

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
УНИВЕРСИТЕТ)**

**ЛАБОРАТОРНАЯ РАБОТА №7**

по курсу объектно-ориентированное программирование I семестр, 2021/22  
уч. год

Студент: Макаров Глеб Александрович, группа М8О-207Б-20

Преподаватель: Дорохов Евгений Павлович

## Условие

Задание: Стэк (Пятиугольник). Используя структуру данных, разработанную для лабораторной работы №4, спроектировать и разработать итератор для динамической структуры данных.

Итератор должен быть разработан в виде шаблона и должен позволять работать с любыми типами фигур, согласно варианту задания.

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.

## Описание программы

Исходный код лежит в 10 файлах:

1. main.cpp: тестирование кода
2. figure.h: родительский класс-интерфейс для фигур
3. point.h: описание класса точки
4. point.cpp: реализация класса точки
5. pentagon.h: описание класса пятиугольника, наследующегося от figure
6. pentagon.cpp: реализация класса пятиугольника
7. titerator.h: описание и реализация итераторов
8. template.cpp: файл для правильного подключения шаблонов класса.
9. tstack.h: структура стэка
10. tstack.cpp: реализация стэка

## **Дневник отладки**

Ошибок не было.

## **Недочёты**

Недочётов не заметил.

## **Вывод**

В данной лабораторной работе были реализованы итераторы для класса стэк. Итераторы необходимы для удобного перемещения по контейнерам. Похожую работу мы писали на первом курсе

## Исходный код

# main.cpp

```
#include "pentagon.h"
```

```
#include "tstack.h"
```

```
void menu(){
```

```
    std::cout << "Select an action" << std::endl;
```

```
    std::cout << "1) Remove an item from the stack" << std::endl;
```

```
    std::cout << "2) Print items from the stack" << std::endl;
```

```
    std::cout << "3) Clear the stack" << std::endl;
```

```
    std::cout << "4) Add an item to the stack" << std::endl;
```

```
    std::cout << "5) Print the stack length" << std::endl;
```

```
    std::cout << "6) Is the stack empty?" << std::endl;
```

```
    std::cout << "7) End the program" << std::endl;
```

```
}
```

```
int main() {
```

```
    std::cout.setf(std::ios_base::boolalpha);
```

```
    TStack<Pentagon> s;
```

```
    char k = 'y';
```

```
    menu();
```

```
    std::cin >> k;
```

```
    while (k != EOF) {
```

```
        switch (k) {
```

```
            case '1': s.Pop();
```

```
                break;
```

```
            case '2': std::cout << s << std::endl;
```

```
                break;
```

```
            case '3': s.Clear();
```

```
                break;
```

```
            case '4': s.Push(std::shared_ptr<Pentagon>(new Pentagon(std::cin)));
```

```
                break;
```

```

    case '5':std::cout << s.Length() << std::endl;
        break;
    case '6':std::cout << (bool)s.Empty() << std::endl;
        break;
    case '7': {
        std::cout << "Have a nice day!" << std::endl;
        return 0;
    }
    default: std::cout << "Input error! Enter a number from the suggested menu!" <<
std::endl;
        break;
    }
    menu();
    std::cin >> k;
}
return 0;
}

```

## figure.h

```

#ifndef MAI_OOP_FIGURE_H
#define MAI_OOP_FIGURE_H
#include "point.h"
#include <memory>

class Figure {
public:
    virtual size_t VertexesNumber() = 0;
    virtual double Area() = 0;
    virtual void Print(std::ostream &os) = 0;

};

#endif //MAI_OOP_FIGURE_H

```

# point.h

```
#ifndef POINT_H
```

```
#define POINT_H
```

```
#include <iostream>
```

```
class Point {
```

```
public:
```

```
Point();
```

```
Point(std::istream &is);
```

```
Point(double x, double y);
```

```
double dist(const Point &other);
```

```
double get_x();
```

```
double get_y();
```

```
friend std::istream &operator>>(std::istream &is, Point &p);
```

```
friend std::ostream &operator<<(std::ostream &os, Point &p);
```

```
bool operator==(const Point &p);
```

```
Point &operator=(const Point &p);
```

```
private:
```

```
double x_;
```

```
double y_;
```

```
};
```

```
#endif // POINT_H
```

