**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 6**

Тема: Основы работы с коллекциями: аллокаторы

Студент: Бутырев Даниил Вячеславович

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

[https://github.com/SteepturN/oop\_exercise\_0](https://github.com/SteepturN/oop_exercise_01)6

1. Постановка задачи

Лабораторная работа № 06

Тема: Основы работы с коллекциями: аллокаторы

Цель:

Изучение основ работы с контейнерами, знакомство концепцией аллокаторов памяти;

Порядок выполнения работы

1. Ознакомиться с теоретическим материалом.

2. Получить у преподавателя вариант задания.

3. Реализовать задание своего варианта в соответствии с поставленными требованиями.

4. Подготовить тестовые наборы данных.

5. Создать репозиторий на GitHub.

6. Отправить файлы лабораторной работы в репозиторий.

7. Отчитаться по выполненной работе путём демонстрации работающей программы на тестовых наборах данных (как подготовленных самостоятельно, так и предложенных преподавателем) и ответов на вопросы преподавателя (как из числа контрольных, так и по реализации программы).

Требования к программе

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

Где login – логин, выбранный студентом для своего репозитория на Github.

Репозиторий должен содержать файлы:

· main.cpp //файл с заданием работы

· CMakeLists.txt // файл с конфигураций CMake

· 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 фигурами вращения, т.е.

равносторонние (кроме трапеции и прямоугольника). Для хранения координат фигур необходимо использовать шаблон std::pair.

Например:

template <class T>

struct Square{

using vertex\_t = std::pair<T,T>;

vertex\_t a,b,c,d;

};

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:

o Стек – pop, push, top;

o Очередь – pop, push, top;

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

4. 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).

5. Коллекция должна использовать аллокатор для выделения и освобождения памяти для своих элементов.

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

7. Реализовать программу, которая:

o позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять

в коллекцию использующую аллокатор;

o позволяет удалять элемент из коллекции по номеру элемента;

o выводит на экран введенные фигуры c помощью std::for\_each.\*/

figure container allocator

19 var - Прямоугольник Очередь Стек.

1. Описание программы

1) в main обработка пользовательского ввода

2) Очередь реализована на умных указателях

3) Аллокатор реализован на стандартном стеке

1. Набор тестов

p 0 0 1 0 1 1

p 0 0 0 1 1 1

p 1 1 3 1 3 3

p 5 5 10 5 10 10

pr

ar 2

d 3

pr

ar 2

d 0

pr

ar 2

p 0 0 1 2 2 0

p 0 0 2 1 4 0

p 1 -4 0 -2 2 -3

pr

ar 3

d 2

pr

ar 3

1. Результаты выполнения тестов

You can use

--put square: p [(point) 3 times]

--delete by figure number: d (number of figure)

--print container: pr

--print number of figures, which area is less then given: ar (area)

--exit

square

sides' length:

side 0: 1

side 1: 1

side 2: 1

side 3: 1

coordinates: (0, 0)(1, 0)(1, 1)(0, 1)

square

sides' length:

side 0: 1

side 1: 1

side 2: 1

side 3: 1

coordinates: (0, 0)(0, 1)(1, 1)(1, 0)

square

sides' length:

side 0: 2

side 1: 2

side 2: 2

side 3: 2

coordinates: (1, 1)(3, 1)(3, 3)(1, 3)

square

sides' length:

side 0: 5

side 1: 5

side 2: 5

side 3: 5

coordinates: (5, 5)(10, 5)(10, 10)(5, 10)

2

square

sides' length:

side 0: 1

side 1: 1

side 2: 1

side 3: 1

coordinates: (0, 0)(1, 0)(1, 1)(0, 1)

square

sides' length:

side 0: 1

side 1: 1

side 2: 1

side 3: 1

coordinates: (0, 0)(0, 1)(1, 1)(1, 0)

square

sides' length:

side 0: 2

side 1: 2

side 2: 2

side 3: 2

coordinates: (1, 1)(3, 1)(3, 3)(1, 3)

2

square

sides' length:

side 0: 1

side 1: 1

side 2: 1

side 3: 1

coordinates: (0, 0)(0, 1)(1, 1)(1, 0)

square

sides' length:

side 0: 2

side 1: 2

side 2: 2

side 3: 2

coordinates: (1, 1)(3, 1)(3, 3)(1, 3)

1

You can use

--put square: p [(point) 3 times]

--delete by figure number: d (number of figure)

--print container: pr

--print number of figures, which area is less then given: ar (area)

--exit

wrong input

wrong input

wrong input

0

wrong input

0

1. Листинг программы

// using const\_pointer = const T\*;

#ifndef ALLOCATOR\_H\_

#define ALLOCATOR\_H\_

#include <iostream>

#include <stack>

#include <memory>

struct Block {

size\_t size;

void\* memory;

Block(size\_t, void\*);

};

template <typename T, int Size>

struct Allocator {

using value\_type = T;

template <typename U>

struct rebind {

using other = Allocator<U, Size>;

};

Allocator();

~Allocator();

template <typename U, int Size2>

Allocator(Allocator<U, Size2> const&);

