

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

**ЛАБОРАТОРНАЯ РАБОТА №3**

по курсу объектно-ориентированное программирование 3 семестр, 2021/22 уч. Год

Студент Абросимов Алексей Дмитриевич, группа М8О-207Б-20  
Преподаватель Дорохов Евгений Павлович

## Условие

Задание: Вариант 3: Динамический массив и прямоугольник

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: □

Требования к классу фигуры аналогичны требованиям из лабораторной работы №1; □

Требования к классу контейнера аналогичны требованиям из лабораторной работы №2; □

Класс-контейнер должен содержать объекты используя `std::shared_ptr<...>`.

## Описание программы

Исходный код лежит в 6 файликах:

1. main.cpp — основная программа, направленная на взаимодействие с пользователем.

2. rectangle.h — описание класса прямоугольник, который наследуется от фигуры.

3. rectangle.cpp — описание методов прямоугольника

4. tvector.h — описание класса вектора

5. tvector.cpp — описание методов вектора

6. figure.h — описание абстрактного класса фигуры

## Дневник отладки

В процессе выполнения данной ЛР столкнулся с непониманием, как работать с этими умными указателями. Не сразу пришло осознание того, что, как и при работе с обычными указателями, можно делать умный указатель на умный указатель. Как только я это понял, я немного переписал класс динамического массива.

Результат работы программы:

```
Rectangle #1coords is Rectangle coords (0,0) (1,1) (0,0) (0,0)
```

```
Rectangle #2coords is Rectangle coords (1,5) (2,6) (3,7) (4,8)
```

```
2
```

```
Rectangle was deleted
```

```
Rectangle was deleted
```

## Недочёты

## Выводы

Данная лабораторная работа позволила мне ознакомиться с такой удивительной вещью, как умные указатели. Умные указатели позволяют писать меньше строк кода и избежать различные утечки памяти.

Ссылка на гитхаб: [https://github.com/yungalexkey/oop\\_labs/tree/main/lab3](https://github.com/yungalexkey/oop_labs/tree/main/lab3)

## Исходный код

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H
#include <iostream>
class figure {
public:
    virtual double Area()=0;
    virtual size_t VertexesNumber()=0;
    virtual bool isit()=0;
};
```

```
#endif // FIGURE_H
main.cpp
```

```
#include <iostream>
#include "rectangle.h"
#include "Figure.h"
#include "tvector.h"
int main()
{
    TVector container;
    container.push_back(std::shared_ptr<rectangle>(new rectangle));
    container.push_back(std::shared_ptr<rectangle>(new rectangle(1,2,3,4,5,6,7,8)));
    std::shared_ptr<rectangle> t;
    std::cout<<container<<std::endl;
    std::cout<<container.size()<<std::endl;
    container.clear();
    std::cout<<container<<std::endl;
    return 0;
}
```

```
rectangle.h
```

```
#ifndef RECTANGLE_H
#define RECTANGLE_H
#include "Figure.h"
#include <iostream>

class rectangle:public figure{
public:
    rectangle();
    rectangle(int x1,int x2,int x3,int x4,int y1,int y2,int y3,int y4);
    size_t VertexesNumber();
    double Area();
    bool isit();
    friend std::ostream& operator<<(std::ostream &out, const rectangle &rec);
    friend std::istream& operator>>(std::istream &in, rectangle &rec);
    rectangle& operator= (rectangle &rec1);
    virtual ~rectangle();
private:
    double x1;
    double y1;
    double x2;
    double y2;
    double x3;
    double y3;
    double x4;
    double y4;
};

#endif // RECTANGLE_H
```

```
rectangle.cpp
```

```
#include "rectangle.h"
#include <math.h>

rectangle::rectangle():x1(0),y1(0),x2(1),y2(1),x3(0),y3(0),x4(0),y4(0){
}
rectangle::rectangle(int x1,int x2,int x3,int x4,int y1,int y2,int y3,int y4){
    this->x1=x1;
    this->x2=x2;
    this->x3=x3;
    this->x4=x4;
    this->y1=y1;
    this->y2=y2;
```

```

this->y3=y3;
this->y4=y4;
}
size_t rectangle::VertexesNumber(){
return 4;
}
rectangle::~rectangle(){
std::cout<<"Rectangle was deleted\n";
}

```

```

bool rectangle::isit(){
double perp;
double perp2;
perp=(x4-x1)*(x2-x1)+(y4-y1)*(y2-y1);
perp2=(x3-x4)*(x3-x2)+(y3-y4)*(y3-y2);
if((perp+perp2)==0) return true;
else return false;
}
double rectangle::Area(){
double r1 = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
double r2 = sqrt((x2 - x3) * (x2 - x3) + (y2 - y3) * (y2 - y3));
double r3 = sqrt((x1 - x3) * (x1 - x3) + (y1 - y3) * (y1 - y3));
double p=(r1+r2+r3)/2;
double s= 2*sqrt((p * (p - r1) * (p - r2) * (p - r3)));
return s;
}

```

