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

(ННГУ)

Институт информационных технологий математики и механики

Отчёт по лабораторной работе

Геометрические фигуры

Выполнил:

Студент ИТММ ПМИ гр. 381903-3

Стеценко Д. М.

Проверил:

Лебедев И. Г.

Нижний Новгород

2020 г.

**Содержание**

[1.](#_gjdgxs) Введение 3

[2.](#_30j0zll) Постановка задачи 4

[3.](#_1fob9te) Руководство пользователя 5

[4. Руководство программиста 6](#_3znysh7)

[4.1. Описание структуры программы 6](#_2et92p0)

[4.2. Описание структур данных 6](#_tyjcwt)

[4.3. Описание алгоритмов](#_3dy6vkm) 13

[5.](#_3as4poj) Эксперименты 14

[6.](#_1pxezwc) Заключение 15

[7.](#_49x2ik5) Литература 16

[8.](#_2p2csry) Приложения 17

1. **Введение**

Работа с геометрическими объектами, это важная и сложная задача. Важно иметь возможность описания и отображения объектов, обладающих геометрическими свойствами.  Поэтому необходимо продумать возможность работать напрямую с такими объектами, как точка, прямая, квадрат, круг и т. д.

1. **Постановка задачи**

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, прямоугольник, треугольник, трапеция.

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

Контейнер должен иметь следующие функции: добавить новый объект, удалить существующий объект, отобразить все имеющиеся объекты, отобразить выбранный объект.

Продемонстрировать (написать в main пример) работоспособность.

Должны быть использованы и продемонстрированы: абстракция, инкапсуляция, наследование, полиморфизм, перегрузка операций, шаблоны.

1. **Руководство пользователя**

Шаги, совершаемые пользователем:

1. Создать объекты типа Point, Line, Circle, Square, Cube, Rectangle, Triangle, в зависимости от необходимости;
2. Наполнить выбранные объекты, используя конструкторы, также существует возможность в дальнейшем изменять размеры объектов при помощи сеттеров;
3. Упаковать объекты в контейнер;
4. Произвести необходимые действия с объектами;

**4. Руководство программиста**

***4.1. Описание структуры программы***

Программа состоит из одного решения.

В решении Shapes определено 10 модулей main.cpp, Container.h, Circle.h, triangle.h, Square.h, Rectangle.h, Line.h, Point.h, Cube.h, Objects.h.

* В модуле main.cpp определена стандартная функция int main(), внутри которой содержится набор действий с геометрическими объектами.
* В модуле Container.h определен класс Container, а также объявлены все его методы и их определения.
* В модуле Objects.h определен класс Objects, а также объявлены все его методы и их определения.
* В модуле Point.h определен класс Point (наследник класса Objects), а также объявлены все его методы и их определения.
* В модуле Line.h определен класс Line (наследник класса Objects), а также объявлены все его методы и их определения.
* В модуле Square.h определен класс Square (наследник класса Objects), а также объявлены все его методы и их определения.
* В модуле Rectangle.h определен класс Rectangle (наследник класса Objects), а также объявлены все его методы и их определения.
* В модуле Cube.h определен класс Cube (наследник класса Objects), а также объявлены все его методы и их определения.
* В модуле triangle.h определен класс Triangle (наследник класса Objects), а также объявлены все его методы и их определения.
* В модуле Circle.h определен класс Circle (наследник класса Objects), а также объявлены все его методы и их определения.

***4.2. Описание структур данных***

В программе определен один абстрактный класс: Objects

Внутри класса Objects определены следующие виртуальные функции:

* virtual ~ Objects (){} – виртуальный деструктор
* virtual float Area() = 0 - виртуальный метод вычисления площади
* virtual ostream& print(ostream& os) = 0; - виртуальный метод вывода принт
* friend ostream& operator<<(ostream& os, Objects& other) – виртуальный метод вывода
* bool operator > (Objects& other) – перегруженный оператор больше, сравнивающий фигуры по площади;
* bool operator < (Objects& other) – перегруженный оператор меньше, сравнивающий фигуры по площади;

В программе определены 7 следующих шаблонных класса: Circle, Triangle, Square, Rectangle, Line, Point, Cube.

