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 *Зубко Дмитрий Валерьевич, 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.

Дневник отладки

Во время выполнения лабораторной работы программа была несколько раз отлажена, так как плохо работала функция удаления из дерева. После нескольких отладок программа стала работать исправно.

Недочёты

Недочётов не было обнаружено.

Выводы

Лабораторная работа №4 - это модернизация последних лабораторных 2 семестра. Если на 1 курсе я реализовывал вектор при помощи структур на языке СИ, то сейчас я реализовал вектор при помощи ООП на языке C++. Лабораторная прошла успешно, я повторил старый материал и узнал, усвоил много нового. Также я освоил работу с выделением и очисткой памяти на языке C++ при помощи команд new и delete.

Исходный код

figure.h

```
#ifndef LAB1_FIGURE_H
#define LAB1_FIGURE_H

#include <cmath>
#include <iostream>
#include "point.h"

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


#endif //LAB1_FIGURE_H
```

main.cpp

```
#include "pentagon.h"
#include "TVector.h"
#include <string>

int main() {

    std::string command;
    TVector v;

    while (std::cin >> command){
        if(command == "print")
            std::cout << v;
        else if(command == "insertlast"){
            Pentagon p;
            std::cin >> p;
            v.InsertLast(p);
        }
        else if(command == "removelast"){
            v.RemoveLast();
```

```
        }
        else if(command == "last"){
            std::cout << v.Last();
        }
        else if(command == "idx"){
            int idx;
            std::cin >> idx;
            std::cout << v[idx];
        }
        else if(command == "length"){
            std::cout << v.Length() << std::endl;
        }
        else if(command == "clear"){
            v.Clear();
        }
        else if(command == "empty"){
            if(v.Empty()) std::cout << "Yes" << std::endl;
            else std::cout << "No" << std::endl;
        }
    }

}
```

pentagon.cpp

```
#include "pentagon.h"

std::istream& operator>>(std::istream& is, Pentagon& p) {
    std::cout << "Enter data:" << std::endl;
    is >> p.a >> p.b >> p.c >> p.d >> p.e;
//    std::cout << "Pentagon created via istream" << std::endl;
    return is;
}

std::ostream& operator<<(std::ostream& os, Pentagon& p) {
    os << "Pentagon: " << p.a << p.b << p.c << p.d << p.e << std::endl;
    return os;
}


Pentagon& Pentagon::operator=(const Pentagon &other) {
    this->a = other.a;
    this->b = other.b;
    this->c = other.c;
    this->d = other.d;
    this->e = other.e;
```

```
    return *this;
}

bool Pentagon::operator==(const Pentagon &other) {
    return a == other.a && b == other.b && c == other.c && d == other.d && e ==
other.e;
}

size_t Pentagon::VertexesNumber() {
    return 5;
}

double Pentagon::SquareTriangle(Point a, Point b, Point c){
    double p = (a.dist(b) + b.dist(c) + c.dist(a)) / 2;
    return sqrt(p * (p - a.dist(b)) * (p - b.dist(c)) * (p - c.dist(a)));
}

double Pentagon::Area() {
    return SquareTriangle(a, b, c) + SquareTriangle(a, c, d) + SquareTriangle(a, d,
e);
}

void Pentagon::Print(std::ostream &os) {
    os << "Pentagon: " << a << b << c << d << e << std::endl;
}

Pentagon::Pentagon(){}

Pentagon::Pentagon(Point a_, Point b_, Point c_, Point d_, Point e_) : a(a_), b(b_),
c(c_), d(d_), e(e_) {}

Pentagon::Pentagon(const Pentagon &other) : Pentagon(other.a, other.b, other.c,
other.d, other.e) {
    std::cout << "Made copy of pentagon" << std::endl;
}

Pentagon::Pentagon(std::istream &is) {
    std::cout << "Enter data:" << std::endl;
    is >> a >> b >> c >> d >> e;
//    std::cout << "Pentagon created via istream" << std::endl;
}

Pentagon::~Pentagon() {
//    std::cout << "Pentagon deleted" << std::endl;
}
```

Pentagon.h

```
#ifndef LAB1_PENTAGON_H
#define LAB1_PENTAGON_H

#include "figure.h"

class Pentagon : Figure{
public:
    friend std::istream& operator>>(std::istream& is, Pentagon& p);
    friend std::ostream& operator<<(std::ostream& os, Pentagon& p);
    size_t VertexesNumber() override;
    double Area() override;
    void Print(std::ostream &os) override;
    bool operator==(const Pentagon& other);
    Pentagon();
    Pentagon(Point a_, Point b_, Point c_, Point d_, Point e_);
    Pentagon(std::istream &is);
    Pentagon(const Pentagon &other);
    Pentagon& operator=(const Pentagon& other);
    virtual ~Pentagon();
private:
    Point a, b, c, d, e;
    double SquareTriangle(Point a, Point b, Point c);
};

#endif //LAB1_PENTAGON_H
```

