) valoarea variabilei denumire_procesor si memorie _RAM

*/

//main
#define  _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include  "Class.h"

int main()
{
	Calculator calc;
	Placa_de_baza placa;
	int RAM;
	char nume[DIM];

	cout << "\nIntroduceti memoria RAM: ";
	cin >> RAM;
	calc.set_RAM(RAM);
	cout << "\nIntroduceti denumirea procesorului: ";
	cin >> nume;
	placa.set_denumire_procesor(nume);

	cout << "\nRAM: " << calc.get_RAM() << "\nProcesor: " <<
placa.get_denumire_procesor();

	cout << "\n\nIntroduceti noua memorie RAM: ";
	cin >> RAM;
	cout << "\nIntroduceti noua denumire a procesorului: ";
	cin >> nume;
	placa.Tehnician_Service(calc, RAM, placa, nume);

	cout << "\nRAM: " << calc.get_RAM() << "\nProcesor: " <<
placa.get_denumire_procesor() << endl << endl;

	return 0;
}

//header Class.h
#pragma once

const int DIM = 30;
class Calculator;

class Placa_de_baza
{
private:
```

```
	char denumire_procesor[DIM];
public:
	Placa_de_baza()
	{
		strcpy(denumire_procesor, " ");
	}

	void set_denumire_procesor(char *n)
	{
		if (n != 0)
			strcpy(denumire_procesor, n);
		else
			strcpy(denumire_procesor, "-");
	}
	char*  get_denumire_procesor(void)
	{
		return denumire_procesor;
	}

	void Tehnician_Service(Calculator &, int, Placa_de_baza &, char *);
};

class Calculator
{
private:
	int memorie_RAM;
public:
	Calculator()
	{
		memorie_RAM = 0;
	}

	void set_RAM(int m)
	{
		memorie_RAM = m;
	}
	int get_RAM(void)
	{
		return memorie_RAM;
	}

	friend void Placa_de_baza::Tehnician_Service(Calculator &, int, Placa_de_baza &,
char *);
};

void Placa_de_baza::Tehnician_Service(Calculator &x, int val, Placa_de_baza &y, char
*nume)
{

	x.memorie_RAM = val;
	strcpy(y.denumire_procesor, nume);
}
```

```
/*  Lab 8  Popa Larisa-Ancuta  Prob 3

-definți o clasă numită Repository care are două variabile private de tip întreg
```

```
-clasa mai conține un constructor explicit vid si unul cu 2 parametri și o metodă accesor
care afișează valorile variabilelor din clasă
-scrieți o clasă numită Mathemathics, friend cu prima clasă, care implementează
operațiile aritmetice elementare (+, -, *, /)
   asupra variabilelor din prima clasa
-fiecare metoda primește ca parametru un obiect al clasei Repository.

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string.h>

using namespace std;

#include "Class.h"

int main()
{
	Mathemathics operatii;
	int v1, v2;

	cout << "\nx=";
	cin >> v1;
	cout << "\ny=";
	cin >> v2;

	Repository var(v1, v2);
	var.afisare();

	cout << "\nAdunare: " << operatii.adunare(var);
	cout << "\nScadere: " << operatii.scadere(var);
	cout << "\nInmultire: " << operatii.inmultire(var);
	cout << "\nImpartire: ";
	if (operatii.impartire(var))
		cout << operatii.impartire(var) << endl << endl;
	else
		cout << "\nInmultirea nu poate fi efectuata!\n\n";
	return 0;
}

//header Class.h
#pragma once

class Repository
{
private:
	int x, y;
	friend class Mathemathics;
public:
	Repository()
	{
		x = 0;
		y = 0;
	}
	Repository(int v1, int v2)
	{
```