Внутри класса Point определено следующее поле:

* T mark – шаблонное значение точки;

Внутри класса Point определен следующий набор public-методов

(плюс конструкторы):

* Point () – конструктор по умолчанию, не принимает никаких параметров, инициализирует все два поля 0, через списки инициализации;
* Point (T \_mark) – конструктор инициализатор, принимает на вход шаблонный параметр и инициализирует поле mark значением переданным в конструктор соответственно;
* Point (const Point& point) – конструктор копирования, принимает на вход объект типа Point, создает объект с теми же характеристиками;
* bool operator ==(const Point& other) – перегрузка оператора = =;
* ostream& print(ostream& os) override - override функция вывода;
* T Get() – метод возвращающий значение точки;
* float Area() override – override функция подсчета площади;

Внутри класса Line определены следующие поля:

* int length – длина отрезка;
* T\* Array – шаблонный динамический массив значений отрезка;

Внутри класса Line определен следующий набор public-методов

(плюс конструкторы и деструктор):

* Line () – конструктор по умолчанию, не принимает никаких параметров, инициализирует поле length 10, а массив заполняет символом \*;
* Line (int len, T \_mark) – конструктор инциализатор, принимает на вход параметр тип int и один шаблонный параметр, инициализирует поле length, и заполняет массив соответственно;
* Line (const Line& line) – конструктор копирования, принимает на вход объект типа Line, создает объект с теми же характеристиками;
* ~Line () – деструктор;
* int GetLen () – метод возвращающий длину отрезка;
* T GetMark () – метод возвращающий значение массива;
* bool operator ==(const Line& other) – перегрузка оператора ==;
* ostream& print(ostream& os) override – override функция вывода;
* float Area () override - метод возвращающий площадь отрезка;

Внутри класса Circle определены следующие поля:

* int radius – радиус круга;
* T\*\* Array – шаблонный двойной динамический массив значений круга;

Внутри класса Circle определен следующий набор public-методов

(плюс конструкторы и деструктор):

* Circle () – конструктор по умолчанию, не принимает никаких параметров, инициализирует поле radius двойкой, а массив заполняет символом \*;
* Circle (int rad, T \_mark) – конструктор инциализатор, принимает на вход параметр типа int и шаблонный параметр, инициализирует поле radius, и заполняет массив соответственно;
* Circle (const Circle& circle) – конструктор копирования, принимает на вход объект типа Circle, создает объект с теми же характеристиками;
* ~Circle () – деструктор;
* int GetRadius() – метод возвращающий радиус круга;
* T GetMark () – метод возвращающий значение массива;
* void SetRadius(int \_radius) – метод изменяющий радиус круга;
* bool operator ==(const Circle& other) – перегрузка оператора ==;
* ostream& print(ostream& os) override – override функция вывода;
* float Area () override - метод возвращающий площадь круга;

Внутри класса Square определены следующие поля:

* int side – сторона квадрата;
* T\*\* Array – шаблонный двойной динамический массив значений квадрата;

Внутри класса Square определен следующий набор public-методов

(плюс конструкторы и деструктор):

* Square () – конструктор по умолчанию, не принимает никаких параметров, инициализирует поле side единицей, а массив заполняет символом \*;
* Square (int a, T \_mark) – конструктор инициализатор, принимает на вход параметр типа int и шаблонный параметр, инициализирует поле side, и заполняет массив соответственно;
* Square (const Square & square) – конструктор копирования, принимает на вход объект типа Square, создает объект с теми же характеристиками;
* ~Square () – деструктор;
* int GetSide() – метод возвращающий длину стороны квадрата;
* T GetMark () – метод возвращающий значение массива;
* void SetSide(int \_side) – метод изменяющий длину стороны квадрата;
* bool operator ==(const Square& other) – перегрузка оператора ==;
* ostream& print(ostream& os) override – override функция вывода;
* float Area () override - метод возвращающий площадь квадрата;

Внутри класса Triangle определены следующие поля:

* int height – высота прямоугольного равнобедренного треугольника +1;
* T\*\* Array – шаблонный двойной динамический массив значений треугольника;

Внутри класса Triangle определен следующий набор public-методов

(плюс конструкторы и деструктор):

