Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа

Дисциплина: «Объектно-ориентированное программирование»

III семестр

Задание 4: «Основы метапрограммирования»

|  |  |
| --- | --- |
| Группа: | М8О-206Б-18, №12 |
| Студент: | Кузьмичев Александр Николаевич |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: | 20.12.2019 |

Москва, 2019

1. **Задание**

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Необходимо настроить сборку лабораторной работы с помощью CMake. Собранная программа должна называться oop\_exercise\_04 (в случае использования Windows oop\_exercise\_04.exe)

Репозиторий должен содержать файлы:

• main.cpp //файл с заданием работы

• CMakeLists.txt // файл с конфигураций CMake

• test\_xx.txt // файл с тестовыми данными. Где xx – номер тестового набора 01, 02 , … Тестовых наборов

• report.doc // отчет о лабораторной работе

Разработать шаблоны классов согласно варианту задания. Параметром шаблона должен являться скалярный тип

данных задающий тип данных для оси координат. Классы должны иметь публичные поля. Фигуры являются

Создать набор шаблонов, создающих функции, реализующие:

1. Вычисление геометрического центра фигуры;

2. Вывод в стандартный поток вывода std::cout координат вершин фигуры;

3. Вычисление площади фигуры;

Параметром шаблона должен являться тип класса фигуры ( например Square<int>). Помимо самого класса фигуры, шаблонная функция должна уметь работать с tuple. Например, std::tuple<std::pair<int,int>, std::pair<int,int>, std::pair<int,int>> должен интерпретироваться как треугольник. std::tuple<std::pair<int,int>, std::pair<int,int>, std::pair<int,int>, std::pair<int,int>> - как квадрат. Каждый std::pair<int,int> - соответствует координатам вершины фигуры вращения.

Создать программу, которая позволяет:

• Вводить из стандартного ввода std::cin фигуры, согласно варианту задания (как в виде класса, так и в виде

std::tuple).

• Вызывать для нее шаблонные функции (1-3).

При реализации шаблонных функций допускается использование вспомогательных шаблонов std::enable\_if, std::tuple\_size, std::is\_same.

1. **Адрес репозитория на GitHub**

https://github.com/poisoned-monkey/OOP\_labs4\_6\_7

1. **Код программы на С++**

*main.cpp*

#include <iostream>  
#include <tuple>  
  
#include "vertex.h"  
#include "trapeze.h"  
#include "rhombus.h"  
#include "pentagon.h"  
#include "templates.h"  
  
**template**<**class** T>  
**void** process() {  
 T object;  
 read(std::cin, object);  
 print(std::cout, object);  
 std::cout << square(object) << std::endl;  
 std::cout << center(object) << std::endl;  
}  
  
**int** main() {  
 std::cout << "Как вы хотите ввести фигуру: " << std::endl;  
 std::cout << "1. Кортеж(Tuple)" << std::endl;  
 std::cout << "2. Класс" << std::endl;  
 **int** menu, angles, figure;  
 std::cin >> menu;  
 std::cout << "Сколько углов у фигуры (4, 5): " << std::endl;  
 std::cin >> angles;  
 **switch** (menu) {  
 **case** 1 :  
 **switch** (angles) {  
 **case** 4:  
 process<std::tuple<Vertex<**double**>, Vertex<**double**>, Vertex<**double**>, Vertex<**double**>>>();  
 **break**;  
 **case** 5:  
 process<std::tuple<Vertex<**double**>, Vertex<**double**>, Vertex<**double**>, Vertex<**double**>, Vertex<**double**>>>();  
 **break**;  
 }  
 **break**;  
 **case** 2:  
 **switch** (angles) {  
 **case** 4:  
 std::cout << "Введите фигуру: " << std::endl;  
 std::cout << "1. Трапеция" << std::endl;  
 std::cout << "2. Ромб" << std::endl;  
 std::cin >> figure;  
 **switch** (figure) {  
 **case** 1:  
 process<Trapeze<**double**>>();  
 **break**;  
 **case** 2:  
 process<Rhombus<**double**>>();  
 **break**;  
 }  
 **break**;  
 **case** 5:  
 process<Pentagon<**double**>>();  
 **break**;  
 }  
 **break**;  
 }  
 system("pause");  
 **return** 0;  
}

*pentagon.h*

#pragma once  
#include<iostream>  
#include"vertex.h"  
**template** <**class** T> **class** Pentagon{  
**private**:  
 Vertex<T> Vertexs[5];  
**public**:  
 **using** vertex\_type = Vertex<T>;  
 Pentagon();  
 Pentagon(std::istream& in);  
 Vertex<T> center() **const**;  
 **double** square() **const**;  
 **void** read(std::istream& in);  
 **void** print(std::ostream& os) **const**;  
};  
  
**template**<**class** T>  
Pentagon<T>::Pentagon() {}  
  
**template**<**class** T> Pentagon<T>::Pentagon(std::istream& in) {  
 **for** (**int** i = 0; i < 5; i++)  
 in >> Vertexs[i];  
}  
**template**<**class** T> **double** Pentagon<T>::square() **const** {  
 **double** Area = 0;  
 **for** (**int** i = 0; i < 5; i++) {  
 Area += (Vertexs[i].x) \* (Vertexs[(i + 1) % 5].y) - (Vertexs[(i + 1) % 5].x)\*(Vertexs[i].y);  
 }  
 Area \*= 0.5;  
 **return** abs(Area);  
}  
  
**template**<**class** T> **void** Pentagon<T>::print(std::ostream& os) **const** {  
 std::cout << "Pentagon: ";  
 **for** (**int** i = 0; i < 5; i++)  
 std::cout << Vertexs[i] << ' ';  
 std::cout << '\n';  
}  
  
**template**<**class** T> Vertex<T> Pentagon<T>::center() **const** {  
 Vertex<T> res = Vertex<T>();  
 **for** (**int** i = 0; i < 5; i++)  
 res += Vertexs[i];  
 **return** res / 5;  
}  
  
**template** <**class** T> **void** Pentagon<T>::read(std::istream& in) {  
 Pentagon<T> res = Pentagon(in);  
 \***this** = res;  
}

*rhombus.h*

#pragma once  
#include<iostream>  
#include"vertex.h"  
**template** <**class** T> **class** Rhombus {  
**private**:  
 Vertex<T> Vertexs[4];  
**public**:  
 **using** vertex\_type = Vertex<T>;  
 Rhombus();  
 Rhombus(std::istream& in);  
 Vertex<T> center() **const** ;  
 **double** square() **const**;  
 **void** read(std::istream& in);  
 **void** print(std::ostream& os) **const**;  
};  
  
**template**<**class** T> Rhombus<T>::Rhombus() {}  
  
**template**<**class** T> Rhombus<T>::Rhombus(std::istream& in) {  
 **for** (**int** i = 0; i < 4; i++)  
 in >> Vertexs[i];  
}  
**template**<**class** T> **double** Rhombus<T>::square() **const** {  
 **double** Area = 0;  
 **for** (**int** i = 0; i < 4; i++) {  
 Area += (Vertexs[i].x) \* (Vertexs[(i + 1) % 4].y) - (Vertexs[(i + 1) % 4].x)\*(Vertexs[i].y);  
 }  
 Area \*= 0.5;  
 **return** abs(Area);  
}  
  
**template**<**class** T> **void** Rhombus<T>::print(std::ostream& os) **const** {  
 std::cout << "Rhombus: ";  
 **for** (**int** i = 0; i < 4; i++)  
 std::cout << Vertexs[i] << ' ';  
 std::cout << '\n';  
}  
  
**template**<**class** T> Vertex<T> Rhombus<T>::center() **const** {  
 Vertex<T> res = Vertex<T>();  
 **for** (**int** i = 0; i < 4; i++)  
 res += Vertexs[i];  
 **return** res / 4;  
}  
  
**template** <**class** T> **void** Rhombus<T>::read(std::istream& in) {  
 Rhombus<T> res = Rhombus<T>(in);  
 \***this** = res;  
}

*templates.h*

#include<tuple>  
#include<type\_traits>  
  
#include"vertex.h"  
  
**template** <**class** T>  
**struct** is\_vertex : std::false\_type {};  
  
**template** <**class** T>  
**struct** is\_vertex<Vertex<T>> : std::true\_type {};  
  
**template**<**class** T>  
**struct** is\_figurelike\_tuple : std::false\_type {};  
  
**template**<**class** Head, **class**... Tail>  
**struct** is\_figurelike\_tuple<std::tuple<Head, Tail...>> : std::conjunction<is\_vertex<Head>, std::is\_same<Head, Tail>...>{};  
  
  
  
