RO: Supraîncărcarea metodelor şi operatorilor.

EN: Methods and operators overloading.

Obiective:

- Capacitatea de a utiliza metode specifice din cadrul clasei si funcţii şi clase prietene (friend) ai unei clase pentru supraîncărcari de operatori;
- Înţelegerea practică a noţiunii de polimorfism prin implementarea de programe cu supraîncărcari de metode şi operatori.

Objectives:

- The ability of using specific member methods and friend classes and functions for operators overloading;
- Practical understanding of polymorphism by implementing programs that overload methods and operators.

Rezumat:

Supraîncărcarea metodelor (funcţiilor membre) (*overloading*) este unul dintre atributele esenţiale ale limbajului C/C++, conferindu-i flexibilitate în implementarea programelor; ea implementează conceptul de *polimorfism* static.

Termenul de *polimorfism* denotă acea caracteristică a unei anumite componente soft de a avea "mai multe forme", adică de a conduce fluxul de execuţie spre o anumită zonă de program care este specializată în prelucrarea datelor de intrare primite.

Operatorii se supraîncarcă prin crearea metodelor/functiilor-friend cu nume *operator*. O metoda/funcţie *operator* defineşte operaţiile specifice pe care le va efectua operatorul respectiv, relativ la clasa în care a fost destinat să lucreze.

Se pot supraincarca aproape toti operatorii cu unele exceptii. Pot fi supraîncărcaţi si operatorii *new* şi *delete*, însă cu metode (funcţii membre) statice, fără a specifica explicit aceasta. Scopul supraîncărcării acestor operatori este de a efectua anumite operaţii speciale de alocare de memorie în momentul creării sau distrugerii unui obiect din clasa în care aceştia au fost supraîncărcaţi.

