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

ЛАБОРАТОРНАЯ РАБОТА №6 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:

Знакомство с шаблонами классов;

Построение шаблонов динамических структур данных.

Задание

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

· Требования к классам фигуры аналогичны требованиям из лабораторной работы №1;

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

·Шаблон класса-контейнера должен содержать объекты используя std::shared_ptr<…>.

Нельзя использовать:

· Стандартные контейнеры std.

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. Познакомился с шаблонами классов.

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<Rectangle> 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 "rectangle.h"

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 = rectangle.len1;
        len2 = rectangle.len2;
        square = rectangle.square;
        return rectangle;
};

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

```
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){
	is >> a >> b >> c >> d;
	len1 = dist(a, b);
	len2 = dist(b, c);
	square = len1 * len2;
}

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::~Rectangle()
{
}
```

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.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_);
}
```

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
```

tlinkedlist.cpp

```
#include "tlinkedlist.h"

template<typename T>
TLinkedList<T>::TLinkedList() {
    len = 0;
    head = nullptr;
}

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

```
template<typename T>
shared_ptr<T> TLinkedList<T>::First() {
    if (len == 0) {
        return nullptr;
    }
    return head->GetVal();
}

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

template<typename T>
void TLinkedList<T>::InsertFirst(shared_ptr<T> figure) {
    shared_ptr<TLinkedListItem<T>> it = make_shared<TLinkedListItem<T>>(figure, head);
    head = it;
    len++;
}

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

template<typename T>
void TLinkedList<T>::Insert(shared_ptr<T> figure, size_t pos) {
    if (pos > len || pos < 0) {
        return;
    }
    shared_ptr<TLinkedListItem<T>> cur = head;
```

```
    shared_ptr<TLinkedListItem<T>> prev = nullptr;
    for (size_t i = 0; i < pos; ++i) {
        prev = cur;
        cur = cur->GetNext();
    }
    shared_ptr<TLinkedListItem<T>> it = make_shared<TLinkedListItem<T>>(figure, cur);
    if (prev) {
        prev->SetNext(it);
    }
    else {
        head = it;
    }
    len++;
}

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

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

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

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

template<typename T>
bool TLinkedList<T>::Empty() {
    return len == 0;
}

template<typename T>
size_t TLinkedList<T>::Length() {
    return len;
}

template<typename T>
std::ostream& operator<<(std::ostream& os, const TLinkedList<T>& list) {
    shared_ptr<TLinkedListItem<T>> cur = list.head;
    os << "List: \n";
    for (size_t i = 0; i < list.len; ++i) {
        os << *cur;
        cur = cur->GetNext();
    }
    return os;
}

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

template<typename T>
```

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

}

template
class TLinkedList<Rectangle>;

template std::ostream& operator<<(std::ostream& os, const TLinkedList<Rectangle>& list);
```

tlinkedlist.h

```
#pragma once
#include "tlinkedlist_i.h"

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

    TLinkedList(const TLinkedList<T>& list);

    shared_ptr<T> First();

    shared_ptr<T> Last();

    void InsertFirst(shared_ptr<T> rectangle);

    void InsertLast(shared_ptr<T> rectangle);

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

    void RemoveFirst();

    void RemoveLast();

    void Remove(size_t pos);

    shared_ptr<T> GetItem(size_t ind);

    bool Empty();

    size_t Length();

    template<typename X>
    friend std::ostream& operator<<(std::ostream& os, const TLinkedList<X>& list);

    void Clear();
```

```
    virtual ~TLinkedList();
};
```

tlinkedlist_i.cpp

```
#include "tlinkedlist_i.h"



template<typename T>
TLinkedListItem<T>::TLinkedListItem(shared_ptr<T> figure,
shared_ptr<TLinkedListItem<T>> nxt) {
    val = figure;
    next = nxt;
}

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

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

template<typename T>
shared_ptr<T> TLinkedListItem<T>::GetVal() {
    return val;
}

template<typename T>
std::ostream& operator<<(std::ostream& os, const TLinkedListItem<T>& item) {
    os << *item.val;
    return os;
}

template class TLinkedListItem<Rectangle>;
template std::ostream& operator<<(std::ostream& os, const TLinkedListItem<Rectangle>&
item);

template<typename T>
TLinkedListItem<T>::~TLinkedListItem() {

}
```

tlinkedlist_i.h

```
#pragma once


#include "rectangle.h"
#include "iostream"
#include "memory"

using std::shared_ptr;
using std::make_shared;

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

    void SetNext(shared_ptr<TLinkedListItem<T>> nxt);

    shared_ptr<TLinkedListItem<T>> GetNext();

    shared_ptr<T> GetVal();

    template<typename T1>
    friend std::ostream& operator<<(std::ostream& os, const TLinkedListItem<T1>& item);

    virtual ~TLinkedListItem();
};
```