**template**<**class** T>  
**inline constexpr bool** is\_figurelike\_tuple\_v = is\_figurelike\_tuple<T>::value;  
  
**template**<**class** T, **class** = **void**>  
**struct** has\_method\_square : std::false\_type {};  
  
**template**<**class** T>  
**struct** has\_method\_square<T, std::void\_t<**decltype**(std::declval<**const** T&>().*square*())>> : std::true\_type {};  
  
**template**<**class** T>  
**inline constexpr bool** has\_method\_square\_v = has\_method\_square<T>::value;  
  
**template**<**class** T>  
std::enable\_if\_t<has\_method\_square\_v<T>, **double**> square(**const** T& object) {  
 **return** object.*square*();  
}  
  
  
**template**< **class** T>  
  
  
**double** compute\_square(**const** T& tuple) {  
 **if constexpr**(std::tuple\_size\_v<T> == 4){  
 **double** Area = 0;  
 **double** a1 = (std::get<0>(tuple).*x*) \* (std::get<1>(tuple).*y*) - (std::get<1>(tuple).*x*)\*(std::get<0>(tuple).*y*);  
 **double** a2 = (std::get<1>(tuple).*x*) \* (std::get<2>(tuple).*y*) - (std::get<2>(tuple).*x*)\*(std::get<1>(tuple).*y*);  
 **double** a3 = (std::get<2>(tuple).*x*) \* (std::get<3>(tuple).*y*) - (std::get<3>(tuple).*x*)\*(std::get<2>(tuple).*y*);  
 **double** a4 = (std::get<3>(tuple).*x*) \* (std::get<0>(tuple).*y*) - (std::get<0>(tuple).*x*)\*(std::get<3>(tuple).*y*);  
 **double** res = 0.5 \* (a1 + a2 + a3 + a4);  
 **return** res;  
 }  
 **else if constexpr** (std::tuple\_size\_v<T> == 5) {  
 **double** a1 = (std::get<0>(tuple).*x*) \* (std::get<1>(tuple).*y*) - (std::get<1>(tuple).*x*)\*(std::get<0>(tuple).*y*);  
 **double** a2 = (std::get<1>(tuple).*x*) \* (std::get<2>(tuple).*y*) - (std::get<2>(tuple).*x*)\*(std::get<1>(tuple).*y*);  
 **double** a3 = (std::get<2>(tuple).*x*) \* (std::get<3>(tuple).*y*) - (std::get<3>(tuple).*x*)\*(std::get<2>(tuple).*y*);  
 **double** a4 = (std::get<3>(tuple).*x*) \* (std::get<4>(tuple).*y*) - (std::get<4>(tuple).*x*)\*(std::get<3>(tuple).*y*);  
 **double** res = 0.5 \* (a1 + a2 + a3 + a4);  
 **return** res;  
 }  
 **return** 0;  
}  
  
**template**<**class** T>  
std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, **double**> square(**const** T& object) {  
 **if constexpr** (std::tuple\_size\_v<T> < 4 || std::tuple\_size\_v<T> > 5) {  
 **return** 0;  
 }  
 **else** {  
 **return** compute\_square(object);  
 }  
}  
  
//-----------------  
**template**<**class** T, **class** = **void**>  
**struct** has\_method\_center : std::false\_type {};  
  
**template**<**class** T>  
**struct** has\_method\_center<T, std::void\_t<**decltype**(std::declval<**const** T&>().*center*())>> : std::true\_type {};  
  
**template**<**class** T>  
**inline constexpr bool** has\_method\_center\_v = has\_method\_center<T>::value;  
  
**template**<**class** T>  
std::enable\_if\_t<has\_method\_center\_v<T>, std::void\_t<**decltype**(std::declval<**const** T&>().*Vertexs*[0])>> center(**const** T& object) {  
 **return** object.*center*();  
}  
  
**template**<**class** T>  
std::enable\_if\_t<has\_method\_center\_v<T>, **typename** T::vertex\_type> center(**const** T& object) {  
 **return** object.*center*();  
}  
  
  
  
**template**<size\_t *Id*, **class** T>  
std::tuple\_element\_t<0,T> compute\_center(**const** T& tuple) {  
 **using** vertex\_type = std::tuple\_element\_t<0, T>;  
 vertex\_type res{};  
 **if constexpr** (std::tuple\_size\_v<T> == 4) {  
 res += std::get<0>(tuple) + std::get<1>(tuple) + std::get<2>(tuple) + std::get<3>(tuple);  
 **return** res / 4;  
 }  
 **else if** (std::tuple\_size\_v<T> == 5) {  
 res = std::get<0>(tuple) + std::get<1>(tuple) + std::get<2>(tuple) + std::get<3>(tuple) + std::get<4>(tuple);  
 **return** res / 5;  
 }  
 **else  
 return** res;  
}  
  
**template**<**class** T>  
std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, std::tuple\_element\_t<0,T>>  
center(**const** T& object) {  
 **using** vertex\_type = std::tuple\_element\_t<0, T>;  
 vertex\_type res{};  
 **if constexpr** (std::tuple\_size\_v<T> < 4 || std::tuple\_size\_v<T> > 5) {  
 **return** res;  
 }  
 **else** {  
 **return** std::tuple\_size\_v<T> == 4 ? compute\_center<3>(object) : compute\_center<4>(object);  
 }  
}  
  
  
**template**<**class** T, **class** = **void**>  
**struct** has\_method\_print : std::false\_type {};  
  
**template**<**class** T>  
**struct** has\_method\_print<T, std::void\_t<**decltype**(std::declval<**const** T&>().*print*(std::cout))>> : std::true\_type {};  
  
**template**<**class** T>  
**inline constexpr bool** has\_method\_print\_v = has\_method\_print<T>::value;  
  
**template**<**class** T>  
std::enable\_if\_t<has\_method\_print\_v<T>, **void**> print(std::ostream& on, **const** T& object) {  
 object.*print*(on);  
}  
  
  
  
**template**<size\_t *Id*, **class** T>  
**void** t\_print(std::ostream& on, T& tuple) {  
 **if constexpr** (*Id* >= std::tuple\_size\_v<T>) {  
 **return**;  
 }  
 **else** {  
 **if constexpr** (std::tuple\_size\_v<T> == 4)  
 **if**(vector\_product(std::get<0>(tuple) - std::get<3>(tuple), std::get<1>(tuple) - std::get<2>(tuple)) == 0) {  
 on << "Trapeze: " << std::get<0>(tuple) << ' ' << std::get<1>(tuple) << ' ' << std::get<2>(tuple) << ' ' << std::get<3>(tuple) << ' ';  
 }  
 **else if constexpr**(std::tuple\_size\_v<T> == 4)  
 **if**(distance(std::get<0>(tuple), std::get<3>(tuple)) == distance(std::get<0>(tuple), std::get<1>(tuple)) && distance(std::get<0>(tuple), std::get<3>(tuple)) == distance(std::get<1>(tuple), std::get<2>(tuple)) && distance(std::get<0>(tuple), std::get<3>(tuple)) == distance(std::get<2>(tuple), std::get<3>(tuple))) {  
 on << "Rhombus: " << std::get<0>(tuple) << ' ' << std::get<1>(tuple) << ' ' << std::get<2>(tuple) << ' ' << std::get<3>(tuple) << ' ';  
 }  
 **else if constexpr** (std::tuple\_size\_v<T> == 5) {  
 on << "Pentagon: " << std::get<0>(tuple) << ' ' << std::get<1>(tuple)<< ' ' << std::get<2>(tuple) << ' ' << std::get<3>(tuple) << ' ' << std::get<4>(tuple);  
 }  
 **return**;  
 }  
}  
  
**template**<**class** T>  
std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, **void**>  
print(std::ostream& in, T& object) {  
 **if constexpr** (std::tuple\_size\_v<T> < 4 || std::tuple\_size\_v<T> > 5) {  
 **return**;  
 }  
 **else** {  
 std::tuple\_size\_v<T> == 4 ? t\_print<3>(std::cout, object) : t\_print<4>(std::cout, object);  
 }  
}  
  
  
//-----------------  
**template**<**class** T, **class** = **void**>  
**struct** has\_method\_read : std::false\_type {};  
  
**template**<**class** T>  
**struct** has\_method\_read<T, std::void\_t<**decltype**(std::declval<T&>().*read*(std::cin))>> : std::true\_type {};  
  
**template**<**class** T>  
**inline constexpr bool** has\_method\_read\_v = has\_method\_print<T>::value;  
  
**template**<**class** T>  
std::enable\_if\_t<has\_method\_read\_v<T>, **void**> read(std::istream& in, T& object) {  
 object.*read*(in);  
}  
  