* Triangle () – конструктор по умолчанию, не принимает никаких параметров, инициализирует поле height тройкой, а массив заполняет символом \*;
* Triangle (int \_height, T \_mark) – конструктор инициализатор, принимает на вход параметр типа int и шаблонный параметр, инициализирует поле height, и заполняет массив соответственно;
* Triangle (const Triangle & square) – конструктор копирования, принимает на вход объект типа Triangle, создает объект с теми же характеристиками;
* ~ Triangle () – деструктор;
* int GetHeight () – метод возвращающий длину высоты треугольника;
* T GetMark () – метод возвращающий значение массива;
* void SetHeight(int \_height) – метод изменяющий длину высоты треугольника;
* bool operator ==(const Triangle & other) – перегрузка оператора ==;
* ostream& print(ostream& os) override – override функция вывода;
* float Area () override - метод возвращающий площадь треугольника;

Внутри класса Rectangle определены следующие поля:

* int side1 – первая сторона прямоугольника;
* int side1 – вторая сторона прямоугольника, отличная от предыдущей;
* T\*\* Array – шаблонный двойной динамический массив значений прямоугольника;

Внутри класса Rectangle определен следующий набор public-методов

(плюс конструкторы и деструктор):

* Rectangle () – конструктор по умолчанию, не принимает никаких параметров, инициализирует поле side1 и side2 единицей, а массив заполняет символом \*;
* Rectangle (int \_side1, int \_side2, T \_mark) – конструктор инциализатор, принимает на вход два параметра типа int и шаблонный параметр, инициализирует поля side1 и side2, и заполняет массив соответственно;
* Rectangle (const Rectangle & rectangle) – конструктор копирования, принимает на вход объект типа Rectangle, создает объект с теми же характеристиками;
* ~ Rectangle () – деструктор;
* int GetSide1() – метод возвращающий длину первой стороны прямоугольника;
* int GetSide1() – метод возвращающий длину второй стороны прямоугольника;
* T GetMark () – метод возвращающий значение массива;
* void SetSide(int \_side1, int \_side2) – метод изменяющий длины сторон прямоугольника;
* bool operator ==(const Rectangle & other) – перегрузка оператора ==;
* ostream& print(ostream& os) override – override функция вывода;
* float Area () override - метод возвращающий площадь прямоугольника;

Внутри класса Cube определены следующие поля:

* int side – ребро куба;
* T\*\*\* Array – шаблонный тройной динамический массив значений куба;

Внутри класса Cube определен следующий набор public-методов

(плюс конструкторы и деструктор):

* Cube () – конструктор по умолчанию, не принимает никаких параметров, инициализирует поле side единицей, а массив заполняет символом \*;
* Cube (int a, T \_mark) – конструктор инциализатор, принимает на вход параметр типа int и шаблонный параметр, инициализирует поле side, и заполняет массив соответственно;
* Cube (const Cube & cube) – конструктор копирования, принимает на вход объект типа Cube, создает объект с теми же характеристиками;
* ~ Cube () – деструктор;
* int GetSide() – метод возвращающий длину ребра куба;
* T GetMark () – метод возвращающий значение массива;
* void SetSide(int \_side) – метод изменяющий длину ребра куба;
* bool operator ==(const Cube & other) – перегрузка оператора ==;
* ostream& print(ostream& os) override – override функция вывода (возвращает проекцию куба на плоскость);
* float Area () override - метод возвращающий площадь поверхности куба;

В программе определен один класс-контейнер: Container

Внутри класса Container, определены следующие поля:

* Objects \*\* Array;
* int size;

Внутри класса Container определен следующий набор public-методов

(плюс конструкторы и деструктор):

* Container () – конструктор по умолчанию, не принимает никаких параметров;
* void Add(A& elem) – принимает на вход шаблонный аргумент, ничего не возвращает, записывает этот объект в Container;
* void Delete(A& elem) – принимает на вход шаблонный аргумент, ищет его в контейнере, и удаляет его;
* ~Container () – деструктор;
* Objects\* operator[](const int i) – перегрузка оператора возврата объекта по индексу;
* friend ostream& operator<<(ostream& out, Container& other) – перегрузка оператора вывода для класса Container;
* void Out(A& elem) - принимает на вход шаблонный аргумент и выводит его на экран.

***4.3. Описание алгоритмов***

Как таковых, каких-то серьезных алгоритмов в нашем проекте не использовалось.

