МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

# ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ
# ПО КУРСУ
# ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Студент группы 8О-206Б: Ефременко Е.А.
№ Варианта: 1
Преподаватель: Поповкин А.В.

Подпись преподавателя:

Москва
2018

ОГЛАВЛЕНИЕ:

# ЛАБОРАТОРНАЯ РАБОТА №1

## ЦЕЛЬ РАБОТЫ:

- Программирование классов на языке C++.
- Управление памятью в языке C++.
- Изучение базовых понятий ООП.
- Знакомство с классами в C++.
- Знакомство с перегрузкой операторов.
- Знакомство с дружественными функциями.
- Знакомство с операциями ввода-вывода из стандартных библиотек.

## ПОСТАНОВКА ЗАДАЧИ:

Необходимо спроектировать и запрограммировать на языке C++ классы фигур, согласно варианту задания. Классы должны удовлетворять следующим правилам:

- Должны иметь общий родительский класс Figure.
- Должны иметь общии виртуальныи метод Print, печатающии параметры фигуры и ее тип в стандартныи поток вывода cout.
- Должны иметь общии виртуальныи метод расчета площади фигуры – Square.
- Должны иметь конструктор, считывающии значения основных параметров фигуры из стандартного потока cin.
- Должны быть расположенны в раздельных фаилах: отдельно заголовки (.h), отдельно описание методов (.cpp).

## ВАРИАНТ ЗАДАНИЯ (№1):

- Фигура №1 – треугольник.
- Фигура №2 – прямоугольник.
- Фигура №3 – квадрат.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ:

**Классы в C++** — это абстракция описывающая методы, свойства, ещё не существующих объектов.
**Объекты** — конкретное представление абстракции, имеющее свои свойства и методы. Созданные объекты на основе одного класса называются экземплярами этого класса. Эти объекты могут иметь различное поведение, свойства, но все равно будут являться объектами одного класса. В ООП существует три основных принципа построения классов:

1. **Инкапсуляция** — это свойство, позволяющее объединить в классе и данные, и методы, работающие с ними и скрыть детали реализации от пользователя.
2. **Наследование** — это свойство, позволяющее создать новый класс-потомок на основе уже существующего, при этом все характеристики класса родителя присваиваются классу-потомку.
3. **Полиморфизм** — свойство классов, позволяющее использовать объекты классов с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

**Перегрузка операторов** — один из способов реализации полиморфизма, заключающийся в возможности одновременного существования в одной области видимости нескольких различных вариантов применения оператора, имеющих одно и то же имя, но различающихся типами параметров, к которым они применяются.
**Дружественная функция** — это функция, которая не является членом класса, но имеет доступ к членам класса, объявленным в полях private или protected.
Частью стандартной библиотеки C++ является библиотека iostream – объектно-ориентированная иерархия классов, где используется и множественное, и виртуальное наследование. В ней реализована поддержка для файлового ввода/вывода данных встроенных типов. Кроме того, разработчики классов могут расширять эту библиотеку для чтения и записи новых типов данных.
Для использования библиотеки iostream в программе необходимо включить заголовочный файл
#include <iostream>
Операции ввода/вывода выполняются с помощью классов istream (потоковый ввод) и ostream (потоковый вывод). Третий класс, iostream, является производным от них и поддерживает двунаправленный ввод/вывод. Для удобства в библиотеке определены три стандартных объекта-потока:

1. cin – объект класса istream, соответствующий стандартному вводу. В общем случае он позволяет читать данные с терминала пользователя;
2. cout – объект класса ostream, соответствующий стандартному выводу. В общем случае он позволяет выводить данные на терминал пользователя;
3. cerr – объект класса ostream, соответствующий стандартному выводу для ошибок. В этот поток мы направляем сообщения об ошибках программы.

Вывод осуществляется, как правило, с помощью перегруженного оператора сдвига влево (<<), а ввод – с помощью оператора сдвига вправо (>>).

КОД ПРОГРАММЫ:

## **Main.cpp**

```
#include <cstdlib>
#include "Triangle.h"
#include "Foursquare.h"
#include "Rectangle.h"

void navigate();

int main(int argc, char const *argv[]) {
int key = 6;
Figure *ptr;

while(key != 0) {

navigate();
std::cin >> key;
switch(key) {
case 1 :
std::cout <<"Print side of Triangle"<< std::endl;
ptr = new Triangle(std::cin);
std::cout <<"Triangle:"<< std::endl;

break;
case 2 :
std::cout <<"Print side of Foursquare"<< std::endl;
ptr = new Foursquare(std::cin);
std::cout <<"Foursquare:"<< std::endl;

break;
case 3 :
std::cout <<"Print side of Rectangle"<< std::endl;
ptr = new Rectangle(std::cin);
std::cout <<"Rectangle:"<< std::endl;

break;

default :
if(key != 0) std::cout <<"Wrong number!"<< std::endl;
}
if(key != 0) {
ptr->Print();
std::cout <<"Square:"<< ptr->Square() << std::endl;
}
}

delete ptr;
return 0;
}

void navigate() {
printf("Exit - 0\n");
printf("Create Triangle - 1\n");
printf("Create Foursquare - 2\n");
printf("Create Rectangle - 3\n");
}
```

## Figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

class Figure {
public:
	virtual double Square() = 0;
	virtual void Print() = 0;
	virtual ~Figure() {};
};
```

## Triangle.h

```
#ifndef TRIANGLE_H
#define TRIANGLE_H

#include <iostream>
#include <cstdlib>
#include "Figure.h"

class Triangle : public Figure {
public:
	Triangle();
	Triangle(std::istream &is);
	Triangle(size_t i,size_t j,size_t k);
	Triangle(const Triangle *orig);
	double Square() override;
	void Print() override;
	virtual ~Triangle();
private:
	size_t side_a; size_t side_b; size_t side_c;
};
#endif
```

## Triangle.cpp

```
#include "Triangle.h"
#include <iostream>
#include <cmath>

Triangle::Triangle() : Triangle(0, 0, 0)  { }

Triangle::Triangle(size_t i, size_t j, size_t k) : side_a(i), side_b(j), side_c(k) {
	std::cout <<"Triangle created: "<< side_a <<", "<< side_b <<", "<< side_c << std::endl;
}

Triangle::Triangle(std::istream &is) {
	is >> side_a;
	is >> side_b;
	is >> side_c;
}

Triangle::Triangle(const Triangle *orig) {
	std::cout <<"Triangle copy created"<< std::endl;
	side_a = orig->side_a;
	side_b = orig->side_b;
	side_c = orig->side_c;
}
double Triangle::Square() {
	double p = double(side_a + side_b + side_c) / 2.0;
	return sqrt(p * (p - double(side_a))*(p - double(side_b))*(p - double(side_c)));
}

void Triangle::Print() {
	std::cout <<"a="<< side_a <<", b="<< side_b <<", c="<< side_c << std::endl;
```

```
}

Triangle::~Triangle() {
	std::cout <<"Triangle deleted"<< std::endl;
}
```

## Rectangle.h

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include <iostream>
#include <cstdlib>
#include "Figure.h"

class Rectangle : public Figure {
public:
	Rectangle();
	Rectangle(std::istream &is);
	Rectangle(size_t i,size_t j);
	Rectangle(const Rectangle& orig);
	double Square() override;
	void Print() override;
	virtual ~Rectangle();
private:
	size_t side_a;
	size_t side_b;
};
```

## Rectangle.cpp

```
#include "Rectangle.h"
#include <iostream>
#include <cmath>

Rectangle::Rectangle() : Rectangle(0, 0)  { }

Rectangle::Rectangle(size_t i, size_t j) : side_a(i), side_b(j) {
	std::cout <<"Rectangle created: "<< side_a <<", "<< side_b << std::endl;
}

Rectangle::Rectangle(std::istream &is) {
	is >> side_a;
	is >> side_b;
}

Rectangle::Rectangle(const Rectangle& orig) {
	std::cout <<"Rectangle copy created"<< std::endl;
	side_a = orig.side_a;
	side_b = orig.side_b;
}
double Rectangle::Square() {
	return side_a * side_b;
}

void Rectangle::Print() {
	std::cout <<"a="<< side_a <<", b="<< side_b <<", c="<< side_a <<", d="<< side_b << std::endl;
}

Rectangle::~Rectangle() {
	std::cout <<"Rectangle deleted"<< std::endl;
}
```

