

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 Медведев Данила Андреевич, группа М80-208Б-20  
Преподаватель Дорохов Евгений Павлович

## Цель работы

Целью лабораторной работы является:

Закрепление навыков работы с классами.

Знакомство с умными указателями.

## Задание

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий **одну** фигуру класса фигуры, 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<...>`.

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.

Вариант №11

Фигура 1: Прямоугольник (Rectangle)

Структура: Связный список

## Дневник отладки

## Недочёты

## Выводы

Продолжил изучение базовых понятий ооп. Ознакомился с умными указателями. Закрепил навыки работы с классами.

## Исходный код

### figure.h

```
#ifndef FIGURE_H
#define FIGURE_H
#include "point.h"

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

#endif
```

### main.cpp

```
#include "rectangle.h"
#include "tlinkedlist_i.h"
#include "tlinkedlist.h"
#include "point.h"
#include <iostream>

using namespace std;

int main() {
```

```

    Point a;
    a.setX(1);
    a.setY(1);
    Point b(2, 2);
    Point c(3, 3);
    Point d(4, 4);

    Rectangle rc(c, a, b, d);
    Rectangle rc1(a, b, c, d);
    Rectangle rc2(d, b, c, d);

    cout << b << endl;
    cout << rc << endl;

    shared_ptr<Rectangle> rec = make_shared<Rectangle>(rc);
    shared_ptr<Rectangle> rec1 = make_shared<Rectangle>(rc1);
    shared_ptr<Rectangle> rec2 = make_shared<Rectangle>(rc2);

    TLinkedList list;
    list.InsertFirst(rec);
    list.InsertFirst(rec1);
    list.InsertLast(rec2);
    cout << list << endl;
    list.RemoveFirst();
    list.RemoveLast();
    cout << list.Length() << endl;
    cout << list << endl;
    cout << list.Empty() << endl;
    list.RemoveLast();
    cout << list.Empty() << endl;

    return 0;
}

```

## rectangle.cpp

```

#include <iostream>
#include "point.h"
#include "rectangle.h"
using namespace std;

Rectangle::Rectangle() : a(0.0, 0.0), b(0.0, 0.0), c(0.0, 0.0), d(0.0, 0.0), len1(0),
len2(0), square(0.0)
{
};

Rectangle& Rectangle::operator= (Rectangle rectangle)
{
    a = rectangle.a;

```

```

        b = rectangle.b;
        c = rectangle.c;
        d = rectangle.d;
        len1 = dist(a, b);
        len2 = dist(b, c);
        square = len1 * len2;
        return rectangle;
};

double Rectangle::Square()
{
    return square;
}

Rectangle& Rectangle::operator= (Rectangle rectangle)
{
    a = rectangle.a;
    b = rectangle.b;
    c = rectangle.c;
    d = rectangle.d;
    len1 = dist(a, b);
    len2 = dist(b, c);
    square = len1 * len2;
    return rectangle;
};

void Rectangle::Print(std::ostream& os)
{
    std::cout << "Rectangle: " << a << " " << b << " " << c << " " << d << endl;
}

size_t Rectangle::VertexesNumber()
{
    return 4;
}

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

    cin >> a >> b >> c >> d;
}

std::istream& operator >>(std::istream& is, Rectangle& rectangle)
{
    is >> rectangle.a >> rectangle.b >> rectangle.c >> rectangle.d;
    return is;
};

std::ostream& operator <<(std::ostream& os, Rectangle rectangle)
{
    os << rectangle.a << " " << rectangle.b << " " << rectangle.c << " " << rectangle.d
    << "\n";
    return os;
};

bool Rectangle::operator== (Rectangle rectangle)
{
    if ((a == rectangle.a) && (b == rectangle.b) && (c == rectangle.c) && (d ==
rectangle.d))
    {
        return true;
    }
}

```

```

    }
    return false;
};

```

## rectangle.h

```

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

class Rectangle : public Figure
{
public:
    Rectangle();
    Rectangle(std::istream& is);
    void Print(std::ostream& os);
    double Square();
    friend std::istream& operator >>(std::istream& is, Rectangle&
rectangle);
    friend std::ostream& operator <<(std::ostream& os, Rectangle
rectangle);
    Rectangle& operator= (Rectangle rectangle);
    bool operator== (Rectangle rectangle);
    size_t VertexesNumber();
    virtual ~Rectangle();

private:
    Point a, b, c, d;
    double len1, len2;
    double square;
};

#endif

