

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

ЛАБОРАТОРНАЯ РАБОТА №5

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). Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру (колонка фигура 1), согласно вариантам задания. Классы должны удовлетворять следующим правилам:

1. Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
2. Требования к классу контейнера аналогичны требованиям из лабораторной работы 2.
3. Класс-контейнер должен содержать объекты используя `std::shared_ptr<...>`.
4. Классы должны быть расположены в отдельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

- Стандартные контейнеры `std`.
- Шаблоны (`template`).
- Объекты «по значению».

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 8 файлах:

`main.cpp`: тестирование кода

`figure.h`: родительский класс-интерфейс для фигур

`point.h`: описание класса точки

`point.cpp`: реализация класса точки

`pentagon.h`: описание класса пятиугольника, наследующегося от `figure`

`pentagon.cpp`: реализация класса пятиугольника

`tstack.h`: структура стэка

`tstack.cpp`: реализация стэка

Дневник отладки

Ошибок не выявлено.

Недочёты

Недочётов не заметил.

Вывод

В данной лабораторной работе был отредактирован код второй лабораторной работы - заменены все указатели на умные указатели `shared_ptr<>`, заменены элементы стека на указатели. Работа оказалась полезной, она позволяет узнать для себя новую единицу языка - умный указатель. Даёт понять, зачем это нужно и почему это удобно.

Исходный код

main.cpp

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

void menu(){
    std::cout << "Select an action" << std::endl;
    std::cout << "1) Print the top element" << std::endl;
    std::cout << "2) Remove an item from the stack" << std::endl;
    std::cout << "3) Print items from the stack" << std::endl;
    std::cout << "4) Clear the stack" << std::endl;
    std::cout << "5) Add an item to the stack" << std::endl;
    std::cout << "6) Print the stack length" << std::endl;
    std::cout << "7) Is the stack empty?" << std::endl;
    std::cout << "8) End the program" << std::endl;
}

int main() {
    std::cout.setf(std::ios_base::boolalpha);
    TStack stack;
    char k = 'y';
    menu();
    std::cin >> k;
    while (k != EOF) {
        switch (k) {
            case '1': {
                if(!stack.Empty()) std::cout << stack.Top() << std::endl;
                else std::cout << std::endl;
                break;
            }
        }
    }
}
```

```

    case '2': stack.Pop();
        break;
    case '3': std::cout << stack << std::endl;
        break;
    case '4': stack.Clear();
        break;
    case '5': stack.Push(std::shared_ptr<Pentagon>(new Pentagon(std::cin)));
        break;
    case '6':std::cout << stack.Length() << std::endl;
        break;
    case '7':std::cout << (bool)stack.Empty() << std::endl;
        break;
    case '8': {
        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.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"

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

public:
    TStack();
    TStack(const TStack &stack);
    size_t Length();
    bool Empty();
    Pentagon &Top();
    void Push(const std::shared_ptr<Pentagon> t);
    void Pop();
    void Clear();
    friend std::istream &operator>>(std::istream &is, TStack &object);
    friend std::ostream &operator<<(std::ostream &os, const TStack &object);

    virtual ~TStack();
};

#endif

```

tstack.cpp

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

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

```
TStack::TStack(const TStack &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;  
}
```

```
size_t TStack::Length() {  
    return (size_t) size;  
}
```

```
bool TStack::Empty() {  
    return (size == 0);  
}
```

```

Pentagon &TStack::Top() {
    return *(top_->data);
}

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

void TStack::Pop() {
    if (size == 0) {
        std::cout << "Unable to perform pop! The queue is empty!" << std::endl;
        return;
    }
    std::shared_ptr<StackItem> item = this->top_;
    top_ = top_->next;
    size--;
}

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

```

```

std::ostream &operator<<(std::ostream &os, const TStack &object) {
    std::shared_ptr<TStack::StackItem> item = object.top_;
    os << "==" << " ";
    while (item->next != nullptr) {
        os << item->data->Area() << " ";
        item = item->next;
    }
    os << "==" << " ";
    return os;
}

```

```

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

```

```

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

```