  
  
**template**<size\_t *Id*, **class** T>  
**void** t\_read(std::istream& in, T& tuple) {  
 **if constexpr** (*Id* >= std::tuple\_size\_v<T>) {  
 **return**;  
 }  
 **else** {  
 **if constexpr** (std::tuple\_size\_v<T> == 4) {  
 in >> std::get<0>(tuple) >> std::get<1>(tuple) >> std::get<2>(tuple) >> std::get<3>(tuple);  
 }  
 **else if constexpr** (std::tuple\_size\_v<T> == 5) {  
 in >> std::get<0>(tuple) >> std::get<1>(tuple) >> std::get<2>(tuple) >> std::get<3>(tuple) >> std::get<4>(tuple);  
 }  
 **return**;  
 }  
}  
  
**template**<**class** T>  
std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, **void**>  
read(std::istream& in, T& object) {  
 **if constexpr** ((std::tuple\_size\_v<T> < 4) || (std::tuple\_size\_v<T> > 5)) {  
 **return**;  
 }  
 **else if constexpr** ((std::tuple\_size\_v<T>) == 4)  
 t\_read<3>(std::cin, object);  
 **else if constexpr** ((std::tuple\_size\_v<T>) == 5)  
 t\_read<4>(std::cin, object);  
}

*trapeze.h*

#pragma once  
#include<iostream>  
#include"vertex.h"  
**template** <**class** T>  
**class** Trapeze {  
**private**:  
 Vertex<T> Vertexs[4];  
**public**:  
 **using** vertex\_type = Vertex<T>;  
 Trapeze();  
 Trapeze(std::istream& in);  
 **void** read(std::istream& in);  
 Vertex<T> center() **const** ;  
 **double** square() **const**;  
 **void** print(std::ostream& os) **const**;  
};  
  
**template**<**class** T> Trapeze<T>::Trapeze() {}  
  
**template**<**class** T> Trapeze<T>::Trapeze(std::istream& in) {  
 **for** (**int** i = 0; i < 4; i++)  
 in >> Vertexs[i];  
}  
**template**<**class** T> **double** Trapeze<T>::square() **const** {  
 **double** Area = 0;  
 **for** (**int** i = 0; i < 4; i++) {  
 Area += (Vertexs[i].x) \* (Vertexs[(i + 1) % 4].y) - (Vertexs[(i + 1) % 4].x)\*(Vertexs[i].y);  
 }  
 Area \*= 0.5;  
 **return** abs(Area);  
}  
  
**template**<**class** T> **void** Trapeze<T>::print(std::ostream& os) **const** {  
 std::cout << "Trapeze: ";  
 **for** (**int** i = 0; i < 4; i++)  
 std::cout << Vertexs[i] << ' ';  
 std::cout << '\n';  
}  
  
**template**<**class** T> Vertex<T> Trapeze<T>::center() **const** {  
 Vertex<T> res = Vertex<T>();  
 **for** (**int** i = 0; i < 4; i++)  
 res += Vertexs[i];  
 **return** res / 4;  
}  
  
**template** <**class** T> **void** Trapeze<T>::read(std::istream& in) {  
 Trapeze<T> res = Trapeze(in);  
 \***this** = res;  
}

*vertex.h*

#pragma once  
#include <cmath>  
#include<iostream>  
**template** <**class** T> **class** Vertex {  
**public**:  
 T x, y;  
 Vertex() : x(0), y(0) {};  
 Vertex(T \_x, T \_y) :x(\_x), y(\_y) {};  
 Vertex& **operator**+=(**const** Vertex& b) {  
 x += b.x;  
 y += b.y;  
 **return** \***this**;  
 }  
 Vertex& **operator**-=(**const** Vertex& b) {  
 x -= b.x;  
 y -= b.y;  
 **return** \***this**;  
 }  
};  
  
**template**<**class** T>  
std::istream& **operator**>> (std::istream& is, Vertex<T>& p) {  
 is >> p.x >> p.y;  
 **return** is;  
}  
  
**template**<**class** T>  
std::ostream& **operator**<< (std::ostream& os, **const** Vertex<T>& p) {  
 os << p.x << ' ' << p.y;  
 **return** os;  
}  
  
**template**<**class** T> Vertex<T> **operator**+(**const** Vertex<T>&a, **const** Vertex<T>& b) {  
 **return** Vertex<T>(a.x + b.x, a.y + b.y);  
}  
  
**template**<**class** T> Vertex<T> **operator**-(**const** Vertex<T>& a, **const** Vertex<T>& b) {  
 **return** Vertex<T>(a.x - b.x, a.y - b.y);  
}  
  
**template**<**class** T> Vertex<T> **operator**/(**const** Vertex<T>& a, **const int** b) {  
 **return** Vertex<T>(a.x / b, a.y / b);  
}  
  
**template** <**class** T> T distance(**const** Vertex<T>& a, **const** Vertex<T>& b) {  
 **return** sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2));  
}  
  
**template** <**class** T> T vector\_product(**const** Vertex<T>& a, **const** Vertex<T>& b) {  
 **return** a.x\*b.y - b.x\*a.y;  
}

1. **Результаты выполнения тестов**

**Test 1:**

Как вы хотите ввести фигуру:

1. Кортеж(Tuple)

2. Класс

1

Сколько углов у фигуры (4, 5):

4

0 0

3 2

4 2

5 0

Trapeze: 0 0 3 2 4 2 5 0 -6

3 1

**Test 2:**

Как вы хотите ввести фигуру:

1. Кортеж(Tuple)

2. Класс

2

Сколько углов у фигуры (4, 5):

5

1 2

2 4

4 6

6 7

8 9

Pentagon: 1 2 2 4 4 6 6 7 8 9

3.5

4.2 5.6

1. **Объяснение результатов работы программы**

Программа спрашивает, каким образом хранить фигуру, заданную несколькими вершинами, после выводится меню для вычисления площадей фигур. Особенностью программы является то, что функции, которые конструируют, выводят или считают площади могут одновременно работать как с обычными фигурами, заданными через экземпляры класса, так и tuple различной длины.

1. **Вывод**

Навыки, полученные в ходе выполнения данной лабораторной работы приодятся в реализации различных шаблонных функций с особенными спецификациями, которые будут способны работать с большим кол-вом параметров и аргументов. Также, при выполнении данной лабораторной работы я получил опыт работы с tuple.

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа

Дисциплина: «Объектно-ориентированное программирование»

III семестр

Задание 6: «Основные работы с коллекциями: итераторы»

|  |  |
| --- | --- |
| Группа: | М8О-206Б-18, №12 |
| Студент: | Кузьмичев Александр Николаевич |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: | 22.12.2019 |

Москва, 2019

**1. Задание**

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Необходимо настроить сборку лабораторной работы с помощью CMake. Собранная программа должна называться oop\_exercise\_06 (в случае использования Windows oop\_exercise\_06.exe)

Необходимо зарегистрироваться на GitHub (если студент уже имеет регистрацию на GitHub то можно использовать ее) и создать репозитарий для задания лабораторной работы.

Преподавателю необходимо предъявить ссылку на публичный репозиторий на Github. Имя репозитория должно быть <https://github.com/login/oop_exercise_06>

Создать шаблон динамической коллекцию, согласно варианту задания:

1. Коллекция должна быть реализована с помощью умных указателей (std::shared\_ptr, std::weak\_ptr).

Опционально использование std::unique\_ptr;

2. В качестве параметра шаблона коллекция должна принимать тип данных;

3. Коллекция должна содержать метод доступа:

Стек – pop, push, top;

Очередь – pop, push, top;

Список,Динамический массив – доступ к элементу по оператору [];

Реализовать аллокатор, который выделяет фиксированный размер памяти (количество блоков памяти – является параметром шаблона аллокатора). Внутри аллокатор должен хранить указатель на используемый блок памяти и динамическую коллекцию указателей на свободные блоки. Динамическая коллекция должна соответствовать варианту задания (Динамический массив, Список, Стек, Очередь);

Коллекция должна использовать аллокатор для выделеления и освобождения памяти для своих элементов.

Аллокатор должен быть совместим с контейнерами std::map и std::list (опционально – vector).

