**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 5**

Тема: Основы работы с коллекциями: итераторы

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)5

1. Постановка задачи

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 (правильные многоугольники),

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

Например:

template <class T>

struct Rhomb {

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

vertex\_t a,b,c,d;

};

Создать шаблон динамической коллекциu, 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 - фигуры;

3. Реализовать forward\_iterator по коллекции;

4. Коллекция должны возвращать итераторы begin() и end();

5. Коллекция должна содержать метод вставки на позицию итератора insert(iterator);

6. Коллекция должна содержать метод удаления из позиции итератора erase(iterator);

7. При выполнении недопустимых операций (например выход за границы коллекции или удаление несуществующего элемента) необходимо генерировать исключения;

8. Итератор должен быть совместим со стандартными алгоритмами (например, std::count\_if)

9. Коллекция должна содержать метод доступа:

стек – pop, push, top;

очередь – pop, push, top;

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

10. Реализовать программу, которая:

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

позволяет удалять элемент из коллекции по номеру элемента;

выводит на экран введенные фигуры c помощью std::for\_each;

выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count\_if).

variant: 5. Ромб, Стек

1. Описание программы

1) в main происходит ввод данных

2) стек, реализованный с помощью умных указателей: элементы с помощью std::shared\_ptr, итератор с помощью std::weak\_ptr

3) итератор является совместимым со стандартными алгритмами

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 rhomb: 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

rhomb

sides' length:

side 0: 5

side 1: 5

side 2: 5

side 3: 5

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

rhomb

sides' length:

side 0: 2

side 1: 2

side 2: 2

side 3: 2

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

rhomb

sides' length:

side 0: 1

side 1: 1

side 2: 1

side 3: 1

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

rhomb

sides' length:

side 0: 1

side 1: 1

side 2: 1

side 3: 1

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

2

rhomb

sides' length:

side 0: 5

side 1: 5

side 2: 5

side 3: 5

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

rhomb

sides' length:

side 0: 2

side 1: 2

side 2: 2

side 3: 2

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

rhomb

sides' length:

side 0: 1

side 1: 1

side 2: 1

side 3: 1

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

1

rhomb

sides' length:

side 0: 2

side 1: 2

side 2: 2

side 3: 2

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

rhomb

sides' length:

side 0: 1

side 1: 1

side 2: 1

side 3: 1

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

1

You can use

--put rhomb: 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

rhomb

sides' length:

side 0: 2.23607

side 1: 2.23607

side 2: 2.23607

side 3: 2.23607

coordinates: (1, -4)(0, -2)(2, -3)(3, -5)

rhomb

sides' length:

side 0: 2.23607

side 1: 2.23607

side 2: 2.23607

side 3: 2.23607

coordinates: (0, 0)(2, 1)(4, 0)(2, -1)

rhomb

sides' length:

side 0: 2.23607

side 1: 2.23607

side 2: 2.23607

side 3: 2.23607

coordinates: (0, 0)(1, 2)(2, 0)(1, -2)

0

rhomb

sides' length:

side 0: 2.23607

side 1: 2.23607

side 2: 2.23607

side 3: 2.23607

coordinates: (1, -4)(0, -2)(2, -3)(3, -5)

rhomb

sides' length:

side 0: 2.23607

side 1: 2.23607

side 2: 2.23607

side 3: 2.23607

coordinates: (0, 0)(2, 1)(4, 0)(2, -1)

0

1. Листинг программы

#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\_

#pragma once

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

#ifndef SQUARE\_H\_

#define SQUARE\_H\_

#include <vector>

#include <utility>

#include <iostream>

#include <cmath>

#define NUM\_OF\_VERTECES 4

template <typename T>

class Rhomb;

template <typename T>

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

template <typename T>

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

template <typename T>

class Rhomb {

public:

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

Rhomb(const Rhomb<T>& t);

Rhomb();

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

void coordinates() const noexcept;

double area() const noexcept;

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

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

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

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

};

//#include "../header/Rhomb.tpp"

#endif // SQUARE\_H\_

#pragma once

template class Rhomb<int>;

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

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

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

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

#ifndef STACK\_H

#define STACK\_H

#include <memory>

#include "../header/StackEl.h"

#include "../header/Rhomb.h"

template <typename T>

class Stack {

public:

//null - > begin

class iterator;

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

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

unsigned int size;

Stack();

~Stack();

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&, Stack<Y>&);

};

#endif // STACK\_H

#include "../header/Stack.tpp"

#pragma once

template class Stack<Rhomb<int>>;

#ifndef LISTEL\_H\_

#define LISTEL\_H\_

#include <memory>

#include "../header/Rhomb.h"

template<typename T>

class StackEl {

public:

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

T val;

std::shared\_ptr<StackEl> next;

};

#endif // LISTEL\_H\_

#pragma once

template class StackEl<Rhomb<int>>;

#ifndef ITERATOR\_H\_

#define ITERATOR\_H\_

#include "../header/StackEl.h"

#include "../header/Stack.h"

#include "../header/Rhomb.h"

#include <memory>

template<typename T>

class Stack<T>::iterator : public std::iterator<std::forward\_iterator\_tag, T> {

public:

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

iterator();

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

iterator& operator++();

T& operator\*();

bool operator!=(iterator);

bool operator==(iterator);

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

};

template class Stack<Rhomb<int>>::iterator;

