Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа
по курсу «ООП»

Тема:
Основы работы с коллекциями: итераторы.

Студент:	Симонов С.Я.
Группа:	М8О-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	21
Оценка:	
Дата:	

Москва
2019

1. Код на C++:

```
#ifndef POINT_H
#define POINT_H 1

#include <iostream>
#include <algorithm>
#include <cmath>


template<class T>
struct TPoint {
	TPoint() {}
	TPoint(T a, T b) : x(a), y(b){}
	T x;
	T y;


};

template<class T>
std::ostream& operator << (std::ostream& os, const TPoint<T>& p)
{
	os << p.x << " " << p.y << " ";
	return os;
}

template <class T>
std::istream& operator >> (std::istream& is, TPoint<T>& p)
{
	is >> p.x >> p.y;
	return is;
}

template <class T>
TPoint<T> operator /= ( TPoint<T>& p, int val)
{
	p.x = p.x / val;
	p.y = p.y / val;
	return p;
}

template <class T>
TPoint<T> operator + (const TPoint<T>& p1, const TPoint<T>& p2)
{
	TPoint<T> p;
	p.x = p1.x + p2.x;
	p.y = p1.y + p2.y;
	return p;
}

template <class T>
```

```
TPoint<T> operator - (const TPoint<T> p1, const TPoint<T> p2)
{
	TPoint<T> p;
	p.x = p1.x - p2.x;
	p.y = p1.y - p2.y;
	return p;
}

#endif
```

trapezoid.h

```
#ifndef TRAPEZOID_H
#define TRAPEZOID_H 1


#include "point.h"
#include "stack.h"
#include <cassert>

template <class T>
struct TTrapezoid {
	TPoint<T> a, b, c, d;
	TTrapezoid(std::istream&);
	double Square() const;
	TPoint<T> Center() const;
	void Print() const;
};

template <class T>
TTrapezoid<T>::TTrapezoid(std::istream& is) {
	is >> a >> b >> c >> d;
	TPoint<T> ab, ad, bc, dc;
	ab.x = b.x - a.x;
	ab.y = b.y - a.y;
	ad.x = d.x - a.x;
	ad.y = d.y - a.y;
	bc.x = c.x - b.x;
	bc.y = c.y - b.y;
	dc.x = c.x - d.x;
	dc.y = c.y - d.y;
	assert(acos((ab.x * dc.x + ab.y * dc.y) / (sqrt(ab.x * ab.x + ab.y * ab.y) * sqrt(dc.x * dc.x
+ dc.y * dc.y))) == 0 || acos((ad.x * bc.x + ad.y * bc.y) / (sqrt(ad.x * ad.x + ad.y * ad.y) *
sqrt(bc.x * bc.x + bc.y * bc.y))) == 0);

}

template <class T>
double TTrapezoid<T>::Square() const {
	TPoint<T> p = this->Center();
	T t1 = 0.5 * fabs((b.x - a.x) * (p.y - a.y) - (p.x - a.x) * (b.y - a.y));
	T t2 = 0.5 * fabs((c.x - b.x) * (p.y - b.y) - (p.x - b.x) * (c.y - b.y));
	T t3 = 0.5 * fabs((d.x - c.x) * (p.y - c.y) - (p.x - c.x) * (d.y - c.y));
	T t4 = 0.5 * fabs((a.x - d.x) * (p.y - d.y) - (p.x - d.x) * (a.y - d.y));
	return t1 + t2 + t3 + t4;
}

template <class T>
TPoint<T> TTrapezoid<T>::Center() const {
	TPoint<T> p;
	T x = (a.x + b.x + c.x + d.x) /4;
	T y = (a.y + b.y + c.y + d.y) /4;
	p.x = x;
	p.y = y;

	return p;
}

template <class T>
void TTrapezoid<T>::Print() const {
	std::cout << a << b << c << d << "\n";
}




#endif
```

