

# Лабораторная работа №4

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: Капичников Ярослав Андреевич, группа М80-207Б-20

Преподаватель: Дорохов Евгений Павлович, каф. 806

## Задание:

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.
- Классы фигур должны содержать набор следующих методов:
  - Перегруженный оператор ввода координат вершин фигуры из потока `std::istream (>>)`; Он должен заменить конструктор, принимающий координаты вершин из стандартного потока;
  - Перегруженный оператор вывода в поток `std::ostream (<<)`, заменяющий метод `Print` из лабораторной работы 1;
  - Оператор копирования `(=)`;
  - Оператор сравнения с такими же фигурами `(==)`.
- Класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке);
- Класс-контейнер должен содержать набор следующих методов:

Метод по добавлению фигуры в контейнер	Метод по получению фигуры из контейнера	Метод по удалению фигуры из контейнера
Очередь: <b>Push</b> Динамический массив: <b>InsertLast</b> Связанный список: <b>InsertFirst, InsertLast, Insert</b> Бинарное дерево: <b>Push</b> N-дерево: <b>Update</b>	Очередь: <b>Top</b> Динамический массив: <b>operator[]</b> Связанный список: <b>First, Last, GetElement</b> Бинарное дерево: <b>GetNotLess</b> N-дерево: <b>GetItem</b>	Очередь: <b>Pop</b> Динамический массив: <b>Remove</b> Связанный список: <b>RemoveFirst, RemoveLast, Remove</b> Бинарное дерево: <b>Pop</b> N-дерево: <b>RemoveSubTree</b>
<ul style="list-style-type: none"> <li>• Перегруженный оператор по выводу контейнера в поток <code>std::ostream (&lt;&lt;)</code>;</li> <li>• Деструктор, удаляющий все элементы контейнера;</li> <li>• Набор специальных методов для класса-контейнера (см. Приложение).</li> </ul>		

Полное описание всех методов можно найти в приложении к лабораторной.

Нельзя использовать:

- Стандартные контейнеры `std`;
- Шаблоны (`template`);
- Различные варианты умных указателей (`unique_ptr`, `shared_ptr`, `weak_ptr`,...).

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

## Описание программы:

Исходный код разделён на несколько файлов:

- `point.h(cpp)` – описание и реализация класса точки.
- `figure.h(cpp)` – описание и реализация класса фигуры.
- `rectangle.h(cpp)` – описание и реализация класса прямоугольника (наследуется от фигуры).
- `tlinkedlist.h(cpp)` - описание и реализация класса связного списка.
- `tlinkedlist_i.h(cpp)` – описание и реализация класса отдельного элемента списка.

№	Дата	Событие	Действие по исправлению
1	01.11.21	Обнаружил, ошибку при выводе списка.	Исправил путем добавления const в функцию вывода.

## Вывод:

Проделав данную работу, я продолжил изучение базовых понятий ооп. В данном варианте мне пришлось вспомнить динамическую структуру, изученную в прошлом семестре, а именно связный список. Но в отличие от прошлого семестра здесь элементами списка являются не числа или буквы, а фигуры. В целом приятно осознавать, что проделанная в прошлом работа пригодилась мне сейчас.

## Исходный код:

### Figure.h

```
#pragma once
```

```
#include <iostream>
#include "point.h"
using namespace std;
class Figure {
public:
    virtual size_t VertexesNumber() = 0;
    virtual double Area() = 0;
    virtual void Print(std::ostream& os) = 0;
protected:
    Point a;
    Point b;
    Point c;
    Point d;
};
```

### 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::dist(Point& other) {
    double dx = (other.x_ - x_);
    double dy = (other.y_ - y_);
    return std::sqrt(dx * dx + dy * dy);
}
double Point::getX()
{
    return x_;
}
double Point::getY()
{
    return y_;
}
```

```

        return y_;
    }
    void Point::setX(double a)
    {
        x_ = a;
    }
    void Point::setY(double a)
    {
        y_ = a;
    }
    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 operator== (Point& p1, Point& p2)
    {
        return (p1.getX() == p2.getY() &&
                p1.getY() == p2.getY());
    }
    bool operator!= (Point& p1, Point& p2)
    {
        return !(p1 == p2);
    }
}

```

## Point.h

```

#pragma
once

```

```

#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point {
public:
    Point();
    Point(std::istream& is);
    Point(double x, double y);
    double dist(Point& other);
    double getX();
    double getY();
    void setX(double a);
    void setY(double a);
    friend std::istream& operator>>(std::istream& is, Point& p);
    friend std::ostream& operator<<(std::ostream& os, const Point& p);
    friend bool operator== (Point& p1, Point& p2);
    friend bool operator!= (Point& p1, Point& p2);
private:
    double x_;
    double y_;
};

#endif

```

## Rectangle.cpp

```

#include
<iostream>

#include "point.h"

#include "rectangle.h"

using namespace std;

```

```

Rectangle::Rectangle(Point a1, Point a2, Point a3, Point a4) {

    a = a1;

    b = a2;

    c = a3;

    d = a4;

