Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт
з лабораторної роботи № 4 з дисципліни
«Основи програмування 2. Методології програмування»
«Перевантаження операторів»

Варіант__16__

Виконав студент __ІП-15,_Куманецька_Ірина_Вікторівна___
Перевірив ___Вечерковська Анастасія Сергіївна_____

Київ 2022

Лабораторна робота 4

Мета – вивчити механізми створення класів з використанням перевантажених операторів(операцій).

Індивідуальне завдання

Варіант 16

16. Побудувати клас BoolVector, що представляє булевий вектор (складається з булевих констант – 0 і 1). Реалізувати для нього декілька конструкторів, геттери, метод перевірки, чи є даний булевий вектор попередником іншого (булевий вектор $\alpha=a_1a_2\ldots a_n$ є попередником булевого вектора $\beta=b_1b_2\ldots b_n$, якщо для будь-якого $i = 1, 2, \ldots, n$ виконується умова $a_i < b_i$). Перевантажити оператори диз'юнкції ("|") та виключаюче АБО ("^") для компонент булевих векторів. Створити чотири булеві вектори (V1, V2, V3, V4), використовуючи різні конструктори. Визначити вектор V3 як диз'юнкцію булевих векторів V1 та V2 (V3 = V1 ∨ V2), а вектор V4 – як результат застосування операції "^"до булевих векторів V1 і V3 (V4 =V3 ^ V1). З'ясувати, чи є булевий вектор V4 попередником булевого вектора V3.

Виконання на C++

main.cpp

```cpp
#include <iostream>
#include "func.h"
#include "BoolVector.h"

using namespace std;

int main() {
    BoolVector vec1(input_vec()), vec2(input_vec());
    BoolVector vec3 = vec1|vec2;
    cout << "Vector 3 (disjunction v1|v2):" << endl;
    print_vec(vec3);
    BoolVector vec4 = vec1^vec3;
    cout << "Vector 4 (exclusive or v1^v3):" << endl;
    print_vec(vec4);
    cout << "V4 is predecessor of V3: ";
    if (vec4.is_prev(vec3)) {cout << "True" << endl;}
    else {cout << "False" << endl;}
}
```

BoolVector.cpp

```cpp
#include "BoolVector.h"

using namespace std;

BoolVector::BoolVector(const BoolVector &obj){
    size = obj.size;
    vec = new bool [size];
```

```
    for (int i = 0; i < size; ++i) {
        vec[i] = obj.vec[i];
    }
}

BoolVector::BoolVector(int n) {
    size = n;
    vec = new bool [n];
}

BoolVector::BoolVector(string line) {
    size = line.length();
    vec = new bool [size];
    for (int i = 0; i < size; i++) {
        vec[i] = (bool)(line[i]-48);
    }
}

BoolVector::BoolVector(const bool* nums, int len) {
    size = len;
    vec = new bool [len];
    for (int i = 0; i < len; i++) {vec[i] = nums[i];}
}

BoolVector BoolVector::operator|(const BoolVector& sec) {
    bool** vecs = get_vecs_same_size(sec);
    bool* f_vec = vecs[0];
    bool* s_vec = vecs[1];
    int length = max(size, sec.size);
    bool* res = new bool [length];
    for (int i = 0; i < length; ++i) {
        res[i] = f_vec[i] | s_vec[i];
    }
    return BoolVector(res, length);
}

BoolVector BoolVector::operator^(const BoolVector& sec) {
    bool** vecs = get_vecs_same_size(sec);
    bool* f_vec = vecs[0];
    bool* s_vec = vecs[1];
    int len = max(size, sec.size);
    bool* res = new bool [len];
    for (int i = 0; i < len; ++i) {
        res[i] = f_vec[i]^s_vec[i];
    }
    return BoolVector(res, len);
}

bool BoolVector::is_prev(const BoolVector& sec) {
    bool** res = get_vecs_same_size(sec);
    int len = max(size, sec.size);
    for (int i = 0; i < len; ++i) {
        if (res[0][i] && !res[1][i]) return false;
    }
    return true;
}

bool** BoolVector::get_vecs_same_size(const BoolVector& sec){
    bool** vecs = new bool* [2];
```

```
    if (size > sec.size){
        bool* s_vec = new bool [size];
        for (int i = 0; i < size-sec.size; i++){s_vec[i] = false;}
        for (int i = size-sec.size; i < size ; i++) { s_vec[i]=sec.vec[i-
size+sec.size];}
        vecs[0] = vec;
        vecs[1] = s_vec;
    }
    else if (sec.size > size){
        bool* f_vec = new bool [sec.size];
        for (int i = 0; i < sec.size-size; i++){f_vec[i] = false;}
        for (int i = sec.size-size; i < sec.size ; i++) { f_vec[i]=vec[i-
sec.size+size];}
        vecs[0] = f_vec;
        vecs[1] = sec.vec;
    }
    else {
        vecs[0] = vec;
        vecs[1] = sec.vec;
    }
    return vecs;
}
```

BoolVector.h

```
#pragma once
#include <iostream>
#include <vector>

using namespace std;

class BoolVector {
    int size;
    bool* vec;
public:
    explicit BoolVector(int n);
    explicit BoolVector(string line);
    explicit BoolVector(const bool* nums, int len);
    BoolVector(const BoolVector &obj);
    ~BoolVector(){delete[] vec;}
    bool* get_vec(){return vec;}
    int get_size() const{return size;}
    bool** get_vecs_same_size(const BoolVector& sec);
    BoolVector operator|(const BoolVector& sec);
    BoolVector operator^(const BoolVector& sec);
    bool is_prev(const BoolVector& sec);
};
```

func.cpp

```
#include "func.h"

bool is_bool(string line){
    for (char c : line) {
        if (c != '0' and c != '1'){return false;}
    }
    return true;
}

string input_vec(){
```

```
    string vec;
    cout << "Enter the vector:" << endl;
    cin >> vec;
    cin.ignore();
    while (!is_bool(vec)){
        cout << "All symbols have to be 0 or 1. Try again:" << endl;
        cin >> vec;
        cin.ignore();
    }
    cout << "Recorded" << endl;
    return vec;
}

void print_vec(BoolVector vec){
    bool* num = vec.get_vec();
    for (int i = 0; i < vec.get_size(); ++i) {cout << num[i];}
    cout << endl;
}
```

func.h

```
#pragma once
#include <iostream>
#include "BoolVector.h"

using namespace std;

string input_vec();
bool is_bool(string line);
void print_vec(BoolVector vec);
```

Висновки

Протягом лабораторної робити було розглянуто роботу з класами та перевантаженими операторами. Також були використані різні види конструкторів: з одним та двома параметрами, конструктор копіювання.