МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №5 по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Борисов Ян Артурович, группа М80-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

### 

### Цель работы

Целью лабораторной работы является:

Закрепление навыков работы с классами.

Знакомство с умными указателями.

### Задание

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий **все три** фигуры класса фигуры, согласно вариантам задания. Классы должны удовлетворять следующим правилам:

Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.  
  
Требования к классу контейнера аналогичны требованиям из лабораторной работы 2.

Класс-контейнер должен соджержать объекты используя std:shared\_ptr<…>.

Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

Стандартные контейнеры std.

Шаблоны (template).

Объекты «по-значению»

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 неисправностей почти не возникало, все было отлажено сразу же.

**Недочёты**  
Недочётов не было обнаружено.

**Выводы**

Лабораторная работа №5 позволила мне полностью осознать концепцию умных указателей в языке С++ и отточить навыки в работе с ними. Я прочел достаточно большое количество источников и узнал много новой информации о работе с умными указателями, множество “подводных камней”.

**Исходный код**

Tvector.h

#pragma once

#include "square.h"

#include <ostream>

#include <memory>

class TVector {

public:

TVector();

TVector(const TVector &);

virtual ~TVector();

inline size\_t Length() const

{

return length\_;

}

inline bool Empty() const

{

return !length\_;

}

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

{

return data\_[index];

}

inline std::shared\_ptr<Square> Last() const

{

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

}

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

void EmplaceLast(const Square &&);

void Remove(const size\_t index);

inline Square RemoveLast()

{

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

}

void Clear();

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

private:

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

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

size\_t length\_, capacity\_;

enum { CAPACITY = 32 };

};

Tvector.cpp

#include "TVector.h"

#include <cstdlib>

TVector::TVector()

: data\_(new std::shared\_ptr<Square>[CAPACITY]),

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

TVector::TVector(const TVector &vector)

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

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

{

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

}

TVector::~TVector()

{

delete[] data\_;

}

void TVector::\_Resize(const size\_t new\_capacity)

{

std::shared\_ptr<Square> \*newdata =

new std::shared\_ptr<Square>[new\_capacity];

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

delete[] data\_;

data\_ = newdata;

capacity\_ = new\_capacity;

}

#define \_EXTEND\_VECTOR\_IF\_NEEDED \

if (length\_ >= capacity\_) \

\_Resize(capacity\_ << 1);

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

{

\_EXTEND\_VECTOR\_IF\_NEEDED

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

}

void TVector::EmplaceLast(const Square &&item)

{

\_EXTEND\_VECTOR\_IF\_NEEDED

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

}

#undef \_EXTEND\_VECTOR\_IF\_NEEDED

void TVector::Remove(const size\_t index)

{

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

--length\_;

}

void TVector::Clear()

{

delete[] data\_;

data\_ = new std::shared\_ptr<Square>[CAPACITY];

length\_ = 0;

capacity\_ = CAPACITY;

}

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

{

const size\_t last = vector.length\_ - 1;

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

os << \*vector.data\_[i] << ((i != last) ? '**\n**' : '**\0**');

return os;

}