**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

Направление подготовки: «Фундаментальная информатика и информационные технологии»

Отчёт по лабораторной работе

**Работа с геометрическими фигурами в N-мерном пространстве**

Выполнил:

студент ИИТММ гр. 381906-2

Шелепина Н. А.

Проверил:

ассистент каф. математического обеспечения и суперкомпьютерных технологий

Лебедев И.Г.

Нижний Новгород

2020 г.

Содержание

[Введение 3](#_Toc26846310)

[1. Постановка задачи 4](#_Toc26846311)

[2. Руководство пользователя 5](#_Toc26846312)

[3. Руководство программиста 6](#_Toc26846313)

[3.1. Описание структуры программы 6](#_Toc26846314)

[3.2. Описание структур данных 6](#_Toc26846315)

[3.3. Описание алгоритмов 11](#_Toc26846316)

[4. Эксперименты 12](#_Toc26846317)

[Заключение 13](#_Toc26846318)

[Литература 14](#_Toc26846319)

[Приложение 15](#_Toc26846320)

# Введение

На сегодняшний день математическое программирование - важная составляющая всего программирования. Одним из главных понятий математики, в частности, геометрии, является геометрический объект. Для работы с геометрическими объектами удобно использовать следующий метод программирования - ООП.

Также для работы нам понадобятся следующие понятия:

* Класс – это тип структуры, позволяющий включать в описание типа не только элементы данных, но и функции (функции-элементы или методы);
* Шаблоны функций – это обобщенное описание поведения функций, которые могут вызываться для объектов разных типов. Другими словами, шаблон функции (шаблонная функция, обобщённая функция) представляет собой семейство разных функций (или описание алгоритма). По описанию шаблон функции похож на обычную функцию: разница в том, что некоторые элементы не определены (типы, константы) и являются параметризованными.
* Шаблоны классов – обобщенное описание пользовательского типа, в котором могут быть параметризованы атрибуты и операции типа. Представляют собой конструкции, по которым могут быть сгенерированы действительные классы путём подстановки вместо параметров конкретных аргументов.

Данная лабораторная работа демонстрирует реализацию работы с простейшими геометрическими объектами, такими, как: точка, линия, круг, треугольник, четырехугольник, сфера, эллипс.

# Постановка задачи

1. Написать структуру данных для работы с геометрическими объектами в N мерном пространстве. Тип данных определяется пользователем (шаблоны).
2. В программе должны быть реализованы простейшие геометрические объекты, такие, как: точка, линия, круг, треугольник, четырехугольник, сфера, эллипс.
3. Кроме самих геометрических объектов должен быть реализован класс, осуществляющий обобщение действий со всеми созданными пользователями объектами - "контейнер".
4. Контейнер должен иметь следующие функции: добавить новый объект, удалить существующий объект, отобразить все имеющиеся объекты, отобразить выбранный объект и т.п.
5. Продемонстрировать (написать в main пример) работоспособность.
6. Должны быть использованы и продемонстрированы: абстракция, инкапсуляция, наследование, полиморфизм, перегрузка операций, шаблоны.
7. Иерархия должна содержать не менее 7 классов.

# Руководство пользователя

Использование программы пользователем:

1. Создать геометрические объекты необходимых типов: Point, Line, Circle, Triangle, Quadrilateral, Sphere, Ellipse;
2. Наполнить выбранные геометрические объекты с помощью сеттеров.
3. Создать объект класса «контейнер» и добавить в него нужные геометрические фигуры.

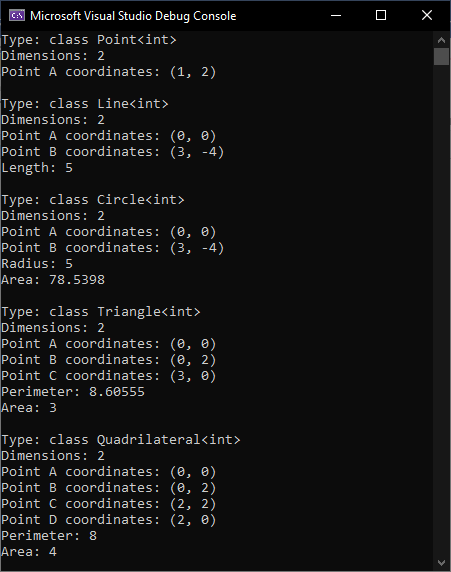


Рисунок 1. Cкриншот интерфейса программы.