```
		x = v1;
		y = v2;
	}
	void afisare()
	{
		cout << "\nAfisare: \nx=" << x << "\ny=" << y << endl;
	}
};

class Mathemathics
{
public:
	int adunare(Repository o)
	{
		return o.x + o.y;
	}
	int scadere(Repository o)
	{
		return o.x - o.y;
	}
	int inmultire(Repository o)
	{
		return o.x*o.y;
	}
	float impartire(Repository o)
	{
		if (o.y != 0)
			return (float)o.x / o.y;
		return 0;
	}
};
```

```
/*  Lab 8  Popa Larisa-Ancuta  Prob 4

-scrieți o aplicație C/C++ care definește într-o clasă variabila publică contor
var_static de tip static întreg
-aceasta se va incrementa în cadrul constructorului
-după o serie de instanțieri, să se afișeze numarul de obiecte create (conținutul
variabilei var_static)

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include "Class.h"

int main()
{
	Contor c;

	cout << "Obiect: " << c.get_obiect() << endl;
```

```
	c.set_contor(66564);
	cout << "Obiect: " << c.get_obiect() << endl;

	c.set_contor(12);
	cout << "Obiect: " << c.get_obiect() << endl;

	c.set_contor(32);
	cout << "Obiect: " << c.get_obiect() << endl;

	c.set_contor(2555);
	cout << "Obiect: " << c.get_obiect() << endl;

	c.afisare();

	return 0;
}

//header Class.h
#pragma once

class Contor
{
	int obiect;
public:
	static int var_static;

	Contor()
	{
		obiect = 0;
		var_static++;
	}
	void set_contor(int val)
	{
		obiect = val;
		var_static++;
	}
	int get_obiect()
	{
		return obiect;
	}
	static void afisare()
	{
		cout << "\nAu fost instantiate " << var_static << " obiecte." << endl <<
endl;
	}
};

int Contor::var_static;
```

```
/* Lab 8  Popa Larisa-Ancuta  Prob 5

-rezolvați problema 4 în cazul în care variabila statică este de tip private
-definiți o metodă accesor care returnează valoarea contorului

*/
```

```cpp
//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;
#include "Class.h"

int main()
{
	Contor c;

	cout << "Obiect: " << c.get_obiect() << endl;

	c.set_contor(66564);
	cout << "Obiect: " << c.get_obiect() << endl;

	c.set_contor(12);
	cout << "Obiect: " << c.get_obiect() << endl;

	c.set_contor(32);
	cout << "Obiect: " << c.get_obiect() << endl;

	c.set_contor(2555);
	cout << "Obiect: " << c.get_obiect() << endl;

	cout << "\nAu fost instantiate " << c.get_var() << " obiecte." << endl << endl;

	return 0;
}
```

```cpp
//header Class.h
#pragma once
class Contor
{
	int obiect;
	static int var_static;
public:
	Contor()
	{
		obiect = 0;
		var_static++;
	}
	void set_contor(int val)
	{
		obiect = val;
		var_static++;
	}
	int get_obiect(void)
	{
		return obiect;
	}
	static int get_var(void)
	{
		return var_static;
	}
};

int Contor::var_static;
```

```
/*  Lab 8  Popa Larisa-Ancuta  Prob 6

-scrieți o aplicație C/C++ în care să implementați clasa Punct cu atributele private x si
y
-implementați o funcție friend care să calculeze aria și perimetrul a doua forme
geometrice definite de două puncte,
   considerând că aceasta are doua puncte ca si parametrii P0(x0,y0) și P1 (x1, y1)
-adăugați funcției un parametru întreg figura prin care să indicați cele doua figuri
geometrice ce sunt definite de punctele (x0, y0)
   si (x1, y1)
-astfel, pentru un cerc, figura=1, coordonatele (x0, y0) și (x1, y1) vor reprezenta două
puncte complementare pe cerc (diametrul)
-dacă este triunghi dreptunghic, punctele definesc ipotenuza iar figura va fi =2
-celelalte laturi ale triunghiului se vor determina pornind de la cele doua puncte
-coordonatele punctelor si apoi selecția figurii geometrice se va realiza introducând de
la tastatura parametrii.