1. **Эксперименты**

В функции Main приведен пример работы с классами геометрических фигур, и содержащим их контейнером.



Рисунок 1: Результаты выполнения функции Main.

1. **Заключение**

По итогу проведения данной работы была реализована программа, позволяющая напрямую работать с такими геометрическими объектами как точка, отрезок, квадрат, круг, куб, прямоугольный равнобедренный треугольник, прямоугольник. Используя принципы: абстракцию, инкапсуляцию, наследование и полиморфизм были созданы удобные объекты для работы с геометрическими фигурами и содержащим их контейнером.

1. **Литература**
2. Бьерн Страуструп. Язык программирования C++, 2-е изд: Пер. с англ. — М.: Издательский дом «Бином», 2017. — 1136 с.
3. Герберт Шилдт. C++ для начинающих. Шаг за шагом, 2-е изд: Пер. с англ – М.: Издательский дом «ЭКОМ Паблишерз», 2013. – 640 с.
4. Липпман Стенли Б., Лажойе Жози, Му Барбара Э. Язык программирования C++, базовый курс, 5-е изд: Пер. с англ. – М.: Издательский дом «Диалектика», 2018. – 1118 с.

1. **Приложения**

**“rectangle.h”**

#pragma once

#include"Objects.h"

#include <iostream>

using namespace std;

template <typename T>

class Rectangle : public Objects{

private:

int side1;

int side2;

T\*\* arr;

public:

Rectangle();

Rectangle(int \_side1, int \_side2, T \_mark);

Rectangle(const Rectangle& rectangle);

~Rectangle();

int GetSide1();

int GetSide2();

T GetMark();

void SetSide(int \_side1, int \_side2);

float A();

ostream& print(ostream& os);

bool operator ==(const Rectangle& other);

};

Rectangle<char>::Rectangle(){

side1 = 1;

side2 = 1;

arr = new char\* [1];

arr[0] = new char[1];

arr[0][0] = '\*';

}

template<typename T>

Rectangle<T>::Rectangle(int \_side1, int \_side2, T \_mark){

side1 = \_side1;

side2 = \_side2;

arr = new T \* [side1];

for (int i = 0; i < side1; i++)

arr[i] = new T[side2];

for (int i = 0; i < side1; i++)

for (int j = 0; j < side2; j++)

arr[i][j] = \_mark;

}

template<typename T>

Rectangle<T>::Rectangle(const Rectangle& rectangle){

if (arr != 0) {

for (int i = 0; i < side1; i++)

delete[] arr[i];

delete[] arr;

}

side1 = rectangle.side1;

side2 = rectangle.side2;

arr = new T \* [side1];

for (int i = 0; i < side1; i++)

arr[i] = new T[side2];

for (int i = 0; i < side1; i++)

for (int j = 0; j < side2; j++)

arr[i][j] = rectangle.arr[i][j];

}

template<typename T>

Rectangle<T>::~Rectangle(){

if (arr != 0) {

for (int i = 0; i < side1; i++)

delete[] arr[i];

delete[] arr;

arr = 0;

side1 = 0;

side2 = 0;

}

}

template<typename T>

int Rectangle<T>::GetSide1(){

return side1;

}

template<typename T>

int Rectangle<T>::GetSide2(){

return side2;

}

template<typename T>

T Rectangle<T>::GetMark(){

return arr[0][0];

}

template<typename T>

void Rectangle<T>::SetSide(int \_side1, int \_side2){

T tmp = arr[0][0];

if (arr != 0){

for (int i = 0; i < side1; i++)

delete[] arr[i];

delete[] arr;

}

side1 = \_side1;

side2 = \_side2;

arr = new T \* [side1];

for (int i = 0; i < side1; i++)

arr[i] = new T[side2];

for (int i = 0; i < side1; i++)

for (int j = 0; j < side2; j++)

arr[i][j] = tmp;

}

template<typename T>

float Rectangle<T>::A(){

return side1\*side2;

}

template<typename T>

ostream& Rectangle<T>::print(ostream& os){

for (int i = 0; i < side1; i++){

for (int j = 0; j < side2; j++)

os << arr[i][j];

os << endl;

}

return os;