# 

# 

# 3. Руководство программиста

## 3.1. Описание структуры программы

Программа состоит из одного решения.

В решении GeometricShapes определено 17 модулей main.cpp, Base.h, Point.h, Point\_imp.h, Line.h, Line\_imp.h, Circle.h, Circle\_imp.h, Triangle.h, Triangle\_imp.h, Quadrilateral.h, Quadrilateral\_imp.h, Sphere.h, Sphere\_imp.h, Ellipse.h, Ellipse\_imp.h, Container.h, Container\_imp.h.

* В модуле main.cpp определена стандартная функция int main(), внутри которой содержится набор действий с геометрическими объектами;
* В модуле Base.h определен абстрактный класс, предок всех геометрических фигур;
* В модуле Point.h определен класс Point (наследник класса Base);
* В модуле Point\_imp.h написана реализация класса Point;
* В модуле Line.h определен класс Line (наследник класса Point);
* В модуле Line\_imp.h написана реализация класса Point;
* В модуле Circle.h определен класс Circle (наследник класса Line);
* В модуле Circle\_imp.h написана реализация класса Circle;
* В модуле Triangle.h определен класс Triangle (наследник класса Line);
* В модуле Triangle\_imp.h написана реализация класса Triangle;
* В модуле Quadrilateral.h определен класс Quadrilateral (наследник класса Triangle);
* В модуле Quadrilateral\_imp.h написана реализация класса Quadrilateral;
* В модуле Sphere.h определен класс Sphere (наследник класса Circle);
* В модуле Sphere\_imp.h написана реализация класса Sphere;
* В модуле Ellipse.h определен класс Ellipse (наследник класса Triangle);
* В модуле Ellipse\_imp.h написана реализация класса Ellipse;
* В модуле Container.h определен класс Container;
* В модуле Container\_imp.h написана реализация класса Container.

## 3.2. Описание структур данных

В программе определен один абстрактный базовый класс под названием Base.

Внутри этого класса определены следующие виртуальные функции:

* virtual ~Base() {} – виртуальный деструктор;
* virtual ostream& print(ostream& os) = 0 – виртуальный метод вывода информации об объекте;
* friend ostream& operator<<(ostream& os, Base& other) – виртуальный метод вывода;

В программе также определено 7 шаблонных классов: Point, Line, Circle, Triangle, Quadrilateral, Sphere, Ellipse.

Внутри класса Point определено следующее поле:

* Int dim – значение размерности;
* T\* A – массив координат точки А.

Внутри класса Point определены следующий набор public-методов:

* Point () – конструктор по умолчанию, не принимает никаких параметров, инициализирует все два поля 0 через списки инициализации;
* Point(int n) – конструктор-инциализатор, принимает на вход шаблонный параметр и инициализирует поле value значением, переданным в конструктор;
* Point(const Point& p) – конструктор копирования, принимает на вход объект типа Point, создает объект с теми же характеристиками;
* virtual ~Point() – виртуальный деструктор;
* T\* GetA() const – геттер массива координат точки А;
* int GetDim() const – геттер значения размерности;
* T\*& SetA() – сеттер точки А;
* int& SetDim() ­– сеттер значение размерности.
* ostream& print(ostream& os) override – перекрытие функции вывода информации об объекте.

Внутри класса Line определены следующие поля:

* T\* B – массив координат точки B.

Внутри класса Line определены следующий набор public-методов:

* Line () – конструктор по умолчанию, не принимает никаких параметров, инициализирует все два поля 0 через списки инициализации;
* Line (int n) – конструктор-инциализатор, принимает на вход шаблонный параметр и инициализирует поле value значением, переданным в конструктор;
* Line (const Line& p) – конструктор копирования, принимает на вход объект типа Line, создает объект с теми же характеристиками;
* virtual ~Line() – виртуальный деструктор;
* double GetLength() const – метод получения длины;
* T\* GetB() const – геттер массива координат точки А;

T\*& SetB() – сеттер точки А;

* ostream& print(ostream& os) override – перекрытие функции вывода информации об объекте.