```

## point.h

```

#ifndef POINT_H
#define POINT_H
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <algorithm>

class Point {
public:

```

```

    Point();
    Point(std::istream& is);
    Point(double x, double y);
    double length(Point& p1, Point& p2);
    friend std::istream& operator>>(std::istream& is, Point& p);
    friend std::ostream& operator<<(std::ostream& os, Point& p);
    bool operator==(Point point);
    friend double dist(Point& p1, Point& p2);

private:
    double x_, y_;
};

#endif

```

## point.cpp

```

#include "point.h"

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 dist(Point& p1, Point& p2){
    double dx = (p1.x_ - p2.x_);
    double dy = (p1.y_ - p2.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, const Point& p) {
    os << "(" << p.x_ << ", " << p.y_ << ")";
    return os;
}

bool Point::operator == (Point point){
    return (x_ == point.x_) && (y_ == point.y_);
}

```

## tlinkedlist.cpp

```

#include "tlinkedlist.h"

TLinkedList::TLinkedList() {
    len = 0;
    head = nullptr;
}

TLinkedList::TLinkedList(const TLinkedList& list) {
    len = list.len;
    if (!list.len) {
        head = nullptr;
        return;
    }
    head = make_shared<TLinkedListItem>(list.head->GetVal(), nullptr);
    shared_ptr<TLinkedListItem> cur = head;
    shared_ptr<TLinkedListItem> it = list.head;
    for (size_t i = 0; i < len - 1; ++i) {
        it = it->GetNext();
        shared_ptr<TLinkedListItem> new_item =
make_shared<TLinkedListItem>(it->GetVal(), nullptr);
        cur->SetNext(new_item);
        cur = cur->GetNext();
    }
}

shared_ptr<Rectangle> TLinkedList::First() {
    if (!len) {
        return nullptr;
    }
    return head->GetVal();
}

shared_ptr<Rectangle> TLinkedList::Last() {
    if (!len) {
        return nullptr;
    }
    shared_ptr<TLinkedListItem> cur = head;
    for (size_t i = 0; i < len - 1; ++i) {
        cur = cur->GetNext();
    }
    return cur->GetVal();
}

void TLinkedList::InsertFirst(shared_ptr<Rectangle> rectangle) {
    shared_ptr<TLinkedListItem> it = make_shared<TLinkedListItem>(rectangle,
head);
    head = it;
    len++;
}

```



```

}

void TLinkedList::InsertLast(shared_ptr<Rectangle> rectangle) {
    if (!len) {
        head = make_shared<TLinkedListItem>(rectangle, nullptr);
        len++;
        return;
    }
    shared_ptr<TLinkedListItem> cur = head;
    for (size_t i = 0; i < len - 1; ++i) {
        cur = cur->GetNext();
    }
    shared_ptr<TLinkedListItem> it = make_shared<TLinkedListItem>(rectangle,
nullptr);
    cur->SetNext(it);
    len++;
}

void TLinkedList::Insert(shared_ptr<Rectangle> rectangle, size_t pos) {
    if (pos > len || pos < 0) return;
    shared_ptr<TLinkedListItem> cur = head;
    shared_ptr<TLinkedListItem> prev = nullptr;
    for (size_t i = 0; i < pos; ++i) {
        prev = cur;
        cur = cur->GetNext();
    }
    shared_ptr<TLinkedListItem> it = make_shared<TLinkedListItem>(rectangle,
cur);
    if (prev) {
        prev->SetNext(it);
    }
    else {
        head = it;
    }
    len++;
}

void TLinkedList::RemoveFirst() {
    if (!len) return;
    shared_ptr<TLinkedListItem> del = head;
    head = head->GetNext();
    len--;
}

void TLinkedList::RemoveLast() {
    if (!len) return;
    if (len == 1) {
        head = nullptr;
        len = 0;
        return;
    }