}

template<typename T>

bool Rectangle<T>::operator==(const Rectangle& other){

if (side1 == other.side1 && side2 == other.side2 && arr[0][0] == other.arr[0][0])

return true;

return false;

}

**“main.cpp”**

#include <iostream>

#include "Container.h"

using namespace std;

int main(){

Container C;

Point<char> point;

Line<char> line(10,'\*');

Square<int> square(3, 1);

Rectangle<int> rectangle(3, 6, 1);

Triangle<char> triangle(7, '\*');

Circle<char> circle(5,'\*');

Cube<char> cube(3, '\*');

C.Add(point);

C.Add(line);

C.Add(square);

C.Add(rectangle);

C.Add(triangle);

C.Add(circle);

C.Add(cube);

cout << "point:" << endl;

C.Out(point);

cout << endl;

cout << "line:" << endl;

C.Out(line);

cout << endl;

cout << "square:" << endl;

C.Out(square);

cout << endl;

cout << "rectangle:" << endl;

C.Out(rectangle);

cout << endl;

cout << "triangle:" << endl;

C.Out(triangle);

cout << endl;

cout << "circle:" << endl;

C.Out(circle);

cout << endl;

cout << "cube:" << endl;

C.Out(cube);

return 0;

}

**“container.h”**

#pragma once

#include"Circle.h"

#include"triangle.h"

#include"Square.h"

#include"Rectangle.h"

#include"Line.h"

#include"Point.h"

#include"Cube.h"

#include"Objects.h"

class Container{

private:

Objects\*\* arr;

int size;

public:

Container();

~Container();

template <typename T>

void Add(T& elem);

template <typename T>

void Delete(T& elem);

template <typename T>

void Out(T& elem);

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

friend ostream& operator<<(ostream& out, Container& other);

};

Container::Container(){

size = 0;

arr = 0;

}

Container::~Container(){

delete[] arr;

}

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

return arr[i];

}

template<typename T>

void Container::Add(T& elem){

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

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

tmp[i] = arr[i];

}

delete[]arr;

size++;

arr = new Objects \*[size];

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

arr[i] = tmp[i];

}

arr[size - 1] = &elem;

delete[]tmp;

}

template<typename T>

void Container::Delete(T& elem){

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

int j = 0;

int k = 0;

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

if (arr[i] == &elem){

k = 1;

}

else{

tmp[j] = arr[i];

j++;

}

}

if (k == 0)

throw - 1;

delete[] arr;

arr = new Objects \*[size - 1];

size = size - 1;

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

arr[i] = tmp[i];

delete tmp;

}

template<typename T>

void Container::Out(T& elem){

cout << elem;

}

ostream& operator<<(ostream& out, Container& other){

for (int i = 0; i < other.size; i++){

out << \*(other[i]) << endl;

}

return out;

}

**“objects.h”**

#pragma once

#include <iostream>

using namespace std;

class Objects{

public:

virtual ~Objects() {}

virtual float A() = 0;

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

friend ostream& operator<<(ostream& os, Objects& other){

return other.print(os);

}

bool operator > (Objects& other){

if (this->A() > other.A())

return true;

return false;

}

bool operator < (Objects& other){

if (this->A() < other.A())

return true;

return false;

}

};

**“triangle.h”**

#pragma once

#include"Objects.h"

#include <iostream>

using namespace std;

template <typename T>

class Triangle : public Objects{

private:

int height;

T\*\* arr;

public:

Triangle();

Triangle(int \_height, T \_mark);

Triangle(const Triangle& triangle);

~Triangle();

int GetHeight();

T GetMark();

void SetHeight(int \_height);

float A();

ostream& print(ostream& os);

bool operator ==(const Triangle& other);

};

Triangle<char>::Triangle(){

height = 3;

arr = new char \* [height];

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

arr[i] = new char[height];

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

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

arr[i][j] = '\*';

}

template<typename T>

Triangle<T>::Triangle(int \_height, T \_mark){

height = \_height+1;

arr = new T \* [height];

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

arr[i] = new T[height];

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

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

arr[i][j] = \_mark;

}

template<typename T>

Triangle<T>::Triangle(const Triangle& triangle){

if (arr != 0){

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

delete[] arr[i];

delete[] arr;

}

height = triangle.height;

arr = new T \* [height];