Exemple:

```
//1.a  Supraincarcarea constructorilor; supraincarcarea operatorilor +, -, si [ ] pentru clasa String
//String_static.h

const int max = 30;

class String {
        char sir[max];
public:
        String() { };//sau se poate folosi constr explicit default, String()=default;
        String(char x[max]) {
                strcpy(sir, x);
        }
        char* getSir() {
                return sir;
        }

        String operator+ (String x1) {          // supraincarcare cu metoda membra
                String rez;
                strcpy(rez.sir, sir);//daca suma depaseste max de caractere, probleme
                strcat(rez.sir, x1.sir);
                return rez;
        }

        //functia prietena care supraincarca operatorul de scadere este definite in exteriorul clasei mai jos
        friend String operator- (String& x1, String& x2);

        char operator[ ](int poz) { // supraincarcarea op [ ] pt. returnarea unui
                char rez;           // caracter din pozitia data in sir
                rez = sir[poz];
```

```
		return rez;
	}
};

// supraincarc. op. prin f-ctie friend, in exteriorul clasei
String operator- (String& x1, String& x2) {
	char* pp;
	pp = strstr(x1.sir, x2.sir);  // adresa unde începe sirul x2.sir in sirul x1.sir
	if (pp == NULL)  // sirul nu se gaseste in x1.sir
		return String(x1.sir);
	else {
		char buf[max];
		strncpy(buf, x1.sir, pp - x1.sir);
		strcpy(buf + (pp - x1.sir), pp + strlen(x2.sir));
		//strcpy_s(buf + (pp - x1.sir), max - (pp - x1.sir), pp + strlen(x2.sir));
		return String(buf);
	}
}

//main()
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
#include "String_static.h"

int main( ) {
	char xx[max];
	char c;
	cout << "\n Dati un sir de caractere (I obiect): ";
	gets_s(xx, _countof(xx));
	String ob1(xx);
	cout << "\n Dati un sir de caractere (al II-lea obiect): ";
	cin.getline(xx, _countof(xx));
	String ob2(xx);
	String ob3;//ob3 instantiat
	ob3 = ob1 + ob2;//assign implicit
	cout << "\nSirul obtinut in urma adunarii: " << ob3.getSir();
	cout << "\n Sirul care va fi scazut din cel initial: ";
	gets_s(xx, _countof(xx));
	String ob4(xx);
	String ob5 = ob1 - ob4;//copy constructor implicit
	cout << "\nSirul obtinut in urma scaderii: " << ob5.getSir();
	c = ob5[1];
	cout << "\n Al 2-lea caracter din sirul de mai sus este: " << c;
	}

//1.b  Supraincarcarea constructorilor; alocare dinamica
//StringA.h

const int dim = 30;

class String {
	char* sir;
public:
	String() {
		sir = new char[dim];
	}
	String(char* x) {
		sir = new char[strlen(x) + 1];//se mai rezerva pentru \0 – destructorul va lucra corect
		strcpy(sir, x);
	}
/*
	String(const String& x) {//copy constructor
```

```
		sir = new char[strlen(x.sir) + 1];
		strcpy(sir, x.sir);
	}
	String& operator= (const String& x) {//assign
		if (this == &x)	return *this;
		delete[] sir;//eliberez sir din obiectul current, echivalent cu: this-> ~String();
		sir = new char[strlen(x.sir) + 1];//aloc spatiu pentru sir pornind de la  sirul de asignat
		strcpy(sir, x.sir);
		return *this;
	}
	~String() {
		delete []sir;
	}
*/
	void setSir(char* x) {
		strcpy(sir, x);
	}
	char* getSir() {
		return sir;
	}
	String operator+ (String x1) {  // supraincarcare cu metoda membra
		String rez;
		rez.~String();
		rez.sir = new char[strlen(this->sir)+strlen(x1.sir) + 1];
		strcpy(rez.sir, sir);
		strcat(rez.sir, x1.sir);
		return rez;
	}

	//functia prietena care supraincarca operatorul de scadere este definite in exteriorul clasei mai jos
	friend String operator- (const String& x1, const String& x2);

	char operator[ ](int poz) { // supraincarcarea op [ ] pt. returnarea unui
		char rez;		// caracter din pozitia data in sir
		rez = sir[poz];
		return rez;
	}
};

// supraincarc. op. prin f-ctie friend, in exteriorul clasei
String operator- (const String& x1, const String& x2) {
	char* pp;
	pp = strstr(x1.sir, x2.sir);  // adresa unde începe sirul x2.sir in sirul x1.sir
	if (pp == NULL)  // sirul nu se gaseste in x1.sir
		return String(x1.sir);
	else {
		//char buf[dim];
		char* buf = new char[strlen(x1.sir) + 1];
		strncpy(buf, x1.sir, pp - x1.sir);
		strcpy(buf + (pp - x1.sir), pp + strlen(x2.sir));
		return String(buf);
	}
}

//main( )
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
#include "StringA.h"

int main() {
	char xx[dim];
	char c;
```

```
	cout << "\nDati un sir de caractere (ob1), mai mic de " << dim << " :";
	gets_s(xx, _countof(xx));
	String ob1(xx);
	cout << "Dati un sir de caractere (ob2), mai mic de " << dim << " :";
	cin.getline(xx, _countof(xx));
	String ob2(xx);
	String ob3;
	ob3 = ob1 + ob2;//assign
	cout << "Sirul (ob3) obtinut in urma adunarii prin supraincarcare operator = : " << ob3.getSir();
	cout << "\n Dati un sir (ob4) care va fi scazut din cel initial, mai mic de " << dim << " :";
	gets_s(xx, _countof(xx));
	String ob4(xx);
	String ob5 = ob1 - ob4;//copy constructor
	cout << "Sirul obtinut (ob5) in urma scaderii cu copy constructor: " << ob5.getSir();
	c = ob5[1];
	cout << "\n Al 2-lea caracter din sirul de mai sus este: " << c;
	String ob6;
	cout << "\n Dati un sir de caractere (ob6) pentru un obiect, mai mic de " << dim << " :";
	cin.getline(xx, _countof(xx));
	ob6.setSir(xx);
	cout << "Sirul : " << ob6.getSir();
	ob3.setSir(xx);
	cout << "\nSirul ob3 setat la ob6: " << ob3.getSir();
}
```