## Foursquare.h

```
#ifndef FOURSQUARE_H
#define FOURSQUARE_H

#include <iostream>
#include <cstdlib>
#include "Figure.h"

class Foursquare : public Figure {
public:
        Foursquare();
        Foursquare(std::istream &is);
        Foursquare(size_t i);
        Foursquare(const Foursquare& orig);
        double Square() override;
        void Print() override;
        virtual ~Foursquare();
private:
        size_t side_a;
};
```

## Foursquare.cpp

```
#include "Foursquare.h"
#include <iostream>
#include <cmath>

Foursquare::Foursquare() : Foursquare(0)  { }

Foursquare::Foursquare(size_t i) : side_a(i) {
        std::cout <<"Foursquare created: "<< side_a << std::endl;
}

Foursquare::Foursquare(std::istream &is) {
        is >> side_a;
}

Foursquare::Foursquare(const Foursquare& orig) {
        std::cout <<"Foursquare copy created"<< std::endl;
        side_a = orig.side_a;
}
double Foursquare::Square() {
        return side_a * side_a;
}

void Foursquare::Print() {
        std::cout <<"a="<< side_a <<", b="<< side_a <<", c="<< side_a <<", d="<< side_a << std::endl;
}

Foursquare::~Foursquare() {
        std::cout <<"Foursquare deleted"<< std::endl;
}
```

# ТЕСТЫ:

```
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab1$ ./progr
Exit - 0
Create Triangle - 1
Create Foursquare - 2
Create Rectangle - 3
1
Print side of Triangle
3 4 5
Triangle:
a=3, b=4, c=5
Square:6
Exit - 0
Create Triangle - 1
Create Foursquare - 2
Create Rectangle - 3
2
Print side of Foursquare
4
Foursquare:
a=4
Square:16
Exit - 0
Create Triangle - 1
Create Foursquare - 2
Create Rectangle - 3
3
Print side of Rectangle
3 4
Rectangle:
a=3, b=4
Square:12
Exit - 0
Create Triangle - 1
Create Foursquare - 2
Create Rectangle - 3
0
Rectangle was deleted
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab1$
```

# ЛАБОРАТОРНАЯ РАБОТА №2

## ЦЕЛЬ РАБОТЫ:

- Закрепление навыков работы с классами.
- Создание простых динамических структур данных.
- Работа с объектами, передаваемыми «по значению».

## ПОСТАНОВКА ЗАДАЧИ:

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру, согласно вариантам задания. Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из л/р 1.
- Классы фигур должны иметь переопределенный оператор вывода в поток std::ostream (<<). Оператор должен распечатывать параметры фигуры.
- Классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream(>>). Оператор должен вводить основные параметры фигуры.
- Класс фигур должны иметь операторы копирования (=).
- Классы фигур должны иметь операторы сравнения и такими же фигурами (==).
- Класс-контейнер должен содержать объекты фигур «по значению» (не по ссылке).
- Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.
- Класс-контейнер должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера).

Нельзя использовать:

- Стандартные контейнеры std.
- Шаблоны (template).
- Различные варианты умных указателей (shared_ptr, weak_ptr).

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.

## ВАРИАНТ ЗАДАНИЯ(№1):

- Контейнер 1-го уровня – массив.
- Фигура №1 – треугольник.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ:

Часто в серьезных программах надо использовать данные, размер и структура которых должны **меняться** в процессе работы. Динамические массивы здесь не выручают, поскольку заранее нельзя сказать, сколько памяти надо выделить – это выясняется только в процессе работы. Например, надо проанализировать текст и определить, какие слова и в каком количестве в нем встречаются, причем эти слова нужно расставить по алфавиту. В таких случаях применяют данные особой структуры, которые представляют собой отдельные элементы, связанные с помощью **ссылок.**
Каждый элемент (**узел**) состоит из двух областей памяти: **поля данных** и **ссылок.** Ссылки – это адреса других узлов этого же типа, с которыми данный элемент логически связан. В языке Си для организации ссылок используются переменные-указатели. При добавлении нового узла в такую структуру выделяется новый блок памяти и (с помощью ссылок) устанавливаются связи этого элемента с уже существующими. Для обозначения конечного элемента в цепи используются **нулевые ссылки (NULL).**
**Связный список** является простейшим типом данных динамической структуры, состоящей из элементов (**узлов**). Каждый узел включает в себя в классическом варианте два поля:

- данные (в качестве данных может выступать переменная, объект класса или структуры и т. д.).
- указатель на следующий узел в списке.

Элементы связанного списка можно помещать и исключать произвольным образом. Доступ к списку осуществляется через указатель, который содержит адрес первого элемента списка, называемый **корнем списка.**
**Перегрузка операторов** — один из способов реализации полиморфизма, заключающийся в возможности одновременного существования в одной области видимости нескольких различных вариантов применения оператора, имеющих одно и то же имя, но различающихся типами параметров, к которым они применяются.
**Дружественная функция** — это функция, которая не является членом класса, но имеет доступ к членам класса, объявленным в полях private или protected.
Частью стандартной библиотеки C++ является библиотека iostream – объектно-ориентированная иерархия классов, где используется и множественное, и виртуальное наследование. В ней реализована поддержка для файлового ввода/вывода данных встроенных типов. Кроме того, разработчики классов могут расширять эту библиотеку для чтения и записи новых типов данных.
Для использования библиотеки iostream в программе необходимо включить заголовочный файл
#include <iostream>
Операции ввода/вывода выполняются с помощью классов istream (потоковый ввод) и ostream (потоковый вывод). Третий класс, iostream, является производным от них и поддерживает двунаправленный ввод/вывод. Для удобства в библиотеке определены три стандартных объекта-потока:

4. cin – объект класса istream, соответствующий стандартному вводу. В общем случае он позволяет читать данные с терминала пользователя;
5. cout – объект класса ostream, соответствующий стандартному выводу. В общем случае он позволяет выводить данные на терминал пользователя;
6. cerr – объект класса ostream, соответствующий стандартному выводу для ошибок. В этот поток мы направляем сообщения об ошибках программы.

Вывод осуществляется, как правило, с помощью перегруженного оператора сдвига влево (<<), а ввод – с помощью оператора сдвига вправо (>>).

КОД ПРОГРАММЫ:

## main.cpp

```
#include <iostream>
#include "Triangle.h"
#include "TVector.h"

void navigate();

int main()
{
int key;
int index;
int a = 0;
int b = 0;
int c = 0;
Triangle tmp;

TVector vector(5);
do{
navigate();
std::cin >> key;
switch(key){
case 1:
std::cout <<"Введите стороны треугольника"<< std::endl;
std::cin >> a >> b >> c ;
tmp.SetSides(a, b, c);
vector.Push(tmp);
break;
case 2:
std::cout <<"Введите индекс треугольника"<< std::endl;
std::cin >> index;
if(index >= vector.GetSize()){
std::cout <<"Треугольник не найден"<< std::endl;
} else {
std::cout << vector.Get(index) << std::endl;
std::cout << vector.Get(index).Square() << std::endl;
}
break;
case 3:
std::cout <<"Введите индекс"<< std::endl;
std::cin >> index;
vector.Delete(index);
break;
case 4:
vector.~TVector();
key = 0;
break;
case 5:
std::cout << vector << std::endl;
break;
case 0:
break;
default:
std::cout <<"Неизвестная команда"<< std::endl;
break;
}
} while(key);
return 0;
}

void navigate() {
std::cout <<"Введите"<< std::endl;
std::cout <<"1) Чтобы добавить треугольник."<< std::endl;
std::cout <<"2) Чтобы получить длины сторон и площадь треугольника"<< std::endl;
```

```
std::cout <<"3) Чтобы удалить треугольник"<< std::endl;
std::cout <<"4) Чтобы удалить все треугольники"<< std::endl;
std::cout <<"5) Чтобы напечатать все треугольники"<< std::endl;
std::cout <<"0) Выход"<< std::endl;
}
```

