Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа**

**по курсу «Объектно-ориентированное программирование»**

**III Семестр**

**Задание 5  
Вариант 8**

**Основы работы с коллекциями: итераторы**

|  |  |
| --- | --- |
| Студент: | Жерлыгин М.А |
| Группа: | М8О-208Б-18 |
| Преподаватель: | Журалвев А.А. |
| Оценка: |  |
| Дата: |  |
| Подпись: |  |

# 1. Код программы на языке С++

**vertex.h:**

#ifndef VERTEX\_H

#define VERTEX\_H

#include <iostream>

#include <type\_traits>

#include <cmath>

template<class T>

struct vertex {

T x;

T y;

vertex<T>& operator=(vertex<T> A);

};

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, vertex<T> p) {

os << '(' << p.x << ", " << p.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>& vertex<T>::operator=(const vertex<T> A) {

this->x = A.x;

this->y = A.y;

return \*this;

}

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>

vertex<T> operator/=(vertex<T>& A, const double B) {

A.x /= B;

A.y /= B;

return A;

}

template<class T>

double length(vertex<T>& A, vertex<T>& B) {

double res = sqrt( pow(B.x - A.x, 2) + pow(B.y - A.y, 2) );

return res;

}

template<class T>

struct is\_vertex : std::false\_type {};

template<class T>

struct is\_vertex<vertex<T>> : std::true\_type {};

#endif //VERTEX\_H

**octagon.h:**

#ifndef OCTAGON\_H\_

#define OCTAGON\_H\_

#include "vertex.h"

#include <iostream>

#include <type\_traits>

template <class T>

class Octagon {

public:

vertex<T> points[8];

int size = 8;

Octagon<T>() = default;

explicit Octagon<T>(std::istream& is) {

for (auto & point : points) {

is >> point;

}

}

void print(std::ostream& os) {

for(int i = 0; i < 8; ++i) {

os << this->points[i];

if(i != size - 1) os << ", ";

}

os << '\n';

}

double area() {

double result = 0;

for(int i = 0; i < 7; ++i) {

result += (points[i].x \* points[i+1].y) - (points[i+1].x \* points[i].y);

}

result = (result + (points[7].x \* points[0].y) - (points[0].x \* points[7].y))/2;

return std::abs(result);

}

void operator<< (std::ostream& os) {

for(int i = 0; i < 8; ++i) {

os << this->points[i];

if(i != size - 1) os << ", ";

}

}

};

#endif // OCTAGON\_H\_

**stack.h:**

#ifndef STACK\_H\_

#define STACK\_H\_

#include <iterator>

#include <memory>

namespace containers {

template<class T>

class stack {

private:

struct element;

size\_t size = 0;

public:

stack() = default;

class forward\_iterator {

private:

element\* ptr\_;

friend stack;

public:

using value\_type = T;

using reference = T&;

using pointer = T\*;

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

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

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;

};

forward\_iterator begin();

forward\_iterator end();

T& top();

void push(const T& value);

void pop();

void insert\_by\_number(size\_t nubmer, T& value);

void insert\_by\_iterator(forward\_iterator insert\_iterator, T& value);

void delete\_by\_iterator(forward\_iterator delete\_iterator);

void delete\_by\_number(size\_t number);

private:

struct element {

T value;

std::unique\_ptr<element> next = nullptr;

forward\_iterator next\_elem();

};

static std::unique\_ptr<element> insert\_impl(std::unique\_ptr<element> current, const T& value);

std::unique\_ptr<element> first\_ = nullptr;

};

template<class T>

T& stack<T>::top() {

if(size == 0) throw std::logic\_error("Stack empty");

return first\_->value;

}

template<class T>

typename stack<T>::forward\_iterator stack<T>::begin() {

if(first\_ == nullptr) return nullptr;

return forward\_iterator(first\_.get());

}

template<class T>

typename stack<T>::forward\_iterator stack<T>::end() {

return forward\_iterator(nullptr);

}

template<class T>

std::unique\_ptr<typename stack<T>::element> stack<T>::insert\_impl(std::unique\_ptr<stack<T>::element> current, const T& value) {

if(current != nullptr) {

current->next = insert\_impl(std::move(current->next), value);

return current;

}

return std::unique\_ptr<element>(new element{value});

}

template<class T>

void stack<T>::delete\_by\_iterator(containers::stack<T>::forward\_iterator delete\_iterator) {

forward\_iterator begin = this->begin();

forward\_iterator end = this->end();

if(delete\_iterator == end) throw std::logic\_error("End of limit!");

if(delete\_iterator == this->begin()) {

this->pop();

return;

}

while((begin.ptr\_ != nullptr) && (begin.ptr\_->next\_elem() != delete\_iterator)) {

++begin;

}

if(begin.ptr\_ == nullptr) throw std::logic\_error("End of limit!");

begin.ptr\_->next = std::move(delete\_iterator.ptr\_->next);

size--;

}

template<class T>

void stack<T>::delete\_by\_number(size\_t number) {

forward\_iterator iterator = this->begin();

for(size\_t i = 0; i < number; i++) {

if(i == number){

break;

}

++iterator;

}

this->delete\_by\_iterator(iterator);

}

template<class T>

void stack<T>::insert\_by\_iterator(containers::stack<T>::forward\_iterator insert\_iterator, T& value) {

auto temp = std::unique\_ptr<element> (new element{value});

forward\_iterator begin = this->begin();

if (insert\_iterator == this->begin()) {

temp->next = std::move(first\_);

first\_ = std::move(temp);

size++;

return;

}

while((begin.ptr\_ != nullptr) && (begin.ptr\_->next\_elem() != insert\_iterator)) {

++begin;