# point.cpp

```
#include "point.h"
```

```
#include <cmath>
```

```
Point::Point() : x_(0.0), y_(0.0) {}
```

```
Point::Point(double x, double y) : x_(x), y_(y) {}
```

```
Point::Point(std::istream &is) {
```

```
    is >> x_ >> y_;
```

```
}
```

```
double Point::get_x() {
```

```
    return x_;
```

```
}
```

```
double Point::get_y() {
```

```
    return y_;
```

```
}
```

```
double Point::dist(const Point &other) {
```

```
    double dx = (other.x_ - x_);
```

```
    double dy = (other.y_ - y_);
```

```
    return std::sqrt(dx * dx + dy * dy);
```

```
}
```

```
std::istream &operator>>(std::istream &is, Point &p) {
```

```
    is >> p.x_ >> p.y_;
```

```
    return is;
```

```
}
```

```
std::ostream &operator<<(std::ostream &os, Point &p) {
```

```
    os << "(" << p.x_ << ", " << p.y_ << ")";
```

```

    return os;
}

bool Point::operator==(const Point &p) {
    if (this->x_ == p.x_ && this->y_ == p.y_) {
        return true;
    } else return false;
}

```

```

Point &Point::operator=(const Point &p) {
    if (this == &p) {
        return *this;
    }
    this->x_ = p.x_;
    this->y_ = p.y_;
    return *this;
}

```

## pentagon.h

```

#ifndef MAI_OOP_PENTAGON_H
#define MAI_OOP_PENTAGON_H
#include "figure.h"

class Pentagon {
private:
    Point a_, b_, c_, d_, e_;
public:
    Pentagon();
    Pentagon(const Pentagon &pentagon);
    Pentagon(std::istream &is);
    size_t VertexesNumber();

```



```

    double Area();
    void Print(std::ostream &os);
    friend std::istream &operator>>(std::istream &is, Pentagon &object);
    friend std::ostream &operator<<(std::ostream &os, Pentagon &object);
    Pentagon &operator=(const Pentagon &object);
    bool operator==(const Pentagon &object);

};

```

```

#endif //MAI_OOP_PENTAGON_H

```

## pentagon.cpp

```

#include "pentagon.h"

```

```

#include <math.h>

```

```

Pentagon::Pentagon() : a_(0, 0), b_(0, 0), c_(0, 0), d_(0, 0), e_(0, 0) {}

```

```

Pentagon::Pentagon(const Pentagon &pentagon) {

```

```

    this->a_ = pentagon.a_;

```

```

    this->b_ = pentagon.b_;

```

```

    this->c_ = pentagon.c_;

```

```

    this->d_ = pentagon.c_;

```

```

    this->e_ = pentagon.c_;

```

```

}

```

```

Pentagon::Pentagon(std::istream &is) {

```

```

    std::cin >> a_ >> b_ >> c_ >> d_ >> e_;

```

```

}

```

```

size_t Pentagon::VertexesNumber() {

```

```

    return (size_t) 5;

```

}

*double* *Pentagon::Area()* {

*double p = fabs(a\_.get\_x()\*b\_.get\_y()-b\_.get\_x()\*a\_.get\_y()+b\_.get\_x()\*c\_.get\_y()-  
c\_.get\_x()\*b\_.get\_x()+c\_.get\_x()\*d\_.get\_y()-d\_.get\_x()\*c\_.get\_y()+d\_.get\_x()\*e\_.get\_y()-  
e\_.get\_x()\*d\_.get\_y()+e\_.get\_x()\*a\_.get\_y()-a\_.get\_x()\*e\_.get\_y())/2;*

*return p;*

}

*void* *Pentagon::Print(std::ostream &os)* {

*std::cout << "Pentagon " << a\_ << b\_ << c\_ << d\_ << e\_ << std::endl;*

}

*std::istream &operator>>(std::istream &is, Pentagon &object)* {

*is >> object.a\_ >> object.b\_ >> object.c\_ >> object.d\_ >> object.e\_;*

*return is;*

}

*std::ostream &operator<<(std::ostream &os, Pentagon &object)* {

*os << "a side = " << object.a\_.dist(object.b\_) << std::endl;*

*os << "b side = " << object.b\_.dist(object.c\_) << std::endl;*

*os << "c side = " << object.c\_.dist(object.d\_) << std::endl;*

*os << "d side = " << object.d\_.dist(object.e\_) << std::endl;*

*os << "e side = " << object.e\_.dist(object.a\_) << std::endl;*

*return os;*

}

*Pentagon &Pentagon::operator=(const Pentagon &object)* {

*this->a\_ = object.a\_;*

*this->b\_ = object.b\_;*

*this->c\_ = object.c\_;*

*this->d\_ = object.d\_;*

```

    this->e_ = object.e_;
    return *this;
}

```

```

bool Pentagon::operator==(const Pentagon &object) {
    if (this->a_ == object.a_ && this->b_ == object.b_ && this->c_ == object.c_ && this->d_ ==
object.d_ && this->e_ == object.e_) {
        return true;
    } else return false;
}