## TVector.h

```
#ifndef TVECTOR_H
#define TVECTOR_H

#include "Triangle.h"

class TVector {
public:
TVector(int vCap); // конструктор класса
void Push(Triangle &temp); // добавление элемента в класс
Triangle& Get(int index); // получение элемента класса
void Delete();// удаление последнего элемента класса
void Delete(int index); // удаление по индексу
friend std::ostream& operator<<(std::ostream& os,const TVector& vector);// вывод в стандартный поток
~TVector(); // деструктор класса
int GetSize(); // получение размера класса
private:
Triangle *ResizeVector(Triangle *array); // изменение размера вектора
int size; // текущее количетсво элементов
int capacity; // выделенное место
Triangle *array; // массив
};
#endif
```

## Tvector.cpp

```
#include "TVector.h"
#include <stdlib.h>

const int INCREASE = 2;

TVector::TVector(int vCap) {
array = (Triangle *)malloc(sizeof(Triangle) * vCap);
size = 0;
capacity = vCap;
}

Triangle *TVector::ResizeVector(Triangle *array) {
return (Triangle *)realloc(array, sizeof(Triangle) * INCREASE * capacity);
}

void TVector::Push(Triangle &temp) { // change &temp -> *temp
if(size == capacity) {
array = ResizeVector(array);
}
array[size] = temp;
// array[size].side_a = temp->side_a;
// array[size].side_b = temp->side_b;
// array[size].side_c = temp->side_c;
++size;
capacity *= INCREASE;
}

Triangle& TVector::Get(int index) {
return array[index];
}

void TVector::Delete() {
// array[--size].SetZero(); 1 version
```

```
array[--size].~Triangle();
}

void TVector::Delete(int index) {
int i;
int j;
Triangle *temp = (Triangle *)malloc(sizeof(Triangle) * (size - 1));
for(j = 0, i = 0; i < size; ++i) {
if(i != index) {
temp[j] = array[i];
++j;
}
}
free(array);
array = temp;
--size;
}

TVector::~TVector() {
free(array);
array = NULL;
capacity = size = 0;
}

std::ostream& operator<<(std::ostream& os, const TVector& vector) {
for(int i = 0; i < vector.size; ++i) {
os << vector.array[i];
}
return os;
}

int TVector::GetSize() {
return size;
}
#endif
```

## Описание класса Triangle осталось таким же, методы класса тоже не подверглись изменениям.

# ТЕСТЫ:

```
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab2$ ./prog
Triangle Created: 0,0,0
Введите
1) Чтобы добавить треугольник.
2) Чтобы получить длины сторон и площадь треугольника
3) Чтобы удалить треугольник
4) Чтобы удалить все треугольники
5) Чтобы напечатать все треугольники
0) Выход
1
Введите стороны треугольника
3 4 5
Введите
1) Чтобы добавить треугольник.
2) Чтобы получить длины сторон и площадь треугольника
3) Чтобы удалить треугольник
4) Чтобы удалить все треугольники
5) Чтобы напечатать все треугольники
0) Выход
1
Введите стороны треугольника
2 3 4
Введите
1) Чтобы добавить треугольник.
2) Чтобы получить длины сторон и площадь треугольника
```

```
3) Чтобы удалить треугольник
4) Чтобы удалить все треугольники
5) Чтобы напечатать все треугольники
0) Выход
1
Введите стороны треугольника
13 15 17
Введите
1) Чтобы добавить треугольник.
2) Чтобы получить длины сторон и площадь треугольника
3) Чтобы удалить треугольник
4) Чтобы удалить все треугольники
5) Чтобы напечатать все треугольники
0) Выход
5
a=3b=4c=5
a=2b=3c=4
a=13b=15c=17

Введите
1) Чтобы добавить треугольник.
2) Чтобы получить длины сторон и площадь треугольника
3) Чтобы удалить треугольник
4) Чтобы удалить все треугольники
5) Чтобы напечатать все треугольники
0) Выход
2
Введите индекс треугольника
0
a=3b=4c=5

6
Введите
1) Чтобы добавить треугольник.
2) Чтобы получить длины сторон и площадь треугольника
3) Чтобы удалить треугольник
4) Чтобы удалить все треугольники
5) Чтобы напечатать все треугольники
0) Выход
3
Введите индекс
1
Введите
1) Чтобы добавить треугольник.
2) Чтобы получить длины сторон и площадь треугольника
3) Чтобы удалить треугольник
4) Чтобы удалить все треугольники
5) Чтобы напечатать все треугольники
0) Выход
5
a=3b=4c=5
a=13b=15c=17

Введите
1) Чтобы добавить треугольник.
2) Чтобы получить длины сторон и площадь треугольника
3) Чтобы удалить треугольник
4) Чтобы удалить все треугольники
5) Чтобы напечатать все треугольники
0) Выход
4
Triangle was deleted
```

# ЛАБОРАТОРНАЯ РАБОТА №3

## ЦЕЛЬ РАБОТЫ:

- Закрепление навыков работы с классами.
- Знакомство с умными указателями.

## ПОСТАНОВКА ЗАДАЧИ:

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий все три фигуры класса фигур, реализованные в л/р 1.
Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из л/р 1.
- Класс-контейнер должен содержать объекты используя std::shared_ptr<…>.
- Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.
- Класс-контейнер должен иметь методы по получению фигуры из контейнера.
- Класс-контейнер должен иметь метод по удалению фигуры из контейнера.
- Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostrem(<<).
- Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно методы (.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.

## ВАРИАНТ ЗАДАНИЯ(№1):

- Контейнер 1-ого уровня – массив.
- Фигура №1 – треугольник.
- Фигура №2 – прямоугольник.
- Фигура №3 – квадрат.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ:

**Умный указатель** – класс (обычно шаблонный), имитирующий интерфейс обычного указателя и добавляющий некую новую функциональность, например проверку границ при доступе или очистку памяти.
**shared_ptr** –это самый популярный и самый широко используемый умный указатель. Он начал своё развитие как часть библиотеки boost. Данный указатель был столь успешным, что его включили в C++ Technical Report 1 и он был доступен в пространстве имен tr1 — std::tr1::shared_ptr<>.
В отличии от рассмотренных выше указателей, shared_ptr реализует подсчет ссылок на ресурс. Ресурс освободится тогда, когда счетчик ссылок на него будет равен 0. Как видно, система реализует одно из основных правил сборщика мусора.

КОД ПРОГРАММЫ:

## main.cpp

```
#include <iostream>
#include "Triangle.h"
#include "Foursquare.h"
#include "Rectangle.h"
#include "TVector.h"
#include <memory>

void navigate();

int main()
{
int key;
int index;
int a = 0;
int b = 0;
int c = 0;
std::shared_ptr<Figure> tri;
TVector vector(20);
do{
navigate();
std::cin >> key;
switch(key){
case 1:
std::cout <<"Введите стороны треугольника"<< std::endl;
std::cin >> a >> b >> c ;
tri = std::shared_ptr<Figure>(new Triangle(a, b, c));
vector.Push(tri);
break;
case 2:
std::cout <<"Введите стороны квадрата"<< std::endl;
std::cin >> a ;
tri = std::shared_ptr<Figure>(new Foursquare(a));
vector.Push(tri);
break;
case 3:
std::cout <<"Введите стороны прямоугольника"<< std::endl;
std::cin >> a >> b ;
tri = std::shared_ptr<Figure>(new Rectangle(a, b));
vector.Push(tri);
break;
case 4:
std::cout <<"Введите индекс фигуры"<< std::endl;
std::cin >> index ;
if(index >= vector.GetSize()){
std::cout <<"Такой фигуры не существует"<< std::endl;
```

```
} else {
vector.Get(index);
}
break;
case 5:
if(vector.GetSize() > 0){
vector.Delete();
} else {
std::cout <<"Такой фигуры не существует"<< std::endl;
}
break;
case 6:
vector.~TVector();
key = 0;
break;
case 7:
std::cout << vector << std::endl;
break;
case 0:
break;
default:
std::cout <<"Неизвестная команда"<< std::endl;
break;
}
} while(key);
//std::cout << vector.GetSize() << std::endl;
return 0;
}

void navigate() {
std::cout <<"Введите"<< std::endl;
std::cout <<"1) Чтобы добавить треугольник."<< std::endl;
std::cout <<"2) Чтобы добавить квадрат."<< std::endl;
std::cout <<"3) Чтобы добавить прямоугольник."<< std::endl;
std::cout <<"4) Чтобы получить фигуру."<< std::endl;
std::cout <<"5) Чтобы удалить фигуру."<< std::endl;
std::cout <<"6) Чтобы удалить все фигуры."<< std::endl;
std::cout <<"7) Чтобы напечать массив."<< std::endl;
std::cout <<"0) Выход"<< std::endl;
}
```