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

arr[i] = new T[height];

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

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

arr[i][j] = triangle.arr[i][j];

}

template<typename T>

Triangle<T>::~Triangle(){

if (arr != 0){

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

delete[] arr[i];

delete[] arr;

arr = 0;

height = 0;

}

}

template<typename T>

int Triangle<T>::GetHeight(){

return height-1;

}

template<typename T>

T Triangle<T>::GetMark(){

return T();

}

template<typename T>

void Triangle<T>::SetHeight(int \_height){

T tmp = this->GetMark();

if (arr != 0) {

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

delete[] arr[i];

delete[] arr;

}

height = \_height+1;

arr = new T \* [height];

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

arr[i] = new T[height];

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

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

arr[i][j] = tmp;

}

template<typename T>

float Triangle<T>::A(){

return ((height-1) \* (height-1)) / 2;

}

template<typename T>

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

for (int i = 0; i < height; i++){

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

os << arr[i][j];

os << endl;

}

return os;

}

template<typename T>

bool Triangle<T>::operator==(const Triangle& other){

if (height == other.height && arr[0][0] == other.arr[0][0])

return true;

return false;

}

“point.h”

#pragma once

#include"Objects.h"

#include <iostream>

using namespace std;

template <typename T>

class Point : public Objects{

private:

T mark;

public:

Point();

Point(T \_mark);

Point(const Point& point);

T Get();

float A() ;

ostream& print(ostream& os) ;

bool operator ==(const Point& other);

};

template<typename T>

Point<T>::Point(){

mark = '.';

}

template<typename T>

Point<T>::Point(T \_mark){

mark = \_mark;

}

template<typename T>

Point<T>::Point(const Point& point){

mark = point.mark;

}

template<typename T>

T Point<T>::Get(){

return mark;

}

template<typename T>

float Point<T>::A(){

return 0;

}

template<typename T>

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

os << this->Get();

os << endl;

return os;

}

template<typename T>

bool Point<T>::operator==(const Point& other){

if (mark==other.mark)

return true;

return false;

}

“square.h”

#pragma once

#include"Objects.h"

#include <iostream>

using namespace std;

template <typename T>

class Square : public Objects{

private:

int side;

T\*\* Array;

public:

Square();

Square(int a, T \_mark);

Square(const Square& square);

~Square();

int GetSide();

T GetMark();

void SetSide(int \_side);

float A();

ostream& print(ostream& os);

bool operator ==(const Square& other);

};

Square<char>::Square(){

side = 1;

Array = new char \* [1];

Array [0] = new char[1];

Array[0][0] = '\*';

}

template<typename T>

Square<T>::Square(int a, T \_mark){

side = a;

Array = new T \* [side];

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

Array[i] = new T[side];

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

for (int j = 0; j < side; j++)

Array[i][j] = \_mark;

}

template<typename T>

Square<T>::Square(const Square& square){

if (Array != 0) {

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

delete[] Array[i];

delete[] Array;

}

side = square.side;

Array = new T \* [side];

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

Array[i] = new T[side];

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

for (int j = 0; j < side; j++)

Array[i][j] = square.Array[i][j];

}

template<typename T>

Square<T>::~Square(){

if (Array != 0){

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

delete[] Array[i];

delete[] Array;

Array = 0;

side = 0;

}

}

template<typename T>

int Square<T>::GetSide(){

return side;

}

template<typename T>

T Square<T>::GetMark(){

return Array[0][0];

}

template<typename T>

void Square<T>::SetSide(int \_side){

T tmp = Array[0][0];

if (Array != 0){

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

delete[] Array[i];

delete[] Array;

}

side = \_side;

Array = new T \* [side];

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

Array[i] = new T[side];

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

for (int j = 0; j < side; j++)

Array[i][j] = tmp;

}

template<typename T>

float Square<T>::A(){

return side \* side;

}

template<typename T>

ostream& Square<T>::print(ostream& os){

for (int i = 0; i < side; i++){

for (int j = 0; j < side; j++)

os << Array[i][j];

os << endl;

}

return os;

}

template<typename T>

bool Square<T>::operator==(const Square& other){

if (side == other.side && Array[0][0]==other.Array[0][0])

return true;

return false;

}

“line.h”

#pragma once

#include"Objects.h"

