 МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Практичне заняття №4**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему: «Перевантаження операцій»

Варіант №11

**Виконав:**

студент гр. БС-52

Розюк П.В.

**Перевірив:**

ст. викладач каф. БМК

Радогуз А.В.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2017

**Завдання:**

1. Вивчити теоретичні основи перевантаження операторів в мові С++.
2. Розробити діаграму класів в нотації UML та програмний додаток для тестування всіх перевантажених операцій класу відповідно до свого варіанту(визначення класів та їх реалізація мають бути розташовані у файлаї \*.h та \*.cpp відповідно; для ініціалізації даниї класів використовувати введення з клавіатури; кожен клас, окрім зазначених у варіанті операцій має містити конструктор з параметрами, конструктор копіювання,деструктор,методи встановлення і отримання значень всіх полів класу):
3. Скласти і захистити звіт по роботі

**🞏 Практичне заняття без зауважень** **🞏 Практичне заняття має зауваження:**

**🞏 несвоєчасний захист**

**🞏 присутні зауваження до UML діаграми:**

**🞏 діаграма класу не відповідає коду**

**🞏 виконані не за стандартом:**

**🞏 атрибути**

**🞏 відношення**

**🞏 багатостатність**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 присутні зауваження до коду:**

**🞏 задача завдання вирішена хибно**

**🞏 код програми не компілюється**

**🞏 хибно задані специфікатори доступу**

**🞏 помилки у визначенні конструкторів / деструкторів**

**🞏 використано глобальні змінні**

**🞏 статичні змінні при роботі з масивами**

**🞏 оформлення коду**

**🞏 присутні зайві символи «{» та «}»**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 невірні відповіді на запитання:**

**🞏 №1 🞏 №2 🞏 №3 🞏 №4 🞏 №5**

**🞏 №6 🞏 №7 🞏 №8 🞏 №9 🞏 №10**

**🞏 маються інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Лістинг програми:**

**Main.cpp**

#include "Bvector.h"

#include <iostream>

#include <conio.h>

using namespace std;

void vector(int size,Bvector C)

{

cout << "Заполните значениями вектор A: ";

Bvector A(size);

A.calculation(size);

cout << "\nЗаполните значениями вектор B: ";

Bvector B(size);

B.calculation(size);

cout << "\nДизъюнкция векторов А и В: ";

C = A | B;

C.print();

cout << "Конъюнкция векторов А и В: ";

C = A & B;

C.print();

cout << "Инверсия A: ";

C = ~A;

C.print();

}

void main()

{

setlocale(LC\_ALL, "rus");

int size;

cout << "Введите значение длины вектора: ";

cin >> size;

Bvector C;

vector(size, C);

\_getch();

}

**Bvector.h**

#pragma once

#include <iostream>

using namespace std;

class Bvector

{

private:

bool \*arr;

int size;

public:

Bvector() {};

Bvector(int size);

Bvector(Bvector &object);

void set\_size(int size);

void set\_arr(int size);

int get\_size();

bool \*get\_arr();

void calculation(int);

void print();

Bvector operator = (Bvector);

Bvector operator | (Bvector);// диз'юнкція

Bvector operator & (Bvector);//кон'юнкція

Bvector operator ~ ();// інверсія

~Bvector();

};

**Bvector.cpp**

#include <iostream>

#include "Bvector.h"

using namespace std;

Bvector::Bvector(int size)

{

this->size = size;

arr = new bool[size];

for (int i = 0; i < size; i++)

{

cin >> arr[i];

}

}

Bvector::Bvector(Bvector &object)

{

size = object.size;

arr = new bool[size];

for (int i = 0; i<size; i++)

{

arr[i] = object.arr[i];

}

}

void Bvector::set\_size(int size)

{

this->size = size;

}

void Bvector::set\_arr(int size)

{

for (int i = 0; i < size; i++)

{

cout << "Введите заначание "<<i<<": ";

cin >> arr[i];

}

}

int Bvector::get\_size()

{

return size;

}

bool \*Bvector::get\_arr()

{

return arr;

}

void Bvector::calculation( int size)

{

int x1=0, x2=0;

for (int i = 0; i < size; i++)

{

if (arr[i] == 0)

x1++;

else x2++;

}

cout << "\nКоличество нулей в векторе: "<<x1<<"\n";

cout << "Количество единиц в векторе: "<< x2 << "\n";