Obs: Pentru functionarea corecta la var. 1.b e nevoie a supraincarca operatorul de asignare pentru clasa String, copy constructorul si s-a implementat un destructor pentru a elibera obiectele.
Tema: *Modificati supraincarcarea operatorului – care sa permita scaderea a mai multor aparitii din sirul initial*

```
//*********************************************************************************************
//2. Supraincarcarea operatorilor new si delete in cazul clasei Pozitie care controleaza pozitia unui punct in plan
//Pozitie.h

class Pozitie {
	int x;
	int y;
public:
	Pozitie( ) { }
	Pozitie(int oriz, int vert) {
		x = oriz;
		y = vert;
	}
	int getX( ) {
		return x;
	}
	int getY( ) {
		return y;
	}

	void *operator new(size_t marime);
	void operator delete(void *p);
};

void *Pozitie :: operator new(size_t marime) {
	cout<<"\n Supraincarc local  operatorul new.";
	return malloc(marime);
}

void Pozitie :: operator delete(void *p) {
	cout<<"\n Supraincarc local operatorul delete.";
	free(p);
}
//main( )
```

```
#include <iostream>
using namespace std;
#include "Pozitie.h"


int main( ) {
	Pozitie *p1, *p2;
	p1 = new Pozitie(100, 200);
	if (!p1) {
		cout<<"\n Nu am putut aloca memorie pt. P1.";
		exit(0);
	}
	p2 = new Pozitie(10, 20);
	if (!p2) {
		cout<<"\n Nu am putut aloca memorie pt. P2.";
		exit(1);
	}
	cout << "\nPozitia in plan P1 are coordonatele: " << p1->getX( ) << " " << p1->getY( );
	cout << "\nPozitia in plan P2 are coordonatele: " << p2->getX( ) << " " << p2->getY( );
	delete(p1);
	delete(p2);
	}


//*************************************************************************
3. Supraincarcarea operatorilor ++ si --
//Time_inc_dec.h


class Time
{
private:
	int hh;
	int mm;
public:
	Time(int m) {
		if (m >= 0) {
			hh = m / 60;
			mm = m % 60;
		}
		else {
			hh = mm = 0;
		}
	}

	Time(int h = 0, int m = 0) {
		if (((h >= 0) && (h < 24)) && ((m >= 0) && (m < 60))) {
			hh = h;
			mm = m;
		}
		else {
			hh = mm = 0;
		}
	}

	int getH( ) {
		return hh;
	}

	void setH(int h) {
		if ((h >= 0) && (h < 24))
			hh = h;
		else
```

```
                hh = 0;
}

int getM( ) {
        return mm;
}

void setM(int m) {
        if ((m >= 0) && (m < 60))
                mm = m;
        else
                mm = 0;
}

void Show( ) {
        cout << hh << ":" << mm << endl;
}

// supraincarcare operator de incrementare prefixata
Time& operator++( ) {
        mm++;
        if (mm == 60)
        {
                hh++;
                if (hh == 24)
                        hh = 0;
                mm = 0;
        }
        return *this;
}

// supraincarcare operator de incrementare postfixata
Time operator++(int) {
        Time temp = *this;
        ++(*this);
        return temp;
}

// supraincarcare operator de decrementare prefixata
Time& operator--( ) {
        if (mm == 0)
        {
                hh--;
                if (hh < 0)
                        hh = 23;
                mm = 59;
        }
        else
                mm--;
        return *this;
}

// supraincarcare operator de decrementare postfixata
Time operator--(int) {
        Time temp = *this;
        --(*this);
        return temp;
}

// supraincarcare operator de atribuire compusa +=
Time& operator+=(int min) {
        int h1, m1;
        h1 = min / 60;
```

```
		m1 = min % 60;
		hh = (hh + h1 + ((mm + m1) / 60)) % 24;
		mm = (mm + m1) % 60;
		return *this;
	}

	// supraincarcare operator de atribuire compusa -=
	Time& operator-=(int min) {
		int h1, m1;
		h1 = min / 60;
		m1 = min % 60;
		hh = hh - h1;
		if (mm < m1)
			hh--;
		if (hh < 0)
			hh = 24 + hh;
		mm = mm - m1;
		if (mm < 0)
			mm = 60 + mm;
		return *this;
	}

	// supraincarcare operator de adunare +
	Time operator+(Time tm) {
		Time rez;
		int nm = mm + tm.mm;
		rez.hh = hh + tm.hh + nm / 60;
		if (rez.hh > 23)
			rez.hh = rez.hh - 24;
		rez.mm = nm % 60;
		return rez;
	}

	// supraincarcare operator de scadere-
	Time operator-(Time tm) {
		Time rez;
		if (mm < tm.mm)
		{
			rez.hh = hh - tm.hh - 1;
			rez.mm = 60 - (tm.mm - mm);
		}
		else
		{
			rez.hh = hh - tm.hh;
			rez.mm = mm - tm.mm;
		}
		if (rez.hh < 0)
			rez.hh = 24 + rez.hh;
		return rez;
	}
};

//main( )
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
#include "Time_inc_dec.h"

int main( )
{
	Time tm1(1, 29);
	cout << "Time 1: ";
	tm1.Show( );
```

```
cout << endl;
Time tm2;
tm2 = --tm1;
cout << "Time 1 (tm2 = --tm1): ";
tm1.Show( );
cout << "Time 2 (tm2 = --tm1): ";
tm2.Show( );
cout << endl;
tm1 = ++tm2;
cout << "Time 1: (tm1 = ++tm2)";
tm1.Show( );
cout << "Time 2: (tm1 = ++tm2)";
tm2.Show( );

cout << endl;
tm1 = tm2++;
cout << "Time 1 (tm1 = tm2++): ";
tm1.Show( );
cout << "Time 2 (tm1 = tm2++): ";
tm2.Show( );
tm2--;
cout << "Time 2 (tm2--): ";
tm2.Show( );

cout << endl;
Time tm3(22, 15);
cout << "Time 3: ";
tm3.Show( );

cout << endl;
tm3 += 75;
cout << "Time 3 (+=75): ";
tm3.Show( );
tm3 -= 86;
cout << "Time 3 (-=86): ";
tm3.Show( );

Time tm4(23, 18);
cout << endl;
cout << "Time 4: ";
tm4.Show( );
cout << "Time 3 + Time 4 : ";
(tm3 + tm4).Show();
cout << "Time 3 - Time 4 : ";
(tm3 - tm4).Show( );
}
```