Внутри класса Circle определены следующий набор public-методов:

* Circle() : Line<T>() – конструктор по умолчанию, унаследованный от Line;
* Circle(int n) : Line<T>(n) – конструктор-инциализатор, унаследованный от Line;
* Circle(const Circle& c) : Line<T>(c) – конструктор копирования, унаследованный от Line;
* virtual ~ Circle () – виртуальный деструктор;
* double GetRadius() const – метод получения радиуса;
* ostream& print(ostream& os) override – перекрытие функции вывода информации об объекте.

Внутри класса Triangle определены следующие поля:

* T\* C – массив координат точки C.

Внутри класса Line определены следующий набор public-методов:

* Triangle () – конструктор по умолчанию, не принимает никаких параметров, инициализирует все два поля 0 через списки инициализации;
* Triangle (int n) – конструктор-инциализатор, принимает на вход шаблонный параметр и инициализирует поле value значением, переданным в конструктор;
* Triangle (const Triangle& p) – конструктор копирования, принимает на вход объект типа Triangle, создает объект с теми же характеристиками;
* virtual ~ Triangle () – виртуальный деструктор;
* virtual double GetPerimeter() const – виртуальный метод получения периметра;
* virtual double GetArea() const – виртуальный метод получения площади;
* T\* GetС() const – геттер массива координат точки С;

T\*& SetС() – сеттер точки С;

* ostream& print(ostream& os) override – перекрытие функции вывода информации об объекте.

Внутри класса Sphere определены следующий набор public-методов:

* Sphere() : Circle<T>() – конструктор по умолчанию, унаследованный от Circle;
* Sphere(int n) : Circle<T>(n) – конструктор-инциализатор, унаследованный от Circle;
* Sphere(const Sphere& s) : Circle<T>(s) – конструктор копирования, унаследованный от Circle;
* virtual ~Sphere() – виртуальный деструктор;
* double GetRadius() const – метод получения радиуса;
* ostream& print(ostream& os) override – перекрытие функции вывода информации об объекте.

Внутри класса Quadrilateral определены следующие поля:

* T\* D – массив координат точки D.

Внутри класса Line определены следующий набор public-методов:

* Quadrilateral () – конструктор по умолчанию, не принимает никаких параметров, инициализирует все два поля 0 через списки инициализации;
* Quadrilateral (int n) – конструктор-инциализатор, принимает на вход шаблонный параметр и инициализирует поле value значением, переданным в конструктор;
* Quadrilateral (const Quadrilateral & p) – конструктор копирования, принимает на вход объект типа Quadrilateral, создает объект с теми же характеристиками;
* virtual ~ Quadrilateral () – виртуальный деструктор;
* double GetPerimeter() const – метод получения периметра;
* double GetArea() const – метод получения площади;
* bool IsSquare() const – метод определения, является ли четырехугольник квадратом;
* T\* GetD() const – геттер массива координат точки D;

T\*& SetD() – сеттер точки D;

* ostream& print(ostream& os) override – перекрытие функции вывода информации об объекте.

Внутри класса Ellipse определены следующий набор public-методов:

* Ellipse (): Triangle<T>() – конструктор по умолчанию, унаследованный от Triangle;
* Ellipse (int n) : Triangle <T>(n) – конструктор-инциализатор, унаследованный от Triangle;
* Ellipse (const Circle& c) : Triangle <T>(c) – конструктор копирования, унаследованный от Triangle;
* virtual ~ Ellipse () – виртуальный деструктор;
* double GetAxisA() const – метод получения длины большой полуоси;
* double GetAxisB() const – метод получения длины малой полуоси;
* double GetPerimeter() const – метод получения периметра;
* double GetArea() const – метод получения площади;
* ostream& print(ostream& os) override – перекрытие функции вывода информации об объекте.

В программе определен один класс-контейнер под названием Container.

Внутри данного класса определены следующие поля:

* Base\*\* Array – массив указателей типа Base;
* vector<string> names – вектор из строк для хранения имен объектов;
* int size – размер контейнера.

