

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 5: «Основы работы с коллекциями: Итераторы»

Группа:	М8О-108Б-18, №6
Студент:	Васильева Василиса Евгеньевна
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	25.11.2019

Москва, 2019

## 1. Задание

Собрать шаблон динамической коллекции согласно варианту задания.  
Вариант 6: Пятиугольник. Стек.

## 2. Адрес репозитория на GitHub

[https://github.com/vasilisavasileva/oop\\_exercise\\_05](https://github.com/vasilisavasileva/oop_exercise_05)

## 3. Код программы на C++

### Vertex.h

```
#pragma once
#include<iostream>
#include<type_traits>
#include<cmath>

template<class T>
struct vertex {
    T x;
    T y;
    vertex<T>& operator=(vertex<T> A);
    vertex() = default;
};

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;
}

template<class T>
struct is_vertex : std::false_type {};

template<class T>
struct is_vertex<vertex<T>> : std::true_type {};

```

## Pentagon.h

```

#pragma once
#include "vertex.h"

template<class T>
class Pentagon {
public:
    vertex<T> vertices[5];
    Pentagon() = default;
    Pentagon(std::istream& in);
    void Read(std::istream& in);
    double Area() const;
    void Print(std::ostream& os) const;
    friend std::ostream& operator<< (std::ostream& out, const
Pentagon<T>& point);
};

template<class T>
Pentagon<T>::Pentagon(std::istream& is) {
    for (int i = 0; i < 5; i++) {
        is >> this->vertices[i];
    }
}

template<class T>
double Pentagon<T>::Area() const {
    double Area = 0;
    for (int i = 0; i < 5; i++) {
        Area += (vertices[i].x) * (vertices[(i + 1) % 5].y) -
(vertices[(i + 1) % 5].x) * (vertices[i].y);
    }
    Area *= 0.5;
    return abs(Area);
}

template<class T>
void Pentagon<T>::Print(std::ostream& os) const {
    for (int i = 0; i < 5; i++) {

```

```

        os << this->vertices[i];
        if (i != 4) {
            os << ',';
        }
    }
    os << std::endl;
}

template<class T>
void Pentagon<T>::Read(std::istream& in) {
    for (int i = 0; i < 5; i++)
        in >> this->vertices[i];
}

template<class T>
std::ostream& operator<<(std::ostream& os, const Pentagon<T>&
point) {
    for (int i = 0; i < 5; i++) {
        os << point.vertices[i];
        if (i != 4) {
            os << ',';
        }
    }
}

```

## 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 {
        public:
            using value_type = T;
            using reference = T&;
            using pointer = T*;
            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 stack;
        };
    };
}

```

```

};

forward_iterator begin();
forward_iterator end();
void push(const T& value);
T& top();
T& bottom();
void pop();
size_t length();
void delete_by_it(forward_iterator d_it);
void delete_by_index(size_t N);
void insert_by_it(forward_iterator ins_it, T& value);
void insert_by_index(size_t N, T& value);
stack& operator=(stack& other);
private:
    struct element {
        T value;
        std::unique_ptr<element> next_element = nullptr;
        forward_iterator next();
    };

    static std::unique_ptr<element>
push_impl(std::unique_ptr<element> cur, const T& value);
    static std::unique_ptr<element>
pop_impl(std::unique_ptr<element> cur);
    std::unique_ptr<element> first = nullptr;
};

template<class T>
typename stack<T>::forward_iterator stack<T>::begin() {
    return forward_iterator(first.get());
}

template<class T>
typename stack<T>::forward_iterator stack<T>::end() {
    return forward_iterator(nullptr);
}

template<class T>
size_t stack<T>::length() {
    return size;
}

template<class T>
void stack<T>::push(const T& value) {
    first = push_impl(std::move(first), value);
    size++;
}

template<class T>
std::unique_ptr<typename stack<T>::element>
stack<T>::push_impl(std::unique_ptr<element> cur, const T& value) {
    if (cur != nullptr) {

```

```

        cur->next_element = push_impl(std::move(cur-
>next_element), value);
        return cur;
    }
    return std::unique_ptr<element>(new element{ value });
}

template<class T>
void stack<T>::pop() {
    if (size == 0) {
        throw std::logic_error("stack is empty");
    }
    first = pop_impl(std::move(first));
    size--;
}

template<class T>
std::unique_ptr<typename stack<T>::element>
stack<T>::pop_impl(std::unique_ptr<element> cur) {
    if (cur->next_element != nullptr) {
        cur->next_element = pop_impl(std::move(cur-
>next_element));
        return cur;
    }
    return nullptr;
}

template<class T>
T& stack<T>::top() {
    if (size == 0) {
        throw std::logic_error("stack is empty");
    }
    forward_iterator i = this->begin();
    while (i.it_ptr->next() != this->end()) {
        i++;
    }
    return *i;
}

template<class T>
T& stack<T>::bottom() {
    return first->value;
}

template<class T>
stack<T>& stack<T>::operator=(stack<T>& other) {
    size = other.size;
    first = std::move(other.first);
}

template<class T>
void
stack<T>::delete_by_it(containers::stack<T>::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();
            return;
        }
        while ((i.it_ptr != nullptr) && (i.it_ptr->next() != d_it))
    {
        ++i;
    }
    if (i.it_ptr == nullptr) throw std::logic_error("out of
borders");
    i.it_ptr->next_element = std::move(d_it.it_ptr-
>next_element);
    size--;
}

