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 *Зубко Дмитрий Валерьевич, 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.

Дневник отладки

Во время выполнения лабораторной работы были некие неисправности в работе шаблонов и компиляции программы, однако окончательный вариант полностью исправен.

Недочёты

Недочётов не было обнаружено.

Выводы

Лабораторная работа №6 позволила мне полностью осознать одну из базовых и фундаментальных концепций языка C++ - работу с так называемыми шаблонами (templates). Благодаря шаблонам упрощается написание кода для структур, классов и функций, от которых требуется принимать не только один тип аргументов. Вместо того, чтобы реализовывать полиморфизм с помощью переопределения вышесказанных вещей, гораздо удобнее применить шаблоны. Поэтому я уверен, что знания, полученные в этой лабораторной работе, обязательно пригодятся мне.

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 <memory>
#include <string>

int main() {

    std::string command;
    TVector<Pentagon> v;

    while (std::cin >> command){
        if(command == "print")
            std::cout << v;
        else if(command == "insertlast"){
```

```
            Pentagon p;
            std::cin >> p;
            std::shared_ptr<Pentagon> d(new Pentagon(p));
            v.InsertLast(d);
        }
        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) {
}

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 <iostream>
#include "TVectorItem.h"

template<class T>
TVectorItem<T>::TVectorItem(const std::shared_ptr<T>& other){
    p = other;
}

template<class T>
TVectorItem<T>::TVectorItem(const std::shared_ptr<TVectorItem<T>>& other){
    p = other->p;
}

template<class T>
std::shared_ptr<T>& TVectorItem<T>::GetPentagon(){
    return p;
}

template<class A>
std::ostream &operator<<(std::ostream &os, TVectorItem<A> &p){
    os << *p.GetPentagon();
    return os;
}

template class TVectorItem<Pentagon>;
template std::ostream& operator<<(std::ostream& os, TVectorItem<Pentagon>& p);
```

TVector.h

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

#ifndef LAB1_TVECTOR_H
#define LAB1_TVECTOR_H
```

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

template<class T>
class TVector {
public:

    TVector();

    TVector(const std::shared_ptr<TVector>& other);

    ~TVector();

    void InsertLast(const std::shared_ptr<T>& pentagon);

    void RemoveLast();

    T& Last();

    T& operator[] (const size_t idx);

    bool Empty();

    size_t Length();

    template<class A> friend std::ostream& operator<<(std::ostream& os, const TVector<A>&
arr);

    void Clear();


private:
    size_t size;
    size_t capacity;
    std::shared_ptr<TVectorItem<T>[]> data;
};

#endif //LAB1_TVECTOR_H
```

TVector.cpp

```
//
```

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

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

template<class T>
TVectorItem<T>::TVectorItem(const std::shared_ptr<T>& other){
    p = other;
}

template<class T>
TVectorItem<T>::TVectorItem(const std::shared_ptr<TVectorItem<T>>& other){
    p = other->p;
}

template<class T>
std::shared_ptr<T>& TVectorItem<T>::GetPentagon(){
    return p;
}

template<class A>
std::ostream &operator<<(std::ostream &os, TVectorItem<A> &p){
    os << *p.GetPentagon();
    return os;
}

template class TVectorItem<Pentagon>;
template std::ostream& operator<<(std::ostream& os, TVectorItem<Pentagon>& p);
```

TBinaryTreeItem.h

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

#ifndef LAB1_TVECTORITEM_H
#define LAB1_TVECTORITEM_H

#include <iostream>
#include <memory>
```

```
#include "pentagon.h"

template<class T>
class TVectorItem {
public:
    TVectorItem(const std::shared_ptr<T>& other);

    TVectorItem(const std::shared_ptr<TVectorItem<T>>& other);

    std::shared_ptr<T>& GetPentagon();

    TVectorItem(){}

    template<class A> friend std::ostream &operator<<(std::ostream &os, TVectorItem<A>
&p);


private:
    std::shared_ptr<T> p;
};

#endif //LAB1_TVECTORITEM_H
```