[[nodiscard]] T\* allocate(int);

void deallocate(T\*, int);

inline constexpr std::size\_t max\_size() const noexcept {return 1;}

//stupid task == stupid allocator

std::shared\_ptr<char[]> storage;

std::shared\_ptr<std::stack<Block>> free\_elements;

};

template<class T1, int Sz1, class T2, int Sz2>

bool operator==(const Allocator<T1, Sz1>& lhs, const Allocator<T2, Sz2>& rhs);

#endif // ALLOCATOR\_H\_

#ifndef ITERATOR\_H\_

#define ITERATOR\_H\_

#include "../header/QueueEl.h"

#include "../header/Queue.h"

#include "../header/Square.h"

#include <memory>

template<typename T, typename Alloc>

class Queue<T, Alloc>::iterator{

public:

using iterator\_category = std::forward\_iterator\_tag;

using value\_type = T;

using difference\_type = std::size\_t;

using pointer = T\*;

using reference = T&;

std::weak\_ptr<QueueEl<T>> prev\_el;

iterator();

iterator(std::shared\_ptr<QueueEl<T>>);

iterator& operator++();

T& operator\*();

bool operator!=(iterator);

bool operator==(iterator);

bool operator==(std::nullptr\_t);

};

template class Queue<Square<int>, Allocator<QueueEl<Square<int> >, 10> >::iterator;

template class Queue<Square<int>>::iterator;

#include "../template/Queue.tpp"

#endif // ITERATOR\_H\_

#ifndef LISTEL\_H\_

#define LISTEL\_H\_

#include <memory>

#include "../header/Square.h"

template<typename T>

class QueueEl {

public:

QueueEl();

QueueEl(const T& val, const std::shared\_ptr<QueueEl>& next);

T val;

std::shared\_ptr<QueueEl> next;

};

#endif // LISTEL\_H\_

#ifndef STACK\_H

#define STACK\_H

#include <memory>

#include "../header/QueueEl.h"

#include "../header/Square.h"

#include "../header/Allocator.hpp"

template <class T, class Alloc = std::allocator<QueueEl<T>>>

class Queue {

public:

//null - > begin

class iterator;

using traits = std::allocator\_traits<Alloc>;

std::shared\_ptr<QueueEl<T>> zero\_el;

std::shared\_ptr<QueueEl<T>> end\_el;

unsigned int size;

Alloc alloc;

Queue();

~Queue();

iterator begin();

iterator end();

void insert(const T& t, iterator i);

void erase(iterator p);

T pop();

void push(const T&);

T top()/\*const\*/;

template <typename Y>

friend std::ostream& operator<<(std::ostream&, Queue<Y>&);

};

#endif // STACK\_H

#include "../template/Queue.tpp"

#ifndef READ\_INPUT\_H\_

#define READ\_INPUT\_H\_

#include <set>

#include <string>

typedef enum {

END\_OF\_FILE,

VALID\_INPUT,

INVALID\_INPUT,

END\_OF\_LINE,

} read\_return\_t;

read\_return\_t get\_command(std::set<std::string>&, char\*);

template <typename T>

read\_return\_t get\_value(T&);

template <>

read\_return\_t get\_value (unsigned int&);

template <>

read\_return\_t get\_value (unsigned long long&);

#endif // READ\_INPUT\_H\_

#ifndef SQUARE\_H\_

#define SQUARE\_H\_

#include <vector>

#include <utility>

#include <iostream>

#include <cmath>

#define NUM\_OF\_VERTECES 4

template <typename T>

class Square;

template <typename T>

std::ostream& operator<<(std::ostream& cout, const Square<T>& t);

template <typename T>

std::istream& operator>>(std::istream& cin, Square<T>& t);

template <typename T>

class Square {

public:

std::vector<std::pair<T, T>> verteces;

Square(const Square<T>& t);

Square();

std::pair<double, double> center() const noexcept;

void coordinates() const noexcept;

double area() const noexcept;

Square<T>& operator=(const Square<T>& t) noexcept;

bool operator==(const Square<T>& t) const noexcept;

friend std::ostream& operator<< <>(std::ostream& cout, const Square<T>& t);

friend std::istream& operator>> <>(std::istream& cin, Square<T>& t);

};

//#include "../template/Square.tpp"

#endif // SQUARE\_H\_

#pragma once

#include <set>

#include <list>

#include "../header/Square.h"

#include "../header/QueueEl.h"

template class Allocator<int, 10>;

template class Allocator<std::\_Rb\_tree\_node<int>, 10>;

template Allocator<std::\_Rb\_tree\_node<int>, 10>::Allocator<int, 10>(Allocator<int, 10> const&);

template class Allocator<std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>;

template Allocator<std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>::Allocator<int, 10>(Allocator<int, 10> const&);

template Allocator<std::\_List\_node<int>, 10>::Allocator<int, 10>(Allocator<int, 10> const&);

template class Allocator<std::\_List\_node<int>, 10>;

template class Allocator<QueueEl<Square<int> >, 10>;

template class Allocator<std::\_Sp\_counted\_ptr\_inplace<QueueEl<Square<int>>, Allocator<QueueEl<Square<int>>, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>;

