**MINISTERUL EDUCAŢIEI, CULTURII ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**UNIVERSITATEA TEHNICĂ A MOLDOVEI**

**Facultatea Calculatoare, Informatică și Microelectronică**

**Departamentul Ingineria Software și Automatică**

**RAPORT**

**la lucrarea de laborator nr.3**

**Tema: „Supraîncărcarea operatorilor”**

**Disciplina: „Clase și obiecte. Constructori și destructor pentru clasa”**

**A elaborat *st. gr. SI-211, Vozian Vladimir***

**A verificat *lect.univ S. Scrob***

**Chișinău 2022**

Varianta 11:

а) Să se creeze clasa *Date* – data, care conţine cîmpurile: ziua, luna, anul. Să se definească operatorii "+" şi "-" ca metode ale clasei, iar operatorii "++" şi "--" în ambele variante(prefixă şi postfixă) ca funcţii prietene. Operatorul "+" trebuie să permită realizarea operaţiilor numai cu variabilele de tip predefinit *int* (x=y+5). De prevăzut prelucrarea corectă a anilor bisecţi.

b) Să se creeze clasa *String –* şir, utilizînd memoria dinamică. Să se definească operatorii "+" – adunarea şirurilor, "=" şi "+=" – atribuirea ca funcţii prietene. Să se definească operatorii de comparare: "==", "!=", "<", ">" ca metode ale clasei. Operatorii trebuie să lucreze atît cu *String*, cît şi cu *char\*.* Să se definească operatorul "*[]*" de acces la fiecare simbol în parte. Să se supraîncarce operatorii "<<" şi ">>" pentru ieşiri/intrări de obiecte.

**Codul programului:**

a)

#include <iostream>

#include <string>

using namespace std;

class Date {

public:

    int an, luna, zi;

    Date(int a = 0, int b = 0, int c = 0) {

        an = a;

        luna = b;

        zi = c;

    }

    bool chek\_bisect(){

        if(an % 400 == 0) return true;

        else if(an % 100 == 0) return false;

        else if(an % 4 == 0) return true;

        else return false;

    }

    friend const Date &operator++(Date &i);

    friend const Date operator++(Date &i, int);

    friend const Date &operator--(Date &i);

    friend const Date operator--(Date &i, int);

    Date operator+(const Date &other) {

        Date temp;

        temp.an = this->an + other.an;

        temp.luna = this->luna + other.luna;

        temp.zi = this->zi + other.zi;

        return temp;

    }

    Date operator-(const Date &other) {

        Date temp;

        temp.an = this->an - other.an;

        temp.luna = this->luna - other.luna;

        temp.zi = this->zi - other.zi;

        return temp;

    }

    void display() {

        cout << "Anul ="<<an<<endl << "Luna =" << luna <<endl<< "Zi =" << zi << endl;

    }

};

const Date operator--(Date &i,int){

    Date oldValue(i.an, i.luna,i.zi);

    i.zi--;

    i.an--;

    i.luna;

    return oldValue;

}

const Date& operator--(Date &i){

    i.an--;

    i.zi--;

    i.luna;

    return i;

}

const Date operator++(Date &i, int) {

    Date oldValue(i.an, i.luna, i.zi);

    i.zi++;

    i.an++;

    i.luna++;

    return oldValue;

}

const Date &operator++(Date &i) {

    i.an++;

    i.luna++;

    i.zi++;

    return i;

}

int main() {

    Date first(2001, 4, 12), second(2003, 5, 9);

    Date resul =first.operator+(second);

    resul=resul+4;

    resul.display();

    bool check=resul.chek\_bisect();

    if (check == true){

        cout << "Anul este bisect";

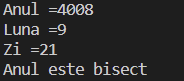
    }

    else cout<<"Anul nu este bisect";

    return 0;

}

Rezultatele:



b)

#include <iostream>

#include <string>

using namespace std;

class MyString {

private:

    string \*our\_string;

public:

    MyString() {

        our\_string = new string;

        cout << "Enter your string->";

        cin >> \*our\_string;

    }

    MyString(string value = "") {

        our\_string = new string;

        if (value == "") {

            cout << "Enter your string->";

            cin >> \*our\_string;

        } else {

            \*our\_string = value;

        }

    }

    void print() {

        cout << "Your string->" << \*our\_string << endl;

    }

//      Supraincarcarea operatorului (=)

    MyString &operator=(const MyString &i) {

        if (our\_string != nullptr) {

            delete our\_string;

        }

        this->our\_string = new string;

        string temp = \*i.our\_string;

        \*this->our\_string = temp;

        return \*this;

    }

//      Supraincarcarea operatorului (+=)

    MyString &operator+=(const MyString &i) {

        string temp1 = \*this->our\_string, temp2 = \*i.our\_string;

        if (our\_string != nullptr) {

            delete our\_string;

        }

        this->our\_string = new string;

        \*this->our\_string = temp1 + temp2;

        return \*this;

    }

//      Supraincarcarea operatorului(+) Posibil scurgere de memorie

//      deoarece cand returnam se activeaza destructorul si ne

//      sterge valorile.

