**Цель работы:** Изучение основ объектно-ориентированного программирования, получение навыков создания дружественных функций перегрузки операций в классе.

**Постановка задачи.**

Дополнить класс Треугольник, описанный в предыдущей работе методами – перегруженными операциями сложения, вычитания, умножения, постфиксной и префиксной формой инкремента и декремента. Одну бинарную и одну унарную операцию реализовать как дружественную функцию.

Описать в классе конструктор копирования, предусмотреть в нём изменение статического поля –количества объектов.

Продемонстрировать работу всех описанных методов на созданных объектах. Один из перегруженных операторов реализовать в виде дружественной функции.

Вывести на экран информацию о количестве объектов.

Указания к работе.

Все перегруженные операции должны реализовывать действия, приводящие к результату, имеющему геометрический смысл.

Например, возможен такой подход. Операции сложения и вычитания возвращают сумму и разность площадей треугольников соответственно, операция умножения реализует масштабирование треугольника, операции декремента и инкремента изменяют значение угла между двумя сторонами, здесь важно не забыть пересчитать значение третей стороны треугольника.

**Теоретический материал.**

Давайте разберем теорию дружественных функций и перегрузки операторов в C++.

Дружественные функции (Friend Functions)

Определение:

Дружественные функции — это функции, которые не являются членами класса, но имеют доступ к его закрытым (private) и защищенным (protected) членам.

Синтаксис:

Чтобы объявить дружественную функцию, используйте ключевое слово friend.

Пример:

#include <iostream>

class MyClass {

private:

int value;

public:

MyClass(int v) : value(v) {}

// Объявление дружественной функции

friend void showValue(const MyClass& obj);

};

void showValue(const MyClass& obj) {

std::cout << "Value: " << obj.value << std::endl; // Доступ к закрытому члену value

}

int main() {

MyClass obj(10);

showValue(obj); // Вывод: Value: 10

return 0;

}

Перегрузка операторов

Определение:

Перегрузка операторов — это возможность определять новое поведение операторов для пользовательских типов данных (классов).

Общий синтаксис для перегрузки операторов:

ReturnType operatorOperatorSymbol (const Parameters) {

// Реализация

}

Пример:

1. Перегрузка оператора сложения (+) для класса Point:

#include <iostream>

class Point {

public:

int x, y;

Point(int x, int y) : x(x), y(y) {}

// Перегрузка оператора +

Point operator+(const Point& p) const {

return Point(this->x + p.x, this->y + p.y);

}

};

// Перегрузка оператора вывода для класса Point

std::ostream& operator<<(std::ostream& os, const Point& p) {

os << "(" << p.x << ", " << p.y << ")";

return os;

}

int main() {

Point p1(1, 2);

Point p2(3, 4);

Point p3 = p1 + p2; // Используем перегруженный оператор +

std::cout << "Сложение: " << p3 << std::endl; // Вывод: (4, 6)

return 0;

}

Основные моменты

1. Ненарушение инварианта:

При перегрузке операторов старайтесь сохранять инварианты класса и логику, присущую перегруженному оператору. Например, например, A + B должно давать результат, который логически соответствует сложению.

2. Параметры и возвращаемые значения:

- Операторы могут принимать параметры (обычно один или два объекта), и могут возвращать новые объекты (например, сложение двух векторов).

- Подумайте о том, нужно ли возвращать объект по значению или по ссылке. Например, для перегрузки оператора =, вы можете вернуть \*this (ссылку на текущий объект).

3. Доступ к приватным членам:

Если вам нужно получить доступ к закрытым членам класса из дружественной функции, вы можете реализовать это как дружественную функцию внутри класса.

4. Сложные операторы:

Некоторые операторы, такие как [], (), ->, обычно перегружаются как члены класса, так как они работают с конкретным экземпляром класса.

5. Стандарты и стиль:

Следуйте стандартам программирования и старайтесь поддерживать читаемость и понятность вашего кода.