stack.h

```
#ifndef STACK_H
#define STACK_H 1

#include <memory>
#include <iostream>
#include <iterator>



namespace containers {


template <class T>
class TStack {
private:

	struct Node;
```

```
public:


	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;
		forward_iterator (Node* ptr) : ptr_(ptr) {};
		T& operator* ();
		forward_iterator& operator++ ();
		forward_iterator operator++ (int);
		bool operator== (const forward_iterator& o) const;
		bool operator!= (const forward_iterator& o) const;

	private:
		Node* ptr_ = nullptr;
		friend TStack;
	};
	forward_iterator begin();
	forward_iterator end();
	void pop();
	T& top();
	void push(const T& value);
	void erase(const forward_iterator& it);
	void insert(forward_iterator& it, const T& val);
	void advance(forward_iterator& it, int idx);
	void print();
private:
	struct Node {
		T value;
		std::shared_ptr<Node> following = nullptr;
		forward_iterator next();
		Node(const T& val, std::shared_ptr<Node> nxt) :
			value(val), following(nxt) {};
	};
	std::shared_ptr<Node> head = nullptr;


};

template <class T>
typename TStack<T>::forward_iterator TStack<T>::Node::next() {
	return following.get();
}

template <class T>
typename TStack<T>::forward_iterator TStack<T>::begin() {
	return head.get();
```

```
}

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

template <class T>
T& TStack<T>::forward_iterator::operator* () {
	return ptr_->value;
}

template <class T>
typename TStack<T>::forward_iterator& TStack<T>::forward_iterator::operator++ () {
	*this = ptr_->next();
	return *this;
}

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

template <class T>
bool TStack<T>::forward_iterator::operator== (const forward_iterator& o) const{
	return ptr_ == o.ptr_;
}

template <class T>
bool TStack<T>::forward_iterator::operator!= (const forward_iterator& o) const{
	return ptr_ != o.ptr_;
}



template <class T>
void TStack<T>::push(const T& value) {
	std::shared_ptr<Node> NewNode(new Node(value, nullptr));
	NewNode->following = head;
	head = NewNode;
}

template<class T>
void TStack<T>::pop() {
	if (head.get() == nullptr) {
		throw std::logic_error("Stack is empty\n");
	} else {
		head = head->following;
	}
}
```

```
template <class T>
T& TStack<T>::top() {
        if (head.get() == nullptr) throw std::logic_error("Stack is empty\n");
        return head->value;
}


template <class T>
void TStack<T>::print() {
        std::shared_ptr<Node> tmp;
        tmp = head;
        while (tmp != nullptr) {
                std::cout << tmp->value << " ";
                tmp = tmp->following;
        }
}


template <class T>
void TStack<T>::insert(forward_iterator& it, const T& value) {
        std::shared_ptr<Node> NewNode(new Node(value, nullptr));
        if (it.ptr_ == head.get()) {
                this->push(value);
                return;
        }
        auto tmp = this->begin();
        auto prev = tmp;
        while (tmp.ptr_ != it.ptr_) {
                if (tmp.ptr_ == nullptr && tmp.ptr_ != it.ptr_) throw std::logic_error("Out of
range");
                prev.ptr_ = tmp.ptr_;
                ++tmp;
        }
        NewNode->following = prev.ptr_->following;
        prev.ptr_->following = NewNode;

        return;

}

template <class T>
void TStack<T>::erase(const forward_iterator& it) {
        if (it.ptr_ == head.get()) {
                this->pop();
                return;
        }
        auto tmp = this->begin();
        auto prev =tmp;
        while (tmp.ptr_ != it.ptr_) {

                prev.ptr_ = tmp.ptr_;
        ++tmp;
        if (tmp.ptr_ == nullptr) {
            throw std::logic_error("Out of range");

        }
    }
    prev.ptr_->following = tmp.ptr_->following;

    return;

}



template <class T>
void TStack<T>::advance(forward_iterator& it, int idx) {
    it = this->begin();
    if (it.ptr_ == nullptr && idx > 0) throw std::logic_error("Out of range");
    int i = 0;
    while (i < idx) {
        if (it.ptr_->following == nullptr && i < idx - 1) {

            throw std::logic_error("Out of range\n");
        }
        ++it;
        ++i;

    }
}

}

#endif
```