Реализовать программу, которая: Позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию использующую аллокатор; Позволяет удалять элемент из коллекции по номеру элемента; Выводит на экран введенные фигуры c помощью std::for\_each;

1. **Адрес репозитория на GitHub**

<https://github.com/poisoned-monkey/OOP_labs4_6_7>

1. **Код программы на С++**

*main.cpp*

#include<iostream>  
#include<algorithm>  
#include<locale.h>  
#include"trapeze.h"  
#include"containers.h"  
#include"allocators.h"  
  
**void** Menu1() {  
 std::cout << "1. Добавить фигуру в список\n";  
 std::cout << "2. Удалить фигуру\n";  
 std::cout << "3. Вывести фигуру\n";  
 std::cout << "4. Вывести все фигуры\n";  
 std::cout << "5. Вывести кол-во фигур чья площаль больше чем ...\n";  
}  
  
**void** Push() {  
 std::cout << "1. Добавить фигуру в начало списка\n";  
 std::cout << "2. Добавить фигуру в конец списка\n";  
 std::cout << "3. Добавить фигуру по индексу\n";  
}  
  
**void** Delete() {  
 std::cout << "1. Удалить фигуру в начале списка\n";  
 std::cout << "2. Удалить фигуру в конце списка\n";  
 std::cout << "3. Удалить фигуру по индексу\n";  
}  
  
**void** Print() {  
 std::cout << "1. Вывести первую фигуру в списке\n";  
 std::cout << "2. Вывести последнюю фигуру в списке\n";  
 std::cout << "3. Вывести фигуру по индексу\n";  
}  
  
**int** main() {  
 std::cout<<**sizeof**(Trapeze<**int**>)<<std::endl;  
 setlocale(LC\_ALL, "rus");  
 containers::list<Trapeze<**int**>, allocators::my\_allocator<Trapeze<**int**>, 500>> MyList;  
 Trapeze<**int**> TempTrapeze;  
 **while** (**true**) {  
 Menu1();  
 **int** n, m, ind;  
 **double** s;  
 std::cin >> n;  
 **switch** (n) {  
 **case** 1:  
 TempTrapeze.read(std::cin);  
 Push();  
 std::cin >> m;  
 **switch** (m) {  
 **case** 1:  
 MyList.push\_front(TempTrapeze);  
 **break**;  
 **case** 2:  
 MyList.push\_back(TempTrapeze);  
 **break**;  
 **case** 3:  
 std::cin >> ind;  
 MyList.insert\_by\_number(ind, TempTrapeze);  
 **default**:  
 **break**;  
 }  
 **break**;  
 **case** 2:  
 Delete();  
 std::cin >> m;  
 **switch** (m) {  
 **case** 1:  
 MyList.pop\_front();  
 **break**;  
 **case** 2:  
 MyList.pop\_back();  
 **break**;  
 **case** 3:  
 std::cin >> ind;  
 MyList.delete\_by\_number(ind);  
 **break**;  
 **default**:  
 **break**;  
 }  
 **break**;  
 **case** 3:  
 Print();  
 std::cin >> m;  
 **switch** (m) {  
 **case** 1:  
 MyList.front().print(std::cout);  
 std::cout << std::endl;  
 **break**;  
 **case** 2:  
 MyList.back().print(std::cout);  
 std::cout << std::endl;  
 **break**;  
 **case** 3:  
 std::cin >> ind;  
 MyList[ind].print(std::cout);  
 std::cout << std::endl;  
 **break**;  
 **default**:  
 **break**;  
 }  
 **break**;  
 **case** 4:  
 std::for\_each(MyList.begin(), MyList.end(), [](Trapeze<**int**> &X) { X.print(std::cout); std::cout << std::endl; });  
 **break**;  
 **case** 5:  
 std::cin >> s;  
 std::cout << std::count\_if(MyList.begin(), MyList.end(), [=](Trapeze<**int**>& X) {**return** X.square() > s; }) << std::endl;  
 **break**;  
 **default**:  
 **return** 0;  
 }  
 }  
 system("pause");  
 **return** 0;  
}

*vertex.h*

#pragma once  
#include<iostream>  
#include<cmath>  
**template**<**class** T>  
**class** Vertex {  
**public**:  
 T x, y;  
};  
  
**template**<**class** T>  
std::istream& **operator**>>(std::istream& is, Vertex<T>& point) {  
 is >> point.x >> point.y;  
 **return** is;  
}  
  
**template**<**class** T>  
std::ostream& **operator**<<(std::ostream& os, Vertex<T> point) {  
 os <<'['<< point.x << ", " << point.y<<']';  
 **return** os;  
}  
  
**template**<**class** T>  
Vertex<T> **operator**+(**const** Vertex<T>& a, **const** Vertex<T>& b) {  
 Vertex<T> res;  
 res.x = a.x + b.x;  
 res.y = a.y + b.y;  
 **return** res;  
}  
  
**template**<**class** T>  
Vertex<T> **operator**+=(Vertex<T> &a, **const** Vertex<T> &b) {  
 a.x += b.x;  
 a.y += b.y;  
 **return** a;  
}  
  
**template**<**class** T>  
**double** distance(**const** Vertex<T> &a, **const** Vertex<T>& b) {  
 **return** sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2));  
}

*trapeze.h*

#pragma once  
#include"vertex.h"  
**template** <**class** T>  
**class** Trapeze {  
**private**:  
 Vertex<T> Vertexs[4];  
**public**:  
 **using** vertex\_type = Vertex<T>;  
 Trapeze();  
 Trapeze(std::istream& in);  
 **void** read(std::istream& in);  
 Vertex<T> center() **const**;  
 **double** square() **const**;  
 **void** print(std::ostream& os) **const**;  
 **friend** std::ostream& **operator**<< (std::ostream &out, **const** Trapeze<T> &point);  
 **friend** std::ostream& **operator**>> (std::istream &in, **const** Trapeze<T> &point);  
};  
  
**template**<**class** T> Trapeze<T>::Trapeze() {}  
  
**template**<**class** T> Trapeze<T>::Trapeze(std::istream& in) {  
 **for** (**int** i = 0; i < 4; i++)  
 in >> Vertexs[i];  
}  
**template**<**class** T> **double** Trapeze<T>::square() **const** {  
 **double** Area = 0;  
 **for** (**int** i = 0; i < 4; i++) {  
 Area += (Vertexs[i].x) \* (Vertexs[(i + 1) % 4].y) - (Vertexs[(i + 1) % 4].x)\*(Vertexs[i].y);  
 }  
 Area \*= 0.5;  
 **return** abs(Area);  
}  
  
**template**<**class** T> **void** Trapeze<T>::print(std::ostream& os) **const** {  
 os << "Trapeze: ";  
 **for** (**int** i = 0; i < 4; i++)  
 os << Vertexs[i] << ' ';  
 os << '\n';  
}  
  
  
**template**<**class** T> Vertex<T> Trapeze<T>::center() **const** {  
 Vertex<T> res = Vertex<T>();  
 **for** (**int** i = 0; i < 4; i++)  
 res += Vertexs[i];  
 **return** res / 4;  
}  
  
**template** <**class** T> **void** Trapeze<T>::read(std::istream& in) {  
 Trapeze<T> res = Trapeze(in);  
 \***this** = res;  
}  
  
**template**<**class** T>  
std::ostream& **operator**<< (std::ostream &out, **const** Trapeze<T> &point) {  
 out << "Trapeze: ";  
 **for** (**int** i = 0; i < 4; i++)  
 out << point.Vertexs[i] << ' ';  
 out << '\n';  
}  
  
**template**<**class** T>  
std::ostream& **operator**>> (std::istream &in, **const** Trapeze<T> &point){  
 **for** (**int** i = 0; i < 4; i++)  
 in >> point.Vertexs[i];  
}

*list.h*

#pragma once  
#include <iterator>  
#include <memory>  
  
  
**namespace** containers {  
  
 **template**<**class** T, **class** Allocator = std::allocator<T>>  
 **class** list {  
 **private**:  
 **struct** element;  
 size\_t size = 0;  
 **public**:  
 list() = **default**;  
  
 **class** forward\_iterator {  
 **public**:  
 **using** value\_type = T;  
 **using** reference = value\_type& ;  
 **using** pointer = value\_type\* ;  
 **using** difference\_type = std::ptrdiff\_t;  
 **using** iterator\_category = std::forward\_iterator\_tag;  
 **explicit** forward\_iterator(element\* ptr);  
 T& **operator**\*();  
 forward\_iterator& **operator**++();  
 forward\_iterator **operator**++(**int**);  
 **bool operator**== (**const** forward\_iterator& other) **const**;  
 **bool operator**!= (**const** forward\_iterator& other) **const**;  
 **private**:  
 element\* it\_ptr;  
 **friend** list;  
 };  
  
 forward\_iterator begin();  
 forward\_iterator **end**();  
 **void** push\_back(**const** T& value);  
 **void** push\_front(**const** T& value);  
 T& front();  
 T& back();  
 **void** pop\_back();  
 **void** pop\_front();  
 size\_t length();  
 **bool** empty();  
 **void** delete\_by\_it(forward\_iterator d\_it);  
 **void** delete\_by\_number(size\_t N);  
 **void** insert\_by\_it(forward\_iterator ins\_it, T& value);  
 **void** insert\_by\_number(size\_t N, T& value);  
 list& **operator**=(list& other);  
 T& **operator**[](size\_t index);  
 **private**:  
 **using** allocator\_type = **typename** Allocator::**template** rebind<element>::other;  
  