    MyString &operator+(const MyString &i) {

        MyString temp\_str("-");

        \*temp\_str.our\_string = \*our\_string + \*i.our\_string;

        return temp\_str;

    }

//      Supraincarcarea operatorului (==)

    bool operator==(const MyString &i) {

        if (\*this->our\_string == \*i.our\_string)  return true;

        else  return false;

    }

//      Supraincarcarea operatorului (!=)

//      returneaza un bool

    bool operator!=(const MyString &i) {

        if (\*this->our\_string != \*i.our\_string) return false;

        else  return true;

    }

//      Supraincarcarea operatorului (<)

    bool operator<(const MyString &i) {

        if (\*this->our\_string < \*i.our\_string) return true;

        else  return false;

    }

//      Supraincarcarea operatorului (>)

    bool operator>(const MyString &i) {

        if (\*this->our\_string > \*i.our\_string) return false;

        else  return true;

    }

//      Supraincarcarea operatorului ([])

    char operator[](int index) {

        string temp = \*this->our\_string;

        int len\_string;

        if (index <= this->our\_string->length()) {

            return temp[index - 1];

        } else {

            return '0';

        }

    }

//      Supraincaracarea operatorului de iesire a datelor

//      cu ajutorul functii prietene.

    friend ostream &operator<<(ostream &out, MyString a);

//      Supraincaracarea operatorului de intrare a datelor

//      cu ajutorul functii prietene.

    friend istream &operator>>(istream &in, MyString &a);

};

istream &operator>>(istream &in, MyString &a) {

    cout << "Enter string->";

    in >> \*a.our\_string;

    return in;

}

ostream &operator<<(ostream &out, MyString a) {

    out << \*a.our\_string << "\n";

    return out;

}

int main() {

    MyString first("ThisIsFirstObj"), second("ThisIsSecondObj");

//    cout<<t;

//      Exemple de supraincarcarea o operatorilor binari.

    if (first == second)

        cout << "Objects are similar" << endl;

    else

        cout << "The objections are different." << endl;

//      Inca un exemplu dar cu negatie.

    if (first != second)

        cout << "The objections are different." << endl;

    else

        cout << "The objects are the same" << endl;

//    Opearatori mai mare mic

    if (first > second) // aici putem verifica inversul. Schimband in loc de '<' '>'

        cout << "First Obj is larger. " << endl;

    else

        cout << "Second Obj is large" << endl;

//    Operatori de esire si intrare.

    cout << "First Obj->" << first;      //exit

    cout << "Second Obj->" << second;     //exit

    cin >> first;             //input

    cin >> second;            //input

//    Verificarea operatorului de indexare.

    int index = 3;

    char temp = first[index];

    cout << "Index " << index << " element in FirstObj->" << temp << endl;

//      Operatorul +=

    MyString result("SoThisIsResult");

    result += first;

    cout << "This is result after result+=first\t" << result;

//      Operatorul (=) atentie este posibila

//      scurgere de memorie

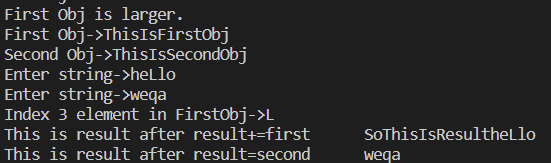
    result = second;

    cout << "This is result after result=second\t" << result;

    return 0;

}

Rezultatele:



**Concluzii:**

Deci sa lucrat cu supraîncărcarea operatorilor in C++ ceea ce permite de a optimiza și de a adapta funcționalul programului la necesitățile noastere. La punctul a sa supraîncărcat doar operatorii ‚+’ si ‚-’, iar la punctul b erau cu mult mai multe sarcini ele fiind mai dificile. Sa lucrat cu operatorii de intrare și ieșire ce permite mai eficient să introducem date și sa le afișăm, operatorul de indexare care afișează anumit element din string.