}

void Bvector::print()

{

for (int i = 0; i < size; i++)

cout << arr[i] << " ";

cout << endl;

}

Bvector Bvector::operator=(Bvector t)

{

size = t.size;

arr = new bool[size];

for (int i = 0; i<size; i++)

{

arr[i] = t.arr[i];

}

return \*this;

}

Bvector Bvector:: operator | (Bvector t)

{

Bvector c;

c.size = size;

c.arr = new bool[size];

if (size == t.size)

{

for (int i = 0; i < size; i++)

{

c.arr[i] = (arr[i] | t.arr[i]);

}

}

return c;

}

Bvector Bvector::operator & (Bvector t)

{

Bvector c;

c.size = size;

c.arr = new bool[size];

if (size == t.size)

{

for (int i = 0; i < size; i++)

{

c.arr[i] = (arr[i] & t.arr[i]);

}

}

return c;

}

Bvector Bvector::operator ~ ()

{

{

for (int i = 0; i < size; i++)

{

arr[i] =!arr[i];

}

}

return \*this;

}

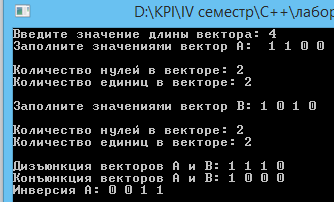
Bvector ::~Bvector()

{

delete arr;

}

**Скрін-шот роботи програми:**



**UML-діаграма:**



**Контрольні питання:**

1. Навіщо потрібне перевантаження операторів?

Перевантаження операторів використовується для розширення застосування операторів на класи, створювані програмістом. Оскільки компілятор має убодовані типи даних з визначеним набором операцій, які можна до нього застосувати ( за допомогою відомих йому операндів +,-,/,// і т.д), але компілятор не знає як застосувати ті чи інші операнди для користувацьких типів даних. Для цього в мові С++ існує механізм перевантаження операторів.

1. Якими двома різними способами визначаються перевантажені операції?

Будь-яка операція може бути визначена двома способами: як компонентна функція класу або як глобальна (дружня) функція класу.

1. Які оператори мови C++ не можуть бути перевантажені?

Не може бути перевантажена операція доступу до членів класу (.), оператор доступу до елементу за покажчиком (\*.), операція розширення області видимості (::), тернарний оператор умови (?:), та оператор знаходження розміру об’кта в байтах (sizeof)

1. Які операції не можна перевантажити за допомогою глобальної дружньої функції? Відповідь пояснити.

Оператор отримання індексу [], оскільки він завжди визначається як член класу і звертається до певого елементу масива, то йому варто повертати посилання.

Оператор розіменування покажчика \*, оскільки він обовязково має повертати посилання на бєкт або покажчик, що дозволяє звернутися до обєкта, то він завжди має визначатися як функція – член.

Оператор присвоєння (=) оскільки він нерозривно пов'язаний з об’єктом, що знаходиться ліворуч від операнда.

1. У яких випадках операцію можна перевантажити тільки за допомогою глобальної функції?

У випадках, коли першим параметром функції є стандартний тип ( якщо для класу а перевантажена операція «+» помилковим буде виклик оператора <int>.operator + (variable), а також операції вводу ( >>) та виводу (<<).

1. У яких випадках глобальна операція-функція повинна бути дружньою?

У випадку необхідності прямого доступу до закритих або захищених даних.

1. Чи обов’язковий у функції operator параметр типу «клас» або «посилання на клас»? Відповідь пояснити.

Параметр типу «клас» або «посилання на клас» є не обовязковим у функції operator, оскільки перевантаження допускається для будь-яких типів даних.

1. Чи можна успадкувати або повторно перевантажити в похідному класі операцію, що була перевантажена в базовому класі?

Так, можна, але при цьому метод базового класу стане прихованим і використовувати його можна буде лише за допомогою ключового слова using::

1. Чому не можна перевантажувати деструктори?

Деструктор не приймає ніяких параметрів і не повертає ніяких значень. Клас може мати тільки один деструктор – перевантаження деструктора не дозволяється ( він є індивідуальним для кожного типу користувача )

1. Скільки необхідно передати параметрів функції-методу класу, що реалізує перевантаження бінарної операції?

Для бінарної операції необхідно передати один параметр, оскільки під виразом a + b використовується перевантаження оператора у іигляді а.operator+(b).