 **struct** deleter {  
 **private**:  
 allocator\_type\* allocator\_;  
 **public**:  
 deleter(allocator\_type\* allocator) : allocator\_(allocator) {}  
  
 **void operator**() (element\* ptr) {  
 **if** (ptr != **nullptr**) {  
 std::allocator\_traits<allocator\_type>::destroy(\*allocator\_, ptr);  
 allocator\_->*deallocate*(ptr, 1);  
 }  
 }  
  
 };  
  
 **using** unique\_ptr = std::unique\_ptr<element, deleter>;  
 **struct** element {  
 T value;  
 unique\_ptr next\_element = { **nullptr**, deleter{**nullptr**} };  
 element\* prev\_element = **nullptr**;  
 element(**const** T& value\_) : value(value\_) {}  
 forward\_iterator next();  
 };  
  
 allocator\_type allocator\_{};  
 unique\_ptr first{ **nullptr**, deleter{**nullptr**} };  
 element\* tail = **nullptr**;  
 };  
  
 **template**<**class** T, **class** Allocator>  
 **typename** list<T, Allocator>::forward\_iterator list<T, Allocator>::begin() {  
 **return** forward\_iterator(first.get());  
 }  
  
 **template**<**class** T, **class** Allocator>  
 **typename** list<T, Allocator>::forward\_iterator list<T, Allocator>::**end**() {  
 **return** forward\_iterator(**nullptr**);  
 }  
 **template**<**class** T, **class** Allocator>  
 size\_t list<T, Allocator>::length() {  
 **return** size;  
 }  
 **template**<**class** T, **class** Allocator>  
 **bool** list<T, Allocator>::empty() {  
 **return** length() == 0;  
 }  
  
 **template**<**class** T, **class** Allocator>  
 **void** list<T, Allocator>::push\_back(**const** T& value) {  
 element\* result = **this**->allocator\_.*allocate*(1);  
 std::allocator\_traits<allocator\_type>::construct(**this**->allocator\_, result, value);  
 **if** (!size) {  
 first = unique\_ptr(result, deleter{ &**this**->allocator\_ });  
 tail = first.get();  
 size++;  
 **return**;  
 }  
 tail->next\_element = unique\_ptr(result, deleter{ &**this**->allocator\_ });  
 element\* temp = tail;  
 tail = tail->next\_element.get();  
 tail->prev\_element = temp;  
 size++;  
 }  
  
 **template**<**class** T, **class** Allocator>  
 **void** list<T, Allocator>::push\_front(**const** T& value) {  
 size++;  
 element\* result = **this**->allocator\_.*allocate*(1);  
 std::allocator\_traits<allocator\_type>::construct(**this**->allocator\_, result, value);  
 unique\_ptr tmp = std::move(first);  
 first = unique\_ptr(result, deleter{ &**this**->allocator\_ });  
 first->next\_element = std::move(tmp);  
 **if**(first->next\_element != **nullptr**)  
 first->next\_element->prev\_element = first.get();  
 **if** (size == 1) {  
 tail = first.get();  
 }  
 **if** (size == 2) {  
 tail = first->next\_element.get();  
 }  
 }  
  
 **template**<**class** T, **class** Allocator>  
 **void** list<T, Allocator>::pop\_front() {  
 **if** (size == 0) {  
 **throw** std::logic\_error("can`t pop from empty list");  
 }  
 **if** (size == 1) {  
 first = **nullptr**;  
 tail = **nullptr**;  
 size--;  
 **return**;  
 }  
 unique\_ptr tmp = std::move(first->next\_element);  
 first = std::move(tmp);  
 first->prev\_element = **nullptr**;  
 size--;  
 }  
  
 **template**<**class** T, **class** Allocator>  
 **void** list<T, Allocator>::pop\_back() {  
 **if** (size == 0) {  
 **throw** std::logic\_error("can`t pop from empty list");  
 }  
 **if** (tail->prev\_element){  
 element\* tmp = tail->prev\_element;  
 tail->prev\_element->next\_element = **nullptr**;  
 tail = tmp;  
 }  
 **else**{  
 first = **nullptr**;  
 tail = **nullptr**;  
 }  
 size--;  
 }  
  
  
 **template**<**class** T, **class** Allocator>  
 T& list<T, Allocator>::front() {  
 **if** (size == 0) {  
 **throw** std::logic\_error("list is empty");  
 }  
 **return** first->value;  
 }  
  
 **template**<**class** T, **class** Allocator>  
 T& list<T, Allocator>::back() {  
 **if** (size == 0) {  
 **throw** std::logic\_error("list is empty");  
 }  
 forward\_iterator i = **this**->begin();  
 **while** ( i.it\_ptr->next() != **this**->**end**()) {  
 i++;  
 }  
 **return** \*i;  
 }  
 **template**<**class** T, **class** Allocator>  
 list<T,Allocator>& list<T, Allocator>::**operator**=(list<T, Allocator>& other) {  
 size = other.size;  
 first = std::move(other.first);  
 }  
  
 **template**<**class** T, **class** Allocator>  
 **void** list<T, Allocator>::delete\_by\_it(containers::list<T, Allocator>::forward\_iterator d\_it) {  
 forward\_iterator i = **this**->begin(), **end** = **this**->**end**();  
 **if** (d\_it == **end**) **throw** std::logic\_error("out of borders");  
 **if** (d\_it == **this**->begin()) {  
 **this**->pop\_front();  
 **return**;  
 }  
 **if** (d\_it.it\_ptr == tail) {  
 **this**->pop\_back();  
 **return**;  
 }  
  
 **if** (d\_it.it\_ptr == **nullptr**) **throw** std::logic\_error("out of broders");  
 **auto** temp = d\_it.it\_ptr->prev\_element;  
 unique\_ptr temp1 = std::move(d\_it.it\_ptr->next\_element);  
 d\_it.it\_ptr->prev\_element->next\_element = std::move(temp1);  
 d\_it.it\_ptr = d\_it.it\_ptr->prev\_element;  
 d\_it.it\_ptr->next\_element->prev\_element = temp;  
  
 size--;  
 }  
  
 **template**<**class** T, **class** Allocator>  
 **void** list<T, Allocator>::delete\_by\_number(size\_t N) {  
 forward\_iterator it = **this**->begin();  
 **for** (size\_t i = 0; i < N; ++i) {  
 ++it;  
 }  
 **this**->delete\_by\_it(it);  
 }  
  
 **template**<**class** T, **class** Allocator>  
 **void** list<T, Allocator>::insert\_by\_it(containers::list<T, Allocator>::forward\_iterator ins\_it, T& value) {  
 element\* tmp = **this**->allocator\_.*allocate*(1);  
 std::allocator\_traits<allocator\_type>::construct(**this**->allocator\_, tmp, value);  
  
 forward\_iterator i = **this**->begin();  
 **if** (ins\_it == **this**->begin()) {  
 **this**->push\_front(value);  
 **return**;  
 }  
 **if**(ins\_it.it\_ptr == **nullptr**){  
 **this**->push\_back(value);  
 **return**;  
 }  
  
 tmp->prev\_element = ins\_it.it\_ptr->prev\_element;  
 ins\_it.it\_ptr->prev\_element = tmp;  
 tmp->next\_element = unique\_ptr(ins\_it.it\_ptr, deleter{ &**this**->allocator\_ });  
 tmp->prev\_element->next\_element = unique\_ptr(tmp, deleter{ &**this**->allocator\_ });  
  
 size++;  
 }  
  
 **template**<**class** T, **class** Allocator>  
 **void** list<T, Allocator>::insert\_by\_number(size\_t N, T& value) {  
 forward\_iterator it = **this**->begin();  
 **if** (N >= **this**->length())  
 it = **this**->**end**();  
 **else  
 for** (size\_t i = 0; i < N; ++i) {  
 ++it;  
 }  
 **this**->insert\_by\_it(it, value);  
 }  
 **template**<**class** T, **class** Allocator>  
 **typename** list<T,Allocator>::forward\_iterator list<T, Allocator>::element::next() {  
 **return** forward\_iterator(**this**->next\_element.get());  
 }  
  