***Obs**: Inlocuiti metoda Show() cu metode getter apelate in main( ).*

```
//*********************************************************************************
//4.a Supraincarcare [ ] pentru acces bazat pe indexare – relatie de dependenta clase friend
//Pers_analize.h
const int max = 30;//nr. caractere
const int dim = 5;//nr. obiecte implicit

class Analize;
class Pers{
        //date personale
        char nume[max];
        double greutate;
        int varsta;
        friend class Analize;
        //metoda de  afisare
```

```
public:
		void tip( );
};// Pers

void Pers::tip( ){
if (this)cout << "\nPersoana : " << nume << "\tGreutatea : " << greutate << "\tVarsta :  "<<varsta;
		else cout << "\nNu exista obiect ";
}

class Analize{
		Pers *sir;
		int n;
public:
		//constructori
		Analize( ){
				n = dim;
				sir = new Pers[dim];
		}
		Analize(int nr){
				n = nr;
				sir = new Pers[n];
		}
		//Supraincarcare [ ]
		Pers* operator[ ](char *);
		Pers* operator[ ](double);
		Pers* operator[ ](int);

		void introdu( );
};//Analize

void Analize::introdu( ){
		cout << "\nIntrodu date :\n";
		for (int i = 0; i<n; i++)
		{
				cout << "Numele persoanei" << (i + 1) << " : ";
				cin >> sir[i].nume;
				cout << "Greutatea : ";
				cin >> sir[i].greutate;
				cout << "Varsta : ";
				cin >> sir[i].varsta;
		}
}//introdu

 //Indexare dupa nume
Pers* Analize:: operator[ ](char *nume){
		for (int i = 0; i<n; i++)
				if (strcmp(sir[i].nume, nume) == 0)return &sir[i];
		return NULL;
}//op[ ]nume

 //Indexare dupa greutate
Pers* Analize:: operator[ ](double gr){
		for (int i = 0; i<n; i++)
				if (sir[i].greutate == gr)return &sir[i];
		return NULL;
}//op[ ]greutate

 //Indexare dupa index
Pers* Analize:: operator[ ](int index){
		if ((index >= 1) && (index <= n))return &sir[index - 1];
		else { cout << "\nIndex gresit"; return NULL; }
}//op[ ]index
```

```
//main()
#include <iostream>
using namespace std;
#include "Pers_analize.h"

int main( ){
	char c;
	int nr;
	cout << "\nIntrodu nr. obiecte ce vor fi create : ";
	cin >> nr;
	Analize t(nr);//creare obiecte-rezervare
	t.introdu();//dau datele
	while (1)
	{
		cout << "\nOptiunea (g-greutate, n-nume, i-index, e-exit) ?";
		cin >> c;
		switch (toupper(c))
		{
		case 'G':double g;
			cout << "Greutatea: ";
			cin >> g;
			t[g]->tip();
			break;
		case 'N':char n[max];
			cout << "Numele: ";
			cin >> n;
			t[n]->tip();
			break;
		case 'I':int i;
			cout << "Nr index: ";
			cin >> i;
			t[i]->tip();
			break;
		case 'E':return 0;
		}//end switch-case
	}//end while
}//main


//4.b varianta 2: Supraincarcare [ ] pentru acces bazat pe indexare – relatie de asociere

//Pers_analize.h

const int max = 30;//nr. caractere
const int dim = 5;//nr. obiecte implicit

class Persoana {
	char nume[max];
	double greutate;
	int varsta;
public:
	Persoana() {
		strcpy(nume, "Necunoscut");
		greutate = 0.0;
		varsta = 0;
	}

	Persoana(char* nume, double greutate, int varsta) {
		strcpy(this->nume, nume);
		this->greutate = greutate;
		this->varsta = varsta;
	}
	char* getNume() {
```

```
		return nume;
	}
	double getGreutate() {
		return greutate;
	}
	int getVarsta() {
		return varsta;
	}

	void display() {
		cout << "\nNume: " << nume;
		cout << "\nGreutate: " << greutate;
		cout << "\nVarsta: " << varsta;
	}
};

//Analize si Persoana sunt in relatie de asociere
class Analize {
	Persoana* p;
	int n;
public:
	Analize() {
		p = new Persoana[dim];
		n = dim;
	}

	Analize(int j) {
		p = new Persoana[j];
		n = j;
	}
	void introduce() {
		int j;
		char nume[max];
		double greutate;
		int varsta;
		for (j = 0; j < n; j++) {
			cout << "\nDatele pentru persoana: " << j + 1; cout << "\nNume: ";