//template Allocator<std::\_Sp\_counted\_ptr\_inplace<QueueEl<Square<int>>, Allocator<QueueEl<Square<int>>, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>::Allocator<QueueEl<Square<int>>(Allocator<QueueEl<Square<int>> const&);

//std::allocator\_traits<Allocator<QueueEl<Square<int> >, 10> >::construct(Allocator<QueueEl<Square<int> >, 10>&, QueueEl<Square<int> >\*)’

//‘std::allocator\_traits<Allocator<QueueEl<Square<int> >, 10> >::\_S\_construct(Allocator<QueueEl<Square<int> >, 10>&, QueueEl<Square<int> >\*&)’

template Allocator<std::\_Sp\_counted\_ptr\_inplace<QueueEl<Square<int> >, Allocator<QueueEl<Square<int> >, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>::Allocator<QueueEl<Square<int> >, 10>(Allocator<QueueEl<Square<int> >, 10> const&);

//std::allocator\_traits<Allocator<QueueEl<Square<int> >, 10> >::construct(Allocator<QueueEl<Square<int> >, 10>&, QueueEl<Square<int> >\*)

//std::allocator\_traits<Allocator<QueueEl<Square<int> >, 10> >::\_S\_construct(Allocator<QueueEl<Square<int> >, 10>&, QueueEl<Square<int> >\*&)’

// /usr/bin/ld: /tmp/ccw0vg9M.o: в функции «main»:

// /home/steep/educat/tries/alloc.cpp:98: неопределённая ссылка на «Allocator<int, 10>::Allocator()»

// /usr/bin/ld: /home/steep/educat/tries/alloc.cpp:98: неопределённая ссылка на «Allocator<int, 10>::~Allocator()»

// /usr/bin/ld: /home/steep/educat/tries/alloc.cpp:98: неопределённая ссылка на «Allocator<int, 10>::~Allocator()»

// /usr/bin/ld: /tmp/ccw0vg9M.o: в функции «std::set<int, std::less<int>, Allocator<int, 10> >::set(Allocator<int, 10> const&)»:

// /usr/include/c++/10/bits/stl\_set.h:252: неопределённая ссылка на «Allocator<int, 10>::~Allocator()»

// /usr/bin/ld: /usr/include/c++/10/bits/stl\_set.h:252: неопределённая ссылка на «Allocator<int, 10>::~Allocator()»

// /usr/bin/ld: /tmp/ccw0vg9M.o: в функции «std::\_Rb\_tree<int, int, std::\_Identity<int>, std::less<int>, Allocator<int, 10> >::\_Rb\_tree\_impl<std::less<int>, true>::~\_Rb\_tree\_impl()»:

// /usr/include/c++/10/bits/stl\_tree.h:677: неопределённая ссылка на «Allocator<std::\_Rb\_tree\_node<int>, 10>::~Allocator()»

// /usr/bin/ld: /tmp/ccw0vg9M.o: в функции «std::\_Rb\_tree<int, int, std::\_Identity<int>, std::less<int>, Allocator<int, 10> >::\_Rb\_tree(Allocator<int, 10> const&)»:

// /usr/include/c++/10/bits/stl\_tree.h:951: неопределённая ссылка на «Allocator<std::\_Rb\_tree\_node<int>, 10>::Allocator<int, 10>(Allocator<int, 10> const&)»

// /usr/bin/ld: /usr/include/c++/10/bits/stl\_tree.h:951: неопределённая ссылка на «Allocator<std::\_Rb\_tree\_node<int>, 10>::~Allocator()»

// /usr/bin/ld: /tmp/ccw0vg9M.o: в функции «std::allocator\_traits<Allocator<std::\_Rb\_tree\_node<int>, 10> >::deallocate(Allocator<std::\_Rb\_tree\_node<int>, 10>&, std::\_Rb\_tree\_node<int>\*, unsigned long)»:

// /usr/include/c++/10/bits/alloc\_traits.h:341: неопределённая ссылка на «Allocator<std::\_Rb\_tree\_node<int>, 10>::deallocate(std::\_Rb\_tree\_node<int>\*, int)»

// /usr/bin/ld: /tmp/ccw0vg9M.o: в функции «std::allocator\_traits<Allocator<std::\_Rb\_tree\_node<int>, 10> >::allocate(Allocator<std::\_Rb\_tree\_node<int>, 10>&, unsigned long)»:

// /usr/include/c++/10/bits/alloc\_traits.h:314: неопределённая ссылка на «Allocator<std::\_Rb\_tree\_node<int>, 10>::allocate(int)»

// collect2: error: ld returned 1 exit status

// /usr/bin/ld: /tmp/cc7BCtvJ.o: в функции «std::\_\_shared\_count<(\_\_gnu\_cxx::\_Lock\_policy)2>::\_\_shared\_count<int, Allocator<int, 10>>(int\*&, std::\_Sp\_alloc\_shared\_tag<Allocator<int, 10> >)»:

