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 объекты

Выполнил:

Студент ИИТММ

Гр. 381906-2

Челебаев Артём

Проверил:

Лебедев И. Г.

Нижний Новгород

2020

**Содержание**

[Введение](#_Введение)

1. [Постановка задачи](#_Постановка_задачи)
2. [Руководство пользователя](#_Руководство_пользователя)
3. [Руководство программиста](#_Руководство_программиста)
   1. [Описание структуры программы](#_Описание_структуры_программы)
   2. [Описание структур данных](#_Описание_структур_данных_1)
   3. [Описание алгоритмов](#_Описание_алгоритмов)
4. [Эксперименты](#_Эксперименты_1)
5. [Заключение](#_Заключение)
6. [Литература](#_Литература_1)
7. [Приложение](#_Приложения)

# **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 на прямую с такими объектами, как точка, прямая, квадрат, круг и т. д.

# **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.

# **Руководство пользователя**

Шаги, совершаемые пользователем:

1. Создаются объекты типа Point, Line, Circle, Square, Cube, Rectangle, Triangle в зависимости от необходимости;
2. Выбранные объекты определяются при помощи конструкторов, помимо этого, в дальнейшем можно изменять размеры объектов с помощью set-методов;
3. Объекты помещаются в контейнер;
4. Производятся необходимые действия с объектами;

# **Руководство программиста**

# **Описание структуры программы**

В проекте PAINt существует 10 файлов: main.cpp, Container.h, Circle.h, Triangle.h, Square.h, Rectangle.h, Line.h, Point.h, Cube.h, Objects.h.

* В модуле main.cpp содержится стандартная функция int main(), внутри которой определён набор действий с геометрическими объектами.
* В модуле Container.h создан класс Container, а также объявлены все его методы и их определения.
* В модуле Objects.h создан класс Objects, а также объявлены все его методы и их определения.
* В модуле Point.h создан класс Point (наследник класса Objects), а также объявлены все его методы и их определения, аналогично с остальными файлами объектов.

# **Описание структур данных**

В программе определен один абстрактный(виртуальный) 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 print
* 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 инциализатор, 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 – длинна 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 инциализатор, 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 инциализатор, 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 есть 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):

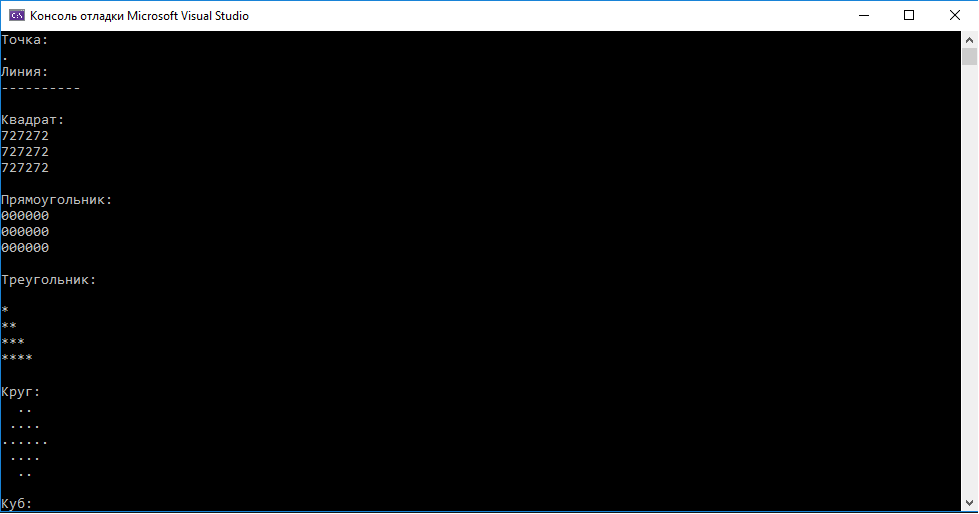
* Container () – конструктор по умолчанию, не принимает никаких параметров;
* void Add(A& elem) – принимает на вход шаблонный аргумент, ничего не возвращает, записывает этот объект в Container;
* void Delete(A& elem) – принимает на вход шаблонный аргумент, ищет его в контейнере, и удаляет его;
* ~Container () – 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.

# **Описание алгоритмов**

Никаких особых алгоритмов в проекте не содержится и не используется.

# 

# **Эксперименты**

Запустим программу:

# **Заключение**

В данной программе получилось реализовать класс-контейнер, содержащий в себе различные геометрические фигуры.