Внутри класса Container определены следующий набор public-методов и конструкторы с деструктором:

* Container () – конструктор по умолчанию, не принимает никаких параметров;
* Container(const Container& c) – конструктор копирования;
* virtual ~Container() – виртуальный деструктор;
* void Add(A& elem, string name) – метод добавления в контейнер элементов, принимает на вход объект и имя;
* void Remove(int r) – метод удаления объекта из контейнера, принимает на вход индекс;
* void ShowAll() – метод для вызова списка всех элементов;
* void Show(int i) – метод для вызова информации конкретного элемента, принимает на вход индекс;
* Shapes\* operator[](const int i) – перегрузка оператора возврата объекта по индексу;
* Base\* operator[](const int i) ***–*** перегрузка оператора индексации.

## 3.3. Описание алгоритмов

В данной программе присутствуют простейшие алгоритмы вычисления длины вектора, вычисления периметра, площади, объема фигур.

# 4. Эксперименты

В функции main приведен пример создания объектов, добавления их в контейнер, вызова списка всех объектов, удаления объекта из контейнера и вызова информации по объекту из контейнера.

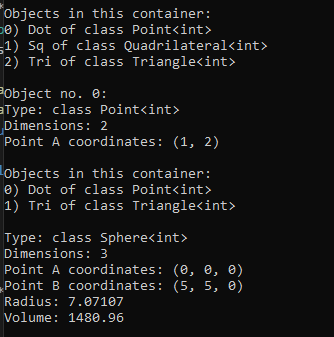


Рисунок 2: Демонстрация работы контейнера.

# Заключение

Таким образом, в рамках данной лабораторной работы была успешно создана программа, которая позволяет работать с геометрическими объектами, такими, как точка, линия, треугольник, круг, четырехугольник, сфера и эллипс в N-мерном пространстве. Были использованы и продемонстрированы основные принципы объектно-ориентированного программирования, а также был реализован контейнер, использующийся для работы с геометрическими объектами. Программа полностью выдержала проверку и готова к использованию.

# Литература

1. Павловская Т.А. C/C++, Программирование на языке высокого уровня, 2003.
2. Официальный сайт Habr. – Режим доступа <https://habr.com/ru/post/339656/>
3. Официальный сайт Microsoft. – Режим доступа <https://docs.microsoft.com/ru-ru/cpp/c-language/main-function-and-program-execution?view=vs-2019>
4. Сайт Stackoverflow.com. – Режим доступа <https://stackoverflow.com>

# Приложение

***Код программы***

### Main.cpp

#include "Point.h"

#include "Line.h"

#include "Triangle.h"

#include "Circle.h"

#include "Quadrilateral.h"

#include "Sphere.h"

#include "Ellipse.h"

#include "Container.h"

#define Add(x) Add(x, #x)

#define SetA(t) SetA()[t]

#define SetB(t) SetB()[t]

#define SetC(t) SetC()[t]

#define SetD(t) SetD()[t]

using namespace std;

int main()

{

Point<int> Dot(2);

Dot.SetA(0) = 1; Dot.SetA(1) = 2;

cout << Dot << endl;

Point<int> Dot2(Dot);

Line<int> Vec(2);

Vec.SetA(0) = 0; Vec.SetA(1) = 0;

Vec.SetB(0) = 3; Vec.SetB(1) = -4;

cout << Vec << endl;

Line<int> Vec2(Vec);

Circle<int> Cc(2);

Cc.SetA(0) = 0; Cc.SetA(1) = 0;

Cc.SetB(0) = 3; Cc.SetB(1) = -4;

cout << Cc << endl;

Triangle<int> Tri(2);

Tri.SetA(0) = 0; Tri.SetA(1) = 0;

Tri.SetB(0) = 0; Tri.SetB(1) = 2;

Tri.SetC(0) = 3; Tri.SetC(1) = 0;

cout << Tri << endl;

Quadrilateral<int> Sq(2);

Sq.SetA(0) = 0; Sq.SetA(1) = 0;

Sq.SetB(0) = 0; Sq.SetB(1) = 2;

Sq.SetC(0) = 2; Sq.SetC(1) = 2;

Sq.SetD(0) = 2; Sq.SetD(1) = 0;

cout << Sq << endl;

Quadrilateral<int> Sk(Sq);

Container Box;

Box.Add(Dot);

Box.Add(Sq);

Box.Add(Tri);

