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 *Васютинский Вадим Александрович, 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**

При достаточном понимании того, что такое шаблоны, фактически, в коде было дописано всего пара строчек. Отлаживать было нечего))

**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 "rectangle.h"

#include "TVector.h"

#include <memory>

#include <string>

int main() {

std::string command;

TVector<Rectangle> v;

while (std::cin >> command) {

if (command == "print")

std::cout << v;

else if (command == "insertlast") {

Rectangle r;

std::cin >> r;

std::shared\_ptr<Rectangle> d(new Rectangle(r));

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 == "clear") {

v.Clear();

}

else if (command == "empty") {

if (v.Empty()) std::cout << "Yes" << std::endl;

else std::cout << "No" << std::endl;

}

}

return 0;

}

rectangle.cpp  
  
#include "rectangle.h"

std::istream& operator>>(std::istream& is, Rectangle& r) {

std::cout << "Enter data: " << std::endl;

is >> r.a >> r.b >> r.c >> r.d;

return is;

}

std::ostream& operator<<(std::ostream& os, Rectangle& r) {

os << "Pentagon: " << r.a << r.b << r.c << r.d;

return os;

}

Rectangle& Rectangle::operator=(const Rectangle &other) {

this->a = other.a;

this->b = other.b;

this->c = other.c;

this->d = other.d;

return \*this;

}

bool Rectangle::operator==(const Rectangle &other) {

return a == other.a && b == other.b && c == other.c;

}

void Rectangle::Print(std::ostream &os) {

os << "Rectangle: " << a << b << c << d << std::endl;

}

size\_t Rectangle::VertexesNumber() {

return 4;

}

double Rectangle::Area() {

return a.dist(b) \* a.dist(d);

}

Rectangle::Rectangle() {}

Rectangle::Rectangle(Point a, Point b, Point c, Point d) : a(a), b(b), c(c), d(d) {}

Rectangle::Rectangle(std::istream &is) {

std::cout << "Enter data:" << std::endl;

is >> a >> b >> c >> d;

std::cout << "Rectangle created via istream" << std::endl;

}

Rectangle::Rectangle(const Rectangle &other) : Rectangle(other.a, other.b, other.c, other.d) {

std::cout << "Made copy of rectangle" << std::endl;

}

Rectangle::~Rectangle() {

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

}

rectangle.h  
  
#ifndef OOP1\_RECTANGLE\_H

#define OOP1\_RECTANGLE\_H

#include "figure.h"

class Rectangle : Figure {

public:

friend std::istream& operator>>(std::istream& is, Rectangle& p);

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

bool operator==(const Rectangle& other);

Rectangle& operator=(const Rectangle& other);

void Print(std::ostream &os) override;

size\_t VertexesNumber() override;

double Area() override;

Rectangle();

Rectangle(Point a, Point b, Point c, Point d);

Rectangle(std::istream &is);

Rectangle(const Rectangle &other);

virtual ~Rectangle();

private:

Point a, b, c, d;

};

#endif //OOP1\_RECTANGLE\_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

#include "TVector.h"

#include "rectangle.h"

#include <cassert>

template <typename T>

TVector<T>::TVector()

: data\_(new std::shared\_ptr<T>[32]),

length\_(0), capacity\_(32) {}

template <typename T>

TVector<T>::TVector(const TVector &vector)

: data\_(new std::shared\_ptr<T>[vector.capacity\_]),

length\_(vector.length\_), capacity\_(vector.capacity\_) {

std::copy(vector.data\_, vector.data\_ + vector.length\_, data\_);

}

template <typename T>

TVector<T>::~TVector() {

delete[] data\_;

}

template <typename T>

void TVector<T>::\_Resize(const size\_t new\_capacity) {

std::shared\_ptr<T> \*newdata = new std::shared\_ptr<T>[new\_capacity];

std::copy(data\_, data\_ + capacity\_, newdata);

delete[] data\_;

data\_ = newdata;

capacity\_ = new\_capacity;

}

template <typename T>

void TVector<T>::InsertLast(const std::shared\_ptr<T> &item) {

if (length\_ >= capacity\_)

\_Resize(capacity\_ << 1);

data\_[length\_++] = item;

}

template <typename T>

void TVector<T>::EmplaceLast(const T &&item) {

if (length\_ >= capacity\_)

\_Resize(capacity\_ << 1);

data\_[length\_++] = std::make\_shared<T>(item);

}

template <typename T>

void TVector<T>::Remove(const size\_t index) {

std::copy(data\_ + index + 1, data\_ + length\_, data\_ + index);

--length\_;

}

template <typename T>

void TVector<T>::Clear() {

delete[] data\_;

data\_ = new std::shared\_ptr<T>[32];

length\_ = 0;

capacity\_ = 32;

}

template <typename T>

std::ostream &operator<<(std::ostream &os, const TVector<T> &vector) {

for (size\_t i = 0; i < vector.length\_; ++i)

os << (\*vector.data\_[i]);

os << std::endl;

return os;

}

template class TVector<Rectangle>;

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

TVector.h

#ifndef OOP2\_TVECTOR\_H

#define OOP2\_TVECTOR\_H

#include <iostream>

#include <memory>

#include <cstdlib>

template <typename T>

class TVector {

public:

TVector();

TVector(const TVector &);

virtual ~TVector();

size\_t Length() const {

return length\_;

}

bool Empty() const {

return !length\_;

}

const std::shared\_ptr<T> &operator[](const size\_t index) const {

return data\_[index];

}

std::shared\_ptr<T> &Last() const {

return data\_[length\_ - 1];

}

void InsertLast(const std::shared\_ptr<T> &);

void EmplaceLast(const T &&);

void Remove(const size\_t index);

T RemoveLast() {

return \*data\_[--length\_];

}

void Clear();

template<typename TF>

friend std::ostream &operator<<(

std::ostream &, const TVector<TF> &);

private:

void \_Resize(const size\_t new\_capacity);

std::shared\_ptr<T> \*data\_;

size\_t length\_, capacity\_;

};

#endif //OOP2\_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);