**Ministerul Educației al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**Departamentul Ingineria Software și Automatică**

**Raport**

Lucrarea de laborator nr.3

Disciplina Programarea orietntată pe obiecte:

Tema: Supraîncărcarea operatorilor

Varianta 5

**Efectuat**: st.gr. TI-207 Bunescu Gabriel

**Verificat lect.univ.Lisnic Inga**.

Chișinău 2021

**Scopul lucrării:**

1. Studierea necesității supraîncărcării operatorilor;
2. Studierea sintaxei de definire a operatorilor;
3. Studierea tipurilor de operatori;
4. Studierea formelor de supraîncărcare;

**Sarcina:**

**Varianta: 5**

а) Să se creeze clasa numerelor reale. Să se definească operatorii "++" şi "+" ca metode ale clasei, iar operatorii "- -" şi "-" – ca funcţii prietene. Operatorii trebuie să permită realizarea operaţiilor atît cu variabilele clasei date, cît şi cu variabilele de tip predefinit *double*.

b) Să se creeze clasa *Vector* – vector, utilizînd memoria dinamică. Să se definească operatorii "+" – adunarea element cu element a vectorilor, "-" – scăderea element cu element a vectorilor, "=" – atribuirea ca metode ale clasei. Să se definească operatorii de comparare: "==", "!=", "<", ">" ca funcţii prietene. Pentru realizarea ultimilor doi operatori să se definească funcţia de calcul a modulului elementelor vectorului. Să se supraîncarce operatorii "<<" şi ">>" pentru ieşiri/intrări de obiecte. Clasa trebuie să conţină toţi constructorii necesari şi destructorul.

**Mersul lucrării:**

**Listingul programului:**

**a)**

#include <iostream>

#include <stdio.h>

class Real{

double nr;

public:

Real(double a);

void afisare(){

printf("Nr = %f\n", this->nr);

}

Real operator+(const Real& a){

return Real(this->nr+a.nr);

}

Real operator++(int){

return Real(this->nr+1);

}

friend Real operator-(const Real& a, const Real& b);

friend Real operator--(const Real& a, int);

};

Real::Real(double a){

this->nr = a;

}

Real operator-(const Real& a, const Real& b){

return Real(a.nr-b.nr);

}

Real operator--(const Real& a, int){

return Real(a.nr-1);

}

int main(){

Real A= 6.5;

A.afisare();

A = A++;

A.afisare();

Real B = 2.8;

B.afisare();

B = B--;

B.afisare();

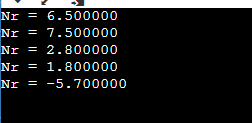
Real C = B - A;

C.afisare();

return 0;

}

**Figura.1.Afișarea datelor introduse.**

****

**b)**

#include <iostream>

using namespace std;

class Vector {

public:

int \*elements = nullptr;

int count = 0;

Vector();

Vector(int \*, int);

void push(int);

int pop();

Vector operator+(const Vector &); // +

Vector &operator=(const Vector &); //=

friend ostream &operator<<(ostream &, Vector &); //cout << Vector

friend istream &operator>>(istream &, Vector &); //cin >> Vector

friend Vector &operator<<(Vector &, int); //Vector << int

friend Vector &operator>>(Vector &, int &); //Vector >> int

friend bool operator==(Vector &, Vector &); // ==

friend bool operator!=(Vector &, Vector &); // !=

friend bool operator<(Vector &, Vector &); // <

friend bool operator>(Vector &, Vector &); // >

};

Vector::Vector() {

this->elements = nullptr;

this->count = 0;

}

Vector::Vector(int \*e, int c) {

this->elements = e;

this->count = c;

}

void Vector::push(int el) {

int \*\_new = new int[this->count + 1];

for (int i = 0; i < this->count; i++) {

\_new[i] = this->elements[i];

}

\_new[this->count] = el;

this->elements = \_new;

this->count++;

}

int Vector::pop() {

if (this->count == 0)

return 0;

int \*\_new = new int[this->count - 1];

for (int i = 1; i < this->count; i++) {

\_new[i - 1] = this->elements[i];

}

int el = this->elements[0];

this->elements = \_new;

this->count--;

return el;

}

Vector Vector::operator+(const Vector &q) {

int \*\_new = new int[this->count + q.count];

for (int i = 0; i < this->count; i++) {

\_new[i] = this->elements[i];

}

for (int i = 0; i < q.count; i++) {

\_new[i + this->count] = q.elements[i];

}

return Vector (\_new,

this->count + q.count

);

}

Vector &Vector::operator=(const Vector &q) = default;

bool operator==(Vector &q1, Vector &q2) {

if (q1.count != q2.count)

return false;

for (int i = 0; i < q1.count; i++) {

if (q1.elements[i] != q2.elements[i])

return false;

}

return true;

}

bool operator!=(Vector &q1, Vector &q2) {

return !(q1 == q2);

}

bool operator<(Vector &q1, Vector &q2) {

return q1.count < q2.count;

}

bool operator>(Vector &q1, Vector &q2) {

return q1.count > q2.count;

}

ostream &operator<<(ostream &out, Vector &q) {

if (q.count == 0)

return out;

for (int i = 0; i < q.count - 1; i++) {

out << q.elements[i] << ", ";

}

out << q.elements[q.count - 1];

return out;

}

istream &operator>>(istream &in, Vector &q) {

int temp;

in >> temp;

q.push(temp);

return in;

}

Vector &operator<<(Vector &q, int el) {

q.push(el);

return q;

}

Vector &operator>>(Vector &q, int &el) {

el = q.pop();

return q;

}

int main() {

Vector v1;

Vector v2;

Vector v3;

v1 << 1 << 2 << 3;

cout << "V1: " << v1 << endl;

int el = 0;

v1 >> el;

cout << "Introduceti vectorul: " << el << endl;

cout << "Introduceti un element nou in V1: ";

cin >> v1;

v2 << 4 << 5 << 6;

cout << "V2: " << v2 << endl;

v3 = v1 + v2;

cout << "V3 (V1 + V2): " << v3 << endl;

cout << "V1 == V2 --- ";

if (v1 == v2)

cout << "true" << endl;

else

cout << "false" << endl;

cout << "V1 != V2 --- ";

if (v1 != v2)

cout << "true" << endl;

else

cout << "false" << endl;

cout << "V1 > V2 --- ";

if (v1 > v2)

cout << "true" << endl;

else

cout << "false" << endl;

cout << "V1 < V2 --- ";

if (v1 < v2)

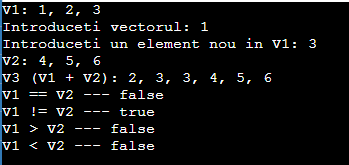
cout << "true" << endl;

else

cout << "false" << endl;

}

**Afișarea rezultatului:**

****

**Concluzie:**

In aceasta lucrare am lucrat la tema supraincarcarea operatorilor in C++. Am supraincarcat multi operatori pentru mai multe tipuri de operatii, operatori cum ar fi +, >, >>. Am folosit cunostinele obtinute individual si la cursuri pentru roezolvarea corecta a problemelor.