#include <iostream>

using namespace std;

template <typename T>

class Line: public Objects{

private:

int length;

T\* Array;

public:

Line();

Line(int len, T \_mark);

Line(const Line& line);

~Line();

int GetLen();

T GetMark();

void SetLen(int len);

float A() ;

ostream& print(ostream& os);

bool operator ==(const Line& other);

};

template<typename T>

Line<T>::Line(){

length = 10;

Array = new char [length];

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

Array[i] = "\*";

}

template<typename T>

Line<T>::Line(int len, T \_mark){

length = len;

Array = new T[length];

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

Array[i] = \_mark;

}

template<typename T>

Line<T>::Line(const Line& line){

if (Array != 0)

delete[] Array;

length = line.length;

Array = new T[length];

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

Array[i] = line.Array[i];

}

template<typename T>

Line<T>::~Line(){

if (Array != 0){

delete[] Array;

Array = 0;

length = 0;

}

}

template<typename T>

int Line<T>::GetLen(){

return length;

}

template<typename T>

T Line<T>::GetMark(){

return Array[0];

}

template<typename T>

void Line<T>::SetLen(int len){

T tmp = Array[0];

if (Array != 0)

delete[] Array;

length = len;

Array = new T[length];

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

Array[i] = tmp;

}

template<typename T>

float Line<T>::A(){

return 0;

}

template<typename T>

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

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

os << Array[i];

os << endl;

return os;

}

template<typename T>

bool Line<T>::operator==(const Line& other){

if (length == other.length && Array[0] == other.Array[0])

return true;

return false;

}

“cube.h”

#pragma once

#include"Objects.h"

#include <iostream>

using namespace std;

template <typename T>

class Cube : public Objects{

private:

int side;

T\*\*\* arr;

public:

Cube();

Cube(int a, T \_mark);

Cube(const Cube& cube);

~Cube();

int GetSide();

T GetMark();

void SetSide(int \_side);

float A();

ostream& print(ostream& os);

bool operator ==(const Cube& other);

};

template<typename T>

Cube<T>::Cube(){

side = 1;

arr = new T \* \*[side];

for (int i = 0; i < side; i++) {

arr[i] = new T \* [side];

for (int j = 0; j < side; j++)

arr[i][j] = new T[side];

}

arr[0][0][0] = '\*';

}

template<typename T>

Cube<T>::Cube(int a, T \_mark){

side = a;

arr = new T \* \*[side];

for (int i = 0; i < side; i++) {

arr[i] = new T \* [side];

for (int j = 0; j < side; j++)

arr[i][j] = new T[side];

}

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

for (int j = 0; j < side; j++)

for (int l = 0; l < side; l++)

arr[i][j][l] = \_mark;

}

template<typename T>

Cube<T>::Cube(const Cube& cube){

if (arr != 0){

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

for (int j = 0; j < side; j++)

delete[] arr[i][j];

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

delete[] arr[i];

delete[] arr;

}

side = cube.side;

arr = new T \* \*[side];

for (int i = 0; i < side; i++) {

arr[i] = new T \* [side];

for (int j = 0; j < side; j++)

arr[i][j] = new T[side];

}

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

for (int j = 0; j < side; j++)

for (int l = 0; l < side; l++)

arr[i][j][l] = cube.arr[i][j][l];

}

template<typename T>

Cube<T>::~Cube(){

if (arr != 0){

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

for (int j = 0; j < side; j++)

delete[] arr[i][j];

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

delete[] arr[i];

delete[] arr;

arr = 0;

side = 0;

}

}

template<typename T>

int Cube<T>::GetSide(){

return side;

}

template<typename T>

T Cube<T>::GetMark(){

return arr[0][0][0];

}

template<typename T>

void Cube<T>::SetSide(int \_side){

T tmp = arr[0][0][0];

if (arr != 0) {

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

for (int j = 0; j < side; j++)

delete[] arr[i][j];

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

delete[] arr[i];

delete[] arr;

}

side = \_side;

arr = new T \* \*[side];

for (int i = 0; i < side; i++) {

arr[i] = new T \* [side];

for (int j = 0; j < side; j++)

arr[i][j] = new T[side];

}

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

for (int j = 0; j < side; j++)

for (int l = 0; l < side; l++)

arr[i][j][l] = tmp;