// /usr/include/c++/10/bits/shared\_ptr\_base.h:679: неопределённая ссылка на «Allocator<std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>::Allocator<int, 10>(Allocator<int, 10> const&)»

// /usr/bin/ld: /usr/include/c++/10/bits/shared\_ptr\_base.h:679: неопределённая ссылка на «Allocator<std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>::~Allocator()»

// /usr/bin/ld: /usr/include/c++/10/bits/shared\_ptr\_base.h:679: неопределённая ссылка на «Allocator<std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>::~Allocator()»

// /usr/bin/ld: /tmp/cc7BCtvJ.o: в функции «std::allocator\_traits<Allocator<std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10> >::allocate(Allocator<std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>&, unsigned long)»:

// /usr/include/c++/10/bits/alloc\_traits.h:314: неопределённая ссылка на «Allocator<std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>::allocate(int)»

// /usr/bin/ld: /tmp/cc7BCtvJ.o: в функции «std::allocator\_traits<Allocator<std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10> >::deallocate(Allocator<std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>&, std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>\*, unsigned long)»:

// /usr/include/c++/10/bits/alloc\_traits.h:341: неопределённая ссылка на «Allocator<std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>::deallocate(std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>\*, int)»

// /usr/bin/ld: /tmp/cc7BCtvJ.o: в функции «std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>::\_M\_destroy()»:

// /usr/include/c++/10/bits/shared\_ptr\_base.h:567: неопределённая ссылка на «Allocator<std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>::Allocator<int, 10>(Allocator<int, 10> const&)»

// /usr/bin/ld: /usr/include/c++/10/bits/shared\_ptr\_base.h:567: неопределённая ссылка на «Allocator<std::\_Sp\_counted\_ptr\_inplace<int, Allocator<int, 10>, (\_\_gnu\_cxx::\_Lock\_policy)2>, 10>::~Allocator()»

// collect2: error: ld returned 1 exit status

// /usr/bin/ld: /tmp/ccX9noX7.o: в функции «std::\_\_cxx11::list<int, Allocator<int, 10> >::list(Allocator<int, 10> const&)»:

// /usr/include/c++/10/bits/stl\_list.h:684: неопределённая ссылка на «Allocator<std::\_List\_node<int>, 10>::Allocator<int, 10>(Allocator<int, 10> const&)»

// /usr/bin/ld: /usr/include/c++/10/bits/stl\_list.h:684: неопределённая ссылка на «Allocator<std::\_List\_node<int>, 10>::~Allocator()»

// /usr/bin/ld: /tmp/ccX9noX7.o: в функции «std::\_\_cxx11::\_List\_base<int, Allocator<int, 10> >::\_List\_impl::~\_List\_impl()»:

// /usr/include/c++/10/bits/stl\_list.h:374: неопределённая ссылка на «Allocator<std::\_List\_node<int>, 10>::~Allocator()»

// /usr/bin/ld: /tmp/ccX9noX7.o: в функции «std::allocator\_traits<Allocator<std::\_List\_node<int>, 10> >::deallocate(Allocator<std::\_List\_node<int>, 10>&, std::\_List\_node<int>\*, unsigned long)»:

// /usr/include/c++/10/bits/alloc\_traits.h:341: неопределённая ссылка на «Allocator<std::\_List\_node<int>, 10>::deallocate(std::\_List\_node<int>\*, int)»

// collect2: error: ld returned 1 exit status

#pragma once

template class QueueEl<Square<int>>;

#pragma once

#include "../header/Allocator.hpp"

template class Queue<Square<int>, Allocator<QueueEl<Square<int>>, 10>>;

template class Queue<Square<int>, std::allocator<QueueEl<Square<int> > > >;

//template class Queue<Square<int>, Allocator<QueueEl<Square<int> >, 10> >::iterator;

#pragma once

template read\_return\_t get\_value (Square<int>& d);

#pragma once

template class Square<int>;

template std::istream& operator>> <int>(std::istream&, Square<int>&);

template std::ostream& operator<< <int>(std::ostream&, const Square<int>&);

template double distance<int>(std::pair<int,int>, std::pair<int,int>);

//template void make\_2\_more\_verteces(Square<int>& t, double side\_length);

#include "../header/Allocator.hpp"

#include <type\_traits>

#include <typeinfo>

#include <string>

Block::Block(size\_t size, void\* memory)

: size(size), memory(memory) {}

template <typename T, int Size>

Allocator<T,Size>::~Allocator(){

// std::cout <<"Allocator destroyed with size = " << Size << " and type = " << sizeof(T) <<'\n';

}

template <typename T, int Size>

Allocator<T, Size>::Allocator() :

storage(new char[Size\*1024]), free\_elements(new std::stack<Block>) {

// std::cout <<"Allocator() used with Size = " << Size << " and type = " << sizeof(T) <<'\n';

if(storage == nullptr) throw std::runtime\_error("can't allocate storage");

if(free\_elements == nullptr) throw std::bad\_alloc();

free\_elements->push(Block(Size\*1024, static\_cast<void\*>(storage.get())));