Вывод

Дружественные функции и перегрузка операторов являются мощными инструментами в C++. Они позволяют создавать более интуитивно понятные и удобные интерфейсы для работы с пользовательскими типами данных. Благодаря этому, вы можете широко применять операторы, такие как +, -, \*, и другие в вашем коде, делая его более естественным для чтения и написания.

**Практическая часть.**

**Triangle.h**

#pragma once

#include <iostream>

using namespace std;

class Triangle

{

private:

//для 1 лабораторной

double str1;

double str2;

double str3;

double angle;

static int count;

void Find();

//для 2 лабораторной

public:

//для 1 лабораторной

Triangle();

Triangle(double str1, double str2, double angle);

//конструктор копирования

Triangle(const Triangle& \_triangle);

double GetAngle();

double GetStr1();

double GetStr2();

double GetP();

double GetS();

void SetAngle(double angle);

void SetStr1(double str1);

void SetStr2(double str2);

void CountTriangl();

~Triangle();

//для 2 лабораторной

double operator + (const Triangle& triangle) const;

double operator - (const Triangle& triangle) const;

Triangle operator \* (double number) const;

Triangle operator ++ (int);

Triangle operator -- (int);

Triangle& operator ++ ();

Triangle& operator --();

friend double operator +(const Triangle& triangle1, const Triangle& triangle2);

double Area() const;

void Print();

};

**Thriangle.cpp**

#include "Triangle.h"

#define \_USE\_MATH\_DEFINES

#include <math.h>

//для 1 лабораторной

//конструктор по умолчанию

Triangle::Triangle() {

this->str1 = 3;

this->str2 = 4;

this->angle = 90;

Find();

CountTriangl();

}

//конструктор с параметрами

Triangle::Triangle(double str1, double str2, double angle) {

this->str1 = str1;

this->str2 = str2;

this->angle = angle;

Find();

CountTriangl();

}

//конструктор копирования

Triangle::Triangle(const Triangle& triangle) {

str1 = triangle.str1;

str2 = triangle.str2;

angle = triangle.angle;

Find();

CountTriangl();

}

void Triangle::Find() {

str3 = sqrt((str1 \* str1) + (str2 \* str2) - (2 \* str1 \* str2 \* cos(angle \* M\_PI / 180)));

}

double Triangle::GetStr1() {

return str1;

}

double Triangle::GetStr2() {

return str2;

}

double Triangle::GetP() {

return str1 + str2 + str3;

}

double Triangle::GetS() {

double S = (str1 \* str2 \* sin(angle \* M\_PI / 180) / 2);

return S;

}

double Triangle::GetAngle() {

return angle;

}

void Triangle::SetAngle(double angle) {

if (angle > 0 && angle < 180) {

this->angle = angle;

}

else {

cout << "Угол не может быть меньше 0 и больше 180 градусов или у Вас он не указан." << endl;

this->angle = 52;

}

}

void Triangle::SetStr1(double str1) {

this->str1 = str1;

Find();

}

void Triangle::SetStr2(double str2) {

this->str2 = str2;

Find();

}

void Triangle::CountTriangl() {

++count;

cout << "Создано " << count << " объект(а)" << endl;

}

Triangle::~Triangle() {

count--;

}

//для 2 лабораторной

//метод для плозади по формуле герона

double Triangle::Area() const{

double resS = (str1 + str2 + str3) / 2;

return sqrt(resS \* (resS - str1) \* (resS - str2) \* (resS - str3));

}

Triangle Triangle::operator\*(double number) const {

return Triangle(str1 \* number, str2 \* number, str3 \* number);

}

double Triangle::operator+(const Triangle& triangle) const {

return this->Area() + triangle.Area();

}

double Triangle::operator-(const Triangle& triangle) const {

return this->Area() - triangle.Area();

}

Triangle& Triangle::operator++() {

angle++;

return \*this;

}

Triangle& Triangle::operator--() {

angle--;

return \*this;