# 

# **Литература**

1. Павловская Т.А. C/C++, Программирование на языке высокого уровня, 2003.
2. Вирт Н. Алгоритмы + структуры данных = программы. — М.: Мир, 1985. — С. 406.
3. Официальный сайт программы PMBK. – Режим доступа <http://www.pmbk.ru/pr/chto-takoe-massivy-v-programmirovanii.html>
4. Официальный сайт Bestreferat. – Режим доступа <https://www.bestreferat.ru/referat-183091.html>
5. Официальный сайт Habr. – Режим доступа <https://habr.com/ru/post/339656/>
6. Официальный сайт Microsoft. – Режим доступа <https://docs.microsoft.com/ru-ru/cpp/c-language/main-function-and-program-execution?view=vs-2019>

# **Приложения**

**Circle.h**

#pragma once

#include"Objects.h"

#include <iostream>

using namespace std;

template <class T>

class Circle : public Objects

{

private:

int radius;

T\*\* Array;

public:

Circle();

Circle(int rad, T \_mark);

Circle(const Circle& circle);

~Circle();

int GetRadius();

T GetMark();

void SetRadius(int \_radius);

float Area() override;

ostream& print(ostream& os) override;

bool operator ==(const Circle& other);

};

inline Circle<char>::Circle()

{

radius = 2;

Array = new char\* [2 \* radius - 1];

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

Array[i] = new char[2 \* radius];

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

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

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

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

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

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

}

template<class T>

inline Circle<T>::Circle(int rad, T \_mark)

{

radius = rad;

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

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

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

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

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

Array[i][j] = \_mark;

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

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

Array[i][j] = \_mark;

}

template<class T>

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

{

if (Array != 0)

{

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

delete[] Array[i];

delete[] Array;

}

radius = circle.radius;

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

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

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

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

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

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

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

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

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

}

template<class T>

inline Circle<T>::~Circle()

{

if (Array != 0)

{

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

delete[] Array[i];

delete[] Array;

Array = 0;

radius = 0;

}

}

template<class T>

inline int Circle<T>::GetRadius()

{

return radius;

}

template<class T>

inline T Circle<T>::GetMark()

{

return Array[radius][radius];

}

template<class T>

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

{

T tmp = this->GetMark();

if (Array != 0)

{

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

delete[] Array[i];

delete[] Array;

}

radius = \_radius;

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

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

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

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

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

Array[i][j] = tmp;

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

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

Array[i][j] = tmp;

}

template<class T>

inline float Circle<T>::Area()

{

return radius \* radius \* 3.14;

}

template<class T>

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

{

for (int i = 0; i < 2 \* radius - 1; 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 << Array[i][j];

else os << " ";

}

os << endl;

}

return os;

}

template<class T>

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

{

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

return true;

return false;

}

**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\*\* Array;

int size;

public:

Container();

~Container();

template <class A>

void Add(A& elem);

template <class A>

void Delete(A& elem);

template <class A>

void Show(A& elem);

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

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

};

Container::Container()

{

size = 0;

Array = 0;

}

Container::~Container()

{

delete[] Array;

}

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

{

return Array[i];

}

template<class A>

void Container::Add(A& elem)

{

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

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

{

tmp[i] = Array[i];

}

delete[]Array;

size++;

Array = new Objects \* [size];

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

{

Array[i] = tmp[i];

}

Array[size - 1] = &elem;

delete[]tmp;

}

template<class A>

void Container::Delete(A& elem)

{

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

int j = 0;

int k = 0;

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

{

if (Array[i] == &elem)

{

k = 1;

}

else

{

tmp[j] = Array[i];

j++;

}

}

if (k == 0)

throw - 1;

delete[] Array;

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

size = size - 1;

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

Array[i] = tmp[i];

delete tmp;

}

template<class A>

void Container::Show(A& elem)

{

cout << elem;

}

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

{

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

{

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

}

return out;

}

**Cube.h**

#pragma once

#include"Objects.h"

#include"Square.h"

#include <iostream>

using namespace std;

template <class T>

class Cube : public Objects

{

private:

int side;

T\*\*\* Array;

public:

Cube();

Cube(int a, T \_mark);

Cube(const Cube& cube);

~Cube();

int GetSide();

T GetMark();

void SetSide(int \_side);

float Area() override;

ostream& print(ostream& os) override;

bool operator ==(const Cube& other);

};

template<class T>

inline Cube<T>::Cube()

{

side = 1;

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

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

{

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

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

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

}

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

}

template<class T>

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

{

side = a;

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

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

{

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

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

Array[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++)

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

}

template<class T>

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

{

if (Array != 0)

{

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

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

delete[] Array[i][j];

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

delete[] Array[i];

delete[] Array;