}

template<class T, int Size> template<class U, int Size2> Allocator<T, Size>::Allocator(const Allocator<U, Size2>& all) :

storage(all.storage), free\_elements(all.free\_elements) {

// std::cout << "Allocator<T,Size>(Allocator<U,Size2>) used with Size = " << Size << " and size of type = " << sizeof(T) <<'\n';

// std::cout << "and Size2 = " <<Size2 <<" and size of type U = " << sizeof(U) <<'\n';

if(storage == nullptr) throw std::runtime\_error("can't allocate storage");

if(free\_elements == nullptr) throw std::bad\_alloc();

}

template <typename T, int Size>

T\* Allocator<T, Size>::allocate(int n) {

// std::cout << "allocate(" << n << ") used with Size = " << Size << " and size of type = " <<sizeof(T) <<'\n';

if(n != 1 || free\_elements->empty()) return nullptr;

char\* allocated\_memory = nullptr;

bool found = false;

while(!found) {

Block block = free\_elements->top();

free\_elements->pop();

if(block.size > sizeof(T)) {

allocated\_memory = static\_cast<char\*>(block.memory);

free\_elements->push(

Block(block.size-sizeof(T),

static\_cast<char\*>(block.memory) + sizeof(T)));

found = true;

} else if (block.size == sizeof(T)) {

allocated\_memory = static\_cast<char\*>(block.memory);

found = true;

} else if (free\_elements->size() == 0) {

throw std::bad\_alloc();

}

}

return reinterpret\_cast<T\*>(allocated\_memory);

}

template <typename T, int Size>

void Allocator<T, Size>::deallocate(T\* p, int n) {

// std::cout << "deallocate(" << p << ") used with Size = " << Size << " and size of type = " <<sizeof(T) <<'\n';

free\_elements->push(Block(n, static\_cast<void\*>(p)));

}

template<class T1, int Sz1, class T2, int Sz2>

bool operator==(const Allocator<T1, Sz1>& lhs, const Allocator<T2, Sz2>& rhs) {

// std::cout << "operator==(" << sizeof(T1) <<' '<<Sz1 << ' ' <<sizeof(T2) << ' ' << Sz2 << ")\n";

return ((Sz1 == Sz2) && (rhs.storage == lhs.storage)

&& (rhs.free\_elements == lhs.free\_elements));

}

#include "../template/Allocator.tpp"

#include "../header/Queue.h"

#include "../header/QueueEl.h"

#include "../header/iterator.h"

#include <memory>

template <typename T, typename Alloc> Queue<T, Alloc>::iterator::iterator()

: prev\_el(std::shared\_ptr<QueueEl<T>>(nullptr)) {}

template <typename T, typename Alloc>

typename Queue<T, Alloc>::iterator& Queue<T, Alloc>::iterator::operator++() {

if(!prev\_el.expired())

prev\_el = prev\_el.lock()->next;

else {

throw std::bad\_weak\_ptr();

}

return \*this;

}

template <typename T, typename Alloc>

Queue<T, Alloc>::iterator::iterator(std::shared\_ptr<QueueEl<T>> el)

: prev\_el(el) {}

template <typename T, typename Alloc>

bool Queue<T, Alloc>::iterator::operator!=(Queue<T, Alloc>::iterator iter) {

if(iter.prev\_el.lock() == nullptr || prev\_el.lock() == nullptr)

throw std::bad\_weak\_ptr();

return (prev\_el.lock()->next != iter.prev\_el.lock()->next);

} //should I compare queues?

template <typename T, typename Alloc>

T& Queue<T, Alloc>::iterator::operator\*() {

if(!prev\_el.expired()) {

std::shared\_ptr<QueueEl<T>> prev = prev\_el.lock();

if(prev->next == NULL)

throw std::out\_of\_range("iterator out of range");

else return prev->next->val;

} else throw std::bad\_weak\_ptr();

}

template <typename T, typename Alloc>

bool Queue<T, Alloc>::iterator::operator==(iterator iter) {

return !(\*this != iter);

}

template <typename T, typename Alloc>

bool Queue<T, Alloc>::iterator::operator==(std::nullptr\_t) {

return (prev\_el.lock() == nullptr);

}

#include "../template/Queue.tpp"

#include <iostream>

#include "../header/Queue.h"

#include "../header/Square.h"

#include "../header/Read\_input.hpp"

#include "../header/iterator.h"

#include "../header/QueueEl.h"

#include "../header/Allocator.hpp"

#include <cstdio>

#include <set>

#include <string>

#include <algorithm>

int main(int argc, char \*argv[]) {

std::string help\_message = "You can use\n\

--put square: p [(point) 3 times]\n\

--delete by figure number: d (number of figure)\n\

--print container: pr\n\

--print number of figures, which area is less then given: ar (area)\n\

--exit\n";