}

Triangle Triangle::operator++(int) {

Triangle temp = \*this;

++(\*this);

return temp;

}

Triangle Triangle::operator--(int) {

Triangle temp = \*this;

--(\*this);

return temp;

}

double operator+(const Triangle& triangle1 , const Triangle& triangle2) {

return triangle1.Area() + triangle2.Area();

}

void Triangle::Print() {

cout << "Треугольник: сторона 1 = " << str1 << ", сторона 2 = " << str2 << ", сторона 3 = " << str3 << ", угол = " << angle << ", площадь = " << Area() << endl;

}

**Lab1.cpp**

#include <iostream>

#include "Triangle.h"

using namespace std;

int Triangle::count;

int main()

{

setlocale(LC\_ALL, "rus");

//2 лабораторная работа

cout << "\n2 лабораторная работа." << endl;

Triangle triangle4;

Triangle triangle5(5, 12, 9);

triangle4.Print();

triangle5.Print();

//сумма и разность площадей

cout << "\nСумма и разность площадей:" << endl;

cout << "Сумма = " << triangle4.Area() + triangle5.Area() << endl;

cout << "Разность = " << triangle4 - triangle5 << endl;

//масштабирование треуголника

cout << "\nМасштабирование треуголника:" << endl;

Triangle triangle6 = triangle5 \* 2;

triangle6.Print();

//изменение значений угла

cout << "\nИзменение значений угла:" << endl;

Triangle triangle8 = --triangle4;

triangle8.Print();

Triangle triangle9 = ++triangle4;

triangle9.Print();

}

**Результат выполнения кода.**

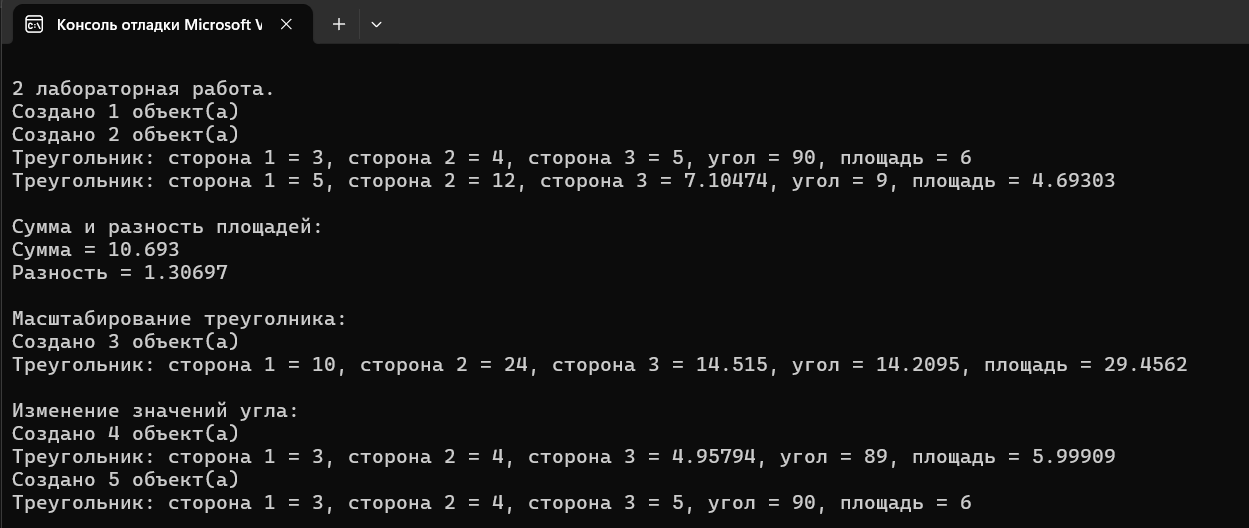


Рисунок 1 – Результат выполнения кода

**Вывод:** в данной лабораторной работе были закреплены навыки работы с классами на языке программирования C++, а также ознакомились с реализацией дружественных функций и перегрузкой операторов.