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

ЛАБОРАТОРНАЯ РАБОТА №4 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.
- Создание простых динамических структур данных.
- Работа с объектами, передаваемыми «по значению».

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 лаб.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).

Нельзя использовать:

- Стандартные контейнеры std.
- Шаблоны (template).
- Различные варианты умных указателей (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

Дневник отладки

Недочёты

Выводы

Реализовал сложную структуру данных, каждым элементом которого является прямоугольник. 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;

	TLinkedList list;
	list.InsertFirst(rc);
	list.InsertFirst(rc1);
	list.InsertLast(rc2);
	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 = dist(a, b);
	len2 = dist(b, c);
	square = len1 * len2;
	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 : Figure {
public:
	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"

#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 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, 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"

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();
};
```