Laborator 9

```
/* Lab 9  Popa Larisa-Ancuta  Prob 5

-să se scrie programul care considera o clasa MyClass cu trei atribute de tip int
-clasa considera pe baza mecanismului de supraincarcare metode publice int myFunction(…),
care în funcţie de numărul de parametri
   primiţi, returnează fie valoarea primită (1 parametru), fie produsul variabilelor de
intrare (0-toti, 2, 3 parametrii)
-instantiati un obiect din clasa in main(), setati atributele cu metode setter adecvate
din clasa si afisati valorile la apelurile metodelor

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include "Class.h"

int main()
{
	MyClass o;
	int val;

	cout << "\nIntroduceti prima valoare: ";
	cin >> val;
	o.set_x(val);
	cout << "\nIntroduceti a doua valoare: ";
	cin >> val;
	o.set_y(val);
	cout << "\nIntroduceti a treia valoare: ";
	cin >> val;
	o.set_z(val);

	cout << "\n\nApel fara parametrii: " << o.myFunction();
	cout << "\nApel cu un parametru: " << o.myFunction(o.get_x());
	cout << "\nApel cu doi parametrii: " << o.myFunction(o.get_x(), o.get_y());
	cout << "\nApel cu doi parametrii: " << o.myFunction(o.get_x(), o.get_y(),
o.get_z()) << endl << endl;

	return 0;
}

//Class.h
#pragma once

class MyClass
{
	int x, y, z;

public:
	void set_x(int val) { x = val; }
	void set_y(int val) { y = val; }
	void set_z(int val) { z = val; }
```

```
        int get_x(void) { return x; }
        int get_y(void) { return y; }
        int get_z(void) { return z; }

        int myFunction()
        {
                return x*y*z;
        }
        int myFunction(int val)
        {
                return val;
        }
        int myFunction(int val1, int val2)
        {
                return val1*val2;
        }
        int myFunction(int val1, int val2, int val3)
        {
                return val1*val2*val3;
        }
};
```

```
/* Lab 9  Popa Larisa-Ancuta  Prob 6

Să se scrie programul care utilizează o clasă numită Calculator și care are în componența
sa metodele publice supraincarcate:
  - int calcul(int x) care returnează pătratul valorii primite;
  - int calcul(int x, int y) care returnează produsul celor două valori primite;
  - double calcul(int x, int y, int z) care returnează rezultatul înlocuirii în formula
f(x,y,z) = (x-y)(x+z)/2. a valorilor primite;
Programul primește din linia de comandă toți parametrii necesari pentru toate aceste
metode ale clasei.
Considerați și cazul în care toate aceste metode sunt statice.
E posibil să aveți în același timp metode publice statice și non-statice?
Analizati si cazul in care clasa are 3 atribute private de tip int, x, y, z, care sunt
modificate cu metode setter adecvate.
Ce trebuie sa modificati pentru a putea efectua operatiile cerute?

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include "Class.h"

int main(int argc, char *argv[])
{
        int x = atoi(argv[1]), y = atoi(argv[2]), z = atoi(argv[3]);
        cout << "x= " << x << "\ny= " << y << "\nz= " << z << endl;

        CalculatorPublic calc1;

        cout << "\nCalculator public:";
```

```
	cout << "\nx^2= " << calc1.calcul(x) << "\ny^2= " << calc1.calcul(y) << "\nz^2= "
<< calc1.calcul(z) << endl;
	cout << "\nx*y= " << calc1.calcul(x, y) << "\ny*z= " << calc1.calcul(y, z) <<
"\nx*z= " << calc1.calcul(x, z) << endl;
	cout << "\nf(x,y,z)= (x-y)(x+z)/2= " << calc1.calcul(x, y, z) << endl;

	CalculatorPrivate calc2;
	calc2.set_x(x);
	calc2.set_y(y);
	calc2.set_z(z);

	cout << "\nCalculator private:";
	cout << "\nx^2= " << calc2.calcul(calc2.get_x()) << "\ny^2= " <<
calc2.calcul(calc2.get_y()) << "\nz^2= " << calc2.calcul(calc2.get_y()) << endl;
	cout << "\nx*y= " << calc2.calcul(calc2.get_x(), calc2.get_y()) << "\ny*z= " <<
calc2.calcul(calc2.get_y(), calc2.get_z()) << "\nx*z= " << calc2.calcul(calc2.get_x(),
calc2.get_z()) << endl;
	cout << "\nf(x,y,z)= (x-y)(x+z)/2= " << calc2.calcul(calc2.get_x(), calc2.get_y(),
calc2.get_z()) << endl << endl;

	return 0;
}

//Class.h
#pragma once

class CalculatorPublic
{
public:
	int calcul(int x) { return x * x; }
	int calcul(int x, int y) { return x * y; };
	double calcul(int x, int y, int z) { return(double)(x - y)*(x + z) / 2; }
};

class CalculatorPrivate
{
	int x, y, z;
public:
	CalculatorPrivate()
	{
		x = y = z = 0;
	}

	void set_x(int val) { x = val; }
	void set_y(int val) { y = val; }