}

if(begin.ptr\_ == nullptr) throw std::logic\_error("End of limit");

temp->next = std::move(begin.ptr\_->next);

begin.ptr\_->next = std::move(temp);

size++;

}

template<class T>

void stack<T>::insert\_by\_number(size\_t number, T& value) {

forward\_iterator iterator = this->begin();

for(size\_t i = 0; i < number; i++) {

if(i == number) {

break;

}

++iterator;

}

this->insert\_by\_iterator(iterator, value);

}

template<class T>

void stack<T>::push(const T& value) {

first\_ = insert\_impl(std::move(first\_), value);

//last\_ = std::move(first\_);

size++;

}

template<class T>

void stack<T>::pop() {

if(size == 0) throw std::logic\_error ("This stack is empty!");

first\_ = std::move(first\_->next);

size--;

}

template<class T>

typename stack<T>::forward\_iterator stack<T>::element::next\_elem() {

return forward\_iterator(this->next.get());

}

template<class T>

stack<T>::forward\_iterator::forward\_iterator(containers::stack<T>::element\* ptr): ptr\_{ptr} {}

template<class T>

typename stack<T>::forward\_iterator& stack<T>::forward\_iterator::operator++() {

if(ptr\_ == nullptr) throw std::logic\_error("End of limit");

\*this = ptr\_->next\_elem();

return \*this;

}

template<class T>

typename stack<T>::forward\_iterator& stack<T>::forward\_iterator::operator++(int) {

forward\_iterator old = \*this;

++\*this;

return old;

}

template<class T>

bool stack<T>::forward\_iterator::operator==(const forward\_iterator& other) const {

return ptr\_ == other.ptr\_;

}

template<class T>

bool stack<T>::forward\_iterator::operator!=(const forward\_iterator& other) const {

return ptr\_ != other.ptr\_;

}

template<class T>

T& stack<T>::forward\_iterator::operator\*() {

return this->ptr\_->value;

}

}

#endif // STACK\_H\_

**main.cpp:**

#include <iostream>

#include <algorithm>

#include "containers/stack.h"

#include "octagon.h"

int main() {

size\_t number;

float S;

//size\_t n = 0;

char option = 'a';

containers::stack<Octagon<int>> q;

Octagon<int> oct{};

while (option != '0') {

std::cout << "0. Exit\n"

<< "1. Add element in stack\n"

<< "2. Add element into number of position\n"

<< "3. Delete element from the number of position\n"

<< "4. Print out current stack\n"

<< "5. Print out number of elements, which area less than value\n";

std:: cin >> option;

switch (option) {

case '0':

break;

case '1': {

std::cout << "enter octagon (have to enter dots consequently): " << std::endl;

oct = Octagon<int>(std::cin);

q.push(oct);

break;

}

case '2': {

std::cout << "enter position to insert to: ";

std::cin >> number;

std::cout << "enter octagon: ";

oct = Octagon<int>(std::cin);

q.insert\_by\_number(number, oct);

break;

}

case '3': {

std::cout << "enter position to delete: ";

std::cin >> number;

q.delete\_by\_number(number);

break;

}

case '4': {

std::for\_each(q.begin(), q.end(), [](Octagon<int> &X) { X.print(std::cout); });

break;

}

case '5': {

std::cout << "enter max area: ";

std::cin >> S;

std::cout <<"number of elements with area < than " << S << ": " << std::count\_if(q.begin(), q.end(), [=](Octagon<int>& X){return X.area() < S;}) << std::endl;

break;

}

default:

std::cout << "Incorrect option." << std::endl;

break;

}

}

return 0;

}

**2. Ссылка на репозиторий на Github**

[https://github.com/mmaxim2710/oop\_exercise\_0](https://github.com/mmaxim2710/oop_exercise_01)5

**3.Набор testcases**

**1)**

1

0 2 1 3 2 3 3 2 3 1 2 0 1 0 0 1

1

1 3 2 5 3 5 4 4 4 2 3 1 2 0 1 2

4

5

2

5

100

3

0

4

**2)**

2

0

0 2 1 3 2 3 3 2 3 1 2 0 1 0 0 1

2

1

1 3 2 5 3 5 4 4 4 2 3 1 2 0 1 2

4

**4. Результат выполнения тестов**

**1)**

(0, 2), (1, 3), (2, 3), (3, 2), (3, 1), (2, 0), (1, 0), (0, 1)

(1, 3), (2, 5), (3, 5), (4, 4), (4, 2), (3, 1), (2, 0), (1, 2)

number of elements with area < than 2: 0

number of elements with area < than 100: 2

(1, 3), (2, 5), (3, 5), (4, 4), (4, 2), (3, 1), (2, 0), (1, 2)

**2)**

(1, 3), (2, 5), (3, 5), (4, 4), (4, 2), (3, 1), (2, 0), (1, 2)

(0, 2), (1, 3), (2, 3), (3, 2), (3, 1), (2, 0), (1, 0), (0, 1)

**5. Объяснение результатов программы**

Коллекция стека расположена в отдельном пространстве имён containers.

В ней содержится private структура element – элемент стека и private член — size, хранящий размер стека, public методы работы со стеком: push – добавить элемент в конец, pop – удаление элемента, insert\_by\_iterator – добавление элемента по итератору, insert\_by\_number – добавление по номеру (с использованием insert\_by\_iterator), delete\_by\_iterator – удаление по итератору, delete\_by\_number – удаление по номеру(с использованием delete\_by\_iterator).

**Вывод:** Проделав данную работу я ознакомился с реализацией итераторов, научился реализовывать совместимость с стандартными алгоритмами работы с колелкциями, изучил принципы работы указателя unique\_ptr.