 МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Компьютерный практикум №4**

з дисципліни «Об'єктно-орієнтоване програмування»

# на тему: «Перевантаження операцій»

Варіант №7

**Виконав:**

студент гр. БС-03

Затуловский Г. А.

**Перевірив:**

ас. каф. БМК Бабенко В.О.

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

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

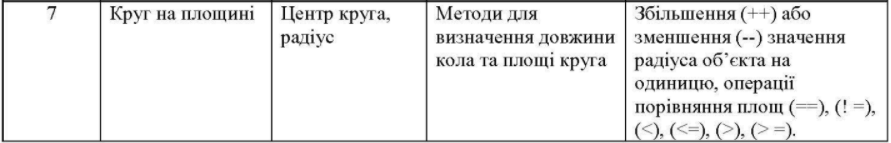
                      (підпис викладача)

Київ-2022

**Завдання:**

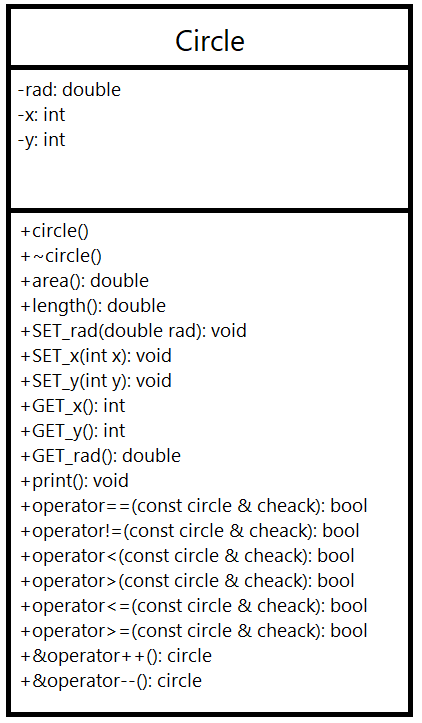
1. Вивчити теоретичні основи перевантаження операторів в мові С++.

2. Розробити діаграму класів в нотації UML та програмний застосунок для тестування всіх перевантажених операцій класу відповідно до свого варіанту (визначення класів та їх реалізації мають бути розташовані у файлах \*.h та \* cpp вiдповiдно; для ініціалізації даних 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. Скрасти і захистити звіт по роботі.

**UML діаграма:**

****

**Код програми:**

**Circle.cpp**

#include "circle.h"

#include <math.h>

circle::circle(double rad, int x, int y)

{

this->rad = rad;

this->x = x;

this->y = y;

}

circle::circle()

{

rad = x = y = 0;

}

circle::~circle()

{

}

double circle::length()

{

double pi = 3.14;

return (rad\*pi\*2);

}

double circle::area()

{

double pi = 3.14;

return (rad \* pow(pi,2));

}

void circle::SET\_rad(double rad)

{

this->rad = rad;

}

void circle::SET\_x(int x)

{

this->x = x;

}

void circle::SET\_y(int y)

{

this->y = y;

}

double circle::GET\_rad()

{

return rad;

}

int circle::GET\_x()

{

return x;

}

int circle::GET\_y()

{

return y;

}

void circle::print()

{

cout << "Circle center coordinates: [" << x << " ; " << y << "]\n"

<< "r = " << rad << " cm\n"

<< "l = " << length() << " cm\n"

<< "S = " << area() << " cm\n"

<<endl;

}

bool circle::operator==(const circle& cheak)

{

return (this->rad == cheak.rad);

}

bool circle::operator!=(const circle& cheak)

{

return (this->rad == cheak.rad);

}

bool circle::operator<(const circle& cheak)

{

return (this->rad < cheak.rad);

}

bool circle::operator>(const circle& cheak)

{

return (this->rad > cheak.rad);

}

bool circle::operator<=(const circle& cheak)

{

return (this->rad <= cheak.rad);

}

bool circle::operator>=(const circle& cheak)

{

return (this->rad >= cheak.rad);

}

circle& circle::operator++()

{

this->x++;

this->y++;

this->rad++;

return \*this;

}

circle& circle::operator--()

{

this->x--;

this->y--;

this->rad--;

return \*this;

}

**Circle.h**

#pragma once

#include <string>

#include <iostream>

using namespace std;

class circle

{

private:

double rad;

int x;

int y;

public:

circle(double rad, int x, int y);

circle();

~circle();

double length();

double area();

void SET\_rad(double rad);

void SET\_x(int x);

void SET\_y(int y);

double GET\_rad();

int GET\_x();

int GET\_y();

void print();

bool operator == (const circle & cheak);

bool operator != (const circle& cheak);

bool operator < (const circle& cheak);

bool operator > (const circle& cheak);

bool operator <= (const circle& cheak);

bool operator >= (const circle& cheak);

circle& operator ++();

circle& operator --();

};

**OOP\_4.cpp**

#include "circle.h"

#include <string>

#include <iostream>

using namespace std;

void Cheak(circle& A, circle& B)

{

cout << "=================\nr - radius" << endl;

cout << "r(A) == r(B) = " << (A == B) << endl;

cout << "r(A) != r(B) = " << (A != B) << endl;

cout << "r(A) > r(B) = " << (A > B) << endl;

cout << "r(A) >= r(B) = " << (A >= B) << endl;

cout << "r(A) < r(B) = " << (A < B) << endl;

cout << "r(A) <= r(B) = " << (A <= B) << endl;

cout << "=================" << endl;