Point.cpp

```
#include "point.h"

#include <cmath>

bool Point::operator==(const Point &other) {
    return (this->x_ == other.x_ && this->y_ == other.y_);
}

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);
}

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;
}
```

Point.h

```
#ifndef LAB1_POINT_H
#define LAB1_POINT_H

#include <iostream>

class Point {
public:
    Point();
    Point(std::istream &is);
    Point(double x, double y);

    double dist(Point& other);

    friend std::istream& operator>>(std::istream& is, Point& p);
    friend std::ostream& operator<<(std::ostream& os, Point& p);
    bool operator==(const Point& other);
private:
    double x_;
    double y_;
};

#endif //LAB1_POINT_H
```

TVector.cpp

```
//
// Created by Dmitriy on 10/11/2021.
//

#include "TVector.h"
```

```
#include <cassert>

TVector::TVector():size(0), data(nullptr), capacity(0) {
}

TVector::TVector(const TVector& other){
    size = other.size;
    capacity = other.capacity;
    data = new TVectorItem[capacity];
    for(int i = 0; i < size; ++i)
        data[i] = other.data[i];
}

TVector::~TVector() {
    if(capacity != 0)
       delete[] data;
}

void TVector::InsertLast(const Pentagon& pentagon){
    if(capacity != 0 && capacity > size){
        data[size++] = pentagon;
    }
    else{
        if(capacity == 0)
            capacity = 1;
        capacity *= 2;
        TVectorItem* data_new = new TVectorItem[capacity];
        for(int i = 0; i < size; ++i){
            data_new[i] = data[i];
        }
        data_new[size++] = pentagon;
        delete[] data;
        data = data_new;
    }
}

void TVector::RemoveLast(){
    if(size > 0)
        --size;
}

Pentagon& TVector::Last(){
    assert(size > 0);
    return data[size - 1].GetPentagon();
}

size_t TVector::Length() {
    return size;
}
```

```
Pentagon& TVector::operator[] (const size_t idx){
    assert(idx >= 0 && idx < size);
    return data[idx].GetPentagon();
}

bool TVector::Empty(){
    return size == 0;
}

void TVector::Clear() {
    if(capacity != 0)
        delete[] data;
    data = nullptr;
    capacity = size = 0;
}

std::ostream& operator<<(std::ostream& os, const TVector& arr){
    for(int i = 0; i < arr.size; ++i){
        os << arr.data[i].GetPentagon();
    }
    return os;
}
```

TBinaryTree.h

```
//
// Created by Dmitriy on 10/11/2021.
//

#ifndef LAB1_TVECTOR_H
#define LAB1_TVECTOR_H

#include <iostream>
#include "TVectorItem.h"
#include "pentagon.h"

class TVector {
public:

    TVector();

    TVector(const TVector& other);

    ~TVector();

    void InsertLast(const Pentagon& pentagon);

    void RemoveLast();
```

```
    Pentagon& Last();

    Pentagon& operator[] (const size_t idx);

    bool Empty();

    size_t Length();

    friend std::ostream& operator<<(std::ostream& os, const TVector& arr);

    void Clear();


private:
    size_t size;
    size_t capacity;
    TVectorItem *data;
};

#endif //LAB1_TVECTOR_H
```

TVectorItem.cpp

```
//
// Created by Dmitriy on 10/11/2021.
//

#include <iostream>
#include "TVectorItem.h"

TVectorItem::TVectorItem(const Pentagon& pentagon){
    p = pentagon;
}

TVectorItem::TVectorItem(const TVectorItem& other){
    p = other.p;
}

Pentagon& TVectorItem::GetPentagon(){
    return p;
}
```

```
std::ostream &operator<<(std::ostream &os, TVectorItem &p){
    os << p;
    return os;
}
```

TVectorItem.h

```
//
// Created by Dmitriy on 10/11/2021.
//

#ifndef LAB1_TVECTORITEM_H
#define LAB1_TVECTORITEM_H

#include <iostream>
#include "pentagon.h"

class TVectorItem {
public:
    TVectorItem(const Pentagon& pentagon);

    TVectorItem(const TVectorItem& other);

    Pentagon& GetPentagon();

    TVectorItem(){}

    friend std::ostream &operator<<(std::ostream &os, TVectorItem &p);


private:
    Pentagon p;
};

#endif //LAB1_TVECTORITEM_H
```