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 С++ - 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