## TVector.h

```
#ifndef TVECTOR_H
#define TVECTOR_H

#include "Figure.h"
#include "Triangle.h"
#include "Foursquare.h"
#include "Rectangle.h"
#include <memory>

class TVector{
public:
TVector(int VecCap);
void Push(std::shared_ptr<Figure>&temp);
void Get(int index);
void Delete();
// void Delete(int index);
friend std::ostream& operator<<(std::ostream& os, const TVector& vector);
~TVector();
int GetSize();
void DeleteAll();

private:
void ResizeVector(std::shared_ptr<Figure> *&array);
int size;
int capacity;
```

```
std::shared_ptr<Figure> *array;
};

#endif
```

## TVector.cpp

```
#include "TVector.h"
#include <stdlib.h>

const int INC = 1.5;

TVector::TVector(int VecCap){
array = new std::shared_ptr<Figure> [VecCap];
size = 0;
capacity = VecCap;
}

void TVector::ResizeVector(std::shared_ptr<Figure> *&array){
capacity *= INC;
std::shared_ptr<Figure> *tmp = new std::shared_ptr<Figure> [capacity];
for(int i = 0; i < size; ++i){
tmp[i] = array[i];
array[i] = nullptr;
}
delete [] array;
array = nullptr;
array = tmp;
tmp = nullptr;
}

void TVector::Push(std::shared_ptr<Figure>&temp){
if(size == capacity) {
ResizeVector(array);
}
array[size] = temp;
++size;
}

void TVector::Get(int index){
array[index]->Print();
}

void TVector::Delete(){
//array[size]->~Figure();
array[size] = nullptr;
size--;
}

/*void TVector::Delete(int index){
int i;
int j;
Triangle *temp = (Triangle*)malloc(sizeof(Triangle) * (size - 1));
for(j = 0, i = 0; i < size; ++i){
if(i != index){
temp[j] = array[i];
j++;
}
}
free(array);
array = temp;
--size;
}*/

void TVector::DeleteAll(){
for(int i = 0; i < GetSize(); i++){
array[i] = nullptr;
}
```

```
delete [] array;
array = nullptr;
capacity = size = 0;
}

TVector::~TVector(){
DeleteAll();
}

int TVector::GetSize(){
return size;
}


std::ostream& operator<<(std::ostream& os, const TVector& vector){
for(int i = 0; i < vector.size; ++i){
os << *vector.array[i];
}
return os;
}
```

## Figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

#include <iostream>
// Superclass Figure
class Figure {
public:
virtual void Print() = 0; // Change for Trg, Rec, Fout.
virtual ~Figure() {};
friend std::ostream& operator <<(std::ostream &os, Figure& obj) {
obj.Print();
return os;
};
};
#endif
```

**Описание классов-фигур осталось неизменным, методы этих классов также не изменились.**

ТЕСТЫ:

```
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab3$ ./prog
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
```

```
0) Выход
1
Введите стороны треугольника
3 4 5
Треугольник создан: 3,4,5
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
2
Введите стороны квадрата
4
Квадрат создан:4
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
3
Введите стороны прямоугольника
3 4
Прямоугольник создан: 3,4
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
7
Это треугольник
a=3, b=4, c=5
Это квадрат
a=4
Это прямоугольник
a=3, b=4

Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
4
Введите индекс фигуры
0
Это треугольник
a=3, b=4, c=5
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
```

6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
5
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
7
Это треугольник
a=3, b=4, c=5
Это квадрат
a=4

Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
6
Треугольник удалён
Квадрат удалён
Прямоугольник удалён

# ЛАБОРАТОРНАЯ РАБОТА №4

## ЦЕЛЬ РАБОТЫ:

- Знакомство с шаблонами классов.
- Построение шаблонов динамических структур данных.

## ПОСТАНОВКА ЗАДАЧИ:

Необходимо спроектировать и запрограммировать на языке С++ шаблон класса-контеинера первого уровня, содержащий все три фигуры класса фигуры. Классы должны удовлетворять следующим правилам:

- Требования к классам фигуры аналогичны требованиям из л/р №1.
- Шаблон класса-контеинера должен содержать объекты используя std:shared_ptr<...>.
- Шаблон класса-контеинера должен иметь метод по добавлению фигуры в контейнер.
- Шаблон класса-контеинера должен иметь методы по получению фигуры из контеинера (определяется структурой контеинера).
- Шаблон класса-контеинера должен иметь метод по удалению фигуры из контеинера (определяется структурой контеинера).
- Шаблон класса-контеинера должен иметь перегруженный оператор по выводу контеинера в поток std::ostream (<<).
- Шаблон класса-контеинера должен иметь деструктор, удаляющий все элементы контеинера.
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контеинера.
- Удалять фигуры из контеинера.

## ВАРИАНТ ЗАДАНИЯ(№1):

- Контейнер 1-ого уровня – массив.
- Фигура №1 – треугольник.
- Фигура №2 – прямоугольник.
- Фигура №3 – квадрат.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ:

Шаблоны (template) предназначены для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).
В C++ возможно создание шаблонов функций и классов.
Шаблоны позволяют создавать параметризованные классы и функции. Параметром может быть любой тип или значение одного из допустимых типов (целое число, enum, указатель на любой объект с глобально доступным именем, ссылка).

КОД ПРОГРАММЫ:

## TVector.h

```
#ifndef TVECTOR_H
#define TVECTOR_H

#include "Figure.h"
#include "Triangle.h"
#include "Foursquare.h"
#include "Rectangle.h"
#include <memory>

template <class T> class TTVector {
public:
TTVector(int VecCap);
void Push(std::shared_ptr<T>&temp);
void Get(int index);
void Delete();
// void Delete(int index);
template <class A> friend std::ostream& operator<<(std::ostream& os, const TTVector<A>& vector);
~TTVector();
int GetSize();
void DeleteAll();

private:
void ResizeVector(std::shared_ptr<T> *&array);
int size;
int capacity;
std::shared_ptr<T> *array;
};

#endif
```

## Tvector.cpp

```
#include "TVector.h"
#include <stdlib.h>
#include "Figure.h"

const int INC = 1.5;

template <class T> TTVector<T>::TTVector(int VecCap){
array = new std::shared_ptr<T> [VecCap];
size = 0;
capacity = VecCap;
}

template <class T> void TTVector<T>::ResizeVector(std::shared_ptr<T> *&array){
capacity *= INC;
std::shared_ptr<T> *tmp = new std::shared_ptr<T> [capacity];
for(int i = 0; i < size; ++i){
```

```
tmp[i] = array[i];
array[i] = nullptr;
}
delete [] array;
array = nullptr;
array = tmp;
tmp = nullptr;
}

template <class T> void TTVector<T>::Push(std::shared_ptr<T>&temp){
if(size == capacity) {
ResizeVector(array);
}
array[size] = temp;
++size;
}

template <class T> void TTVector<T>::Get(int index){
array[index]->Print();
}

template <class T> void TTVector<T>::Delete(){
//array[size]->~Figure();
array[size] = nullptr;
size--;
}

/*void TVector::Delete(int index){
int i;
int j;
Triangle *temp = (Triangle*)malloc(sizeof(Triangle) * (size - 1));
for(j = 0, i = 0; i < size; ++i){
if(i != index){
temp[j] = array[i];
j++;
}
}
free(array);
array = temp;
--size;
}*/

template <class T> void TTVector<T>::DeleteAll(){
for(int i = 0; i < GetSize(); i++){
array[i] = nullptr;
}
delete [] array;
array = nullptr;
capacity = size = 0;
}

template <class T> TTVector<T>::~TTVector(){
DeleteAll();
}

template <class T> int TTVector<T>::GetSize(){
return size;
}



template <class T> std::ostream& operator<<(std::ostream& os, const TTVector<T>& vector){
for(int i = 0; i < vector.size; ++i){
vector.array[i]->Print();
}
return os;
}
```

