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

Института информационных технологий, математики и механики

Отчёт по лабораторной работе

Геометрические объекты в пространстве.

Выполнил:

студент ИИТММ гр. 381903-3

Зинкин К.С

Проверил:

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 1](#_Toc26846316)0

[4. Эксперименты 11](#_Toc26846317)

[Заключение 12](#_Toc26846318)

[Литература 13](#_Toc26846319)

[Приложение 14-30](#_Toc26846320)

**Введение**

**Структура — это некоторое объединение различных переменных в том числе с различными типами данных, которому можно присвоить имя. К примеру, можно объединить данные о геометрических объектах: точка, прямая, квадрат, круг и т.д. То есть, проще говоря, мы можем объединить в единую совокупность данные, о чем угодно.**

**Постановка задачи**

Необходимо написать структуру данных для работы с геометрическими объектами в N мерном пространстве. Тип данных определяется пользователем (шаблоны).

Должны быть реализованы простейшие геометрические объекты, такие как: точка, линия, круг, квадрат, многоугольник, куб, симплекс и т.д.

Кроме самих геом. объектов должен быть реализован класс, осуществляющий обобщение действий со всеми созданными пользователями объектами - "контейнер".

Контейнер дол иметь следующие функции: добавить новый объект, удалить существующий объект, отобразить все имеющиеся объекты, отобразить выбранный объект и т.д. и т.п.

Продемонстрировать (написать в main пример) работоспособность.

Должны быть использованы и продемонстрированы: абстракция, инкапсуляция, наследование, полиморфизм, перегрузка операций, шаблоны. Иерархия должна содержать не менее 7 классов.

**Руководство пользователя**

**Использование программы пользователем:**

**1.Создайте классы необходимых объектов.**

**2.Наполните их с помощью сетторов.**

**3.Создайте контейнер и добавьте в него класс.**

**Руководство пользователя**

3.1. Описание структуры программы

Программа состоит из одного решения.

В решении GeometricShapes определено 10 модулей main.cpp, Base.h, Point.h, Line.h, Circle.h, Triangle.h, Quadrilateral.h, Sphere.h, Ellipse.h, Container.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 Line.h определен класс Line (наследник класса Point);

-В модуле Circle.h определен класс Circle (наследник класса Line);

-В модуле Triangle.h определен класс Triangle (наследник класса Line);

-В модуле Quadrilateral.h определен класс Quadrilateral (наследник класса Triangle);

-В модуле Sphere.h определен класс Sphere (наследник класса Circle);

-В модуле Ellipse.h определен класс Ellipse (наследник класса Triangle);

-В модуле Container.h определен класс 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.

**Описание алгоритмов**

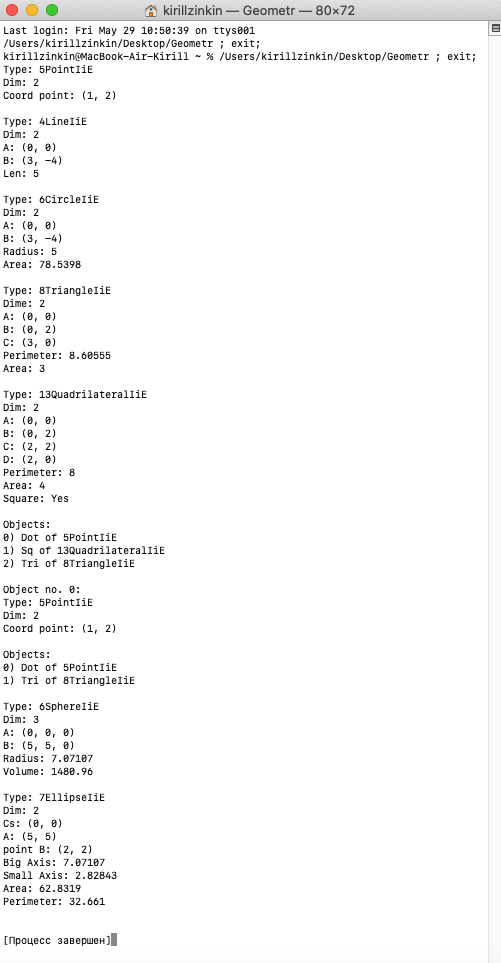
В данной программе используются простейшие алгоритмы вычисления длины

вектора, площади, периметра, объема фигур.