Queue<Square<int>, Allocator<QueueEl<Square<int>>, 10>> queue;

char ch(' ');

char command[20];

std::set<std::string> valid\_commands = {"p", "pr", "d", "exit", "ar"};

std::cout << help\_message;

do {

bool valid\_input = false;

do{

read\_return\_t answer = get\_command(valid\_commands, command);

switch(answer) {

case END\_OF\_FILE: return 0;

case END\_OF\_LINE: continue;

case VALID\_INPUT: valid\_input = true; break;

case INVALID\_INPUT:

do ch=getchar(); while((ch != EOF) && (ch != '\n'));

std::cout << "wrong input" << std::endl;

if(ch == EOF) return 0;

else break;

}

} while(!valid\_input);

std::string&& command\_string = static\_cast<std::string>(command);

if(command\_string == "exit") return 0;

if(command\_string == "p") {

Square<int> square;

if(get\_value<Square<int>>(square) != VALID\_INPUT)

std::cout << "wrong input\n";

else {

queue.push(square);

}

} else if(command\_string == "pr") {

std::for\_each(queue.begin(), queue.end(), [](auto&& queue\_el){

std::cout << queue\_el << std::endl;

});

// for(auto queue\_el : queue) {

// std::cout << queue\_el << std::endl;

// }

} else if(command\_string == "d") {

unsigned int input\_figure\_number = 0;

if(get\_value<unsigned int>(input\_figure\_number) != VALID\_INPUT ||

input\_figure\_number >= queue.size) { //if there would be EOF

std::cout << "wrong input";

} else {

bool all\_done = false;

auto i = queue.begin();

while(!all\_done) {

if(input\_figure\_number == 0) {

all\_done = true;

queue.erase(i);

} else {

++i;

--input\_figure\_number;

}

}

}

} else if(command\_string == "ar"){

unsigned int area = 0;

if(get\_value<unsigned int>(area) != VALID\_INPUT)

std::cout << "wrong input\n";

else

std::cout << std::count\_if(queue.begin(), queue.end(),

[area](const Square<int>& r)

{return r.area() < area;})

<< std::endl;

}

do ch = getchar(); while((ch != '\n') && (ch != EOF));

if(ch == EOF) return 0;

} while(true);

return 0;

}

#include "../header/Square.h"

#include "../header/Queue.h"

#include "../header/QueueEl.h"

#include "../header/iterator.h"

#include "../header/Allocator.hpp"

#include <memory>

//""

/\*

‘std::shared\_ptr<QueueEl<Square<int> > >::reset(Square<int>\*&, Queue<T, Alloc>::Queue<Square<int>, std::allocator<Square<int> > >::<lambda(Square<int>\*)>&, std::allocator<Square<int> >&)’

13 | zero\_el.reset(pointer, deleter, alloc);

| ~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~

home/steep/educat/instit/oop/LR6/source/Queue.cpp: In instantiation of ‘Queue<T, Alloc>::Queue() [with T = Square<int>; Alloc = std::allocator<Square<int> >]’:

/home/steep/educat/instit/oop/LR6/source/../header/../template/Queue.tpp:2:16: required from here

/home/steep/educat/instit/oop/LR6/source/Queue.cpp:16:15: error: no matching function for call to ‘std::shared\_ptr<QueueEl<Square<int> > >::reset(Square<int>\*&, Queue<T, Alloc>::Queue<Square<int>, std::allocator<Square<int> > >::<lambda(QueueEl<Square<int> >\*)>&, std::allocator<Square<int> >&)’

16 | zero\_el.reset(pointer, deleter, alloc);

| ~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~

\*/

template <typename T, typename Alloc> Queue<T, Alloc>::Queue()

: zero\_el(), end\_el(), size(0), alloc() {

zero\_el = std::allocate\_shared<QueueEl<T>>(alloc);

end\_el = zero\_el;

}

template <typename T, typename Alloc> Queue<T, Alloc>::~Queue() {

end\_el = nullptr;

zero\_el = nullptr;

}

template <typename T, typename Alloc> typename Queue<T, Alloc>::iterator Queue<T, Alloc>::begin() {

return iterator(zero\_el);

}

template <typename T, typename Alloc> typename Queue<T, Alloc>::iterator Queue<T, Alloc>::end() {

return iterator(end\_el);

}

template <typename T, typename Alloc> void Queue<T, Alloc>::insert(const T& t, Queue<T, Alloc>::iterator i) { //inserts before

if(i == nullptr) throw std::invalid\_argument("null iterator");

auto cur = i.prev\_el.lock();

cur->next =

std::allocate\_shared<QueueEl<T>>(alloc, t, cur->next);

if(cur == end\_el) end\_el = cur->next;

++size;

}

template <typename T, typename Alloc> void Queue<T, Alloc>::erase(iterator i) {

if(i == nullptr) throw std::invalid\_argument("null iterator");

auto cur = i.prev\_el.lock();

if(cur->next == end\_el) end\_el = cur;

cur->next = cur->next->next;

--size;

}

template <typename T, typename Alloc> T Queue<T, Alloc>::pop() {

if(zero\_el->next == nullptr) throw std::out\_of\_range("empty queue");

T element = \*(begin());

erase(begin());

return element;