main.cpp

```
#include <iostream>
#include <string>
#include <algorithm>
#include "stack.h"
#include "trapezoid.h"
#include "point.h"

int main() {
    containers::TStack<TTrapezoid<int>> s;
    std::string cmd;
    int index;
```

```
std::cout << "push - to push figure to stack\n"
          << "insert - to insert figure to stack\n"
          << "pop - to pop figure from Stack\n"
          << "erase - to delete figure from Stack\n"
          << "top - to show first figure\n"
          << "for_each - to print figures\n"
          << "count_if - to print quantity of figures with square less then given\n"
          << "exit - to finish execution of program\n";
while (true) {
    std::cin >> cmd;
    if (cmd == "push") {
        std::cout << "enter coordinates\n";
        TTrapezoid<int> fig(std::cin);
        s.push(fig);
    } else if (cmd == "insert") {
        std::cout << "enter index\n";
        std::cin >> index;
        auto p = s.begin();
        try {
            s.advance(p, index);
        } catch (std::exception& err) {
            std::cout << err.what() << "\n";
            continue;
        }
        std::cout << "enter coordinates\n";
        TTrapezoid<int> fig(std::cin);
        s.insert(p, fig);

    } else if (cmd == "pop") {
        try {
            s.pop();
        } catch (std::exception& err) {
            std::cout << err.what() << "\n";
            continue;
        }
    } else if (cmd == "erase") {
        std::cout << "enter index\n";
        std::cin >> index;
        auto p = s.begin();
        try {
            s.advance(p, index);
        } catch (std::exception& err) {
            std::cout << err.what() << "\n";
            continue;
        }
        try {
            s.erase(p);
        } catch (std::exception& err) {
            std::cout << err.what() << "\n";
        }
```

```
		} else if (cmd == "top") {
			try {
				s.top();
			} catch (std::exception& err) {
				std::cout << err.what() << "\n";
				continue;
			}
			TTrapezoid<int> figure = s.top();
			figure.Print();

		} else if (cmd == "for_each") {
			std::for_each(s.begin(), s.end(), [] (TTrapezoid<int> tmp) {return
tmp.Print();});
		} else if (cmd == "count_if") {
			int less;
			std::cout << "enter square\n";
			std::cin >> less;
			std::cout << std::count_if(s.begin(), s.end(), [less](TTrapezoid<int> t )
{return t.Square() < less;}) << "\n";

		} else if (cmd == "exit") {
			break;
		} else {
			std::cout << "wrong comand\n";
			continue;
		}
	}

}
```

2. Ссылка на Github

https://github.com/vebcreatex7/oop_exercise_05

3. Набор тестов

```
emil@emil-Aspire-E5-576G:~/Документы/kirill/oop_exercise_
05$ ./oop_exercise_05
push - to push figure to stack
insert - to insert figure to stack
pop - to pop figure from Stack
erase - to delete figure from Stack
top - to show first figure
for_each - to print figures
count_if - to print quantity of figures with square less then given
exit - to finish execution of program
push
enter coordinates
0 0 1 1 2 1 3 0
```

```
insert
enter index
1
enter coordinates
0 0 2 2 4 2 6 0
top
0 0 1 1 2 1 3 0
for_each
0 0 1 1 2 1 3 0
0 0 2 2 4 2 6 0
push
enter coordinates
0 0 3 3 6 3 9 0
count_if
enter square
10
2
pop
top
0 0 1 1 2 1 3 0
erase
enter index
2
Out of range
erase 1
enter index
top
0 0 1 1 2 1 3 0
exit
```

4. Объяснение результатов работы программы:

Стек реализован в виде односвязного списка на итераторах. В main.cpp push добавляет элемент в список, pop удаляет, insert вставляет по индексу, erase удаляет по индексу for_each выводит все элементы за счет того, что для стека реализованны итераторы, count_if выводи количество фигур, площадь которых меньше данной.

6. Вывод:

В данной лабораторной работе я освоил основы работы с коллекциями и итераторами. Создал свой STL контейнер основанный на умных указателях.