

Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа
по курсу «ООП»**

Тема:
Итераторы и умные указатели.

Студент:	Касимов М.М.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	6
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

Pentagon.h:

#pragma once

#include "point.h"

```
template<class T>
struct pentagon {
private:
    point<T> a1,a2,a3,a4,a5;
public:
    point<T> center() const;
    void print(std::ostream& os) const ;
    double area() const;
    pentagon(std::istream& is);
};
```

```
template<class T>
double pentagon<T>::area() const {
    return (0.5) * abs(((a1.x*a2.y + a2.x*a3.y + a3.x*a4.y + a4.x*a5.y + a5.x*a1.y) - (
a1.y*a2.x + a2.y*a3.x + a3.y*a4.x + a4.y*a5.x + a5.y*a1.x )));
}
```

```
template<class T>
pentagon<T>::pentagon(std::istream& is) {
    is >> a1 >> a2 >> a3 >> a4 >> a5;
}
```

```
template<class T>
void pentagon<T>::print(std::ostream& os) const {
    os << "coordinate:\n" << "{" << "\n" << a1 << "\n" << a2 << "\n" << a3 << "\n" << a4 << "\n"
<< a5 << "\n" << "}" << "\n";
}
```

```
template<class T>
point<T> pentagon<T>::center() const {
    T x,y;
    x = (a1.x + a2.x + a3.x + a4.x + a5.x) / 5;
    y = (a1.y + a2.y + a3.y + a4.y + a5.y) / 5;
    return {x,y};
}
```

Point.h:

#pragma once

```
#include <iostream>
```

```
template<class T>
```

```
struct point {
```

```
    T x;
```

```
    T y;
```

```
};
```

```
template<class T>
```

```
std::istream& operator>> (std::istream& is, point<T>& p) {
```

```
    is >> p.x >> p.y;
```

```
    return is;
```

```
}
```

```
template<class T>
```

```
std::ostream& operator<< (std::ostream& os, const point<T>& p) {
```

```
    os << p.x << ' ' << p.y;
```

```
    return os;
```

```
}
```

Stack.h:

```
#pragma once
```

```
#include <iterator>
```

```
#include <memory>
```

```
namespace cntr {
```

```
    template<class T>
```

```
    class Stack {
```

```
    private:
```

```
        class StackNode;
```

```
    public:
```

```
        class ForwardIterator {
```

```
        public:
```

```
            using value_type = T;
```

```
            using reference = T &;
```

```
            using pointer = T *;
```

```
            using difference_type = ptrdiff_t;
```

```
            using iterator_category = std::forward_iterator_tag;
```

```
            ForwardIterator(StackNode *node) : Ptr(node) { };
```

```
            T &operator*();
```

```

ForwardIterator &operator++();

ForwardIterator operator++(int);

bool operator==(const ForwardIterator &it) const;

bool operator!=(const ForwardIterator &it) const;

private:
    StackNode *Ptr;

    friend class Stack;
};

ForwardIterator begin();

ForwardIterator end();

T Top();

void Insert(const ForwardIterator &it, const T &value);

void Erase(const ForwardIterator &it);

void Pop();

Stack() = default;

void Push(const T &value);

size_t Size = 0;
private:
    class StackNode {
    public:
        T Value;
        std::unique_ptr<StackNode> NextNode = nullptr;

        ForwardIterator next();

        StackNode(const T &value, std::unique_ptr<StackNode> next) : Value(value),
NextNode(std::move(next)) {};
    };

    std::unique_ptr<StackNode> Head = nullptr;
Stack(const Stack&);