```
template class TTVector<Figure>;
template std::ostream& operator<<(std::ostream& os, const TTVector<Figure>& vector);
```

## Описание классов-фигур и их методы остались неизменными. Функция main.cpp изменениям также не подверглась.

# ТЕСТЫ:

```
Enter:
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab4$ ./prog
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
1
Введите стороны треугольника
5 6 7
Треугольник создан: 5,6,7
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
2
Введите стороны квадрата
4
Квадрат создан:4
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
3
Введите стороны прямоугольника
10 14
Прямоугольник создан: 10,14
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
7
Это треугольник
a=5, b=6, c=7
Это квадрат
a=4, b=4, c=4, d=4
Это прямоугольник
a=10, b=14, c=10, d=14
```

```
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
4
Введите индекс фигуры
0
Это треугольник
a=5, b=6, c=7
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
5
Прямоугольник удалён
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
1
Введите стороны треугольника
7
6
8
Треугольник создан: 7,6,8
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
7
Это треугольник
a=5, b=6, c=7
Это квадрат
a=4, b=4, c=4, d=4
Это треугольник
a=7, b=6, c=8

Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
0) Выход
6
Треугольник удалён
```

```
Квадрат удалён
Треугольник удалён
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab4$
```

# ЛАБОРАТОРНАЯ РАБОТА №5

## ЦЕЛЬ РАБОТЫ:

- Закрепление навыков работы с шаблонами классов.
- Построение итераторов для динамических структур данных.

## ПОСТАНОВКА ЗАДАЧИ:

Используя структуры данных, разработанные для предыдущеи лабораторнои работы спроектировать и разработать Итератор для динамическои структуры данных.
Итератор должен быть разработан в виде шаблона и должен уметь работать со всеми типами фигур, согласно варианту задания.
Итератор должен позволять использовать структуру данных в операторах типа for.
Например:

```
for(auto i : stack) std::cout << *i << std::endl;
```

Нельзя использовать:

- Стандартные контеинеры std.

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контеинер.
- Распечатывать содержимое контеинера.
- Удалять фигуры из контеинера.

## ВАРИАНТ ЗАДАНИЯ(№1):

- Контейнер 1-ого уровня – массив.
- Фигура №1 – треугольник.
- Фигура №2 – прямоугольник.
- Фигура №3 – квадрат.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ:

Для доступа к элементам некоторого множества элементов используют специальные объекты, называемые итераторами. В контейнерных типах stl они доступны через методы класса (например, begin() в шаблоне класса vector). Функциональные возможности указателей и итераторов близки, так что обычный указатель тоже может использоваться как итератор.
Категории итераторов:

- Итератор ввода (input iterator) - используется потоками ввода.
- Итератор вывода (output iterator) - используется потоками вывода.
- Однонаправленный итератор (forward iterator) - для прохода по элементам в одном направлении.
- Двунаправленный итератор (bidirectional iterator) - способен пройтись по элементам в любом направлении. Такие итераторы реализованы в некоторых контейнерных типах stl (list, set, multiset, map, multimap).
- Итераторы произвольного доступа (random access) - через них можно иметь доступ к любому элемента. Такие итераторы реализованы в некоторых контейнерных типах stl (vector, deque, string, array).

КОД ПРОГРАММЫ:

## Iterator.h

```
#ifndef TITER_H
#define TITER_H

#include <memory>
#include <iostream>
#include "TVector.h"

template <class T> class TIter{ // класс итератор
public:

TIter(std::shared_ptr<T> *n){ // конструктор итератора
ptr = n;
}

std::shared_ptr<T> operator * (){
return *ptr;
}

std::shared_ptr<T> operator -> (){
return ptr->GetValue();
}

void operator ++ (){
++ptr;
}

TIter operator ++ (int){
TIter iter(*this);
++(*this);
return iter;
}

bool operator == (TIter const& i){
return ptr == i.ptr;
}

bool operator != (TIter const& i){
```

```
return !(*this == i);
}

private:
std::shared_ptr<T> *ptr;
};

#endif
```

## Все классы-фигуры остаются неизменными.

# ТЕСТЫ:

```
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab5$ ./prog
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
1
Введите стороны треугольника
3 4 5
Треугольник создан: 3,4,5
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
2
Введите стороны квадрата
4
Квадрат создан:4
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
3
Введите стороны прямоугольника
3 4
Прямоугольник создан: 3,4
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
8
Это треугольник
a=3, b=4, c=5
Это квадрат
a=4, b=4, c=4, d=4
Это прямоугольник
a=3, b=4, c=3, d=4
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
```

```
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
4
Введите индекс фигуры
1
Это квадрат
a=4, b=4, c=4, d=4
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
5
Прямоугольник удалён
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
8
Это треугольник
a=3, b=4, c=5
Это квадрат
a=4, b=4, c=4, d=4
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
6
Треугольник удалён
Квадрат удалён
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab5$
```

# ЛАБОРАТОРНАЯ РАБОТА №6

## ЦЕЛЬ РАБОТЫ:

- Закрепление навыков по работе с памятью в C++.
- Создание аллокаторов памяти для динамических структур данных

## ПОСТАНОВКА ЗАДАЧИ:

Используя структуры данных, разработанные для предыдущей лабораторной работы ( ЛРNo5) спроектировать и разработать аллокатор памяти для динамической структуры данных.
Цель построения аллокатора – минимизация вызова операции malloc. Аллокатор должен выделять большие блоки памяти для хранения фигур и при создании новых фигур-объектов выделять место под объекты в этой памяти.
Алокатор должен хранить списки использованных/свободных блоков. Для хранения списка свободных блоков нужно применять динамическую структуру данных (контейнер 2-го уровня, согласно варианта задания).
Для вызова аллокатора должны быть переопределены оператор new и delete у классов-фигур.

Нельзя использовать:

- Стандартныеконтейнеры std.

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатыватьсодержимое контейнера.
- Удалять фигуры из контейнера.
-

## ВАРИАНТ ЗАДАНИЯ(№1):

- Контейнер 1-ого уровня – массив.
- Контейнер 2-ого уровня – массив.
- Фигура №1 – треугольник.
- Фигура №2 – прямоугольник.
- Фигура №3 – квадрат.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ:

Allocator (англ) - лицо, ведающее распределением или разверсткой

Программисты должны учитывать последствия динамического выделения памяти и дважды обдумать использование функции malloc или оператора new. Легко убедить себя, что вы не делаете так уж много аллокаций, а значит большого значения это не имеет, но такой тип мышления распространяется лавиной по всей команде, и приводит к медленной мучительной смерти. Фрагментация и потери в производительности, связанные с использованием динамической памяти, не будучи пресечёнными в зародыше, могут иметь катастрофические трудноразрешаемые последствия в вашем дальнейшем цикле разработки. Проекты, где управление и распределение памяти не продумано надлежащим образом, часто страдают от случайных сбоев после длительной игровой сессии из-за нехватки памяти (которые, кстати, практически невозможно воспроизвести) и стоят сотни часов работы программистов, пытающихся освободить память и реорганизовать её выделение.

КОД ПРОГРАММЫ:

## Tallocate.h

```
#ifndef TALLOCATE_H
#define TALLOCATE_H
#include <cstdlib>

class TAllocate{
public:

TAllocate(size_t size, size_t count);// конструктор блока

void *allocate(); // аллоцирование под запрос

void deallocate(void *pointer); //возврат памяти

bool free_blocks(); // проверка на свободные блоки

virtual ~TAllocate(); //деструктор блока

private:

size_t _size; //размер выделенного
size_t _count; // текущий размер

char *_used_blocks; //используемые блоки
void **_free_blocks;// свободные блоки

size_t _free_count; //количество свободных
};


#endif
```

## Tallocate.cpp

```
#include "TAllocate.h"
#include "iostream"