	void set_z(int val) { z = val; }

	int get_x() { return x; }
	int get_y() { return y; }
	int get_z() { return z; }

	int calcul(int x) { return x * x; }
	int calcul(int x, int y) { return x * y; };
	double calcul(int x, int y, int z) { return (double)(x - y)*(x + z) / 2; }
};
```

```
/*  Lab 9  Popa Larisa-Ancuta  Prob 8

-să se implementeze clasa Complex care supraîncarcă operatorii aritmetici cu scopul de a
efectua adunări, scăderi, inmulțiri și împărțiri
   de numere complexe (folosind metode membre (+, -) și funcții friend (*, /))
-Observație: numerele complexe vor fi definite ca având o parte reală și una imaginară,
ambii coeficienți fiind reprezentați prin numere reale

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include "Complex.h"

int main()
{
	int Re, Im;
	Complex z1, z2;

	cout << "\nPrimul numar complex: \n  Re=";
	cin >> Re;
	cout << "  Im=";
	cin >> Im;
	z1.setIm(Im);
	z1.setRe(Re);

	cout << "\nAl doilea numar complex: \n  Re=";
	cin >> Re;
	cout << "  Im=";
	cin >> Im;
	z2.setIm(Im);
	z2.setRe(Re);

	Complex sum = z1 + z2;
	cout << "\n\nSuma este: " << sum.getRe() << "+" << sum.getIm() << "i";

	Complex dif = z1 - z2;
	cout << "\nDiferenta este: " << dif.getRe() << "+" << dif.getIm() << "i";

	Complex prod = z1 * z2;
	cout << "\nProdusul este: " << prod.getRe() << "+" << prod.getIm() << "i";

	Complex div = z1 / z2;
	cout << "\nCatul este: " << div.getRe() << "+" << div.getIm() << "i" << endl <<
endl;

	return 0;
}

//Complex.h
#pragma once

class Complex
{
```

```
private:
	double Re, Im;
public:
	Complex()
	{
		Re = 0;
		Im = 0;
	}

	void setRe(double r) { Re = r; }
	void setIm(double i) { Im = i; }

	double getRe() { return Re; }
	double getIm() { return Im; }

	Complex operator+(Complex z)
	{
		Complex sum;

		sum.Re = Re + z.Re;
		sum.Im = Im + z.Im;

		return sum;
	}
	Complex operator-(Complex z)
	{
		Complex dif;

		dif.Re = Re - z.Re;
		dif.Im = Im - z.Im;

		return dif;
	}

	friend Complex operator*(Complex z1, Complex z2);
	friend Complex operator/(Complex z1, Complex z2);
};

Complex operator*(Complex z1, Complex z2)
{
	Complex prod;

	prod.Re = z1.Re*z2.Re - z1.Im*z2.Im;
	prod.Im = z1.Re*z2.Im + z1.Im*z2.Re;

	return prod;
}

Complex operator/(Complex z1, Complex z2)
{
	Complex div;

	div.Re = (z1.Re*z2.Re + z1.Im*z2.Im) / (z2.Re*z2.Re + z2.Im*z2.Im);
	div.Im = (z1.Im*z2.Re - z1.Re*z2.Im) / (z2.Re*z2.Re + z2.Im*z2.Im);

	return div;
}
```

```
/*  Lab 9  Popa Larisa-Ancuta  Prob 9

-definți o clasă numită Number care are o variabilă private de tip double
-clasa mai conține un constructor explicit vid și unul cu un parametru și o metodă
accesor care afișează valoarea variabilei din clasă
-scrieți o clasă numită Mathemathics, care supraîncarcă operatorii specifici operațiilor
aritmetice elementare (+, -, *, /)
-clasa are ca atribut un obiect instanțiat din prima clasa
-operațiile artimetice se efectuează asupra datelor obținute din obiectul de tip Number

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include "Number.h"

int main()
{
	double val;

	cout << "\nIntroduceti prima valoare: ";
	cin >> val;
	Number nr1(val);
	Mathematics M1(nr1);

	cout << "\nIntroduceti a doua valoare: ";
	cin >> val;
	Number nr2(val);
	Mathematics M2(nr2);

	Mathematics sum = M1 + M2;
	cout << "\n\nSuma este: " << sum.o.getNr();

	Mathematics dif = M1 - M2;
	cout << "\nDiferenta este: " << dif.o.getNr();

	Mathematics prod = M1 * M2;
	cout << "\nProdusul este: " << prod.o.getNr();

	Mathematics div = M1 / M2;
	cout << "\nCatul este: " << div.o.getNr() << endl << endl;

	return 0;
}

//Number.h
#pragma once

class Number
{
private:
	double nr;
public:
	Number()
```

```
	{
		nr = 0;
	}
	Number(double val)
	{
		nr = val;
	}

	void setNr(double val) { nr = val; }
	double getNr(void) { return nr; }

};

class Mathematics
{
public:
	Number o;

	Mathematics() { o; }
	Mathematics(Number x) { o = x; }

	Number getO(void) { return o; }

	Mathematics operator+(Mathematics m)
	{
		Number x = o.getNr() + m.o.getNr();

		return x;
	}

	Mathematics operator-(Mathematics m)
	{
		Number x = o.getNr() - m.o.getNr();

		return x;
	}

	Mathematics operator*(Mathematics m)
	{
		Number x = o.getNr() * m.o.getNr();

		return x;
	}

	Mathematics operator/(Mathematics m)
	{
		Number x = o.getNr() / m.o.getNr();

		return x;
	}
};
```