```

```

    }
    shared_ptr<TLinkedListItem> cur = head;
    for (size_t i = 0; i < len - 2; ++i) {
        cur = cur->GetNext();
    }
    shared_ptr<TLinkedListItem> del = cur->GetNext();
    cur->SetNext(nullptr);
    len--;
}

void TLinkedList::Remove(size_t pos) {
    if (!len) return;
    if (pos < 0 || pos >= len) return;
    shared_ptr<TLinkedListItem> cur = head;
    shared_ptr<TLinkedListItem> prev = nullptr;
    for (size_t i = 0; i < pos; ++i) {
        prev = cur;
        cur = cur->GetNext();
    }
    if (prev) {
        prev->SetNext(cur->GetNext());
    }
    else {
        head = cur->GetNext();
    }
    len--;
}

shared_ptr<Rectangle> TLinkedList::GetItem(size_t ind) {
    if (ind < 0 || ind >= len) {
        return nullptr;
    }
    shared_ptr<TLinkedListItem> cur = head;
    for (size_t i = 0; i < ind; ++i) {
        cur = cur->GetNext();
    }
    return cur->GetVal();
}

bool TLinkedList::Empty() {
    return len == 0;
}

size_t TLinkedList::Length() {
    return len;
}

std::ostream& operator<<(std::ostream& os, const TLinkedList& list) {
    shared_ptr<TLinkedListItem> cur = list.head;
    os << "List: \n";

```

```

        for (size_t i = 0; i < list.len; ++i) {
            os << *cur;
            cur = cur->GetNext();
        }
        return os;
    }

    void TLinkedList::Clear() {
        while (!(this->Empty())) {
            this->RemoveFirst();
        }
    }

    TLinkedList::~TLinkedList() {
        while (!(this->Empty())) {
            this->RemoveFirst();
        }
    }
}

```

## tlinkedlist.h

```

#pragma once
#include "rectangle.h"
#include "tlinkedlist_i.h"
#include "iostream"

class TLinkedList {
private:
    size_t len;
    shared_ptr<TLinkedListItem> head;
public:
    TLinkedList();

    TLinkedList(const TLinkedList& list);

    shared_ptr<Rectangle> First();

    shared_ptr<Rectangle> Last();

    void InsertFirst(shared_ptr<Rectangle> rectangle);

    void InsertLast(shared_ptr<Rectangle> rectangle);

    void Insert(shared_ptr<Rectangle> rectangle, size_t pos);

    void RemoveFirst();

    void RemoveLast();
}

```

```

    void Remove(size_t pos);

    shared_ptr<Rectangle> GetItem(size_t ind);

    bool Empty();

    size_t Length();

    friend std::ostream& operator<<(std::ostream& os, const TLinkedList&
list);

    void Clear();

    virtual ~TLinkedList();
};

```

## tlinkedlist\_i.cpp

```

#include "tlinkedlist_i.h"

TLinkedListItem::~TLinkedListItem() {

}

TLinkedListItem::TLinkedListItem(shared_ptr<Rectangle> rectangle,
shared_ptr<TLinkedListItem> nxt) {
    val = rectangle;
    next = nxt;
}

shared_ptr<TLinkedListItem> TLinkedListItem::GetNext() {
    return next;
}

void TLinkedListItem::SetNext(shared_ptr<TLinkedListItem> nxt) {
    next = nxt;
}

shared_ptr<Rectangle> TLinkedListItem::GetVal() {
    return val;
}

std::ostream& operator<<(std::ostream& os, const TLinkedListItem& item) {
    os << "[" << *item.val << "]" ";
    return os;
}

```

## tlinkedlist\_i.h

```
#pragma once
#include "rectangle.h"
#include "iostream"
#include "memory"
using std::shared_ptr;
using std::make_shared;

class TLinkedListItem {
private:
    shared_ptr<Rectangle> val;
    shared_ptr<TLinkedListItem> next;
public:
    TLinkedListItem(shared_ptr<Rectangle> rectangle, shared_ptr<TLinkedListItem> nxt);

    void SetNext(shared_ptr<TLinkedListItem> nxt);

    shared_ptr<TLinkedListItem> GetNext();

    shared_ptr<Rectangle> GetVal();

    friend std::ostream& operator<<(std::ostream& os, const TLinkedListItem& item);

    virtual ~TLinkedListItem();
};
```