TAllocate::TAllocate(size_t size, size_t count):_size(size),_count(count){
_used_blocks = (char*)malloc(_size*_count);
_free_blocks = (void**)malloc(sizeof(void*) * _count);

for(size_t i = 0; i < _count; i++) _free_blocks[i] = _used_blocks+i*_size;
_free_count = count;
std::cout <<"Allocator: Память инициализирована"<< std::endl;
}

void *TAllocate::allocate() {
void *result = nullptr;

if(_free_count > 0) {
result = _free_blocks[_free_count-1];
_free_count--;
std::cout <<"Allocator: Выделено "<< (_count-_free_count) <<
" из "<< _count << std::endl;
} else {
std::cout <<"Allocator: Нет памяти"<< std::endl;
}
return result;
}

void TAllocate::deallocate(void *pointer) {
std::cout <<"Allocator: Возвращение памяти "<< std::endl;
```

```
_free_blocks[_free_count] = pointer;
_free_count ++;
}

bool TAllocate::free_blocks() {
return _free_count>0;
}

TAllocate::~TAllocate() {
if(_free_count<_count) {
std::cout <<"Allocator: Утечка памяти"<< std::endl;
} else {
std::cout <<"Allocator: Память освобождена"<< std::endl;
}
delete _free_blocks;
delete _used_blocks;
}
```

## main.cpp

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

#include "Figure.h"
#include "TVector.h"
#include "TAllocate.h"

void navigate();

int main()
{
int key;
int index;
int a = 0;
int b = 0;
int c = 0;
std::shared_ptr<Figure> tri;
TTVector<Figure> *vector = new TTVector<Figure> (100);
//TTVector<Figure> vector(20);
Triangle *t = nullptr;
Rectangle *r = nullptr;
Foursquare *f = nullptr;
do{
navigate();
std::cin >> key;
switch(key){
case 1:
std::cout <<"Введите стороны треугольника"<< std::endl;
std::cin >> a >> b >> c ;
t = new Triangle(a, b, c);
//tri = std::shared_ptr<Figure>(new Triangle(a, b, c));
tri = std::shared_ptr<Figure>(t);
vector->Push(tri);
break;
case 2:
std::cout <<"Введите стороны квадрата"<< std::endl;
std::cin >> a ;
f = new Foursquare(a);
//tri = std::shared_ptr<Figure>(new Foursquare(a));
tri = std::shared_ptr<Figure>(f);
vector->Push(tri);
break;
case 3:
std::cout <<"Введите стороны прямоугольника"<< std::endl;
std::cin >> a >> b ;
r = new Rectangle(a, b);
```

```
//tri = std::shared_ptr<Figure>(new Rectangle(a, b));
tri = std::shared_ptr<Figure>(r);
vector->Push(tri);
break;
case 4:
std::cout <<"Введите индекс фигуры"<< std::endl;
std::cin >> index ;
if(index >= vector->GetSize()){
std::cout <<"Такой фигуры не существует"<< std::endl;
} else {
vector->Get(index);
}
break;
case 5:
if(vector->GetSize() > 0){
vector->Delete();
} else {
std::cout <<"Такой фигуры не существует"<< std::endl;
}
break;
case 6:
vector->~TTVector();
key = 0;
break;
case 7:
std::cout << *vector << std::endl;
break;
case 8:
for(auto i: *vector){
(*i).Print();
}
break;
case 0:
break;
default:
std::cout <<"Неизвестная команда"<< std::endl;
break;
}
} while(key);
//std::cout << vector.GetSize() << std::endl;
delete vector;
return 0;
}

void navigate() {
std::cout <<"Введите"<< std::endl;
std::cout <<"1) Чтобы добавить треугольник."<< std::endl;
std::cout <<"2) Чтобы добавить квадрат."<< std::endl;
std::cout <<"3) Чтобы добавить прямоугольник."<< std::endl;
std::cout <<"4) Чтобы получить фигуру."<< std::endl;
std::cout <<"5) Чтобы удалить фигуру."<< std::endl;
std::cout <<"6) Чтобы удалить все фигуры."<< std::endl;
std::cout <<"7) Чтобы напечать массив."<< std::endl;
std::cout <<"8) Цикл 'for' с итератором"<< std::endl;
std::cout <<"0) Выход"<< std::endl;
}
```

## В TVector.h и Tvector.cpp появляются новые(переопределенные) методы для вызова аллокатора:

## Переопределенный оператор new

```
template <class T> void * TTVector<T>::operator new(size_t size){
return vector_allocator.allocate();
}
```

## И оператор delete

```
template <class T> void TTVector<T>::operator delete(void *p){
return vector_allocator.deallocate(p);
}
```

## Остальные методы остаются неизменными.

# ТЕСТЫ:

```
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab6$ ./prog
Allocator: Память инициализирована
Allocator: Выделено 1 из 100
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
1
Введите стороны треугольника
3 4 5
Треугольник создан: 3,4,5
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
2
Введите стороны квадрата
4
Квадрат создан:4
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
3
Введите стороны прямоугольника
3 4
Прямоугольник создан: 3,4
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
8
Это треугольник
a=3, b=4, c=5
Это квадрат
a=4, b=4, c=4, d=4
Это прямоугольник
a=3, b=4, c=3, d=4
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
```

```
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
4
Введите индекс фигуры
0
Это треугольник
a=3, b=4, c=5
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
5
Прямоугольник удалён
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить квадрат.
3) Чтобы добавить прямоугольник.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Цикл 'for' с итератором
0) Выход
6
Треугольник удалён
Квадрат удалён
Allocator: Возвращение памяти
Allocator: Память освобождена
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab6$
```

# ЛАБОРАТОРНАЯ РАБОТА №7

## ЦЕЛЬ РАБОТЫ:

- Создание сложных динамических структур данных.
- Закрепление принципа ОСР.

## ПОСТАНОВКА ЗАДАЧИ:

Необходимо реализовать динамическую структуру данных – «Хранилище объектов» и алгоритм работы с ней. «Хранилище объектов» представляет собой контейнер, одного из следующих видов (Контейнер 1-го уровня). При этом должно выполняться правило, что количество объектов в контейнере второго уровня не больше 5. Т.е. если нужно хранить больше 5 объектов, то создается еще один контейнер второго уровня. Объекты в контейнерах второго уровня должны быть отсортированы по возрастанию площади объекта (в том числе и для деревьев).
При удалении объектов должно выполняться правило, что контейнер второго уровня не должен быть пустым. Т.е. если он становится пустым, то он должен удалится.
Нельзя использовать:

- Стандартныеконтейнеры std.

Программа должна позволять:

1. Вводить произвольное количество фигур и добавлять их в контейнер.
2. Распечатывать содержимое контейнера (1-го и 2-го уровня).
3. Удалять фигуры из контейнера по критериям:
   - По типу (например, все квадраты).
   - По площади (например, все объекты с площадью меньше чем заданная).

## ВАРИАНТ ЗАДАНИЯ(№1):

- Контейнер 1-ого уровня – массив.
- Контейнер 2-ого уровня – массив.
- Фигура №1 – треугольник.
- Фигура №2 – прямоугольник.
- Фигура №3 – квадрат.

## ТЕОРЕТИЧЕСКАЯ ЧАСТЬ:

Контейнер в программировании — структура (АТД), позволяющая инкапсулировать в себе объекты любого типа. Объектами (переменными) контейнеров являются коллекции, которые уже могут содержать объекты одного определённого типа.

Например, в языке C++, std::list (шаблонный класс) является контейнером, а объект его класса-конкретизации (англ. instantiation), как, например, std::list<int> my_list является коллекцией.

Среди «широких масс» программистов наиболее известны контейнеры, построенные на основе шаблонов, однако существуют и реализации в виде библиотек (наиболее широко известна библиотека GLib). Кроме того, применяются и узкоспециализированные решения. Примерами контейнеров в C++ являются контейнеры из стандартной библиотеки (STL) — map, vector и др. В контейнерах часто встречается реализация алгоритмов для них. В ряде языков программирования (особенно скриптовых типа Perl или PHP) контейнеры и работа с ними встроена в язык.