```

## tstack.h

```

#ifndef MAI_OOP_TSTACK_H
#define MAI_OOP_TSTACK_H
#include "pentagon.h"
#include "titerator.h"

template <class T>
class TStack {
private:
    struct StackItem {
        std::shared_ptr<T> data;
        std::shared_ptr<StackItem> next;
    };
    size_t size;
    std::shared_ptr<StackItem> top_;

public:
    TStack();
    TStack(const TStack<T> &stack);
    size_t Length();
    bool Empty();

```

```

    T Top();
    void Push(const std::shared_ptr<T> t);
    void Pop();
    void Clear();
    template<typename Y>
    friend std::istream &operator>>(std::istream &is, TStack<Y> &object);
    template<typename Y>
    friend std::ostream &operator<<(std::ostream &os, const TStack<Y> &object);

    TIterator<StackItem, T> top();

    virtual ~TStack();
};

#endif

```

## tstack.cpp

```
#include "tstack.h"
```

```

template<typename T>
TStack<T>::TStack() {
    top_ = std::shared_ptr<StackItem>(new StackItem);
    top_->next = nullptr;
    size = 0;
}

```

```

template<typename T>
TStack<T>::TStack(const TStack<T> &stack) {
    std::shared_ptr<StackItem> top = stack.top_;

    while (top->next != nullptr) {
        top_->data = top->data;
        std::shared_ptr<StackItem> item1(new StackItem);
        item1->next = nullptr;
        top_->next = item1;
        top = top->next;
    }
    size = stack.size;
}

```

```

template<typename T>
size_t TStack<T>::Length() {
    return (size_t) size;
}

```

```

template<typename T>
bool TStack<T>::Empty() {
    return (size == 0);
}

```

```

template <typename T>
T TStack<T>::Top() {
    return *(top_->data);
}

```

```

template<typename T>
void TStack<T>::Push(const std::shared_ptr<T> t) {
    std::shared_ptr<StackItem> item(new StackItem);
    item->data = t;
    item->next = top_;
    top_ = item;
    size++;
}

```

```

template<typename T>
void TStack<T>::Pop() {
    if (size == 0) {
        return;
    }
    std::shared_ptr<StackItem> item = this->top_;
    top_ = top_->next;
    size--;
}

```

```

template<typename T>
std::istream &operator>>(std::istream &is, TStack<T> &object) {
    std::shared_ptr<Pentagon> t(new Pentagon);
    is >> *t;
    object.Push(t);
    return is;
}

```

```

template<typename T>
std::ostream &operator<<(std::ostream &os, const TStack<T> &object) {
    std::shared_ptr<typename TStack<T>::StackItem> item = object.top_;

```

```

    os << "==" << " ";
    while (item->next != nullptr) {
        os << item->data->Area() << " ";
        item = item->next;
    }
    os << "==" << " ";
    return os;
}

```

```

template<typename T>
void TStack<T>::Clear() {
    while (top_->next != nullptr) {
        std::shared_ptr<StackItem> item = this->top_;
        top_ = top_->next;
    }
    size = 0;
    top_->next = nullptr;
}

```

```

template<typename T>
TStack<T>::~~TStack() {
    while (top_->next != nullptr) {
        std::shared_ptr<StackItem> item = this->top_;
        top_ = top_->next;
    }
    size = 0;
}

```

```

template <class T>
TIterator<typename TStack<T>::StackItem, T> TStack<T>::top() {
    return TIterator<typename TStack<T>::StackItem, T>(top_);
}

```

## titerator.h

```

#ifndef TITERATOR_H
#define TITERATOR_H

```

```

#include <iostream>
#include <memory>

```

```

template <class node,

```

```

class T>
class TIterator {
public:

TIterator(std::shared_ptr<node> n) { node_ptr =
n; }

std::shared_ptr<T>
operator*() { return
(node_ptr->data); }

std::shared_ptr<T>
operator->() { return
(node_ptr->data); }

void operator++() {
node_ptr = node_ptr-
>next; }

TIterator
operator++(int) {
    TIterator iter(*this);
    ++(*this);
    return iter;
}

bool
operator==(TIterator
const& i) { return
node_ptr == i.node_ptr;
}

bool

```

```
operator!=(TIterator  
const& i) { return !(*this  
== i); }
```

```
private:
```

```
std::shared_ptr<node>  
node_ptr;
```

```
};
```

```
#endif //  
TITERATOR_H
```

## **template.h**

```
#include "tstack.h"
```

```
#include "tstack.cpp"
```

```
template class  
TStack<Pentagon>;
```

```
template std::ostream&  
operator<<  
<Pentagon>(std::ostrea  
m&, TStack<Pentagon>  
const&);
```