 **template**<**class** T, **class** Allocator>  
 list<T, Allocator>::forward\_iterator::forward\_iterator(containers::list<T, Allocator>::element \*ptr) {  
 it\_ptr = ptr;  
 }  
  
 **template**<**class** T, **class** Allocator>  
 T& list<T, Allocator>::forward\_iterator::**operator**\*() {  
 **return this**->it\_ptr->value;  
 }  
 **template**<**class** T, **class** Allocator>  
 T& list<T, Allocator>::**operator**[](size\_t index) {  
 **if** (index < 0 || index >= size) {  
 **throw** std::out\_of\_range("out of list's borders");  
 }  
 forward\_iterator it = **this**->begin();  
 **for** (size\_t i = 0; i < index; i++) {  
 it++;  
 }  
 **return** \*it;  
 }  
  
 **template**<**class** T, **class** Allocator>  
 **typename** list<T, Allocator>::forward\_iterator& list<T, Allocator>::forward\_iterator::**operator**++() {  
 **if** (it\_ptr == **nullptr**) **throw** std::logic\_error("out of list borders");  
 \***this** = it\_ptr->next();  
 **return** \***this**;  
 }  
  
 **template**<**class** T, **class** Allocator>  
 **typename** list<T, Allocator>::forward\_iterator list<T, Allocator>::forward\_iterator::**operator**++(**int**) {  
 forward\_iterator old = \***this**;  
 ++\***this**;  
 **return** old;  
 }  
  
 **template**<**class** T, **class** Allocator>  
 **bool** list<T, Allocator>::forward\_iterator::**operator**==(**const** forward\_iterator& other) **const** {  
 **return** it\_ptr == other.it\_ptr;  
 }  
  
 **template**<**class** T, **class** Allocator>  
 **bool** list<T, Allocator>::forward\_iterator::**operator**!=(**const** forward\_iterator& other) **const** {  
 **return** it\_ptr != other.it\_ptr;  
 }  
}

*allocator.h*

#pragma once  
  
#include <cstdlib>  
#include <iostream>  
#include <type\_traits>  
#include <queue>  
  
**namespace** allocators {  
  
 **template**<**class** T, size\_t ALLOC\_SIZE>//ALLOC\_SIZE - размер, который требуется выделить  
 **struct** my\_allocator {  
  
 **private**:  
 **char**\* pool\_begin; //указатель на начало хранилища  
 **char**\* pool\_end;//указатель на конец хранилища  
 **char**\* pool\_tail;//указатель на конец заполненного пространства  
 std::queue<**char**\*> free\_blocks;  
 **public**:  
 **using** value\_type = T;  
 **using** size\_type = std::size\_t;  
 **using** difference\_type = std::ptrdiff\_t;  
 **using** is\_always\_equal = std::false\_type;  
  
 **template**<**class** U>  
 **struct** rebind {  
 **using** other = my\_allocator<U, *ALLOC\_SIZE*>;  
 };  
  
 my\_allocator() :  
 pool\_begin(**new char**[*ALLOC\_SIZE*]),  
 pool\_end(pool\_begin + *ALLOC\_SIZE*),  
 pool\_tail(pool\_begin)  
 {}  
  
 my\_allocator(**const** my\_allocator&) = **delete**;  
 my\_allocator(my\_allocator&&) = **delete**;  
  
 ~my\_allocator() {  
 **delete**[] pool\_begin;  
 }  
  
 T\* allocate(std::size\_t n);  
 **void** deallocate(T\* ptr, std::size\_t n);  
  
 };  
  
 **template**<**class** T, size\_t *ALLOC\_SIZE*>  
 T\* my\_allocator<T, *ALLOC\_SIZE*>::allocate(std::size\_t n) {  
 **if** (n != 1) {  
 **throw** std::logic\_error("can`t allocate arrays");  
 }  
 **if** (size\_t(pool\_end - pool\_tail) < **sizeof**(T)) {  
 **if** (free\_blocks.size()) {//ищем свободное место в райне отданном пространстве  
 **char**\* ptr = free\_blocks.front();  
 free\_blocks.pop();  
 **return reinterpret\_cast**<T\*>(ptr);  
 }  
 std::cout<<"Bad Alloc"<<std::endl;  
 **throw** std::bad\_alloc();  
 }  
 T\* result = **reinterpret\_cast**<T\*>(pool\_tail);//приведение к типу  
 pool\_tail += **sizeof**(T);  
 **return** result;  
 }  
  
 **template**<**class** T, size\_t ALLOC\_SIZE>  
 **void** my\_allocator<T, ALLOC\_SIZE>::deallocate(T\* ptr, std::size\_t n) {  
 **if** (n != 1) {  
 **throw** std::logic\_error("can`t allocate arrays, thus can`t deallocate them too");  
 }  
 **if** (ptr == **nullptr**) {  
 **return**;  
 }  
 free\_blocks.push(**reinterpret\_cast**<**char**\*>(ptr));  
 }  
  
}

*quenue.h*

#pragma once  
  
#include <memory>  
  
#include "allocators.h"  
  
**namespace** oop  
{  
 **template**<**typename** Q>  
 **class** queue\_forward\_iterator  
 {  
  
 };  
  
 **template** <**typename** T, **typename** TBaseAllocator = std::allocator<T>>  
 **class** queue  
 {  
  
 **struct** node;  
  
 **using** allocator = **typename** std::allocator\_traits<TBaseAllocator>::**template** rebind\_alloc<node>;  
  
  
 **struct** deleter  
 {  
 **explicit** deleter(allocator& al) **noexcept** : al\_(al)  
 {}  
  
 **void operator**()(node\* ptr)  
 {  
 std::allocator\_traits<allocator>::destroy(al\_, ptr);  
 al\_.*deallocate*(ptr, 1);  
 }  
  
 **private**:  
 allocator& al\_;  
 };  
  
  
 **struct** node  
 {  
 T value;  
 std::shared\_ptr<node> next;  
  
 **explicit** node(**const** T& v, deleter& d) **noexcept** : value(v)  
 , next(**nullptr**, d)  
 {}  
 };  
  
 **public**:  
 queue()  
 : deleter\_(al\_)  
 , first\_(**nullptr**, deleter\_)  
 , last\_(**nullptr**)  
 , size\_(0)  
 {}  
  
 **void** pop()  
 {  
 **if** (first\_)  
 {  
 first\_ = std::move(first\_->next);  
 }  
  
 --size\_;  
 }  
  
 **void** push(**const** T& v)  
 {  
 node\* obj = al\_.*allocate*(1);  
 std::allocator\_traits<allocator>::construct(al\_, obj, v, deleter\_);  
 **if** (last\_)  
 {  
 last\_->next.reset(obj);  
 }  
 **else** {  
 first\_.reset(obj);  
 }  
 last\_ = obj;  
  
 ++size\_;  
 }  
  
 [[nodiscard]] T& top()  
 {  
 **return** first\_->value;  
 }  
  
 [[nodiscard]] size\_t size() **const noexcept** {  
 **return** size\_;  
 }  
  
 **private**:  
 allocator al\_;  
 deleter deleter\_;  
  
 std::shared\_ptr<node> first\_;  
 node\* last\_;  
 size\_t size\_;  
  
 **friend struct** node;  
 };  
}

1. **Результаты выполнения тестов**

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

1

1 0

0 0

0 0

0 0

1. Добавить фигуру в начало списка

2. Добавить фигуру в конец списка

3. Добавить фигуру по индексу

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

1

2 0

0 0

0 0

0 0

1. Добавить фигуру в начало списка

2. Добавить фигуру в конец списка

3. Добавить фигуру по индексу

2

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

1

1 2

2 3

3 4

5 0

1. Добавить фигуру в начало списка

2. Добавить фигуру в конец списка

3. Добавить фигуру по индексу

3

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

4

Trapeze: [1, 0] [0, 0] [0, 0] [0, 0]

Trapeze: [1, 2] [2, 3] [3, 4] [5, 0]

Trapeze: [2, 0] [0, 0] [0, 0] [0, 0]

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

1

3 0

0 0

0 0

0 0

1. Добавить фигуру в начало списка

2. Добавить фигуру в конец списка

3. Добавить фигуру по индексу

2

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

4

Trapeze: [1, 0] [0, 0] [0, 0] [0, 0]

Trapeze: [1, 2] [2, 3] [3, 4] [5, 0]

Trapeze: [2, 0] [0, 0] [0, 0] [0, 0]

Trapeze: [3, 0] [0, 0] [0, 0] [0, 0]