}

side = cube.side;

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

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

{

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

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

Array[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++)

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

}

template<class T>

inline Cube<T>::~Cube()

{

if (Array != 0)

{

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

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

delete[] Array[i][j];

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

delete[] Array[i];

delete[] Array;

Array = 0;

side = 0;

}

}

template<class T>

inline int Cube<T>::GetSide()

{

return side;

}

template<class T>

inline T Cube<T>::GetMark()

{

return Array[0][0][0];

}

template<class T>

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

{

T tmp = Array[0][0][0];

if (Array != 0)

{

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

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

delete[] Array[i][j];

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 j = 0; j < side; j++)

Array[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++)

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

}

template<class T>

inline float Cube<T>::Area()

{

return side \* side \* 6;

}

template<class T>

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

{

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

tmp.print(os);

return os;

}

template<class T>

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

{

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

return true;

return false;

}

**Line.h**

#pragma once

#include"Objects.h"

#include <iostream>

using namespace std;

template <class 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 Area() override;

ostream& print(ostream& os) override;

bool operator ==(const Line& other);

};

template<class T>

inline Line<T>::Line()

{

length = 10;

Array = new char[length];

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

Array[i] = "\*";

}

template<class T>

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

{

length = len;

Array = new T[length];

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

Array[i] = \_mark;

}

template<class T>

inline 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<class T>

inline Line<T>::~Line()

{

if (Array != 0)

{

delete[] Array;

Array = 0;

length = 0;

}

}

template<class T>

inline int Line<T>::GetLen()

{

return length;

}

template<class T>

inline T Line<T>::GetMark()

{

return Array[0];

}

template<class T>

inline 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<class T>

inline float Line<T>::Area()

{

return 0;

}

template<class T>

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

{

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

os << Array[i];

os << endl;

return os;

}

template<class T>

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

{

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

return true;

return false;

}

**Objects.h**

#pragma once

#include <iostream>

using namespace std;

class Objects

{

public:

virtual ~Objects() {}

virtual float Area() = 0;

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

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

{

return other.print(os);

}

bool operator > (Objects& other)

{

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

return true;

return false;

}

bool operator < (Objects& other)

{

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

return true;

return false;

}

};

**Point.h**

#pragma once

#include"Objects.h"

#include <iostream>

using namespace std;

template <class T>

class Point : public Objects

{

private:

T mark;

public:

Point();

Point(T \_mark);

Point(const Point& point);

T Get();

float Area() override;

ostream& print(ostream& os) override;

bool operator ==(const Point& other);

};

template<class T>

inline Point<T>::Point()

{

mark = '.';

}

template<class T>

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

{

mark = \_mark;

}

template<class T>

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

{

mark = point.mark;

}

template<class T>

inline T Point<T>::Get()

{

return mark;

}

template<class T>

inline float Point<T>::Area()

{

return 0;

}

template<class T>

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

{

os << this->Get();

return os;

}

template<class T>

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

{

if (mark == other.mark)

return true;

return false;

}

**Rectangle.h**

#pragma once

#include"Objects.h"

#include <iostream>

using namespace std;

template <class T>

class Rectangle : public Objects

{

private:

int side1;

int side2;

T\*\* Array;

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 Area() override;

ostream& print(ostream& os) override;

bool operator ==(const Rectangle& other);

};

inline Rectangle<char>::Rectangle()

{

side1 = 1;

side2 = 1;

Array = new char\* [1];

Array[0] = new char[1];

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

}

template<class T>

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

{

side1 = \_side1;

side2 = \_side2;

Array = new T \* [side1];

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

Array[i] = new T[side2];

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

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

Array[i][j] = \_mark;

}

template<class T>

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

{

if (Array != 0)

{

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

delete[] Array[i];

delete[] Array;

}

side1 = rectangle.side1;

side2 = rectangle.side2;

Array = new T \* [side1];

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

Array[i] = new T[side2];

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

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

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

}

template<class T>

inline Rectangle<T>::~Rectangle()

{

if (Array != 0)

{

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

delete[] Array[i];

delete[] Array;

Array = 0;

side1 = 0;

side2 = 0;

}

}

template<class T>

inline int Rectangle<T>::GetSide1()

{

return side1;

}

template<class T>

inline int Rectangle<T>::GetSide2()

{

return side2;

}

template<class T>

inline T Rectangle<T>::GetMark()

{

return Array[0][0];

}

template<class T>

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

{

T tmp = Array[0][0];

if (Array != 0)

{

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

delete[] Array[i];

delete[] Array;

}

side1 = \_side1;

side2 = \_side2;