#endif // ITERATOR\_H\_

#include <iostream>

#include <string>

#include <cstdio>

#include <set>

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

#include "../header/Rhomb.h"

#include "../header/Stack.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 "../header/Read\_input.tpp"

#include <vector>

#include <utility>

#include <iostream>

#include <cmath>

#include "../header/Rhomb.h"

template <typename T>

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

: verteces(obj.verteces) {}

template <typename T>

Rhomb<T>::Rhomb()

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

template <typename T>

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

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

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

}

return \*this;

}

template <typename T>

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

cout << "rhomb\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&, Rhomb<int>&)»

template <typename T>

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

enum {

FIRST\_AXIS = 0,

SECOND\_AXIS = 1,

};

char ch(' ');

Rhomb<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])) {

std::cout << distance(t.verteces[0], t.verteces[1]) << ' ' <<

distance(t.verteces[1], t.verteces[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(Rhomb<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&, Rhomb<int>&)

// template <typename T>

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

// enum {

// FIRST\_AXIS = 0,

// SECOND\_AXIS = 1,

// };

// char ch(' ');

// Rhomb<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> Rhomb<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 Rhomb<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 Rhomb<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 "../header/Rhomb.tpp"

#include "../header/Rhomb.h"

#include "../header/Stack.h"

#include "../header/StackEl.h"

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

//""

template <typename T> Stack<T>::Stack()

: zero\_el(new StackEl<T>(T(), nullptr)), end\_el(zero\_el), size(0){}

template <typename T> Stack<T>::~Stack() {

zero\_el = nullptr;

end\_el = nullptr;

}

template <typename T> typename Stack<T>::iterator Stack<T>::begin() {

return iterator(zero\_el);

}

template <typename T> typename Stack<T>::iterator Stack<T>::end() {

return iterator(end\_el);

}

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

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

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

cur->next =

std::make\_shared<StackEl<T>>(t, cur->next);

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

++size;

}

template <typename T> void Stack<T>::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> T Stack<T>::pop() {

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

T element = \*(begin());

erase(begin());

return element;

}

template <typename T> void Stack<T>::push(const T& el) {

insert(el, begin());

}

template <typename T> T Stack<T>::top() /\*const\*/{

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

T el = \*(begin());

return el;

}

template <typename T> std::ostream& operator<<(std::ostream& cout, Stack<T>& stack) {

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

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

}

return cout;

}

#include "../header/StackEl.h"

#include "../header/Rhomb.h"

template<typename T>

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

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

#include "../header/StackEl.tpp"

#include "../header/Stack.h"

#include "../header/StackEl.h"

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

#include <memory>

template <typename T> Stack<T>::iterator::iterator()

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

template <typename T>

typename Stack<T>::iterator& Stack<T>::iterator::operator++() {

if(!prev\_el.expired())

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

else {

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

}

return \*this;

}

template <typename T> Stack<T>::iterator::iterator(std::shared\_ptr<StackEl<T>> el)

: prev\_el(el) {}

template <typename T> bool Stack<T>::iterator::operator!=(Stack<T>::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 stacks?

template <typename T> T& Stack<T>::iterator::operator\*() {

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

std::shared\_ptr<StackEl<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> bool Stack<T>::iterator::operator==(iterator iter) {

return !(\*this != iter);

}

template <typename T> bool Stack<T>::iterator::operator==(std::nullptr\_t) {

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

}

/\*

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 (правильные многоугольники),

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

Например:

template <class T>

struct Rhomb {

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

vertex\_t a,b,c,d;

};

Создать шаблон динамической коллекциu, 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 - фигуры;

3. Реализовать forward\_iterator по коллекции;

4. Коллекция должны возвращать итераторы begin() и end();

5. Коллекция должна содержать метод вставки на позицию итератора insert(iterator);

6. Коллекция должна содержать метод удаления из позиции итератора erase(iterator);

7. При выполнении недопустимых операций (например выход за границы коллекции или удаление несуществующего элемента) необходимо генерировать исключения;

8. Итератор должен быть совместим со стандартными алгоритмами (например, std::count\_if)

9. Коллекция должна содержать метод доступа:

стек – pop, push, top;

очередь – pop, push, top;

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

10. Реализовать программу, которая:

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

позволяет удалять элемент из коллекции по номеру элемента;

выводит на экран введенные фигуры c помощью std::for\_each;

выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count\_if).

variant: 5. Ромб, Стек

\*/

#include <iostream>

#include "../header/Stack.h"

#include "../header/Rhomb.h"

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

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

#include "../header/StackEl.h"

#include <cstdio>

#include <set>

#include <string>

#include <algorithm>

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

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

--put rhomb: 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";

Stack<Rhomb<int>> stack;

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") {

Rhomb<int> rhomb;

if(get\_value<Rhomb<int>>(rhomb) != VALID\_INPUT)

std::cout << "wrong input";

else {

stack.push(rhomb);

}

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

std::for\_each(stack.begin(), stack.end(),

[](auto stack\_el){

std::cout << stack\_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 >= stack.size) { //if there would be EOF

std::cout << "wrong input";

} else {

bool all\_done = false;

Stack<Rhomb<int>>::iterator i = stack.begin();

while(!all\_done) {

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

all\_done = true;

stack.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";

else

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

[area](const Rhomb<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;

}

1. Вывод

Можно создать совместимые со стандартными алгоритмами контейнеры и итераторы.

ЛИТЕРАТУРА

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