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

5

0

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

2

1. Удалить фигуру в начале списка

2. Удалить фигуру в конце списка

3. Удалить фигуру по индексу

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

2

1. Удалить фигуру в начале списка

2. Удалить фигуру в конце списка

3. Удалить фигуру по индексу

2

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

2

1. Удалить фигуру в начале списка

2. Удалить фигуру в конце списка

3. Удалить фигуру по индексу

31

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

4

Trapeze: [1, 2] [2, 3] [3, 4] [5, 0]

Trapeze: [2, 0] [0, 0] [0, 0] [0, 0]

Trapeze: [3, 0] [0, 0] [0, 0] [0, 0]

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

2

1. Удалить фигуру в начале списка

2. Удалить фигуру в конце списка

3. Удалить фигуру по индексу

3

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

4

Trapeze: [1, 2] [2, 3] [3, 4] [5, 0]

Trapeze: [2, 0] [0, 0] [0, 0] [0, 0]

Trapeze: [3, 0] [0, 0] [0, 0] [0, 0]

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

2

1. Удалить фигуру в начале списка

2. Удалить фигуру в конце списка

3. Удалить фигуру по индексу

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

4

Trapeze: [2, 0] [0, 0] [0, 0] [0, 0]

Trapeze: [3, 0] [0, 0] [0, 0] [0, 0]

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

2

1. Удалить фигуру в начале списка

2. Удалить фигуру в конце списка

3. Удалить фигуру по индексу

3

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

4

Trapeze: [3, 0] [0, 0] [0, 0] [0, 0]

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

1. **Объяснение результатов работы программы**

Программа выводит меню со всеми применимыми к фигурам функциями – вставкой, удалением и выводом фигур из трех различных мест, а также подсчетом фигур с площадью большей чем заданное число. Функционально программа не изменилась, однако для реализованного ранее списка был написан аллокатор, который более грамотно распоряжается памятью, отведенной для хранения списка фигур.

1. **Вывод**

С помощью пользовательских аллокаторов программист может более эффективно распоряжаться отданной для хранения фигур памятью, сам следить за процессом выделения и очистки памяти, конструирования и деконструирования объектов.

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа

Дисциплина: «Объектно-ориентированное программирование»

III семестр

Задание 7: «Проектирование структуры классов»

|  |  |
| --- | --- |
| Группа: | М8О-206Б-18, №12 |
| Студент: | Кузьмичев Александр Николаевич |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: | 22.02.2020 |

Москва, 2020

**1. Задание**

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Необходимо настроить сборку лабораторной работы с помощью CMake. Собранная программа должна называться oop\_exercise\_07 (в случае использования Windows oop\_exercise\_07.exe)

Спроектировать простейший графический векторный редактор.

Требование к функционалу редактора:

• создание нового документа

• импорт документа из файла

• экспорт документа в файл

• создание графического примитива (согласно варианту задания)

• удаление графического примитива

• отображение документа на экране (печать перечня графических объектов и их характеристик)

• реализовать операцию undo, отменяющую последнее сделанное действие. Должно действовать для операций добавления/удаления фигур.

Требования к реализации:

• Создание графических примитивов необходимо вынести в отдельный класс – Factory.

• Сделать упор на использовании полиморфизма при работе с фигурами;

• Взаимодействие с пользователем (ввод команд) реализовать в функции main;

1. **Адрес репозитория на GitHub**

https://github.com/poisoned-monkey/OOP\_labs4\_6\_7

1. **Код программы на С++**

*Figure.h*

#pragma once  
#include"sdl.h"  
#include<iostream>  
  
  
**struct** color {  
 color(): r(255), g(0), b(0) {}  
 int32\_t r, g, b;  
 color(**int** r\_, **int** g\_, **int** b\_) :r(r\_), g(g\_), b(b\_) {}  
 **void** set\_color(**int** r\_, **int** g\_, **int** b\_) { r = r\_, g = g\_, b = b\_; }  
};  
**struct** figure {  
 **virtual void** render(**const** sdl::renderer& renderer) **const** = 0;  
 **virtual void** save(std::ostream& os) **const** = 0;  
 **virtual** ~figure() = **default**;  
  
  
 color color\_{};  
 **virtual void** set\_color(**int** r, **int** g, **int** b) {  
 color\_.r = r;  
 color\_.g = g;  
 color\_.b = b;  
 }  
 **virtual void** set\_color(color c) {  
 color\_.r = c.r;  
 color\_.g = c.g;  
 color\_.b = c.b;  
 }  
};  
  
  
**struct** vertex {  
 int32\_t x, y;  
};

*Figures.h*

#pragma once  
#include"pentagon.h"  
#include"rhombus.h"  
#include"trapeze.h"  
#include"line.h"  
#include"curve\_line.h"

*Main.cpp*

#include <array>  
#include <fstream>  
#include <iostream>  
#include <memory>  
#include <vector>  
  
#include "sdl.h"  
#include "lib/imgui/imgui.h"  
#include "tools.h"  
#include "loader.h"  
#include "canvas.h"  
  
  
**int** main() {  
 canvas canv;  
 color figure\_color{};  
 sdl::renderer renderer("Editor");  
 **bool** quit = **false**;  
 std::unique\_ptr<builder> active\_builder = **nullptr**;  
 **const** int32\_t file\_name\_length = 128;  
 **char** file\_name[file\_name\_length] = "";  
 int32\_t remove\_id = 0;  
 **while**(!quit){  
 renderer.set\_color(0, 0, 0);  
 renderer.clear();  
  
 sdl::event event;  
  
 **while**(sdl::event::poll(event)){  
 sdl::quit\_event quit\_event;  
 sdl::mouse\_button\_event mouse\_button\_event;  
 **if**(event.extract(quit\_event)){  
 quit = **true**;  
 **break**;  
 }**else if**(event.extract(mouse\_button\_event)){  
 **if**(active\_builder && mouse\_button\_event.button() == sdl::mouse\_button\_event::left &&  
 mouse\_button\_event.type() == sdl::mouse\_button\_event::down){  
 std::unique\_ptr<figure> figure =  
 active\_builder->add\_vertex(vertex{mouse\_button\_event.x(), mouse\_button\_event.y()});  
 **if**(figure){  
 (\*figure).set\_color(figure\_color);  
 canv.add\_figure(std::move(figure));  
 active\_builder = **nullptr**;  
 }  
 }  
 }  
 }  
  
 **for** (**int** i = 0; i < canv.figures.size(); ++i) {  
 **if**(canv.figures[i])  
 canv.figures[i]->render(renderer);  
 }  
  
 ImGui::Begin("Menu");  
 **if** (ImGui::Button("New canvas")) {  
 canv.figures.clear();  
 }  
 ImGui::InputText("File name", file\_name, file\_name\_length - 1);  
 **if**(ImGui::Button("Save")){  
 std::ofstream os(file\_name);  
 **if**(os){  
 **for** (**int** i = 0; i < canv.figures.size();++i) {  
 canv.figures[i]->save(os);  
 }  
 }  
 }  
 ImGui::SameLine();  
 **if** (ImGui::Button("Load")) {  
 std::ifstream is(file\_name);  
 **if** (is) {  
 loader loader;  
 canv.figures = loader.load(is);  
 }  
 }  
  
 ImGui::InputInt("R", &figure\_color.r);  
 ImGui::InputInt("G", &figure\_color.g);  
 ImGui::InputInt("B", &figure\_color.b);  
 **if** (ImGui::Button("Red")) {  
 figure\_color.set\_color(255, 0, 0);  
 }  
 ImGui::SameLine();  
 **if** (ImGui::Button("Green")) {  
 figure\_color.set\_color(0, 255, 0);  
 }  
 ImGui::SameLine();  
 **if** (ImGui::Button("Blue")) {  
 figure\_color.set\_color(0, 0, 255);  
 }  
  
 **if** (ImGui::Button("Line")) {  
 active\_builder = std::make\_unique<line\_builder>();  
 }  
 ImGui::SameLine();  
 **if**(ImGui::Button("Trapeze")){  
 active\_builder = std::make\_unique<trapeze\_builder>();  
 }  
  
 ImGui::SameLine();  
 **if** (ImGui::Button("Rhombus")) {  
 active\_builder = std::make\_unique<rhombus\_builder>();  
 }  
 ImGui::SameLine();  
 **if** (ImGui::Button("Pentagon")) {  
 active\_builder = std::make\_unique<pentagon\_builder>();  
 }  
 ImGui::SameLine();  
 **if** (ImGui::Button("Curve Line")) {  
 active\_builder = std::make\_unique<curve\_line\_builder>();  
 }  
  