}

void working\_with\_circle()

{

int x = 0, y = 0;

double rad = 0;

cout << "Input circle A center coordinates: x = ";

cin >> x;

cout << "Input circle A center coordinates: y = ";

cin >> y;

cout << "Input radius A: ";

while (rad <= 0)

{

cin >> rad;

}

circle A(rad, x, y);

rad = 0;

cout << "\nCircle A" << endl;

A.print();

cout << "Input circle B center coordinates: x = ";

cin >> x;

cout << "Input circle B center coordinates: y = ";

cin >> y;

cout << "Input radius B: ";

while (rad <= 0)

{

cin >> rad;

}

circle B(rad,x, y);

rad = 0;

cout << "\nCircle B" << endl;

B.print();

Cheak(A,B);

A.operator--();

B.operator++();

cout << "\nCircle A after - 1" << endl;

A.print();

cout << "\nCircle B after + 1" << endl;

B.print();

Cheak(A, B);

}

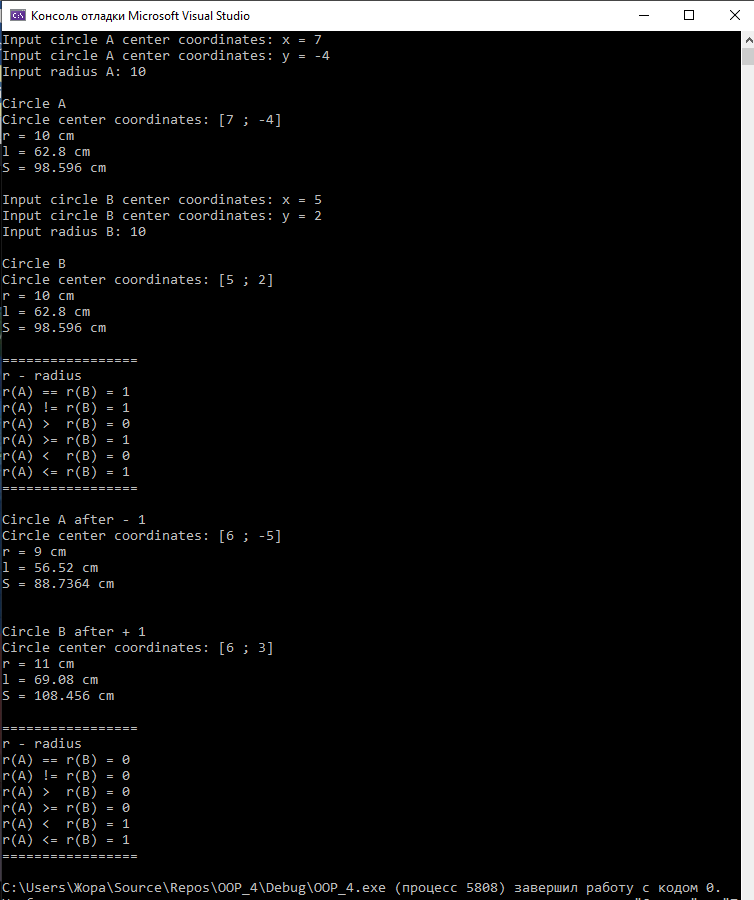
int main()

{

working\_with\_circle();

}

**Результати:**



**Контрольні запитання:**

**1. Навіщо потрібне перевантаження операторів?**

Завдяки перевантаження операторів програміст, що почав писати код програми, змінити семантику операторів, то інші розробники, що будуть працювати далі на кодом, можуть зіткнутись із неочікуваною поведінкою операторів.

**2. Якими двома різними способами визначаються перевантажені операції?**

1. Завдяки вбудованих функцій

2. Завдяки глобальним змінним або функціям

**3. Які оператори мови С++ не можуть бути перевантажені?**

До таких операторів належать :: (оператор дозволу області видимості), . (“точка” — оператор доступу до члена класу), ?: (тернарний оператор), .\* (доступ до розіменованого вказівника-члена класу), sizeof, typeid, static\_cast, dynamic\_cast, const\_cast і reinterpret\_cast.

**4. Які операції не можна перевантажити за допомогою глобальної дружньої функції? Вiдповiдь пояснити.**

Операції: ( ), [], -> та = . Функції перевантаження цих операцій повинні бути оголошені як елементи класу.

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

Опереацію можна перевантажити за допомогою глобальної функції лише в випадку коли з правої сторони перевантаженого оператора розташований об’єкт, а по ліву існуючий тип даних.

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

У випадках якщо не має доступу до private полів, то функція повина бути дружнью.

**7. Чи обов'язковий у функції operator параметр класового типу або посилання на клас? Вiдповiдь пояснити.**

Ні. Функції operator перевантаження префіксного варіанту операцій інкремента та декремента не містить параметрів.

**circle& operator ++();**

**circle& operator --();**

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

Успадкувати в похідному класу операцію, що була перевантажена в базовому класі, не можна. Повторно перевантажити можна.

**9. Чому не можна перевантажувати деструктори?**

Деструктори не можна перевантажувати через відсутність у них аргументів.

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

Необхідно передати один параметр