**Эсперимнты**

**В функции main были созданы и проверены в действии объекты, добавлены в контейнер, также и удаление объекта из контейнера вызова, вызов информации по объекту из контейнера.**

См. Рисунок:



**Заключение**

Результатом проделанной работы стала программа, позволяющая пользователям делать расчеты у различных геометрических объектов в N-мерном пространстве. И что не мало важно, у пользователя есть возможность объединить все в Container.

**Литература**

**1. Т.А.Павловская. C/C++ Программирование на языке высокого уровня изд; ПИТЕР. 2019.**

**2. Керниган Б., Ритчи Д. Язык программирования C, 2-е изд: Пер. с англ. — М.: Издательский дом «Вильямс», 2017. — 288 с.**

**Приложение**

### 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;

};

template<class T>

inline Point<T>::Point()

{

dim = 0;

A = new T[0];

}

template<class T>

inline Point<T>::Point(int n)

{

dim = n;

A = new T[n];

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

A[i] = 0;

}

template<class T>

inline Point<T>::Point(const Point& p)

{

dim = p.dim;

A = new T[p.dim];

for (int i = 0; i < p.dim; i++)

A[i] = p.A[i];

}

template<class T>

inline Point<T>::~Point()

{

delete[] A;

dim = 0;

}

template<class T>

inline T\* Point<T>::GetA() const

{

return A;

}

template<class T>

inline T\*& Point<T>::SetA()

{

return A;

}

template<class T>

inline int& Point<T>::SetDim()

{

return dim;

}

template<class T>

inline int Point<T>::GetDim() const

{

return dim;

}

template<class T>

inline ostream& Point<T>::print(ostream& os)

{

os << "Type: " << typeid(\*this).name() << endl <<

"Dim: " << GetDim() << endl <<

"Coord point: (";

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

{

if (i != GetDim() - 1)

os << GetA()[i] << ", ";

else

os << GetA()[i];

}

os << ")" << endl;

return os;

}

### 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;

};

template<class T>

inline Line<T>::Line()

{

this->SetDim() = 0;

this->SetA() = new T[0];

B = new T[0];

}

template<class T>

inline Line<T>::Line(int n)

{

this->SetDim() = n;

this->SetA() = new T[n];

B = new T[n];

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

{

this->SetA()[i] = 0;

B[i] = 0;

}

}

template<class T>

inline Line<T>::Line(const Line& l)

{

this->SetDim() = l.GetDim();

this->SetA() = new T[this->dim];

B = new T[this->dim];

for (int i = 0; i < this->dim; i++)

{

this->SetA()[i] = l.GetA()[i];

B[i] = l.B[i];

}

}

template<class T>

inline Line<T>::~Line()

{

delete[] B;

}

template<class T>

inline double Line<T>::GetLength() const

{

double sum = 0;

for (int i = 0; i < this->GetDim(); i++)

sum += pow((B[i] - this->GetA()[i]), 2);

return sqrt(sum);

}

template<class T>

inline T\* Line<T>::GetB() const

{

return B;

}

template<class T>

inline T\*& Line<T>::SetB()

{

return B;

}

template<class T>

inline ostream& Line<T>::print(ostream& os)

{

os << "Type: " << typeid(\*this).name() << endl <<

"Dim: " << this->GetDim() << endl <<

"A: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->GetA()[i] << ", ";

else

os << this->GetA()[i];

}

os << ")" << endl;

os << "B: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << GetB()[i] << ", ";

else

os << GetB()[i];