			cin >> nume;
			cout << "\nGreutate: ";
			cin >> greutate;
			cout << "\nVarsta: ";
			cin >> varsta;
			p[j] = Persoana(nume, greutate, varsta);
		}
	}

	void operator[ ](char* nume) {
		int j;
		for (j = 0; j < n; j++)
			if (strcmp(nume, p[j].getNume()) == 0) p[j].display();
	}

	void operator[ ](double greutate) {
		int j;
		for (j = 0; j < n; j++)
			if (greutate == p[j].getGreutate()) p[j].display();
	}

	void operator[ ](int varsta) {
		int j;
		for (j = 0; j < n; j++)
			if (varsta == p[j].getVarsta()) p[j].display();
	}
```

```
};

//main( )
#define _CRT_SECURE_NO_WARNINGS#include <iostream>
using namespace std;
#include "Pers_analize.h"

int main() {
	int n;
	char c;
	char nume[max];
	double greutate;
	int varsta;
	cout << "\nCate persoane? ";
	cin >> n;
	Analize a(n);
	a.introduce();
	cout << "Cautare dupa (v = varsta, g = greutate, n = nume, e=exit)? ";
	cin >> c;
	switch (toupper(c)) {
	case 'V': { cout << "\nVarsta: "; cin >> varsta; a[varsta]; break; }
	case 'G': { cout << "\nGreutate: "; cin >> greutate; a[greutate]; break; }
	case 'N': { cout << "\nNume: "; cin >> nume; a[nume]; break; }
	case 'E': return 0;
	}
}

//*****************************************************************************
//5. Implementarea clasei Matrix pt. efectuarea unor operatii cu matrici
// prin supraincarcarea unor operatori (=, +, -, *, () )
//Matrix.h

const int linii = 2;
const int coloane = 3;

class Matrix {
	int rows;
	int cols;
	int* elems;
public:
	Matrix();
	Matrix(int rows, int cols);
	Matrix(const Matrix&);
	~Matrix(void) { delete [ ] elems; }
	int& operator () (int row, int col);
	Matrix& operator=(const Matrix&);
	friend Matrix operator+(Matrix&, Matrix&);
	friend Matrix operator-(Matrix&, Matrix&);
	friend Matrix operator*(Matrix&, Matrix&);
	int getRows(void) { return rows; }
	int getCols(void) { return cols; }
	void init(int r, int c);
	void citire();
	void afisare();
};//Matrix

Matrix::Matrix() : rows(linii), cols(coloane)
{
	elems = new int[rows * cols];
}
Matrix::Matrix(int r, int c) : rows(r), cols(c)
{
	elems = new int[rows * cols];
```

```
}

Matrix::Matrix(const Matrix& m) : rows(m.rows), cols(m.cols)
{
	int n = m.rows * m.cols;
	elems = new int[n];
	for (int i = 0; i < n; i++)
		elems[i] = m.elems[i];
}
Matrix& Matrix:: operator=(const Matrix &m) {
	if (this != &m){
	delete [ ] elems;// this-> ~Matrix();
	rows=m.rows; cols=m.cols;
		int n = rows * cols;
		elems = new int[n];
		for (int i = 0; i < n; i++)
			elems[i] = m.elems[i];
	}
	return *this;
}

void Matrix::init(int r, int c) {
	rows = r;
	cols = c;
	elems = new int[rows * cols];
}
int& Matrix:: operator()(int row, int col)
{
	return elems[row * cols + col];
}
Matrix operator+(Matrix& p, Matrix& q) {
	Matrix m(p.rows, p.cols);
	for (int r = 0; r < p.rows; ++r)
		for (int c = 0; c < p.cols; ++c)
			m(r, c) = p(r, c) + q(r, c);
	return m;
}
Matrix operator-(Matrix& p, Matrix& q) {
	Matrix m(p.rows, p.cols);
	for (int r = 0; r < p.rows; ++r)
		for (int c = 0; c < p.cols; ++c)
			m(r, c) = p(r, c) - q(r, c);
	return m;
}//op-
Matrix operator*(Matrix& p, Matrix& q) {
	Matrix m(p.rows, q.cols);
	for (int r = 0; r < p.rows; ++r)
		for (int c = 0; c < q.cols; ++c) {
			m(r, c) = 0;
			for (int i = 0; i < p.cols; ++i)
				m(r, c) += p(r, i) * q(i, c);
		}
	return m;
}//op*
void Matrix::citire() {
	for (int i = 0; i < rows; i++)
		for (int j = 0; j < cols; j++) {
			cout << "Dati elem. [" << i << "][" << j << "]  ";
			cin >> elems[cols * i + j];
		}
}//citire
void Matrix::afisare() {
	for (int i = 0; i < rows; i++)
```

```
	{
		for (int j = 0; j < cols; j++)
			cout << elems[cols * i + j] << "\t";
		cout << endl;
	}
}//afisare