}

template <typename T, typename Alloc> void Queue<T, Alloc>::push(const T& el) {

insert(el, end());

}

template <typename T, typename Alloc> T Queue<T, Alloc>::top() /\*const\*/{

if(zero\_el->next == nullptr) throw std::out\_of\_range("empty queue");

T el = \*(begin());

return el;

}

template <typename T, typename Alloc> std::ostream& operator<<(std::ostream& cout, Queue<T, Alloc>& queue) {

for(auto i = queue.begin(); i != queue.end(); ++i) {

cout << \*i << '\n';

}

return cout;

}

#include "../template/Queue.tpp"

#include "../header/QueueEl.h"

#include "../header/Square.h"

#include <memory>

template<typename T>

QueueEl<T>::QueueEl(const T& val, const std::shared\_ptr<QueueEl>& next)

: val(val), next(next){}

template<typename T>

QueueEl<T>::QueueEl() :

val(T()), next(nullptr) {}

#include "../template/QueueEl.tpp"

#include <iostream>

#include <string>

#include <cstdio>

#include <set>

#include "../header/Read\_input.hpp"

#include "../header/Square.h"

#include "../header/Queue.h"

read\_return\_t get\_command(std::set<std::string>& valid\_commands, char\* command)

{

char ch(' ');

while((ch == '\t') || (ch == ' ')) {

std::cin >> ch;

if(std::cin.eof()) exit(0);

if(ch == '\n') ch = ' ';

}

std::cin.unget();

std::cin >> command;

if(std::cin.fail()) {

std::cin.clear();

return INVALID\_INPUT;

}

if(valid\_commands.count(command)) return VALID\_INPUT;

else return INVALID\_INPUT;

}

template <typename T>

read\_return\_t get\_value(T& d)

{

char ch(' ');

while((ch == '\t') || (ch == ' ')) {

std::cin >> ch;

if(std::cin.eof()) exit(0);

if(ch == '\n') ch = ' ';

}

std::cin.unget();

std::cin >> d;

if(std::cin.fail()) {

std::cin.clear();

return INVALID\_INPUT;

}

return VALID\_INPUT;

}

template <>

read\_return\_t get\_value (unsigned int& d) {

char ch(' ');

while((ch == '\t') || (ch == ' ')) {

std::cin >> ch;

if(std::cin.eof()) exit(0);

if(ch == '\n') ch = ' ';

}

std::cin.unget();

if(ch == '-') return INVALID\_INPUT;

std::cin >> d;

if(std::cin.fail()) {

std::cin.clear();

return INVALID\_INPUT;

}

return VALID\_INPUT;

}

template <>

read\_return\_t get\_value (unsigned long long& d) {

char ch(' ');

while((ch == '\t') || (ch == ' ')) {

std::cin >> ch;

if(std::cin.eof()) exit(0);

if(ch == '\n') ch = ' ';

}

std::cin.unget();

if(ch == '-') return INVALID\_INPUT;

std::cin >> d;

if(std::cin.fail()) {

std::cin.clear();

return INVALID\_INPUT;

}

return VALID\_INPUT;

}

#include "../template/Read\_input.tpp"

#include <vector>

#include <utility>

#include <iostream>

#include <cmath>

#include "../header/Square.h"

template <typename T>

Square<T>::Square(const Square<T>& obj)

: verteces(obj.verteces) {}

template <typename T>

Square<T>::Square()

: verteces(std::vector<std::pair<T, T>>(NUM\_OF\_VERTECES)) {}

template <typename T>

Square<T>& Square<T>::operator=(const Square<T>& obj) noexcept {

for(int i = 0; i < NUM\_OF\_VERTECES; ++i) {

verteces[i] = obj.verteces[i];

}

return \*this;

}

template <typename T>

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

bool result = true;

for(int i = 0; i < NUM\_OF\_VERTECES; ++i) {

if(verteces[i] != obj.verteces[i]) {

result = false;

break;

}

}

return result;

}

template <typename T>

std::ostream& operator<<(std::ostream& cout, const Square<T>& obj) {

cout << "square\nsides' length:\n";

for(int i = 0; i < NUM\_OF\_VERTECES; ++i) {

auto v1 = obj.verteces[i];

auto v2 = obj.verteces[(i + 1) % NUM\_OF\_VERTECES];

cout << "side " << i << ": " << sqrt(pow(v1.first - v2.first, 2) +

pow(v1.second - v2.second, 2)) << '\n';

}

cout << "coordinates: ";

obj.coordinates();

return cout;

}

template <typename T>

double distance(std::pair<T, T> o1, std::pair<T, T> o2) {

return sqrt(pow(o1.first - o2.first, 2)+pow(o1.second - o2.second, 2));

}

//«std::istream& operator>><int>(std::istream&, Square<int>&)»

template <typename T>

std::istream& operator>>(std::istream& cin, Square<T>& t) {

enum {

FIRST\_AXIS = 0,

SECOND\_AXIS = 1,

};

char ch(' ');

Square<T> copy = t;

for(int i = 0, cur\_axis = 0; i < ( NUM\_OF\_VERTECES - 1 ) \* 2;