}

```

```

Rectangle::Rectangle() {

    a.setX(0);

    a.setY(0);

    b.setX(0);

    b.setY(0);

    c.setX(0);

    c.setY(0);

    d.setX(0);

    d.setY(0);

}

```

```

double Rectangle::Area() {

    double A = a.dist(b);

    double B = b.dist(c);

    return A * B;

}

```

```

void Rectangle::Print(std::ostream& os)

{

    std::cout << "Rectangle: " << a << " " << b << " " << c << " " << d << endl;

}

```

```

size_t Rectangle::VertexesNumber()

{

    return (size_t)4;

}

```

```

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

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

}

```

```

    }

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

        is >> p.a >> p.b >> p.c >> p.d;

        return is;

    }

    std::ostream& operator<<(std::ostream& os,const Rectangle& p) {

        os << p.a << " " << p.b << " " << p.c << " " << p.d;

        return os;

    }

    bool operator==(Rectangle& p1, Rectangle& p2)

    {

        return (p1.a == p2.a &&

                p1.b == p2.b && p1.c == p2.c && p1.d == p2.d);

    }

    bool operator!=(Rectangle& p1, Rectangle& p2)

    {

        return !(p1 == p2);

    }

```

## Rectangle.h

```

#pragma
once

```

```

#include <iostream>
#include"point.h"
#include"figure.h"
class Rectangle : Figure {
public:

    double Area();
    void Print(std::ostream& os);
    size_t VertexesNumber();
    Rectangle(Point a1, Point a2, Point a3, Point a4);
    Rectangle(std::istream& is);
    Rectangle();

    friend std::istream& operator>>(std::istream& is, Rectangle& p);
    friend std::ostream& operator<<(std::ostream& os,const Rectangle& p);
    friend bool operator==(Rectangle& r1, Rectangle& r2);
    friend bool operator!=(Rectangle& r1, Rectangle& r2);

private:

};

```

## 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 = new TLinkedListItem(list.head->GetVal(), nullptr);
    TLinkedListItem* cur = head;
    TLinkedListItem* it = list.head;
    for (size_t i = 0; i < len - 1; ++i) {
        it = it->GetNext();
        TLinkedListItem* new_item = new TLinkedListItem(it->GetVal(), nullptr);
        cur->SetNext(new_item);
        cur = cur->GetNext();
    }
}

const Rectangle& TLinkedList::First() {
    if (!len) {
        std::cout << "The list is empty!\n";
        return Rectangle();
    }
    return head->GetVal();
}

const Rectangle& TLinkedList::Last() {
    TLinkedListItem* cur = head;
    for (size_t i = 0; i < len - 1; ++i) {
        cur = cur->GetNext();
    }
    return cur->GetVal();
}

void TLinkedList::InsertFirst(const Rectangle& rectangle) {
    TLinkedListItem* it = new TLinkedListItem(rectangle, head);
    head = it;
    len++;
}

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

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

void TLinkedList::RemoveFirst() {

```

```

        if (!len) return;
        TLinkedListItem* del = head;
        head = head->GetNext();
        delete del;
        len--;
    }

    void TLinkedList::RemoveLast() {
        if (!len) return;
        if (len == 1) {
            head = nullptr;
            len = 0;
            return;
        }
        TLinkedListItem* cur = head;
        for (size_t i = 0; i < len - 2; ++i) {
            cur = cur->GetNext();
        }
        TLinkedListItem* del = cur->GetNext();
        cur->SetNext(nullptr);
        delete del;
        len--;
    }

    void TLinkedList::Remove(size_t pos) {
        if (!len) return;
        if (pos < 0 || pos >= len) return;
        TLinkedListItem* cur = head;
        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();
        }
        delete cur;
        len--;
    }

    const Rectangle& TLinkedList::GetItem(size_t ind) {
        if (ind < 0 || ind >= len) {
            std::cout << "NOT FOUND\n";
            return Rectangle();
        }
        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) {
        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;

    TLinkedListItem* head;

public:

    TLinkedList();

    TLinkedList(const TLinkedList& list);

    const Rectangle& First();

    const Rectangle& Last();

    void InsertFirst(const Rectangle& rectangle);

    void InsertLast(const Rectangle& rectangle);

    void Insert(const Rectangle& rectangle, size_t pos);

    void RemoveFirst();

    void RemoveLast();

    void Remove(size_t pos);

    const 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(const Rectangle& rectangle, TLinkedListItem* nxt) {

    val = rectangle;

    next = nxt;

}

TLinkedListItem* TLinkedListItem::GetNext() {

    return next;

}

void TLinkedListItem::SetNext(TLinkedListItem* nxt) {

    next = nxt;

}

const 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"
```

```
class TLinkedListItem {
```

```
private:
```

```
    Rectangle val;
```

```
    TLinkedListItem* next;
```

```
public:
```

```
    TLinkedListItem(const Rectangle& rectangle, TLinkedListItem* nxt);
```

```
    void SetNext(TLinkedListItem* nxt);
```

```
    TLinkedListItem* GetNext();
```

```
    const Rectangle& GetVal();
```

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

```
    virtual ~TLinkedListItem();
```

```
};
```