}

os << ")" << endl;

os << "Len: " << GetLength() << endl;

return os;

}

### 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);

};

template<class T>

inline double Circle<T>::GetRadius() const

{

return this->GetLength();

}

template<class T>

inline double Circle<T>::GetArea() const

{

return pow(this->GetLength(), 2) \* M\_PI;

}

template<class T>

inline ostream& Circle<T>::print(ostream& os)

{

os << "Type: " << typeid(\*this).name() << endl <<

"Dim: " << this->GetDim() << endl <<

"A: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->GetA()[i] << ", ";

else

os << this->GetA()[i];

}

os << ")" << endl;

os << "B: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->GetB()[i] << ", ";

else

os << this->GetB()[i];

}

os << ")" << endl;

os << "Radius: " << GetRadius() << endl;

os << "Area: " << GetArea() << endl;

return os;

}

### 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);

};

template<class T>

inline double Sphere<T>::GetVolume() const

{

return (4.0 / 3) \* pow(this->GetRadius(), 3) \* M\_PI;

}

template<class T>

inline ostream& Sphere<T>::print(ostream& os)

{

os << "Type: " << typeid(\*this).name() << endl <<

"Dim: " << this->GetDim() << endl <<

"A: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->GetA()[i] << ", ";

else

os << this->GetA()[i];

}

os << ")" << endl;

os << "B: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->GetB()[i] << ", ";

else

os << this->GetB()[i];

}

os << ")" << endl;

os << "Radius: " << this->GetRadius() << endl;

os << "Volume: " << GetVolume() << endl;

return os;

}

### 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;

};

template<class T>

inline Triangle<T>::Triangle()

{

this->SetDim() = 0;

this->SetA() = new T[0];

this->SetB() = new T[0];

C = new T[0];

}

template<class T>

inline Triangle<T>::Triangle(int n)

{

this->SetDim() = n;

this->SetA() = new T[n];

this->SetB() = new T[n];

C = new T[n];

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

{

this->SetA()[i] = 0;

this->SetB()[i] = 0;

C[i] = 0;

}

}

template<class T>

inline Triangle<T>::Triangle(const Triangle& t)

{

this->SetDim() = t.GetDim();

this->SetA() = new T[t.GetDim()];

this->SetB() = new T[t.GetDim()];

C = new T[t.GetDim()];

for (int i = 0; i < t.GetDim(); i++)

{

this->SetA()[i] = t.GetA()[i];

this->SetB()[i] = t.GetB()[i];

C[i] = t.C[i];

}

}

template<class T>

inline Triangle<T>::~Triangle()

{

delete[] C;

}

template<class T>

inline double Triangle<T>::GetPerimeter() const

{

double sum = 0, temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetB()[i] - this->GetA()[i]), 2);

sum += sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((C[i] - this->GetB()[i]), 2);

sum += sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetA()[i] - C[i]), 2);

sum += sqrt(temp);

temp = 0;

return sum;

}

template<class T>

inline double Triangle<T>::GetArea() const

{

double p = GetPerimeter() \* 0.5;

double temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetB()[i] - this->GetA()[i]), 2);

double a = sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((C[i] - this->GetB()[i]), 2);

double b = sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetA()[i] - C[i]), 2);

double c = sqrt(temp);

temp = 0;

return sqrt(p \* (p - a) \* (p - b) \* (p - c));

}

template<class T>

inline T\* Triangle<T>::GetC() const

{

return C;

}

template<class T>

inline T\*& Triangle<T>::SetC()

{

return C;

}

template<class T>

inline ostream& Triangle<T>::print(ostream& os)

{

os << "Type: " << typeid(\*this).name() << endl <<

"Dime: " << this->GetDim() << endl <<

"A: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->GetA()[i] << ", ";

else

os << this->GetA()[i];

}

os << ")" << endl;

os << "B: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->GetB()[i] << ", ";

else

os << this->GetB()[i];