//main()
#include<iostream>
using namespace std;
#include "Matrix.h"

int main( ) {
	int i, j;
	Matrix m(linii, coloane);
	cout << endl << "Supraincarcarea operatorului() pentru atribuirea unei valori pentru fiecare element din
matrice: \n";
	for (int i = 0; i < linii; i++)
		for (int j = 0; j < coloane; j++)
			m(i, j) = i + (j + 1) * 10;
	for (i = 0; i < linii; i++)
	{
		for (j = 0; j < coloane; j++)
			cout << m(i, j) << "\t";
		cout << endl;
	}
	int l, c;
cout << "Verificarea supraincarcarii operatorului () pentru un element de pe o pozitie citita de la tastatura" << endl;
	cout << "Dati numarului liniei (>=1): ";
	cin >> l;
	cout << "Dati numarului coloanei (>=1): ";
	cin >> c;
	if ((l >= 1 && l <= m.getRows()) && (c >= 1 && c <= m.getCols()))
		cout << "Elementul m[" << l << "," << c << "]=" << m(l - 1, c - 1) << endl;
	else
		cout << "Indici eronati!" << endl;
	cout << endl << "Utilizare constructor de copiere:" << endl;
	if (m.getRows() > 0 && m.getCols() > 0) {
		Matrix mcopy = m;
		cout << "Matricea \"mcopy\" este:" << endl;
		mcopy.afisare();
	}
	else cout << "Dimensiuni invalide pentru matricea care se copiaza la instantiere!" << endl;
	cout << endl << "Instantiem un nou obiect matrice \"n\" ";
	Matrix n(linii, coloane);
	cout << endl << "Dati matricea:" << endl;
	n.citire();
	cout << endl << "Matricea \"n\" este:" << endl;
	n.afisare();
	cout << endl << "Supraincarcarea operatorului =, copiere matrice \"m\" in matrice \"n\"" << endl;
	if (m.getRows() == n.getRows() && m.getCols() == n.getCols()) {
		n = m;
		//n.afisare();
		for (i = 0; i < linii; i++) {
			for (j = 0; j < coloane; j++)
				cout << n(i, j) << "\t";//afisare prin Supraincarcarea operatorului()
			cout << endl;
		}///end for
	}
	else
	cout << "Matricile nu au aceleasi dimensiuni, deci nu pot fi copiate" << endl;
	cout << endl << "Instantiem un nou obiect matrice \"m1\" ";
	Matrix m1(linii, coloane);
```

```
	cout << endl << "Dati matricea:" << endl;
	m1.citire();
	cout << endl << "Matricea \"m1\" este:" << endl;
	m1.afisare();
	Matrix m2(linii, coloane);
	cout << endl << "Supraincarcarea operatorului +" << endl;
	if (m.getRows() == m1.getRows() && m.getCols() == m1.getCols()) {
		m2 = m + m1;
		cout << endl << "Matricea rezultata din suma matricilor m+m1 este: " << endl;
	}
	m2.afisare();
	cout << endl << "Supraincarcarea operatorului - " << endl;
	if (m.getRows() == m1.getRows() && m.getCols() == m1.getCols()) {
		m2 = m - m1;
		cout << endl << "Matricea rezultata din diferenta matricilor m-m1 este: " << endl;
	}
	m2.afisare();
/*matricea m are 2 linii si 3 coloane deci pentru a fi posibil produsul m3 trebuie sa aiba 3 linii si 2 coloane*/
	cout << endl << "Dati matricea pentru produs \"m3\' (matricea trebuie sa aiba numarul de linii egal cu
numarul de coloane al matricii \"m\")" << endl;
	cout << "Numar de linii: ";
	cin >> l;
	cout << "Numar coloane: ";
	cin >> c;
	Matrix m3;
	if (l > 0 && c > 0) m3.init(l, c);
	else cout << endl << "Dimensiuni negative (gresite)! Se vor folosi pentru instantiere valorile initiale
implicite (2 linii, 3 coloane)" << endl;
	m3.citire();
	cout << endl << "Matricea \"m3\" este:" << endl;
	m3.afisare();
	cout << endl << "Supraincarcarea operatorului * ";
// pentru inmultire m*m3  nr. de coloane al matricii m trebuie sa fie egal cu numarul de randuri al matricii m3
	if (m.getCols() == m3.getRows())
	{
		Matrix m4(m.getRows(), m3.getCols());
		m4 = m * m3;
		cout << endl << "Matricea rezultata prin inmultirea matricilor m*m3 este: " << endl;
		m4.afisare();
	}
	else
	cout << endl << "Matricile nu pot fi inmultite - numarul de linii nu e egal cu numarul de coloane";
	return 0;
}//end main
```

