# Лабораторная работа №3

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: *Медведев Данила Андреевич, М8О-208Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

**Задание:**

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;
- Требования к классу контейнера аналогичны требованиям из лабораторной работы №2;
- Класс-контейнер должен содержать объекты используя std::shared_ptr.

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

**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(с помощью умных указателей).
- tlinkedlist_i.h(cpp ) – описание и реализация класса отдельного элемента списка(с помощью умных указателей).

**Дневник отладки**

| № | Дата | Событие | Действие по исправлению |
|---|---|---|---|
| 1 | | | |

**Вывод:**

Проделав данную работу, я продолжил изучение базовых понятий ооп. Теперь я ознакомился с умными указателями. Их главная суть заключается в том, что они сами могут удалять выделенную им память. И это очень полезная функция, особенно для крупных программ, над которыми трудится несколько человек.

**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_;
}

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 = 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();
};
```