++i, cur\_axis = (cur\_axis + 1)%2) {

while((ch == '\t') || (ch == ' ') || (ch == '\n')) {

cin >> ch;

if(cin.eof()) {

t = copy;

return cin;

}

}

cin.unget();

ch = ' ';

if(cur\_axis == FIRST\_AXIS)

cin >> t.verteces[i/2].first;

else //(cur\_axis == SECOND\_AXIS)

cin >> t.verteces[i/2].second;

if(cin.fail()) {

t = copy;

return cin;

}

}

if((distance(t.verteces[0], t.verteces[1]) !=

distance(t.verteces[1], t.verteces[2])) ||

(distance(t.verteces[0], t.verteces[2])

!= distance(t.verteces[0], t.verteces[1])\*sqrt(2))){

t = copy;

cin.setstate(std::ios\_base::failbit);

return cin;

}

t.verteces[3].first = t.verteces[0].first - t.verteces[1].first

+ t.verteces[2].first;

t.verteces[3].second = t.verteces[0].second - t.verteces[1].second

+ t.verteces[2].second;

return cin;

}

// template <typename T>

// void make\_2\_more\_verteces(Square<T>& t, double side\_length) {

// double \_distance = distance(t.verteces[0], t.verteces[2]);

// std::pair<double, double> center(

// ( t.verteces[0].first + t.verteces[2].first) / 2.0,

// ( t.verteces[0].second + t.verteces[2].second ) / 2.0);

// double alpha = std::atan(( t.verteces[2].second - t.verteces[0].second ) /

// ( t.verteces[2].first - t.verteces[0].first ));

// double side = sqrt(pow( side\_length, 2 )

// - pow( \_distance / 2.0 , 2 ));

// std::cout << center.first - side \* sin(alpha) << ' ' << center.second + side \* cos(alpha) << '+' <<

// center.first + side \* sin(alpha) << ' ' <<

// center.second - side \* cos(alpha);

// t.verteces[1] = std::make\_pair<T, T>(

// static\_cast<T>( center.first - side \* sin(alpha) ),

// static\_cast<T>( center.second + side \* cos(alpha) ));

// t.verteces[3] = std::make\_pair<T, T>(

// static\_cast<T>(center.first + side \* sin(alpha)),

// static\_cast<T>(center.second - side \* cos(alpha)));

// }

// //std::istream& operator>><int>(std::istream&, Square<int>&)

// template <typename T>

// std::istream& operator>> (std::istream& cin, Square<T>& t) {

// enum {

// FIRST\_AXIS = 0,

// SECOND\_AXIS = 1,

// };

// char ch(' ');

// Square<T> copy = t;

// for(int i = 0, cur\_axis = 0; i < ( NUM\_OF\_VERTECES / 2 ) \* 2;

// ++i, cur\_axis = (cur\_axis + 1)%2) {

// while((ch == '\t') || (ch == ' ') || (ch == '\n')) {

// cin >> ch;

// if(cin.eof()) {

// t = copy;

// return cin;

// }

// }

// cin.unget();

// ch = ' ';

// if(cur\_axis == FIRST\_AXIS)

// cin >> t.verteces[ i / 2 ].first; //0 & 2 points

// else //(cur\_axis == SECOND\_AXIS)

// cin >> t.verteces[ i / 2 ].second;

// if(cin.fail()) {

// t = copy;

// return cin;

// }

// }

// t.verteces[2] = t.verteces[1];

// double side\_length;

// cin >> side\_length;

// if(cin.fail() ||

// distance(t.verteces[0], t.verteces[1]) > 2 \* side\_length) {

// t = copy;

// return cin;

// }

// make\_2\_more\_verteces(t, side\_length);

// return cin;

// }

template <typename T>

std::pair<double, double> Square<T>::center() const noexcept{

std::pair<double, double> \_center(0, 0);

for(int i = 0; i < NUM\_OF\_VERTECES; ++i) {

\_center.first+=verteces[i].first;

\_center.second+=verteces[i].second;

\_center.first/=static\_cast<double>(NUM\_OF\_VERTECES);

}

\_center.second/=static\_cast<double>(NUM\_OF\_VERTECES);

return \_center;

}

template <typename T> void Square<T>::coordinates() const noexcept {

for(int i = 0; i < NUM\_OF\_VERTECES; ++i)

std::cout << '(' << verteces[i].first << ", "

<< verteces[i].second << ')';

}

template <typename T> double Square<T>::area() const noexcept {

double area = 0;

for(int i = 0; i < NUM\_OF\_VERTECES; ++i) {

area +=

verteces[ i ].first \* verteces[( i + 1 ) % NUM\_OF\_VERTECES].second

- verteces[( i + 1 ) % NUM\_OF\_VERTECES].first \* verteces[ i ].second;

}

area /= 2.0;

return area < 0 ? -area : area;

}

#include "../template/Square.tpp"

ЛИТЕРАТУРА

1. Б. Страуструп "Язык программирования С++", четвёртое издание, 2013.