template<class T>
void stack<T>::delete_by_index(size_t N) {
    forward_iterator it = this->begin();
    for (size_t i = 1; i <= N; ++i) {
        if (i == N) break;
        ++it;
    }
    this->delete_by_it(it);
}

template<class T>
void
stack<T>::insert_by_it(containers::stack<T>::forward_iterator ins_it,
T& value) {
    auto tmp = std::unique_ptr<element>(new element{ value });
    forward_iterator i = this->begin();
    if (ins_it == this->begin()) {
        tmp->next_element = std::move(first);
        first = std::move(tmp);
        size++;
        return;
    }
    while ((i.it_ptr != nullptr) && (i.it_ptr->next() !=
ins_it)) {
        ++i;
    }
    if (i.it_ptr == nullptr) throw std::logic_error("out of
borders");
    tmp->next_element = std::move(i.it_ptr->next_element);
    i.it_ptr->next_element = std::move(tmp);
    size++;
}

template<class T>
void stack<T>::insert_by_index(size_t N, T& value) {
    forward_iterator it = this->begin();
    for (size_t i = 1; i <= N; ++i) {
        if (i == N) break;
        ++it;
    }
    this->insert_by_it(it, value);
}

```

```

        template<class T>
        typename stack<T>::forward_iterator stack<T>::element::next() {
            return forward_iterator(this->next_element.get());
        }

        template<class T>
        stack<T>::forward_iterator::forward_iterator(containers::stack<T>
::element* ptr) {
            it_ptr = ptr;
        }

        template<class T>
        T& stack<T>::forward_iterator::operator*() {
            return this->it_ptr->value;
        }

        template<class T>
        typename stack<T>::forward_iterator&
stack<T>::forward_iterator::operator++() {
            if (it_ptr == nullptr) throw std::logic_error("out of
stack");
            *this = it_ptr->next();
            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 it_ptr == other.it_ptr;
        }

        template<class T>
        bool stack<T>::forward_iterator::operator!=(const
forward_iterator& other) const {
            return it_ptr != other.it_ptr;
        }
    }
}

```

## Main.cpp

```

#include<iostream>
#include<algorithm>
#include<locale.h>
#include"Pentagon.h"
#include"stack.h"

```



```

void Menu1() {
    std::cout << "1. Добавить фигуру в стек\n";
    std::cout << "2. Удалить фигуру\n";
    std::cout << "3. Вывести фигуру\n";
    std::cout << "4. Вывести все фигуры\n";
    std::cout << "5. Вывести фигуру если площадь больше чем ...\n";
}

void DeleteMenu() {
    std::cout << "1. Удалить фигуру в конце стека\n";
    std::cout << "2. Удалить фигуру по индексу\n";
}

void PrintMenu() {
    std::cout << "1. Вывести первую фигуру в стеке\n";
    std::cout << "2. Вывести последнюю фигуру в стеке\n";
}

int main() {
    setlocale(LC_ALL, "rus");
    containers::stack<Pentagon<int>> Mystack;

    Pentagon<int> TempPentagon;

    while (true) {
        Menu1();
        int n, m;
        size_t ind;
        double s;
        std::cin >> n;
        switch (n) {
            case 1:
                TempPentagon.read(std::cin);
                TempPentagon.Print(std::cout);
                Mystack.push(TempPentagon);
                break;
            case 2:
                DeleteMenu();
                std::cin >> m;
                switch (m) {
                    case 1:
                        Mystack.pop();
                        break;
                    case 2:
                        std::cin >> ind;
                        Mystack.delete_by_number(ind);
                        break;
                    default:
                        break;
                }
                break;
            case 3:
                PrintMenu();
                std::cin >> m;
                switch (m) {
                    case 1:
                        Mystack.bottom().Print(std::cout);
                        std::cout << std::endl;

```

```

        break;
    case 2:
        Mystack.top().Print(std::cout);
        std::cout << std::endl;
        break;
    default:
        break;
    }
    break;
case 4:
    std::for_each(Mystack.begin(), Mystack.end(),
[ ](Pentagon<int>& X) { X.Print(std::cout); std::cout << std::endl; });
    break;
case 5:
    std::cin >> s;
    std::cout << std::count_if(Mystack.begin(),
Mystack.end(), [=](Pentagon<int>& X) {return X.Area() > s; }) <<
std::endl;
    break;
    default:
        return 0;
    }
}
system("pause");
return 0;
}

```

#### 4. Объяснение результатов работы программы

Выводящееся меню предлагает пользователю добавить в стек фигуру, вывести все фигуры или фигуру по индексу в стеке, а также удалить фигуру. Вершинами фигуры являются структуры vertex. В классе фигуры определены такие методы, как считывание координат с потока ввода, подсчет площади, вывод координат на экран. В классе стека описан forward\_iterator, при помощи которого осуществляются передвижение и доступ к элементам стека. В классе присутствуют такие методы, как вставка и удаление элемента в текущем месте нахождения итератора, а также вставка и удаление элемента с верхушки стека. В соответствии с выбором пользователя приводится в исполнение определенный метод класса стек.

#### 5. Вывод

Итератор – это такая структура данных, которая используется для доступа к элементам контейнера, чтобы производить над ними какие-то действия. В нашем случае итератор еще и имеет возможность перемещаться по индексам контейнера для обеспечения более удобного доступа. Таким образом, нам не нужно прописывать разные алгоритмы для доступа к отдельным ячейкам нашего контейнера, потому что для выполнения этой функции у нас есть итератор.