```

std::ostream& operator<<(std::ostream &out, const rectangle &rec){
out << "Rectangle coords " << "(" << rec.x1 << ", " << rec.y1 << ")" << " " << "(" << rec.x2 << ", " <<
rec.y2 << ")" << " " << "(" << rec.x3 << ", " << rec.y3 << ")" << " " << "(" << rec.x4 << ", " << rec.y4 <<
)" << std::endl;
return out;
}
std::istream& operator>>(std::istream &in, rectangle &rec){
in >> rec.x1;
in >> rec.y1;
in >> rec.x2;
in >> rec.y2;
in >> rec.x3;
in >> rec.y3;
in >> rec.x4;
in >> rec.y4;
return in;
}
rectangle& rectangle::operator= (rectangle &rec){
this->x1=rec.x1;
this->x2=rec.x2;
this->x3=rec.x3;
this->x4=rec.x4;
this->y1=rec.y1;
this->y2=rec.y2;
this->y3=rec.y3;
this->y4=rec.y4;
return *this;
}

```

tvector.cpp

```

#include "tvector.h"
#include "rectangle.h"
TVector::TVector():length(0),count(0)
{ }

```

```

int TVector::size(){
return this->length;
}

```

```

}
bool TVector::empty(){
if(this->length>0) return true;
else return false;
}

void TVector::push_back(std::shared_ptr<rectangle> newrec){
if(count==length){
length++;
count++;
std::shared_ptr<std::shared_ptr<rectangle>>[]> narr(new std::shared_ptr<rectangle>[length]);
for(int i=0;i<length-1;i++) narr[i]=arr[i];

narr[length-1]= newrec;
//free(arr);
arr=narr;
}
else if(count<length){
arr[count]=newrec;
count++;
}
}
TVector::~TVector(){

}

std::shared_ptr<rectangle> TVector::pop_back(){
std::shared_ptr<std::shared_ptr<rectangle>>[]> narr(new std::shared_ptr<rectangle>[length]);
for(int i=0;i<count-1;i++){
narr[i]=arr[i];
}
std::shared_ptr<rectangle> tmp=arr[count-1];
count--;
length--;
arr=narr;
return tmp;
}

void TVector::resize(int newlength){
if(newlength==length) return;
if(newlength>length){
std::shared_ptr<std::shared_ptr<rectangle>>[]> narr(new std::shared_ptr<rectangle>[length]);
for(int i=0;i<length;i++)
narr[i]=arr[i];
arr=narr;
length=newlength;
}
else {
std::shared_ptr<std::shared_ptr<rectangle>>[]> narr(new std::shared_ptr<rectangle>[length]);
for(int i=0;i<newlength;i++)
narr[i]=arr[i];
arr=narr;
count=newlength;
}
}

void TVector::clear(){
resize(1);
pop_back();
length=0;
count=0;
}

void TVector::erase(int pos){
if(count==0)

```

```

{
std::cout<<"Container is empty"<<std::endl;
return;
}
std::shared_ptr<std::shared_ptr<rectangle>[]> narr(new std::shared_ptr<rectangle>[length]);
int current_index=0;
for(int i=0;i<count;i++){
if(i!=pos-1) {
narr[current_index]=arr[i];
current_index++;
}
}
count--;
length--;
arr=narr;
}

//перепрыжка операций
std::shared_ptr<rectangle> TVector::operator[] (int i)
{
if(i >= 0 && i < this->length)
return this->arr[i];
}

```

```

std::ostream& operator<<(std::ostream &out, TVector &cont){
for(int i=0;i<cont.count;i++){
out<<"Rectangle #"<< i+1<<"coords is " << *cont[i];
}
return out;
}

```

tvector.h

```

#ifndef TVECTOR_H
#define TVECTOR_H
#include "rectangle.h"
#include <memory>

class TVector
{
private:
int length;
int count;
std::shared_ptr<std::shared_ptr<rectangle>[]> arr;
public:
TVector();
~TVector();
int size();
bool empty();
void resize(int nindex);
void push_back(std::shared_ptr<rectangle> newrec);
void erase(int pos);
std::shared_ptr<rectangle> pop_back();
void clear();

std::shared_ptr<rectangle> operator[] (int i) ;
friend std::istream& operator>>(std::istream &in, TVector &cont);
friend std::ostream& operator<<(std::ostream &out, TVector &cont);
};

#endif // TVECTOR_H

```