Teme:

1. Pornind de la exemplul 4b, introduceti metode de tip set la atributele clasei *Persoana*, astfel incat introducerea datelor sa fie facuta cu metoda *introdu( )* si metode setter in loc de constructor. Preluati optiunile cu confirmare, la fel ca la exemplul 4a. Continuati la optiune gresita dand un mesaj adecvat. Asigurati consistenta supraincarcarii operatorilor de indexare (cand nu se gaseste obiectul). Considerati atributul *nume* de tip *char **, alocarea spatiului fiind facuta in constructori. Definiti copy constructorul si supraincarcati operatorul de asignare in cadrul clasei *Persoana*. Introduceti destructori in ambele clase. Considerati acum procesul de sortare dupa aceleasi chei ca si la cautare cu afisarea rezultatelor in ordine descrescatoare. Verificati functionalitatea elementelor introduse.
2. Pornind de la exemplul 5 verificati/implementati următoarele cerințe:
    a. citirea/scrierea unei matrici, unde dimensiunile sunt preluate de la tastatură
    b. testați toți operatorii supraîncărcați. Implementati variante in care se folosesc metode membre la supraincarcare.
    c. afișați elementele de pe diagonala principală și secundară
    d. implementati operatiile cu matrici folsind metode membre.

3. Să se supraîncarce operatorul *[ ]* astfel încât, folosit fiind asupra unor obiecte din clasa *Departament*, ce contine un tablou de obiecte de tip *Angajat* (clasa *Angajat* contine variabilele *nume* (şir de caractere) şi *salariu* (double)), să returneze toată informaţia legată de angajatul al cărui număr de ordine este trimis ca parametru.
4. Să se supraîncarce operatorii *new* şi *delete* într-una din clasele cu care s-a lucrat anterior, în vederea alocarii şi eliberarii de memorie pentru un obiect din clasa respectivă.
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.
6. Să se scrie programul care utilizează o clasă numită *Calculator* şi care are în componenţa 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 primeşte din linia de comandă toţi 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?
7. Să se definească clasa *Student* având ca date membre private: *numele* (șir de caractere), *note* (pointer de tip întreg) și *nr_note* (int). Clasa mai conține un constructor cu parametri, un constructor de copiere, o metoda de supraîncărcare a operatorului de atribuire, o metodă de setare a notelor, o metodă de afișare a atributelor și un destructor. Să se instanțieze obiecte folosind constructorul cu parametri, un alt obiect va fi obtinut folosind constructorul de copiere, afișând de fiecare data atributele obiectului creat. Realizați o operație de copiere a unui obiect în alt obiect, ambele fiind create in prealabil. Afișati rezultatul copierii. Analizati metodele utilizate. Realizati o alta implementare in care numele e dat printr-un pointer catre un sir de caractere.
8. Să se implementeze clasa *Complex* care supraîncarcă operatorii aritmetici cu scopul de a efectua adunări, scăderi, inmulţiri şi împărţiri de numere complexe (folosind metode membre (+, -) și funcții friend (*, /)).
   Observaţie: numerele complexe vor fi definite ca având o parte reală şi una imaginară, ambii coeficienţi fiind reprezentați prin numere reale.
9. Definţi o clasă numită *Number* care are o variabilă private de tip *double*. Clasa mai conţine 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*.
10. 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 clasei. Declarati o metoda *simplifica()* care simplifica un obiect *Fractie*. 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 înainte de apel va verifica posibilitatea definirii unei fractii (*b*!=0). Definiti un destructor explicit care afiseaza un mesaj si contorul *icount*. Supraincarcati operatorii de adunare, scadere, inmultire si impartire (+,-,*,/) a fractiilor folosind functii *friend* care si simplifica daca e cazul rezultatele obtinute. Instantiati doua obiecte de tip *Fractie* cu date citite de la tastatura. Afisati atributele initiale ale obiectelor. Printr-o metoda accesor, afisati contorul *icount*. Efectuati operatiile implementate prin functiile *friend*, initializand alte 4 obiecte cu rezultatele obtinute. Afisati rezultatele si contorul dupa ultima operatie folosind o metoda accesor adecvata.
11. Folosind aceeasi clasa *Fractie*, definiti supraincarcarea operatorilor compusi de asignare si adunare, scadere, inmultire si impartire(+=, -=, *=, /=) cu metode membre. Supraincarcati operatorii de incrementare si decrementare post/prefixati care aduna/scade valoarea *1* la un obiect de tip *Fractie* cu functii membre (metode). Instantiati doua obiecte de tip *Fractie* cu date citite de la tastatura. Realizați o copie a lor in alte doua obiecte. Efectuați operațiile compuse implementate prin metodele clasei folosind copiile obiectelor, asignând rezultatele obtinute la alte 4 obiecte. Afișați cele 4 rezultate, iar apoi afișați rezultatele după incrementare/decrementare post/prefixat la cele 4 obiecte obținute.

