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 С++ 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 содержать 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 C++ и отточить навыки в работе с ними. Всё прошло успешно.

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;
}
```

```cpp
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;
}
```