

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**gr. IA-231, Chistol Maxim**

**Raport**

**pentru lucrarea de laborator Nr.3**

***la cursul de “Programarea Orientată pe Obiecte”***

Verificat:

Kulev Mihail dr., conf. univ.

Departamentul Informatică şi IS,

Facultatea FCIM, UTM

**Chișinău 2024**

**Tema:** Supraîncărcarea operatorilor

**Scopul lucrării:**

• Studierea necesităţii supraîncărcării operatorilor;

• Studierea sintaxei de definire a operatorilor;

• Studierea tipurilor de operatori;

• Studierea formelor de supraîncărcare;

Noţiuni de bază

Avantajul utilizării operatorilor (reducerea codului) în acelaşi timp complică înţelegerea codului, deoarece nu este posibil întotdeauna sau este complicat să se urmărească dacă se utilizează un operator predefinit sau supraîncărcat. Însă supraîncărcarea operatorilor este necesară. Sunt probleme care se soluţionează numai prin supraîncărcarea operatorilor,cum ar fi operatorul de atribuire. Desigur, compilatorul poate să genereze codul necesar, sau să definim metoda, de exemplu Assign(Java). Dar ambele soluţii nu sunt ideale.

Neajunsul primei metode constă în utilizarea copierii bit cu bit, ceea ce este satisfăcător atît timp cît în clasă nu se utilizează pointerii:

|  |  |
| --- | --- |
| **Codul:** | **Output:** |
| **#include <iostream>**  **#include <string>**  **#include <algorithm>**  **using namespace std;**  **class BigInteger {**  **private:**  **string value; // Reprezentația internă a numărului mare ca șir de caractere**  **public:**  **// Constructori**  **BigInteger() : value("0") {} // Constructor implicit care inițializează valoarea la "0"**  **BigInteger(const string& val) : value(val) {} // Constructor care inițializează cu un șir de caractere**  **BigInteger(long val) : value(to\_string(val)) {} // Constructor care inițializează cu un număr întreg**  **// Destructor (nu este strict necesar în acest caz, dar este adăugat pentru completitudine)**  **~BigInteger() {}**  **// Operatorul de adunare**  **BigInteger operator+(const BigInteger& other) const {**  **string result = "";**  **int carry = 0;**  **int len1 = value.size();**  **int len2 = other.value.size();**  **int maxLength = max(len1, len2);**  **for (int i = 0; i < maxLength; i++) {**  **int digit1 = (len1 - i - 1 >= 0) ? value[len1 - i - 1] - '0' : 0;**  **int digit2 = (len2 - i - 1 >= 0) ? other.value[len2 - i - 1] - '0' : 0;**  **int sum = digit1 + digit2 + carry;**  **carry = sum / 10;**  **result.push\_back((sum % 10) + '0');**  **}**  **if (carry > 0) result.push\_back(carry + '0');**  **reverse(result.begin(), result.end());**  **return BigInteger(result);**  **}**  **// Operatorul de înmulțire**  **BigInteger operator\*(const BigInteger& other) const {**  **int len1 = value.size();**  **int len2 = other.value.size();**  **string result(len1 + len2, '0');**  **for (int i = len1 - 1; i >= 0; i--) {**  **int carry = 0;**  **for (int j = len2 - 1; j >= 0; j--) {**  **int temp = (result[i + j + 1] - '0') + (value[i] - '0') \* (other.value[j] - '0') + carry;**  **result[i + j + 1] = temp % 10 + '0';**  **carry = temp / 10;**  **}**  **result[i] += carry;**  **}**  **size\_t startpos = result.find\_first\_not\_of("0");**  **if (string::npos != startpos) {**  **return BigInteger(result.substr(startpos));**  **}**  **return BigInteger("0");**  **}**  **// Operatorul de scădere (funcție prietenă)**  **friend BigInteger operator-(const BigInteger& a, const BigInteger& b) {**  **string result = "";**  **int carry = 0;**  **int len1 = a.value.size();**  **int len2 = b.value.size();**  **for (int i = 0; i < len1; i++) {**  **int digit1 = a.value[len1 - i - 1] - '0';**  **int digit2 = (len2 - i - 1 >= 0) ? b.value[len2 - i - 1] - '0' : 0;**  **int diff = digit1 - digit2 - carry;**  **if (diff < 0) {**  **diff += 10;**  **carry = 1;**  **} else {**  **carry = 0;**  **}**  **result.push\_back(diff + '0');**  **}**  **reverse(result.begin(), result.end());**  **size\_t startpos = result.find\_first\_not\_of("0");**  **return BigInteger(startpos == string::npos ? "0" : result.substr(startpos));**  **}**  **// Operatorul de împărțire (funcție prietenă)**  **friend BigInteger operator/(const BigInteger& a, const BigInteger& b) {**  **if (b.value == "0") throw invalid\_argument("Division by zero");**  **BigInteger result;**  **BigInteger current;**  **for (char digit : a.value) {**  **current = BigInteger(current.value + digit);**  **int count = 0;**  **while (!(current < b)) {**  **current = current - b;**  **count++;**  **}**  **result.value += to\_string(count);**  **}**  **size\_t startpos = result.value.find\_first\_not\_of("0");**  **return BigInteger(startpos == string::npos ? "0" : result.value.substr(startpos));**  **}**  **// Incrementare prefixată**  **BigInteger& operator++() {**  **\*this = \*this + BigInteger(1);**  **return \*this;**  **}**  **// Incrementare postfixată**  **BigInteger operator++(int) {**  **BigInteger temp = \*this;**  **++(\*this);**  **return temp;**  **}**  **// Decrementare prefixată**  **BigInteger& operator--() {**  **\*this = \*this - BigInteger(1);**  **return \*this;**  **}**  **// Decrementare postfixată**  **BigInteger operator--(int) {**  **BigInteger temp = \*this;**  **--(\*this);**  **return temp;**  **}**  **// Comparare mai mic decât**  **bool operator<(const BigInteger& other) const {**  **if (value.size() != other.value.size())**  **return value.size() < other.value.size();**  **return value < other.value;**  **}**  **// Suprascriere operator << pentru afișare**  **friend ostream& operator<<(ostream& out, const BigInteger& bigInt) {**  **out << bigInt.value;**  **return out;**  **}**  **};**  **int main() {**  **int optiune;**  **BigInteger a, b;**  **while (true) {**  **cout << "\nMeniu:\n";**  **cout << "1. Adunare (a + b)\n";**  **cout << "2. Scadere (a - b)\n";**  **cout << "3. Inmultire (a \* b)\n";**  **cout << "4. Impartire (a / b)\n";**  **cout << "5. Incrementare (prefix ++a)\n";**  **cout << "6. Decrementare (postfix b--)\n";**  **cout << "7. Iesire\n";**  **cout << "Introduceti optiunea dorita: ";**  **cin >> optiune;**  **if (optiune == 7) break;**  **if (optiune >= 1 && optiune <= 4) {**  **string valoareA, valoareB;**  **cout << "Introduceti valoarea pentru a: ";**  **cin >> valoareA;**  **cout << "Introduceti valoarea pentru b: ";**  **cin >> valoareB;**  **a = BigInteger(valoareA);**  **b = BigInteger(valoareB);**  **} else if (optiune == 5 || optiune == 6) {**  **string valoare;**  **cout << "Introduceti valoarea pentru a: ";**  **cin >> valoare;**  **a = BigInteger(valoare);**  **}**  **switch (optiune) {**  **case 1:**  **cout << "a + b = " << (a + b) << endl;**  **break;**  **case 2:**  **cout << "a - b = " << (a - b) << endl;**  **break;**  **case 3:**  **cout << "a \* b = " << (a \* b) << endl;**  **break;**  **case 4:**  **try {**  **cout << "a / b = " << (a / b) << endl;**  **} catch (const invalid\_argument& e) {**  **cout << "Eroare: " << e.what() << endl;**  **}**  **break;**  **case 5:**  **++a;**  **cout << "++a = " << a << endl;**  **break;**  **case 6:**  **b--;**  **cout << "b-- = " << b << endl;**  **break;**  **default:**  **cout << "Optiune invalida. Incercati din nou." << endl;**  **break;**  **}**  **}**  **cout << "Program terminat." << endl;**  **return 0;**  **}** | Meniu:  1. Adunare (a + b)  2. Scadere (a - b)  3. Inmultire (a \* b)  4. Impartire (a / b)  5. Incrementare (prefix ++a)  6. Decrementare (postfix b--)  7. Iesire  Introduceti optiunea dorita: 1  Introduceti valoarea pentru a: 2300  Introduceti valoarea pentru b: 4354  a + b = 6654  Meniu:  1. Adunare (a + b)  2. Scadere (a - b)  3. Inmultire (a \* b)  4. Impartire (a / b)  5. Incrementare (prefix ++a)  6. Decrementare (postfix b--)  7. Iesire  Introduceti optiunea dorita: 2  Introduceti valoarea pentru a: 54 64  Introduceti valoarea pentru b: a - b = 90  Meniu:  1. Adunare (a + b)  2. Scadere (a - b)  3. Inmultire (a \* b)  4. Impartire (a / b)  5. Incrementare (prefix ++a)  6. Decrementare (postfix b--)  7. Iesire  Introduceti optiunea dorita: 3  Introduceti valoarea pentru a: 534  Introduceti valoarea pentru b: 54  a \* b = 28836  Meniu:  1. Adunare (a + b)  2. Scadere (a - b)  3. Inmultire (a \* b)  4. Impartire (a / b)  5. Incrementare (prefix ++a)  6. Decrementare (postfix b--)  7. Iesire  Introduceti optiunea dorita: 5  Introduceti valoarea pentru a: 54  ++a = 55  Meniu:  1. Adunare (a + b)  2. Scadere (a - b)  3. Inmultire (a \* b)  4. Impartire (a / b)  5. Incrementare (prefix ++a)  6. Decrementare (postfix b--)  7. Iesire |