Homework:

1. Starting from examples 4b, enter *set*-type methods to the attributes of the *Person* class, so entering data is done in *introdu( )* method using setter methods instead of the constructor. Take the options with confirmation as in 4a example. At wrong option, continue with an adequate message. Ensure consistency of overloading for indexing operators (when object is not find with a specific message). Consider the *name* as *char* * attribute, the space allocation is done in constructors. Define the copy constructor and overload the assign operator within the *Person* class. define destructors in both classes. Consider now a sorting process based on the same keys as at searching the results being displayed in reverse order. Verify all introduced functionalities.

2. Starting with the 5-th example, verify/resolve the following tasks:

a. reading/writing a matrix with dimensions from KB;

b. test all the overloaded operators. Implement variants in which member methods are used for overloading of operators.

c. displays the elements located on both diagonals.

d. implement matrix operations using member methods.

3. Overload the *[ ]* operator for the *Department* class that contains an array of *Employee* objects (that has as variables the *name* (character array) and the *salary* (float)). When the operator is applied to a *Department* object, it returns (or displays) all the data related to the *Employee* object with that index.

4. Overload the *new* and *delete* operators for one of the classes implemented before, in order to allocate / de-allocate the necessary amount of memory.

5. Write the program that considers a *MyClass* class with three *int*-type attributes. The class considers on the basis of the overload mechanism *public* methods *int myFunction (…),* which depending on the number of parameters received, returns either the value received (1 parameter) or the product of the input variables (0- all, 2, 3 parameters). Instantiate an object of the class in *main(), set* the attributes using dedicated setter methods from the class, and display the values at method calls.

6. Write the program that uses a class called *Calculator* that has as overloaded public methods:

- *int calcul(int x)* that returns the square value of *x*;
- *int calcul(int x, int y)* that returns the product of *x* and *y*;
- *double calcul(int x, int y, int z)* that returns the result of *f(x,y,z) = (x-y)(x+z)/2.;*

The program receives the parameters from the command line.

Consider the case when all the methods are static. Is it possible to have in the same time static and non-static public methods? Analyze the case that the class has 3 *private* attributes of type *int, x, y, z,* that are modified with setter adequate methods. What must be modified to perform the required operations?

7. Define a class called *Student,* containing as private member: *name* (character array), *marks* (integer pointer) and *nr_marks* (int). The class also contains a constructor with parameters, a copy constructor, a method for assign operator overloading, a method for marks setting, a display method and a destructor. Create some objects using the constructor with parameters, another one using the copy constructor, displaying each time the attributes of the created object. Copy an object into another one, both of them being created. Display the result of the copy operation. Analyze the used methods. Make another implementation in which the name is given by a pointer to a character string.

8. Implement a class called *Complex* that overloads the arithmetical operators (+, -, *, /) for performing the corresponding operations when applied to *Complex* instances (use both friend functions (*, /) and member methods (+, -)).

Note: the complex numbers will have a real and an imaginary part, both coefficients being represented as real numbers.

9. Define a class named *Number* that has as private attribute a *double* variable. The class contains an explicit empty constructor, a constructor with a parameter and an accessor method that displays the value of the stored variable. Write a class called *Mathematics* that has as attribute an instance created from the first class and overloads the arithmetical operators (+, -, /, *). Each method calculates the appropriate result by considering the data extracted from the *Number* object.

10. Consider a class named *Fraction* that has two integer private attributes *a* and *b* for the denominator and nominator. Define *set( )* and *get( )* methods for all the class's attributes. Declare a method named *simplify( )* that simplifies a fraction. Consider a *private* static integer variable *icount* that will be initialized with 0 and incremented in the constructors. Declare two explicit constructors, one without parameters that initializes *a* with 0 and *b* with 1 and the other that has two integer parameters which will verify before if the fraction can be defined (b! =0). Define an explicit destructor that will display a message and the *icount* counter. Overload the arithmetic operators (+, -, *, /) using *friend* functions. The results will be displayed after being simplified. Instantiate 2 *Fraction* objects and read the appropriate data from the keyboard. Display the original values of the nominators and denominators. Using a specific accessor method, display the value of *icount*. Apply the implemented *friend* functions and *initialize* other 4 objects with the obtained results. Display the characteristics of the final objects and the value of *icoun*t.

11. Using the same *Fraction* class, overload the composed arithmetical operators (+=, -=, *=, /=) using member functions (methods). Overload the *pre*-de/incrementation and *post*-de/incrementation operators that will subtract/add the value 1 to an already existent *Fraction* object. Instantiate 2 *Fraction* objects and read the appropriate data from the keyboard. Copy the objects in other 2 temporary objects. Apply the overloaded operators using the copied objects and *assign* the results to other 4 objects. Display the characteristics of the final objects. Display the results obtained after decrementing/incrementing post/prefixed of the final 4 results.