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include "Class.h"

int main()
{
	Punct P1, P2;
	int x, y;

	cout << "\nCoordonatele punctului 1:";
	cout << "\nx= ";
	cin >> x;
	P1.set_x(x);
	cout << "y= ";
	cin >> y;
	P1.set_y(y);

	cout << "\nCoordonatele punctului 2:";
	cout << "\nx= ";
	cin >> x;
	P2.set_x(x);
	cout << "y= ";
	cin >> y;
	P2.set_y(y);

	int figura;
	cout << "\nIntroduceti figura dorita(1-cerc,2-triunghi): ";
	cin >> figura;

	cout << "\nPerimetrul: " << perimetru(P1, P2, figura);
	cout << "\nAria: " << aria(P1, P2, figura) << endl << endl;

	return 0;
}
```

```
//header Class.h
#pragma once

class Punct
{
private:
	int x, y;
public:
	Punct()
	{
		x = 0;
		y = 0;
	}
	void set_x(int val)
	{
		x = val;
	}
	void set_y(int val)
	{
		y = val;
	}
	int get_x(void)
	{
		return x;
	}
	int get_y(void)
	{
		return y;
	}

	friend float distanta(Punct, Punct);
	friend float perimetru(Punct, Punct, int);
	friend float aria(Punct, Punct, int);
};

float distanta(Punct P1, Punct P2)
{
	return sqrt((P1.x - P2.x)*(P1.x - P2.x) + (P1.y - P2.y)*(P1.y - P2.y));
}

float perimetru(Punct P1, Punct P2,int figura)
{
	if (figura == 1)
	{
		float raza = distanta(P1, P2);
		return 2 * 3.14*raza;
	}
	if (figura == 2)
	{
		Punct O;
		return distanta(P1, P2) + distanta(P1, O) + distanta(P2, O);
	}

	return 0;
}

float aria(Punct P1, Punct P2, int figura)
{
```

```
	if (figura == 1)
	{
		float raza = distanta(P1, P2);
		return 3.14*pow(raza, 2);
	}

	if (figura == 2)
	{
		Punct O;

		return (distanta(P1, O) * distanta(P2, O)) / distanta(P1, P2);
	}

	return 0;
}
```

```
/* Lab 8  Popa Larisa-Ancuta  Prob 8

-considerati clasa Fractie care are doua atribute intregi private a si b pentru numarator
si numitor, doua metode de tip set( )
   respectiv get( ) pentru fiecare din atributele clase
-declarati o functie friend simplifica( ) care are ca si parametru un obiect al clasei,
returnand un alt obiect simplificat
-considerați o variabila private statica întreagă 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 fractii (b!=0)
-definiti un destructor explicit care afiseaza si decrementeaza contorul icount
-definti o functie friend _f_aduna_fractie(...) care are ca si parametri doua obiecte de
tip Fractie si returneaza suma obiectelor
   in alt obiect Fractie
-analog definiti functii friend pentru scadere, inmultire si impartire
-instantiati doua obiecte de tip Fractie cu date citite de la tastatura
-sfisati atributele initiale si cele obtinute dupa apelul functiei simplifica( )
-printr-o metoda accesor, afisati contorul icount
-efectuati operatiile implementate prin functiile friend ale clasei, initializand alte 4
obiecte cu rezultatele obtinute
-afisati rezultatele si contorul dupa ultima operatie folosind o metoda accesor adecvata