Массив (в некоторых языках программирования также таблица, ряд, матрица) — тип или структура данных в виде набора компонентов (элементов массива), расположенных в памяти непосредственно друг за другом. При этом доступ к отдельным элементам массива осуществляется с помощью индексации, то есть через ссылку на массив с указанием номера (индекса) нужного элемента. За счёт этого, в отличие от списка, массив является структурой данных, пригодной для осуществления произвольного доступа к её ячейкам.

КОД ПРОГРАММЫ:

## main.cpp

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

#include "Figure.h"
#include "Vector.h"

using std::cin;
using std::cout;
using std::endl;

void menu();

int main() {
int key;
int a = 0;
int b = 0;
int c = 0;

std::shared_ptr<Figure> tri;
TMyVector<TMyVector <Figure>> *vector = new TMyVector <TMyVector <Figure>> (1);

Triangle *t = nullptr;
Rectangle *r = nullptr;
Foursquare *f = nullptr;
do {
menu();
cin >> key;
switch(key) {
case 1:
cout <<"Введите стороны треугольника:"<< endl;
cin >> a >> b >> c;
t = new Triangle(a, b, c);
tri = std::shared_ptr<Figure>(t);
vector->SubPush(tri);
break;
case 2:
cout <<"Введите стороны прямоугольника"<< endl;
cin >> a >> b;
r = new Rectangle(a, b);
tri = std::shared_ptr<Figure>(r);
vector->SubPush(tri);
break;
case 3:
cout <<"Введите стороны квадрата"<< endl;
cin >> a;
f = new Foursquare(a);
tri = std::shared_ptr<Figure>(f);
vector->SubPush(tri);
break;
case 4:
if(vector->GetSize() > 0) {
int gg;
cout <<"Удалить по площади (1) или по типу (2)?"<< endl;
cin >> gg;
if(gg == 1) {
cout <<"Введите площадь: "<< endl;
int s;
cin >> s;
vector->SDelete(s);
} else if(gg == 2) {
cout <<"Введите тип фигуры: треугольник (1), прямоугольник (2), квадрат (3)."<< endl;
```

```
int type;
cin >> type;
vector->TDelete(type);
} else {
cout <<"Попробуйте ещё раз"<< endl;
}
} else {
cout <<"Фигура не найдена"<< endl;
}
break;
case 5:
vector->~TMyVector();
key = 0;
break;
case 6:
cout << *vector;
break;
case 0:
break;
default:
cout <<"Неизвестная команда"<< endl;
break;
}
} while(key);
delete vector;
return 0;
}

void menu() {
std::cout <<"Введите"<< std::endl;
std::cout <<"1) Чтобы добавить треугольник."<< std::endl;
std::cout <<"2) Чтобы добавить прямоугольник."<< std::endl;
std::cout <<"3) Чтобы добавить квадрат."<< std::endl;
//   std::cout <<"4) Чтобы получить фигуру."<< std::endl;
std::cout <<"4) Чтобы удалить фигуру."<< std::endl;
std::cout <<"5) Чтобы удалить все фигуры."<< std::endl;
std::cout <<"6) Чтобы напечать массив."<< std::endl;
// std::cout <<"8) Цикл 'for' с итератором"<< std::endl;
std::cout <<"0) Выход"<< std::endl;
}
```

# Vector.h

```
#ifndef VECTOR_H
#define VECTOR_H

#include <memory>

#include "Figure.h"
#include "Triangle.h"
#include "Foursquare.h"
#include "Rectangle.h"
#include "Iterator.h"
#include "TAllocationBlock.h"
#include <stdlib.h>

template <class T> class TMyVector {
public:
TMyVector(int vCap); // конструктор класса
void Push(std::shared_ptr<T>&temp); // добавление элемента в класс
void SubPush(std::shared_ptr<Figure>&temp); // добавление элемента в класс
void Get(int index1, int index2); // получение элемента класса
void Delete(); // удаление последнего элемента класса
void DDelete(int index); // удаление последнего элемента класса
void TDelete(int index); // удаление по индексу
void SDelete(double index); // удаление по индексу
```

```
template <class A> friend std::ostream& operator<<(std::ostream& os,const TMyVector<A>& vector); // вывод в стандартный
поток
~TMyVector(); // деструктор класса
TIterator<T> begin(); // Установка итератора на начало
TIterator<T> end(); // Установка итератора на конец
int GetSize(); // получение размера класса
int GetCapacity(); // получить капасити
std::shared_ptr<T> *GetArray(); // получить массив
void MySort(int l, int r, std::shared_ptr<Figure> *array);
void * operator new (size_t size); // переопределенный оператор выделения памяти из кучи
void operator delete(void *p); // возвращение памяти в кучу

private:
void ResizeVector(std::shared_ptr<T> *&array); // изменение размера вектора
int size; // текущее количетсво элементов
int capacity; // выделенное место
std::shared_ptr<T> *array; // массив

static TAllocationBlock vector_allocator;
};

const int INCREASE = 2;
// конструктор класса
template <class T> TMyVector<T>::TMyVector(int vCap) {
array = new std::shared_ptr<T> [vCap];
size = 0;
capacity = vCap;
}

template <class T> std::shared_ptr<T> *TMyVector<T>::GetArray() {
return array;
}

template <class T> TAllocationBlock TMyVector<T>::vector_allocator(sizeof(TMyVector<T>), 100);
// быстрая сортировка
template <class T> void TMyVector<T>::MySort(int l, int r, std::shared_ptr<Figure> *array) {
int x = l + (r - l) / 2;
int i = l;
int j = r;

while(i <= j) {
while(array[i]->Square() < array[x]->Square()) {
i++;
}
while(array[j]->Square() > array[x]->Square()) {
j--;
}
if(i <= j) {
std::shared_ptr<Figure> tmp = array[i];
array[i] = array[j];
array[j] = tmp;
i++;
j--;
}
}
if (i < r) {
MySort(i, r, array);
}

if (l < j) {
MySort(l, j, array);
}
}

// изменение размера вектора
template <class T> void TMyVector<T>::ResizeVector(std::shared_ptr<T> *&array) {
capacity *= INCREASE;
std::shared_ptr<T> *tmp = new std::shared_ptr<T> [capacity];
```

```
for(int i = 0; i < size; ++i) {
tmp[i] = array[i];
array[i] = nullptr;
}
delete [] array;
array = nullptr;
array = tmp;
tmp = nullptr;
}
// добавление элемента в класс
template <class T> void TMyVector<T>::Push(std::shared_ptr<T>&temp) {
if(size == capacity) {
ResizeVector(array);
}
array[size] = temp;
++size;
}

template <class T> void TMyVector<T>::SubPush(std::shared_ptr<Figure>&temp) {
int i = 0;
int end = size;
while (i == 0 || i < end) {
if (size == 0 || (array[i]->GetSize() == 5 && i + 1 == end)) {
TMyVector<Figure> *slon = new TMyVector <Figure> (5);
std::shared_ptr<TMyVector<Figure>> pp = std::shared_ptr<TMyVector<Figure>> (slon);
Push(pp);
++i;
break;
} else if (array[i]->GetSize() != 5) {
break;
}
++i;
}
if(size == i) {
--i;
}
printf("i = %d\n", i);
(*(array + i))->Push(temp);
if (array[i]->GetSize() > 1) {
MySort(0, array[i]->GetSize() - 1, (*(array + i))->GetArray());
}
}
// получение элемента класса
template <class T> void TMyVector<T>::Get(int index1, int index2) {
(*(*array[index1]) + index2)->Print();
}
// удаление последнего элемента класса
template <class T> void TMyVector<T>::Delete() {
array[--size]->~T();
}
// new для своего аллокатора
template <class T> void * TMyVector<T>::operator new (size_t size) {
return vector_allocator.allocate();
}
// delete для своего аллокатора
template <class T> void TMyVector<T>::operator delete(void *p) {
vector_allocator.deallocate(p);
}
// удаление по типу
template <class T> void TMyVector<T>::TDelete(int index) {
for(int i = 0; i < size; ++i) {
for(int j = 0; j < (*(array + i))->GetSize(); ++j) {
if ((*((*(array + i))->GetArray() + j))->What() == index ) {
(*(array + i))->DDelete(j);
}
}
}
for(int i = 0; i < size; ++i) {
```