  
 ImGui::InputInt("Remove id", &remove\_id);  
 **if**(ImGui::Button("Remove")){  
 **if** (remove\_id < canv.figures.size()) {  
 **if**(canv.figures[remove\_id])  
 canv.remove\_figure(remove\_id);  
 }  
 }  
  
 **if** (ImGui::Button("Undo")) {  
 canv.undo();  
 }  
 ImGui::End();  
  
 renderer.present();  
 }  
}

*Pentagon.h*

#pragma once  
#include"tools.h"  
#include"figure.h"  
#include"journal.h"  
#include<memory>  
**struct** pentagon : figure {  
 pentagon() {}  
 pentagon(**const** std::array<vertex, 5>& vertices) : vertices\_(vertices) {}  
  
 **void** render(**const** sdl::renderer& renderer) **const** override {  
 renderer.set\_color(color\_.r, color\_.g, color\_.b);  
 **for** (int32\_t i = 0; i < 5; ++i) {  
 renderer.draw\_line(vertices\_[i].x, vertices\_[i].y,  
 vertices\_[(i + 1) % 5].x, vertices\_[(i + 1) % 5].y);  
 }  
 }  
  
 **void** save(std::ostream& os) **const** override {  
 os << "pentagon" << std::endl;  
 **for** (int32\_t i = 0; i < 5; ++i) {  
 os << vertices\_[i].x << ' ' << vertices\_[i].y << '\n';  
 }  
  
 os << **this**->color\_.r << ' ' << **this**->color\_.g << ' ' << **this**->color\_.b << std::endl;  
 }  
  
**private**:  
 std::array<vertex, 5> vertices\_;  
  
};  
  
  
**struct** pentagon\_builder : builder {  
 std::unique\_ptr<figure> add\_vertex(**const** vertex& v) {  
 vertices\_[n\_] = v;  
 n\_ += 1;  
 **if** (n\_ != 5) {  
 **return nullptr**;  
 }  
 jl.push(1, **nullptr**);  
 **return** std::make\_unique<pentagon>(vertices\_);  
 }  
**private**:  
 int32\_t n\_ = 0;  
 std::array<vertex, 5> vertices\_;  
};

*Rhombus.h*

#pragma once  
#include"tools.h"  
#include"figure.h"  
#include<memory>  
**struct** rhombus : figure {  
 rhombus(**const** std::array<vertex, 4>& vertices) : vertices\_(vertices) {}  
  
 **void** render(**const** sdl::renderer& renderer) **const** override {  
 renderer.set\_color(color\_.r, color\_.g, color\_.b);  
 **for** (int32\_t i = 0; i < 4; ++i) {  
 renderer.draw\_line(vertices\_[i].x, vertices\_[i].y,  
 vertices\_[(i + 1) % 4].x, vertices\_[(i + 1) % 4].y);  
 }  
 }  
  
 **void** save(std::ostream& os) **const** override {  
 os << "rhombus" << std::endl;  
 **for** (int32\_t i = 0; i < 4; ++i) {  
 os << vertices\_[i].x << ' ' << vertices\_[i].y << '\n';  
 }  
  
 os << **this**->color\_.r << ' ' << **this**->color\_.g << ' ' << **this**->color\_.b << std::endl;  
 }  
  
**private**:  
 std::array<vertex, 4> vertices\_;  
  
};  
  
  
**struct** rhombus\_builder : builder {  
 std::unique\_ptr<figure> add\_vertex(**const** vertex& v) {  
 vertices\_[n\_] = v;  
 n\_ += 1;  
 **if** (n\_ != 4) {  
 **return nullptr**;  
 }  
 jl.push(1, **nullptr**);  
 **return** std::make\_unique<rhombus>(vertices\_);  
 }  
  
**private**:  
 int32\_t n\_ = 0;  
 std::array<vertex, 4> vertices\_;  
  
};

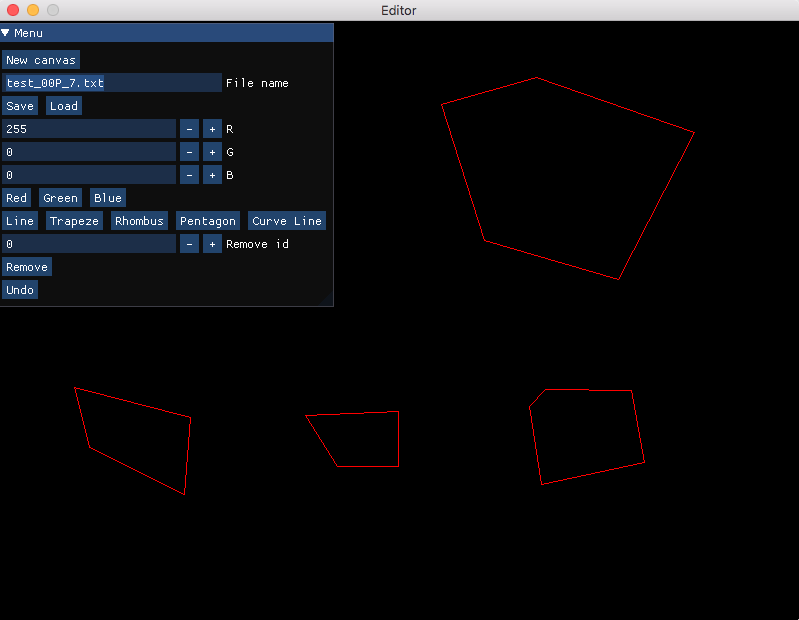
*Tools.h*

#pragma once  
#include<cmath>  
#include<memory>  
#include<stack>  
#include<string>  
#include<iostream>  
#include<vector>  
#include<array>  
#include"sdl.h"  
#include"figures/figure.h"  
  
  
**struct** builder {  
 **virtual** std::unique\_ptr<figure> add\_vertex(**const** vertex& v) = 0;  
  
 **virtual** ~builder() = **default**;  
  
};

*Trapeze.h*

#pragma once  
#include"tools.h"  
#include"figure.h"  
#include<memory>  
**struct** trapeze : figure {  
 trapeze(**const** std::array<vertex, 4>& vertices) : vertices\_(vertices) {}  
  
 **void** render(**const** sdl::renderer& renderer) **const** override {  
 renderer.set\_color(color\_.r, color\_.g, color\_.b);  
 **for** (int32\_t i = 0; i < 4; ++i) {  
 renderer.draw\_line(vertices\_[i].x, vertices\_[i].y,  
 vertices\_[(i + 1) % 4].x, vertices\_[(i + 1) % 4].y);  
 }  
 }  
  
 **void** save(std::ostream& os) **const** override {  
 os << "trapeze" << std::endl;  
 **for** (int32\_t i = 0; i < 4; ++i) {  
 os << vertices\_[i].x << ' ' << vertices\_[i].y << '\n';  
 }  
  
 os << **this**->color\_.r << ' ' << **this**->color\_.g << ' ' << **this**->color\_.b << std::endl;  
 }  
  
**private**:  
 std::array<vertex, 4> vertices\_;  
  
};  
  
  
  
**struct** trapeze\_builder : builder {  
 std::unique\_ptr<figure> add\_vertex(**const** vertex& v) {  
 vertices\_[n\_] = v;  
 n\_ += 1;  
 **if** (n\_ != 4) {  
 **return nullptr**;  
 }  
 jl.push(1, **nullptr**);  
 **return** std::make\_unique<trapeze>(vertices\_);  
 }  
  
**private**:  
 int32\_t n\_ = 0;  
 std::array<vertex, 4> vertices\_;  
  
};

1. **Результаты выполнения тестов**



*Test\_OOP\_7.txt*

trapeze

75 366

90 426

185 473

191 396

255 0 0

rhombus

306 394

338 445

399 445

399 390

255 0 0

pentagon

530 385

542 463

645 441

632 369

546 368

255 0 0

pentagon

485 219

619 258

695 111

537 56

442 83

255 0 0

1. **Объяснение результатов работы программы**

Программа запускается с появлением черного полотна и меню с функциями для рисования и удаления различных фигур. Пользователь может нарисовать сразу несколько видов фигур а затем удалить, либо сохранить их. Также есть возможность отмены прошлых действий при помощи кнопки undo. Внутри программы было реализовано несколько классов для построения, рендеринга и сохранения фигур, журнал изменения состояния полотна.

1. **Вывод**

Выполнив данную лабораторную работу студент может улучшить свои навыки в проектировании более сложных программ. Умение проектировать структуру классов позволяет сделать дальнейшую разработку более гибкой и простой, повысить читаемость кода. Кроме того, в ходе выполнения работы студент может познакомится с графическими библиотеками и написать свой пользовательский интерфейс. Мне было крайне интересно работать с SDL.