Array = new T \* [side1];

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

Array[i] = new T[side2];

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

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

Array[i][j] = tmp;

}

template<class T>

inline float Rectangle<T>::Area()

{

return side1 \* side2;

}

template<class T>

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

{

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

{

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

os << Array[i][j];

os << endl;

}

return os;

}

template<class T>

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

{

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

return true;

return false;

}

**Square.h**

#pragma once

#include"Objects.h"

#include <iostream>

using namespace std;

template <class 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 Area() override;

ostream& print(ostream& os) override;

bool operator ==(const Square& other);

};

inline Square<char>::Square()

{

side = 1;

Array = new char\* [1];

Array[0] = new char[1];

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

}

template<class T>

inline 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<class T>

inline 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<class T>

inline Square<T>::~Square()

{

if (Array != 0)

{

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

delete[] Array[i];

delete[] Array;

Array = 0;

side = 0;

}

}

template<class T>

inline int Square<T>::GetSide()

{

return side;

}

template<class T>

inline T Square<T>::GetMark()

{

return Array[0][0];

}

template<class T>

inline 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<class T>

inline float Square<T>::Area()

{

return side \* side;

}

template<class T>

inline 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<class T>

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

{

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

return true;

return false;

}

**Triangle.h**

#pragma once

#include"Objects.h"

#include <iostream>

using namespace std;

template <class T>

class Triangle : public Objects

{

private:

int height;

T\*\* Array;

public:

Triangle();

Triangle(int \_height, T \_mark);

Triangle(const Triangle& triangle);

~Triangle();

int GetHeight();

T GetMark();

void SetHeight(int \_height);

float Area() override;

ostream& print(ostream& os) override;

bool operator ==(const Triangle& other);

};

inline Triangle<char>::Triangle()

{

height = 3;

Array = new char\* [height];

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

Array[i] = new char[height];

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

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

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

}

template<class T>

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

{

height = \_height + 1;

Array = new T \* [height];

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

Array[i] = new T[height];

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

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

Array[i][j] = \_mark;

}

template<class T>

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

{

if (Array != 0)

{

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

delete[] Array[i];

delete[] Array;

}

height = triangle.height;

Array = new T \* [height];

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

Array[i] = new T[height];

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

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

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

}

template<class T>

inline Triangle<T>::~Triangle()

{

if (Array != 0)

{

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

delete[] Array[i];

delete[] Array;

Array = 0;

height = 0;

}

}

template<class T>

inline int Triangle<T>::GetHeight()

{

return height - 1;

}

template<class T>

inline T Triangle<T>::GetMark()

{

return T();

}

template<class T>

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

{

T tmp = this->GetMark();

if (Array != 0)

{

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

delete[] Array[i];

delete[] Array;

}

height = \_height + 1;

Array = new T \* [height];

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

Array[i] = new T[height];

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

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

Array[i][j] = tmp;

}

template<class T>

inline float Triangle<T>::Area()

{

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

}

template<class T>

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

{

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

{

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

os << Array[i][j];

os << endl;

}

return os;

}

template<class T>

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

{

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

return true;

return false;

}

**main.cpp**

#include <iostream>

#include "Container.h"

#include"Circle.h"

#include"Triangle.h"

#include"Square.h"

#include"Rectangle.h"

#include"Line.h"

#include"Point.h"

#include"Cube.h"

#include"Objects.h"

#include <locale.h>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Rus");

Container Container;

Point<char> point;

Line<char> line(10, '-');

Square<int> square(3, 'H');

Rectangle<int> rectangle(3, 6, 0);

Triangle<char> triangle(4, '\*');

Circle<char> circle(3, '.');

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

Container.Add(point);

Container.Add(line);

Container.Add(square);

Container.Add(rectangle);

Container.Add(triangle);

Container.Add(circle);

Container.Add(cube);

cout << "Точка:" << endl;

Container.Show(point);

cout << endl;

cout << "Линия:" << endl;

Container.Show(line);

cout << endl;

cout << "Квадрат:" << endl;

Container.Show(square);

cout << endl;

cout << "Прямоугольник:" << endl;

Container.Show(rectangle);

cout << endl;

cout << "Треугольник:" << endl;

Container.Show(triangle);

cout << endl;

cout << "Круг:" << endl;

Container.Show(circle);

cout << endl;

cout << "Куб:" << endl;

Container.Show(cube);

if (square.Area() > cube.Area())

Container.Delete(cube);

if (square.Area() < cube.Area())

Container.Delete(square);

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*А вот что лежит в Контейнере\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << Container;

return 0;

}