```

```
Stack& operator=(const Stack&)
```

```
};
```

```
template<class T>
typename Stack<T>::ForwardIterator Stack<T>::StackNode::next() {
    return {NextNode.get()};
}
```

```
template<class T>
T &Stack<T>::ForwardIterator::operator*() {
    return Ptr->Value;
}
```

```
template<class T>
typename Stack<T>::ForwardIterator &Stack<T>::ForwardIterator::operator++() {
    *this = Ptr->next();
    return *this;
}
```

```
template<class T>
typename Stack<T>::ForwardIterator Stack<T>::ForwardIterator::operator++(int)
{
    ForwardIterator prev = *this;
    ++(*this);
    return prev;
}
```

```
template<class T>
bool Stack<T>::ForwardIterator::operator!=(const ForwardIterator &other) const {
    return Ptr != other.Ptr;
}
```

```
template<class T>
bool Stack<T>::ForwardIterator::operator==(const ForwardIterator &other) const {
    return Ptr == other.Ptr;
}
```

```
template<class T>
typename Stack<T>::ForwardIterator Stack<T>::begin() {
    return Head.get();
}
```

```
template<class T>
typename Stack<T>::ForwardIterator Stack<T>::end() {
```

```

    return nullptr;
}

template<class T>
void Stack<T>::Insert(const ForwardIterator &it, const T &value) {
    std::unique_ptr<StackNode> newNode(new StackNode(value, nullptr));
    if (it.Ptr == nullptr && Size != 0) {
        throw std::logic_error("Out of bounds");
    }
    if (Size == 0) {
        Head = std::move(newNode);
        ++Size;
    } else {
        newNode->NextNode = std::move(it.Ptr->NextNode);
        it.Ptr->NextNode = std::move(newNode);
        ++Size;
    }
}

```

```

template<class T>
void Stack<T>::Push(const T &value) {
    std::unique_ptr<StackNode> newNode(new StackNode(value, nullptr));
    newNode->NextNode = std::move(Head);
    Head = std::move(newNode);
    ++Size;
}

```

```

template<class T>
T Stack<T>::Top() {
    if (Head.get()) {
        return Head->Value;
    } else {
        throw std::logic_error("Stack is empty");
    }
}

```

```

template<class T>
void Stack<T>::Pop() {
    if (Head.get() == nullptr) {
        throw std::logic_error("Stack is empty");
    }
    Head = std::move(Head->NextNode);
    --Size;
}

```

```

template<class T>
void Stack<T>::Erase(const Stack<T>::ForwardIterator &it) {
    if (it.Ptr == nullptr) {
        throw std::logic_error("Invalid iterator");
    }
    if (it == this->begin()) {
        Head = std::move(Head->NextNode);

    } else {
        auto tmp = this->begin();
        while (tmp.Ptr->next() != it.Ptr) {
            ++tmp;
        }
        tmp.Ptr->NextNode = std::move(it.Ptr->NextNode);
    }
}
}

```

Main.cpp:

```

#include <iostream>
#include "stack.h"
#include "pentagon.h"
#include <algorithm>

void menu() {
    std::cout << "1 - add(1 - push, 2 - insert by iterator(enter index new elem))\n"
               << "2 - delete(1 - pop, 2 - delete by iterator(enter index))\n"
               << "3 - top\n"
               << "4 - print\n"
               << "5 - count if(enter max area)\n"
               << "6 - exit\n";
}

void usingStack() {
    int command, minicommand, index;
    double val;
    cntr::Stack<pentagon<double>> st;
    for (;;) {
        std::cin >> command;
        if (command == 1) {
            std::cin >> minicommand;
            if (minicommand == 1) {
                pentagon<double> p(std::cin);
                st.Push(p);
            }
        }
    }
}

```

```

} else if (minicommand == 2) {
    std::cin >> index;
    try {
        if (index < 0 || index >= st.Size) {
            throw std::logic_error("Out of bounds\n");
        }
        if (index == st.Size - 1) {
            st.Push(p);
        } else {
            auto it = st.begin();
            for (int i = 0; i < index; ++i) {
                ++it;
            }
            st.Insert(it, p);
        }
    } catch (std::logic_error &e) {
        std::cout << e.what() << std::endl;
        continue;
    }
}
} else if (command == 6) {
    break;
} else if (command == 2) {
    std::cin >> minicommand;
    if (minicommand == 1) {
        try {
            st.Pop();
        } catch (std::logic_error &e) {
            std::cout << e.what() << std::endl;
            continue;
        }
    }
}
if (minicommand == 2) {
    std::cin >> index;
    try {
        if (index < 0 || index > st.Size) {
            throw std::logic_error("Out of bounds\n");
        }
        auto it = st.begin();
        for (int i = 0; i < index; ++i) {
            ++it;
        }
        st.Erase(it);
    }
    catch (std::logic_error &e) {