}

os << ")" << endl;

os << "C: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << C[i] << ", ";

else

os << C[i];

}

os << ")" << endl;

os << "Perimeter: " << GetPerimeter() << endl;

os << "Area: " << GetArea() << endl;

return os;

}

### 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;

};

template<class T>

inline Quadrilateral<T>::Quadrilateral()

{

this->SetDim() = 0;

this->SetA() = new T[0];

this->SetB() = new T[0];

this->SetC() = new T[0];

D = new T[0];

}

template<class T>

inline Quadrilateral<T>::Quadrilateral(int n)

{

this->SetDim() = n;

this->SetA() = new T[n];

this->SetB() = new T[n];

this->SetC() = new T[n];

D = new T[n];

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

{

this->SetA()[i] = 0;

this->SetB()[i] = 0;

this->SetC()[i] = 0;

D[i] = 0;

}

}

template<class T>

inline Quadrilateral<T>::Quadrilateral(const Quadrilateral& q)

{

this->SetDim() = q.GetDim();

this->SetA() = new T[q.GetDim()];

this->SetB() = new T[q.GetDim()];

this->SetC() = new T[q.GetDim()];

D = new T[q.GetDim()];

for (int i = 0; i < q.GetDim(); i++)

{

this->SetA()[i] = q.GetA()[i];

this->SetB()[i] = q.GetA()[i];

this->SetC()[i] = q.GetA()[i];

D[i] = q.D[i];

}

}

template<class T>

inline Quadrilateral<T>::~Quadrilateral()

{

delete[] D;

}

template<class T>

inline T\* Quadrilateral<T>::GetD() const

{

return D;

}

template<class T>

inline T\*& Quadrilateral<T>::SetD()

{

return D;

}

template<class T>

inline double Quadrilateral<T>::GetPerimeter() const

{

double sum = 0, temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetB()[i] - this->GetA()[i]), 2);

sum += sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetC()[i] - this->GetB()[i]), 2);

sum += sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetC()[i] - D[i]), 2);

sum += sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetA()[i] - D[i]), 2);

sum += sqrt(temp);

temp = 0;

return sum;

}

template<class T>

inline double Quadrilateral<T>::GetArea() const

{

double p = GetPerimeter() \* 0.5;

double temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetB()[i] - this->GetA()[i]), 2);

double a = sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetC()[i] - this->GetB()[i]), 2);

double b = sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetC()[i] - D[i]), 2);

double c = sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetA()[i] - D[i]), 2);

double d = sqrt(temp);

temp = 0;

return sqrt((p - a) \* (p - b) \* (p - c) \* (p - d));

}

template<class T>

inline bool Quadrilateral<T>::IsSquare() const

{

double temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetA()[i] - this->GetB()[i]), 2);

double a = sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetB()[i] - this->GetC()[i]), 2);

double b = sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetC()[i] - D[i]), 2);

double c = sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetA()[i] - D[i]), 2);

double d = sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetA()[i] - this->GetC()[i]), 2);

double ac = sqrt(temp);

temp = 0;

for (int i = 0; i < this->GetDim(); i++)

temp += pow((this->GetB()[i] - D[i]), 2);

double bd = sqrt(temp);

temp = 0;

if ((a == b) & (b == c) & (c == d) & (ac == bd))

return true;

else

return false;

}

template<class T>

inline ostream& Quadrilateral<T>::print(ostream& os)

{

os << "Type: " << typeid(\*this).name() << endl <<

"Dim: " << this->GetDim() << endl <<

"A: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->GetA()[i] << ", ";

else

os << this->GetA()[i];

}

os << ")" << endl;

os << "B: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->GetB()[i] << ", ";

else

os << this->GetB()[i];

}

os << ")" << endl;

os << "C: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->C[i] << ", ";

else

os << this->C[i];

}

os << ")" << endl;

os << "D: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->D[i] << ", ";

else

os << this->D[i];