Box.ShowAll();

cout << endl;

Box.Show(0);

Box.Remove(1);

Box.ShowAll();

cout << endl;

Sphere<int> Sph(3);

Sph.SetA(0) = 0; Sph.SetA(1) = 0; Sph.SetA(2) = 0;

Sph.SetB(0) = 5; Sph.SetB(1) = 5; Sph.SetB(2) = 0;

cout << Sph << endl;

Ellipse<int> El(2);

El.SetA(0) = 5; El.SetA(1) = 5;

El.SetB(0) = 2; El.SetB(1) = 2;

El.SetC(0) = 0; El.SetC(1) = 0;

cout << El << endl;

return 0;

}

### Base.h

#pragma once

#include <iostream>

#define \_USE\_MATH\_DEFINES

#include <cmath>

#include <math.h>

#include <typeinfo>

using namespace std;

class Base

{

public:

virtual ~Base() {}

virtual ostream& print(ostream& os) = 0;

friend ostream& operator<<(ostream& os, Base& other)

{

return other.print(os);

}

};

### Point.h

#pragma once

#include "Base.h"

template<class T>

class Point : public Base

{

protected:

int dim;

T\* A;

public:

Point();

Point(int n);

Point(const Point& p);

virtual ~Point();

T\* GetA() const;

int GetDim() const;

T\*& SetA();

int& SetDim();

ostream& print(ostream& os) override;

};

#include "Point\_imp.h"

### Line.h

#pragma once

#include "Point.h"

template<class T>

class Line : public Point<T>

{

protected:

T\* B;

public:

Line();

Line(int n);

Line(const Line& l);

virtual ~Line();

double GetLength() const;

T\* GetB() const;

T\*& SetB();

ostream& print(ostream& os) override;

};

#include "Line\_imp.h"

### Triangle.h

#pragma once

#include "Line.h"

template<class T>

class Triangle : public Line<T>

{

protected:

T\* C;

public:

Triangle();

Triangle(int n);

Triangle(const Triangle& t);

virtual ~Triangle();

virtual double GetPerimeter() const;

virtual double GetArea() const;

T\* GetC() const;

T\*& SetC();

ostream& print(ostream& os) override;

};

#include "Triangle\_imp.h"

### Circle.h

#pragma once

#include "Line.h"

template<class T>

class Circle : public Line<T>

{

using Line<T>::GetLength;

public:

Circle() : Line<T>() {}

Circle(int n) : Line<T>(n) {}

Circle(const Circle& c) : Line<T>(c) {}

virtual ~Circle() {};

double GetRadius() const;

double GetArea() const;

ostream& print(ostream& os);

};

#include "Circle\_imp.h"

### Quadrilateral.h

#pragma once

#include "Triangle.h"

template<class T>

class Quadrilateral : public Triangle<T>

{

protected:

T\* D;

public:

Quadrilateral();

Quadrilateral(int n);

Quadrilateral(const Quadrilateral& q);

virtual ~Quadrilateral();

T\* GetD() const;

T\*& SetD();

double GetPerimeter() const;

double GetArea() const;

bool IsSquare() const;

ostream& print(ostream& os) override;

};

#include "Quadrilateral\_imp.h"

### Sphere.h

#pragma once

#include "Circle.h"

template<class T>

class Sphere : public Circle<T>

{

using Circle<T>::GetArea;

public:

Sphere() : Circle<T>() {}

Sphere(int n) : Circle<T>(n)

{

if (n < 3)

throw "Yikes";

}

Sphere(const Sphere& s) : Circle<T>(s) {}

virtual ~Sphere() {};

double GetVolume() const;

ostream& print(ostream& os);

};

#include "Sphere\_imp.h"

### Ellipse.h

#pragma once

#include "Triangle.h"

template<class T>

class Ellipse : public Triangle<T>

{

public:

Ellipse() : Triangle<T>() {}

Ellipse(int n) : Triangle<T>(n) {}

Ellipse(const Ellipse& e) : Triangle<T>(e) {}

virtual ~Ellipse() {};

double GetAxisA() const;

double GetAxisB() const;

double GetPerimeter() const;

double GetArea() const;

ostream& print(ostream& os);

};

#include "Ellipse\_imp.h"