|  |  |
| --- | --- |
| **Codul:** | **Output:** |
| **#include <iostream>**  **#include <cmath>**  **using namespace std;**  **class Matrix {**  **private:**  **int rows, cols;**  **double\*\* data;**  **public:**  **// Constructori**  **Matrix(int r = 0, int c = 0) : rows(r), cols(c) {**  **data = new double\*[rows];**  **for (int i = 0; i < rows; ++i) {**  **data[i] = new double[cols]{0}; // Inițializare cu 0**  **}**  **}**  **// Constructor de copiere**  **Matrix(const Matrix& other) : rows(other.rows), cols(other.cols) {**  **data = new double\*[rows];**  **for (int i = 0; i < rows; ++i) {**  **data[i] = new double[cols];**  **for (int j = 0; j < cols; ++j) {**  **data[i][j] = other.data[i][j];**  **}**  **}**  **}**  **// Destructor**  **~Matrix() {**  **for (int i = 0; i < rows; ++i) {**  **delete[] data[i];**  **}**  **delete[] data;**  **}**  **// Operator "=" pentru atribuirea unei matrice**  **Matrix& operator=(const Matrix& other) {**  **if (this == &other) return \*this;**  **// Curățăm memoria existentă**  **for (int i = 0; i < rows; ++i) delete[] data[i];**  **delete[] data;**  **// Realocăm și copiem datele**  **rows = other.rows;**  **cols = other.cols;**  **data = new double\*[rows];**  **for (int i = 0; i < rows; ++i) {**  **data[i] = new double[cols];**  **for (int j = 0; j < cols; ++j) {**  **data[i][j] = other.data[i][j];**  **}**  **}**  **return \*this;**  **}**  **// Operator "+" pentru adunarea matricilor**  **Matrix operator+(const Matrix& other) const {**  **Matrix result(rows, cols);**  **for (int i = 0; i < rows; ++i) {**  **for (int j = 0; j < cols; ++j) {**  **result.data[i][j] = data[i][j] + other.data[i][j];**  **}**  **}**  **return result;**  **}**  **// Operator "-" pentru scăderea matricilor**  **Matrix operator-(const Matrix& other) const {**  **Matrix result(rows, cols);**  **for (int i = 0; i < rows; ++i) {**  **for (int j = 0; j < cols; ++j) {**  **result.data[i][j] = data[i][j] - other.data[i][j];**  **}**  **}**  **return result;**  **}**  **// Funcție pentru calculul normei elementelor matricei**  **double norm() const {**  **double sum = 0;**  **for (int i = 0; i < rows; ++i) {**  **for (int j = 0; j < cols; ++j) {**  **sum += data[i][j] \* data[i][j];**  **}**  **}**  **return sqrt(sum);**  **}**  **// Operatorii de comparare**  **friend bool operator==(const Matrix& a, const Matrix& b) {**  **if (a.rows != b.rows || a.cols != b.cols) return false;**  **for (int i = 0; i < a.rows; ++i) {**  **for (int j = 0; j < a.cols; ++j) {**  **if (a.data[i][j] != b.data[i][j]) return false;**  **}**  **}**  **return true;**  **}**  **// operatorul !=**  **friend bool operator!=(const Matrix& a, const Matrix& b) {**  **return !(a == b);**  **}**  **// operatorul <**  **friend bool operator<(const Matrix& a, const Matrix& b) {**  **return a.norm() < b.norm();**  **}**  **// operatorul >**  **friend bool operator>(const Matrix& a, const Matrix& b) {**  **return a.norm() > b.norm();**  **}**  **// Operator "[]" pentru accesul la elemente (suport [][])**  **double\* operator[](int index) const {**  **return data[index];**  **}**  **// Operator de ieșire "<<"**  **friend ostream& operator<<(ostream& out, const Matrix& m) {**  **for (int i = 0; i < m.rows; ++i) {**  **for (int j = 0; j < m.cols; ++j) {**  **out << m.data[i][j] << " ";**  **}**  **out << endl;**  **}**  **return out;**  **}**  **// Operator de intrare ">>"**  **friend istream& operator>>(istream& in, Matrix& m) {**  **for (int i = 0; i < m.rows; ++i) {**  **for (int j = 0; j < m.cols; ++j) {**  **in >> m.data[i][j];**  **}**  **}**  **return in;**  **}**  **};**  **// Testare**  **int main() {**  **int sel;**  **int n;**  **Matrix m1(n, n), m2(n, n);**  **cout << "Introduceti elementele pentru matricea m1 (2x2):" << endl;**  **cin >> m1;**  **cout << "Introduceti elementele pentru matricea m2 (2x2):" << endl;**  **cin >> m2;**  **Matrix sum = m1 + m2;**  **Matrix diff = m1 - m2;**  **cout << "m1 + m2 = " << endl << sum;**  **cout << "m1 - m2 = " << endl << diff;**  **cout << "Comparare m1 si m2:" << endl;**  **cout << "m1 == m2? " << (m1 == m2 ? "Da" : "Nu") << endl;**  **cout << "m1 != m2? " << (m1 != m2 ? "Da" : "Nu") << endl;**  **cout << "m1 < m2? " << (m1 < m2 ? "Da" : "Nu") << endl;**  **cout << "m1 > m2? " << (m1 > m2 ? "Da" : "Nu") << endl;**    **while (true) {**  **cout <<"\nMeniu:\n";**  **cout <<"0. Intoroducere a matricilor\n1. Adunare\n2. Scaderea\n3. Compararea\n4. Exit";**    **cin >> sel;**  **switch (sel){**    **case 0:**  **cout << "Introduceti demensiunile matriciilor";**  **cin >> n;**  **cout << "Introduceti elementele pentru matricea m1 :" << endl;**  **cin >> m1;**  **cout << "Introduceti elementele pentru matricea m2 :" << endl;**  **cin >> m2;**  **case 1:**    **}**  **}**  **return 0;**  **}** | **Meniu:**  **1. Introducere dimensiuni si elemente matrice**  **2. Adunarea matricilor m1 + m2**  **3. Scăderea matricilor m1 - m2**  **4. Compararea matricilor**  **5. Ieșire**  **Alegeți o opțiune: 1**  **Introduceți dimensiunile matricilor (rânduri și coloane): 2 2**  **Introduceți elementele pentru matricea m1:**  **2 -4**  **3 5**  **Introduceți elementele pentru matricea m2:**  **3 6**  **3 5**  **Meniu:**  **1. Introducere dimensiuni si elemente matrice**  **2. Adunarea matricilor m1 + m2**  **3. Scăderea matricilor m1 - m2**  **4. Compararea matricilor**  **5. Ieșire**  **Alegeți o opțiune: 2**  **m1 + m2 =**  **5 2**  **6 10**  **Meniu:**  **1. Introducere dimensiuni si elemente matrice**  **2. Adunarea matricilor m1 + m2**  **3. Scăderea matricilor m1 - m2**  **4. Compararea matricilor**  **5. Ieșire**  **Alegeți o opțiune: 3**  **m1 - m2 =**  **-1 -10**  **0 0**  **Meniu:**  **1. Introducere dimensiuni si elemente matrice**  **2. Adunarea matricilor m1 + m2**  **3. Scăderea matricilor m1 - m2**  **4. Compararea matricilor**  **5. Ieșire**  **Alegeți o opțiune: 4**  **Comparare m1 și m2:**  **m1 == m2? Nu**  **m1 != m2? Da**  **m1 < m2? Da**  **m1 > m2? Nu** |

# **Concluzii:**

Supraîncărcarea operatorilor este o tehnică esențială în programarea orientată pe obiecte, permițând utilizarea intuitivă și eficientă a operatorilor pentru tipurile de date definite de utilizator. Prin supraîncărcare, operatorii comuni (cum ar fi +, -, <, >, ==) pot fi adaptați pentru a funcționa cu clase și structuri personalizate, simplificând astfel codul și îmbunătățind lizibilitatea. Studiul sintaxei și formelor de supraîncărcare evidențiază flexibilitatea acestei tehnici și demonstrează cum poate facilita manipularea și compararea obiectelor într-un mod natural și ușor de înțeles pentru utilizator. De asemenea, cunoașterea tipurilor de operatori ce pot fi supraîncărcați ajută la crearea unor interfețe intuitive și eficiente pentru clasele definite de utilizator.

# **Bibliografie:**

1. Lucrare de laborator 3:https://ocw.cs.pub.ro/courses/sd-ca/2019/laboratoare/lab-03 Acesat pe 10.11.2024
2. POO:<https://www.youtube.com/watch?v=rZcTaRU7AAw&list=PLQOaTSbfxUtBm7DxblJZShqBQnBAVzlXX> Acesat pe 10.11.2024