}

os << ")" << endl;

os << "Perimeter: " << GetPerimeter() << endl;

os << "Area: " << GetArea() << endl;

auto x = IsSquare() ? "Yes" : "No";

os << "Square: " << x << endl;

return os;

}

### 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);

};

template<class T>

inline double Ellipse<T>::GetAxisA() const

{

double sum = 0;

for (int i = 0; i < this->GetDim(); i++)

sum += pow((this->C[i] - this->GetA()[i]), 2);

return sqrt(sum);

}

template<class T>

inline double Ellipse<T>::GetAxisB() const

{

double sum = 0;

for (int i = 0; i < this->GetDim(); i++)

sum += pow((this->C[i] - this->GetB()[i]), 2);

return sqrt(sum);

}

template<class T>

inline double Ellipse<T>::GetPerimeter() const

{

return 4 \* (M\_PI \* GetAxisA() \* GetAxisB() + pow((GetAxisA() - GetAxisB()), 2)) /(GetAxisA() + GetAxisB());

}

template<class T>

inline double Ellipse<T>::GetArea() const

{

return M\_PI \* GetAxisA() \* GetAxisB();

}

template<class T>

inline ostream& Ellipse<T>::print(ostream& os)

{

os << "Type: " << typeid(\*this).name() << endl <<

"Dim: " << this->GetDim() << endl <<

"Cs: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->GetC()[i] << ", ";

else

os << this->GetC()[i];

}

os << ")" << endl;

os << "A: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->GetA()[i] << ", ";

else

os << this->GetA()[i];

}

os << ")" << endl;

os << "point B: (";

for (int i = 0; i < this->GetDim(); i++)

{

if (i != this->GetDim() - 1)

os << this->B[i] << ", ";

else

os << this->B[i];

}

os << ")" << endl;

os << "Big Axis: " << GetAxisA() << endl;

os << "Small Axis: " << GetAxisB() << endl;

os << "Area: " << GetArea() << endl;

os << "Perimeter: " << GetPerimeter() << endl;

return os;

}

### Container.h

#pragma once

#include <vector>

#include "Base.h"

using namespace std;

class Container

{

protected:

Base\*\* Array;

vector<string> names;

int size;

public:

Container();

Container(const Container& c);

virtual ~Container();

template <class A>

void Add(A& elem, string name);

void Remove(int r);

void ShowAll();

void Show(int i);

Base\* operator[](const int i);

};

Container::Container()

{

size = 0;

Array = 0;

names = {};

}

Container::Container(const Container& c)

{

size = c.size;

Array = c.Array;

names = c.names;

}

Container::~Container()

{

delete[] Array;

names.clear();

}

Base\* Container::operator[](const int i)

{

return Array[i];

}

template<class A>

void Container::Add(A& elem, string name)

{

Base\*\* tmp = new Base \* [size];

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

{

tmp[i] = Array[i];

}

delete[]Array;

size++;

Array = new Base \* [size];

for (int i = 0; i < size - 1; i++)

{

Array[i] = tmp[i];

}

Array[size - 1] = &elem;

delete[]tmp;

names.push\_back(name);

}

void Container::Remove(int r)

{

Base\*\* tmp = new Base \* [size];

int j = 0;

int k = 0;

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

{

if (i == r)

{

k = 1;

}

else

{

tmp[j] = Array[i];

j++;

}

}

if (k == 0)

throw - 1;

delete[] Array;

Array = new Base \* [size - 1];

size = size - 1;

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

Array[i] = tmp[i];

delete[] tmp;

names.erase(names.begin() + r);

}

void Container::ShowAll()

{

cout << "Objects: " << endl;

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

cout << i << ") " << names[i] << " of " << typeid(\*Array[i]).name() <<endl;

inline void Container::Show(int i)

{

cout << "Object no. " << i << ":" << endl << \*Array[i] << endl;

}