```



```

        std::cout << e.what() << std::endl;
        continue;
    }
}
} else if (command == 3) {
    try {
        st.Top().print(std::cout);
    }
    catch (std::logic_error &e) {
        std::cout << e.what() << std::endl;
        continue;
    }
} else if (command == 4) {
    for (auto elem: st) {
        elem.print(std::cout);
    }
} else if (command == 5) {
    std::cin >> val;
    std::cout << std::count_if(st.begin(), st.end(), [val](pentagon<double> r) {
return r.area() < val; })
        << std::endl;
    } else {
        std::cout << "Error command\n";
        continue;
    }
}
}

int main() {
    menu();
    usingStack();
    return 0;
}

```

2. Ссылка на репозиторий на GitHub.

https://github.com/magomed2000kasimov/oop_exercise_05

3. Набор тестов.

test_01.test:

```

3
4
5 7
2 1
2 2 1

```

6

test_02.test:

1 1

0 0

0 3

2 3

4 3

4 0

1 2 3

1 2 0

0 0

0 2

2 2

3 1

2 0

3

4

1 2 0

0 0

0 2

2 2

2 0

1 0

4

5 4.3

6

test_03.test:

1 1

3 3

3 3

3 3

3 3

3 3

1 1

2 2

2 2

2 2
2 2
2 2

1 1

1 1
1 1
1 1
1 1
1 1

4
2 2 1
4
2 1
4
2 2 0
3
5 0
6

4. Результаты выполнения тестов.

test_01.result:

1 - add(1 - push, 2 - insert by iterator(enter index new elem)
2 - delete(1 - pop, 2 - delete by iterator(enter index)
3 - top
4 - print
5 - count if(enter max area)
6 - exit
Stack is empty
0
Stack is empty
Out of bounds

test_02.result:

1 - add(1 - push, 2 - insert by iterator(enter index new elem)
2 - delete(1 - pop, 2 - delete by iterator(enter index)
3 - top
4 - print
5 - count if(enter max area)
6 - exit
Out of bounds

coordinate:

```
{  
0 0  
0 3  
2 3  
4 3  
4 0  
}
```

coordinate:

```
{  
0 0  
0 3  
2 3  
4 3  
4 0  
}
```

coordinate:

```
{  
0 0  
0 2  
2 2  
3 1  
2 0  
}
```

coordinate:

```
{  
0 0  
0 3  
2 3  
4 3  
4 0  
}
```

coordinate:

```
{  
0 0  
0 2  
2 2  
2 0  
1 0  
}
```

coordinate:

```
{  
0 0  
0 2  
2 2
```

```
3 1
2 0
}
1
```

test_03.result:

```
1 - add(1 - push, 2 - insert by iterator(enter index new elem)
2 - delete(1 - pop, 2 - delete by iterator(enter index)
3 - top
4 - print
5 - count if(enter max area)
6 - exit
```

coordinate:

```
{
1 1
1 1
1 1
1 1
1 1
}
```

coordinate:

```
{
2 2
2 2
2 2
2 2
2 2
}
```

coordinate:

```
{
3 3
3 3
3 3
3 3
3 3
}
```

coordinate:

```
{
1 1
1 1
1 1
1 1
1 1
}
```

coordinate:

```
{
```

```
3 3
3 3
3 3
3 3
3 3
}
coordinate:
{
3 3
3 3
3 3
3 3
3 3
}
Stack is empty
0
```

5. Объяснение результатов работы программы.

Площадь для пятиугольника находится методом Гаусса (метод шунтирования). В 1 тесте я выходил за границу и пытался удалить объект, которого в стеке нет. Как в результирующем файле все ошибки корректно обрабатываются благодаря конструкции try catch. Вставка по итератору производится за текущим объектом, а удаление по текущему. Во 2 и 3 тесте происходят вставки и удаления, а после вызывается count_if и поэлементный for .

6. Вывод.

Выполняя данную лабораторную, я получил опыт работы с итераторами и умными указателями в C++, с системой сборки meson, с системой контроля версий git. Узнал о применении итераторов в STL. Понял, как важно использовать свое пространство имен при создании своего контейнера. Мне было трудно работать с умными указателями, но, если научиться грамотно использовать их, можно избежать неприятных утечек памяти.

7. Литература.

- 1) лекции по ООП МАИ.
- 2) Г.Шилдт «C++».
- 3) Б.Страуструп «C++ специальное издание».