}

template<typename T>

float Cube<T>::A(){

return side\*side\*6;

}

template<typename T>

ostream& Cube<T>::print(ostream& os){

Square<T> tmp(side, arr[0][0][0]);

tmp.print(os);

return os;

}

template<typename T>

bool Cube<T>::operator==(const Cube& other){

if (side == other.side && arr[0][0][0] == other.arr[0][0][0])

return true;

return false;

}

“circle.h”

#pragma once

#include"Objects.h"

#include <iostream>

using namespace std;

template <typename T>

class Circle : public Objects{

private:

int radius;

T\*\* arr;

public:

Circle();

Circle(int r, T \_mark);

Circle(const Circle& circle);

~Circle();

int GetRadius();

T GetMark();

void SetRadius(int \_radius);

float A() ;

ostream& print(ostream& os);

bool operator ==(const Circle& other);

};

Circle<char>::Circle(){

radius = 0;

arr = new char \* [radius];

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

arr[i] = new char[radius];

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

for (int j = radius - i; j < radius + i + 1; j++)

arr[i][j] = '\*';

for (int i = radius; i < 2 \* radius; i++)

for (int j = 1 + i - radius; j < 3 \* radius - i; j++)

arr[i][j] = '\*';

}

template<typename T>

Circle<T>::Circle(int r, T \_mark){

radius = r;

arr = new T\* [2 \* radius - 1];

for (int i = 0; i < 2 \* radius - 1; i++)

arr[i] = new T[2 \* radius];

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

for (int j = radius - 1 - i; j < radius + i + 1; j++)

arr[i][j] = \_mark;

for (int i = radius; i < 2 \* radius - 1; i++)

for (int j = 1 + i - radius; j < 3 \* radius - 1 - i; j++)

arr[i][j] = \_mark;

}

template<typename T>

Circle<T>::Circle(const Circle& circle){

if (arr != 0) {

for (int i = 0; i < 2 \* radius; i++)

delete[] arr[i];

delete[] arr;

}

radius = circle.radius;

arr = new T \* [2 \* radius];

for (int i = 0; i < 2 \* radius; i++)

arr[i] = new T[2 \* radius];

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

for (int j = radius; j < radius + i + 1; j++)

arr[i][j] = circle.arr[i][j];

for (int i = radius; i < 2 \* radius ; i++)

for (int j = 1 + i - radius; j < 3 \* radius; j++)

arr[i][j] = circle.arr[i][j];

}

template<typename T>

Circle<T>::~Circle(){

if (arr != 0) {

for (int i = 0; i < 2 \* radius ; i++)

delete[] arr[i];

delete[] arr;

arr = 0;

radius = 0;

}

}

template<typename T>

int Circle<T>::GetRadius(){

return radius;

}

template<typename T>

T Circle<T>::GetMark(){

return arr[radius][radius];

}

template<typename T>

void Circle<T>::SetRadius(int \_radius){

T tmp = this->GetMark();

if (arr != 0){

for (int i = 0; i < 2 \* radius; i++)

delete[] arr[i];

delete[] arr;

}

radius = \_radius;

arr = new T \* [2 \* radius];

for (int i = 0; i < 2 \* radius; i++)

arr[i] = new T[2 \* radius];

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

for (int j = radius - i; j < radius + i + 1; j++)

arr[i][j] = tmp;

for (int i = radius; i < 2 \* radius ; i++)

for (int j = 1 + i - radius; j < 3 \* radius - i; j++)

arr[i][j] = tmp;

}

template<typename T>

float Circle<T>::A(){

return radius\*radius\*3.14;

}

template<typename T>

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

for (int i = 0; i < 2 \* radius; i++){

for (int j = 0; j < 2 \* radius; j++){

if ((i < radius && j >=(radius - 1 - i) && j < (radius + i + 1)) || (i >= radius && i < (2 \* radius - 1) && j >=(1 + i - radius) && j < (3 \* radius - 1 - i)))

os << arr[i][j];

else os << " ";

}

os << endl;

}

return os;

}

template<typename T>

bool Circle<T>::operator==(const Circle& other){

if (radius == other.radius && arr[radius][radius] == other.arr[radius][radius])

return true;

return false;

}