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 *Зинин Владислав Владимирович, 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 соджержать объекты фигур “по значению” (не по ссылке).

Класс-контейнер должен содержать набор следующих методов:

TODO: по поводу методов в личку

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.

Описание программы

Исходный код лежит в 10 файлах:

1. main.cpp - основная программа, взаимодействие с пользователем посредством команд из меню
2. include/figure.h - описание абстрактного класса фигур
3. include/point.h - описание класса точки
4. include/TVector.cpp - реализация функций контейнера первого уровня (в моем случае вектора)
5. include/TVector.h – реализация класса контейнера первого уровня (в моем случае вектора)
6. include/rhombus.h - описание класса ромба, наследующегося от figures
7. include/point.cpp - реализация класса точки
8. include/TVectorItem.cpp – реализация функций вспомогательного класса для контейнера
9. include/TVectorItem.h – описание вспомогательного класса для контейнера

10. include/rhombus.cpp: реализация класса ромба, наследующегося от figure

Дневник отладки

Во время выполнения лабораторной работы программа была несколько раз отлажена, так как плохо работали некоторые функции вектора. После нескольких отладок программа стала работать исправно.

Недочёты

Недочётов не было обнаружено.

Выводы

Лабораторная работа №4 -э то модернизация последних лабораторных 2 семестра. Если на 1 курсе я реализовывал бинарное дерево при помощи структур на языке СИ, то сейчас я реализовал бинарное дерево при помощи ООП на языке С++. Лабораторная прошла успешно, я повторил старый материал и узнал, усвоил много нового.

Исходный код

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H
#include <iostream>
#include "point.h"

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

                    #endif //FIGURE_H
```

point.h

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point
{
public:
Point();
Point(double x, double y);
Point(std::istream &is);
double dist(Point &other);
friend double get_x(Point &other);
friend double get_y(Point &other);

friend std::istream& operator>>(std::istream& is, Point& p);
friend std::ostream& operator<<(std::ostream& os, Point& p);

private:
```

```
        double x_, y_;
};
;

                #endif //POINT_H
```

point.cpp

```
        #include <iostream>
        #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_;
        }

        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;
        }

        double get_x(Point &other)
        {
        return other.x_;
        }

        double get_y(Point &other)
        {
return other.y_;
```

rhombus.h

```
        #ifndef RHOMBUX_H
```

```
#define RHOMBUX_H
#include <iostream>
#include "point.h"
#include "figure.h"

class Rhombus : public Figure
{
public:
Rhombus(std::istream &is);
double Area();
void Print(std::ostream &os);
size_t VertexesNumber();

virtual ~Rhombus();


private:
Point a, b, c, d;
};



                                    #endif //RHOMBUX_H
```

rhombus.cpp

```
#include <iostream>
#include "rhombus.h"
#include <math.h>

Rhombus::Rhombus(std::istream &is)
{
is >> a;
is >> b;
is >> c;
is >> d;
```

```
}

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

{

os << "Rhombus" << std::endl;

os << a << ',' << b << ',' << c << ',' << d << std::endl;

}

double Rhombus::Area()

{

return 0.5 * fabs(get_x(a)*get_y(b) + get_x(b)*get_y(c) + get_x(c)*get_y(d) + get_x(d)*get_y(a) - get_x(b)*get_y(a) - get_x(
- get_x(d)*get_y(c) - get_x(a)*get_y(d));

}

Rhombus::~Rhombus()

{

std::cout << "Rhombus deleted" << std::endl;

}

size_t Rhombus::VertexesNumber()

{

return 4;




}
```

Main.cpp

```
#include <iostream>

#include "TVector.h"


int main()

{

TVector list;

/*-----Test InsertLast---*/

list.InsertLast(Rhombus());
```

```
list.InsertLast(Rhombus(Point(1,2), Point(3,4), Point(5,6), Point(7,8)));
list.InsertLast(Rhombus(Point(1,2), Point(3,4), Point(5,4), Point(7,8)));
list.InsertLast(Rhombus(Point(1,0), Point(3,2), Point(4,5), Point(9,9)));
list.InsertLast(Rhombus(Point(1,0), Point(3,2), Point(4,5), Point(9,9)));
list.InsertLast(Rhombus(Point(1,0), Point(3,2), Point(4,5), Point(9,9)));
list.InsertLast(Rhombus());
std::cout << list << std::endl;
/*-----Test RemoveLast---*/
list.RemoveLast();
std::cout << list << std::endl;

list.RemoveLast();
std::cout << list << std::endl;
/*-----Test push_back---*/
// list.push_front(Rhombus(Point(2,3), Point(2,3), Point(2,3), Point(2,3)));
// std::cout << list << std::endl;
// /*-----Test pop_back---*/
// list.pop_front();
// std::cout << list << std::endl;
/*-----Test Remove---*/
// list.pop_back();
// std::cout << list << std::endl;

// list.push_front(Rhombus(Point(2,3), Point(2,3), Point(2,3), Point(2,3)));
// std::cout << list << std::endl;
// /*-----Test insert---*/
// list.insert(Rhombus(Point(0,1), Point(2,3), Point(4,5), Point(6,7)), 1);
// std::cout << list << std::endl;
// list.insert(Rhombus(Point(0,1), Point(2,3), Point(4,5), Point(6,7)), 3);
// std::cout << list << std::endl;
// list.insert(Rhombus(Point(0,1), Point(2,3), Point(4,5), Point(6,7)), 2);
// std::cout << list << std::endl;
```

```
/*-----Test Remove---*/
std::cout << "--------------" << std::endl;
list.Remove(1);
std::cout << list << std::endl;
std::cout << list.Length() << std::endl;
std::cout << "--------------" << std::endl;
list.Resize(2);
std::cout << list << std::endl;
std::cout << "--------------" << std::endl;
std::cout << list.Length() << std::endl;
std::cout << list << std::endl;
std::cout << list[2] << std::endl;
list.Resize(4);
std::cout << list << std::endl;
list.Resize(4);
std::cout << list << std::endl;
return 0;
}
```