*/

//main
#define  _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include  "Class.h"

int main()
{
	int val1, val2;

	cout << "Prima fractie:\n";
	cin >> val1;
```

```
	cin >> val2;
	Fractie f1(val1, val2);

	cout << "A doua fractie:\n";
	cin >> val1;
	cin >> val2;
	Fractie f2(val1, val2);

	cout << "\nPrima fractie: " << f1.get_a() << "/" << f1.get_b();
	cout << "\nA doua fractie: " << f2.get_a() << "/" << f2.get_b();

	cout << "\n\nDupa simplificare\nPrima fractie: " << simplifica(f1).get_a() << "/"
<< simplifica(f1).get_b();
	cout << "\nA doua fractie: " << simplifica(f2).get_a() << "/" <<
simplifica(f2).get_b();

	cout << "\n\nIcount: " << f1.get_icount() << endl;

	Fractie S = aduna_fractie(f1, f2);
	cout << "\nSuma: " << S.get_a() << "/" << S.get_b();

	Fractie D = scade_fractie(f1, f2);
	cout << "\nDiferenta: " << D.get_a() << "/" << D.get_b();

	Fractie P = inmultire_fractie(f1, f2);
	cout << "\nProdus: " << P.get_a() << "/" << P.get_b();

	Fractie C = impartire_fractie(f1, f2);
	cout << "\nCat: " << C.get_a() << "/" << C.get_b();

	cout << "\n\nIcount: " << f1.get_icount() << endl << endl;
	return 0;
}

//header Class.h
#pragma once

class Fractie
{
private:
	int a, b;
	static int icount;
public:
	Fractie()
	{
		a = 0;
		b = 1;
		icount++;
	}
	Fractie(int val1, int val2)
	{
		if (val2 != 0)
		{
			a = val1;
			b = val2;
			icount++;
		}
		else
```

```
                cout << "\nNumitorul trebuie sa fie diferit de 0!\n\n";
        }
        /*~Fractie()
        {
                cout << "\n\nicount= " << icount << "\n\n";
                icount = 0;
        }*/

        void set_a(int val)
        {
                a = val;
        }
        void set_b(int val)
        {
                b = val;
        }

        int get_a()
        {
                return a;
        }
        int get_b()
        {
                return b;
        }
        static int get_icount()
        {
                return icount;
        }

        friend Fractie simplifica(Fractie &);

        friend Fractie aduna_fractie(Fractie, Fractie);
        friend Fractie scade_fractie(Fractie, Fractie);
        friend Fractie inmultire_fractie(Fractie, Fractie);
        friend Fractie impartire_fractie(Fractie, Fractie);
};

Fractie simplifica(Fractie &fr)
{
        Fractie new_fr;
        int c, x = fr.a, y = fr.b;

        while (y)
        {
                c = x % y;
                x = y;
                y = c;
        }

        new_fr.a = fr.a / x;
        new_fr.b = fr.b / x;

        return new_fr;
}
```

```
Fractie aduna_fractie(Fractie fr1, Fractie fr2)
{
	Fractie suma;

	if (fr1.b != fr2.b)
	{
		suma.a = fr1.a*fr2.b + fr1.b*fr2.a;
		suma.b = fr1.b*fr2.b;
	}
	else
	{
		suma.a = fr1.a + fr2.a;
		suma.b = fr1.b;
	}

	return suma;
}

Fractie scade_fractie(Fractie fr1, Fractie fr2)
{
	Fractie dif;

	if (fr1.b != fr2.b)
	{
		dif.a = fr1.a*fr2.b - fr1.b*fr2.a;
		dif.b = fr1.b*fr2.b;
	}
	else
	{
		dif.a = fr1.a - fr2.a;
		dif.b = fr1.b;
	}

	return dif;
}

Fractie inmultire_fractie(Fractie fr1, Fractie fr2)
{
	Fractie prod;

	prod.a = fr1.a*fr2.a;
	prod.b = fr1.b*fr2.b;

	return prod;
}

Fractie impartire_fractie(Fractie fr1, Fractie fr2)
{
	Fractie imp;

	imp.a = fr1.a*fr2.b;
	imp.b = fr1.b*fr2.a;

	return imp;
}

int Fractie::icount = 0;
```