```
if((*(array + i))->GetSize() == 0) {
DDelete(i);
}
}
}
// удаление по площади
template <class T> void TMyVector<T>::SDelete(double index) {
for(int i = 0; i < size; ++i) {
for(int j = 0; j < (*(array + i))->GetSize(); ++j) {
if ((*((*(array + i))->GetArray() + j))->Square() == index ) {
(*(array + i))->DDelete(j);
}
}
}
for(int i = 0; i < size; ++i) {
if((*(array + i))->GetSize() == 0) {
DDelete(i);
}
}
}

template <class T> void TMyVector<T>::DDelete(int index) {
std::shared_ptr<T> tmp;
for(int i = index; i < size - 1; ++i) {
tmp = array[i];
array[i] = array[i + 1];
}
Delete();
}

// Установка итератора на начало
template <class T> TIterator<T> TMyVector<T>::begin() {
return TIterator<T>(array);
}
// Установка итератора на конец
template <class T> TIterator<T> TMyVector<T>::end() {
return TIterator<T>(array + size);
}
// деструктор класса
template <class T> TMyVector<T>::~TMyVector() {
for(int i = 0; i < size; ++i) {
array[i] = nullptr;
}
delete [] array;
array = nullptr;
capacity = size = 0;
}
// вывод в стандартный поток
template <class T> std::ostream& operator<<(std::ostream& os, const TMyVector<T>& vector) {
for(int j = 0; j < vector.size; ++j) {
printf("Array %d:\n", j);
printf("Size %d:\n", vector.array[j]->size);
for(int i = 0; i < vector.array[j]->size; ++i) {
vector.array[j]->array[i]->Print();
}
printf("================\n");
}
return os;
}
// получение размера класса
template <class T> int TMyVector<T>::GetSize() {
return size;
}

template <class T> int TMyVector<T>::GetCapacity() {
return capacity;
}
```

```
#endif
```

## Figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

#include <iostream>
// Superclass Figure
class Figure {
public:
virtual void Print() = 0; // Change for Trg, Rec, Fout.
virtual double Square() = 0; // Change for Trg, Rec, Fout.
virtual int What() = 0;
virtual ~Figure() {};
// virtual friend std::ostream& operator<<(std::ostream& os, const Figure& obj);
};

#endif
```

## В классе фигур добавляется метод, определяющий тип фигуры:

```
int Triangle::What() {
return 1;
}

int Rectangle::What() {
return 2;
}

int Foursquare::What() {
return 3;
}
```

# ТЕСТЫ:

```
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab7$ ./lab7
TAllocationBlock: Память инициализирована
TAllocationBlock: Память инициализирована
TAllocationBlock: Веделено 1 из 100
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
1
Введите стороны треугольника:
3 4 5
Треугольник создан 3, 4, 5
TAllocationBlock: Веделено 1 из 100
i = 0
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
2
Введите стороны прямоугольника
3 4
Прямоугольник создан: 3, 4
i = 0
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
3
Введите стороны квадрата
5
Квадрат создан: 5
i = 0
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
6
Array 0:
Size 3:
a=3, b=4, c=5
a=3, b=4, c=3, d=4
a=5, b=5, c=5, d=5
================
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
```

```
0) Выход
4
Удалить по площади (1) или по типу (2)?
1
Введите площадь:
25
Квадрат удалён
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
4
Удалить по площади (1) или по типу (2)?
2
Введите тип фигуры: треугольник (1), прямоугольник (2), квадрат (3).
2
Прямоугольник удалён
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
6
Array 0:
Size 1:
a=3, b=4, c=5
================
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
3
Введите стороны квадрата
5
Квадрат создан: 5
i = 0
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
2
Введите стороны прямоугольника
7 8
Прямоугольник создан: 7, 8
i = 0
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
```

```
6
Array 0:
Size 3:
a=3, b=4, c=5
a=5, b=5, c=5, d=5
a=7, b=8, c=7, d=8
================
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
2
Введите стороны прямоугольника
4 5
Прямоугольник создан: 4, 5
i = 0
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
1
Введите стороны треугольника:
5 6 7
Треугольник создан 5, 6, 7
i = 0
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
6
Array 0:
Size 5:
a=3, b=4, c=5
a=5, b=6, c=7
a=4, b=5, c=4, d=5
a=5, b=5, c=5, d=5
a=7, b=8, c=7, d=8
================
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
2
Введите стороны прямоугольника
2 3
Прямоугольник создан: 2, 3
TAllocationBlock: Веделено 2 из 100
i = 1
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
```

```
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
6
Array 0:
Size 5:
a=3, b=4, c=5
a=5, b=6, c=7
a=4, b=5, c=4, d=5
a=5, b=5, c=5, d=5
a=7, b=8, c=7, d=8
================
Array 1:
Size 1:
a=2, b=3, c=2, d=3
================
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы удалить фигуру.
5) Чтобы удалить все фигуры.
6) Чтобы напечать массив.
0) Выход
0
Треугольник удалён
Треугольник удалён
Прямоугольник удалён
Квадрат удалён
Прямоугольник удалён
Прямоугольник удалён
TAllocationBlock: Блок был возвращён
TAllocationBlock: Блок был возвращён
TAllocationBlock: Блок был возвращён
TAllocationBlock: Память освобождена
TAllocationBlock: Память освобождена
```

# ЛАБОРАТОРНАЯ РАБОТА №8

## ЦЕЛЬ РАБОТЫ:

- Знакомство с параллельным программированием в C++.

## ПОСТАНОВКА ЗАДАЧИ:

Используя структуры данных, разработанные для лабораторной работы No6 (контейнер первого уровня и классы-фигуры) разработать алгоритм быстрой сортировки для класса -контейнера .

Необходимо разработать два вида алгоритма:

- Обычный, без параллельных вызовов.
- С использованием параллельных вызовов. В этом случае, каждый рекурсивный вызов сортировки должен создаваться в отдельном потоке.
-

Для создания потоков использовать механизмы:

- future
- packaged_task/async
-

Для обеспечения потоко-безопасности структур данных использовать:

- mutexlock_guard

Нельзя использовать:

- Стандартныеконтейнеры std.

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.
- Проводить сортировку контейнера

## ВАРИАНТ ЗАДАНИЯ(№1):

- Контейнер 1-ого уровня – массив.
- Фигура №1 – треугольник.
- Фигура №2 – прямоугольник.
- Фигура №3 – квадрат.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ:

Параллельное программирование - это техника программирования, которая использует преимущества многоядерных или многопроцессорных компьютеров и является подмножеством более широкого понятия многопоточности (multithreading).
Параллельное программирование может быть сложным, но его легче понять, если считать не "трудным", а просто "немного иным". Оно включает в себя все черты более традиционного, последовательного программирования, но в параллельном программировании имеются три дополнительных, четко определенных этапа.
Определение параллелизма: анализ задачи с целью выделить подзадачи, которые могут выполняться одновременно
Выявление параллелизма: изменение структуры задачи таким образом, чтобы можно было эффективно выполнять подзадачи. Для этого часто требуется найти зависимости между подзадачами и организовать исходный код так, чтобы ими можно было эффективно управлять
Выражение параллелизма: реализация параллельного алгоритма в исходном коде с помощью системы обозначений параллельного программирования.

КОД ПРОГРАММЫ:

## Vector.h

```
#ifndef VECTOR_H
#define VECTOR_H

#include <iostream>
#include <functional>
#include <memory>
#include <future>
#include <mutex>
#include <thread>


#include "Figure.h"
#include "Triangle.h"
#include "Foursquare.h"
#include "Rectangle.h"
#include "Iterator.h"
#include "TAllocationBlock.h"

template <class T> class TMyVector {
public:
TMyVector(int vCap); // конструктор класса
void Push(std::shared_ptr<T>&temp); // добавление элемента в класс
void Get(int index); // получение элемента класса
void Delete(); // удаление последнего элемента класса
// void Delete(int index); // удаление по индексу
template <class A> friend std::ostream& operator<<(std::ostream& os,const TMyVector<A>& vector); // вывод в стандартный
поток
~TMyVector(); // деструктор класса
TIterator<T> begin(); // Установка итератора на начало
TIterator<T> end(); // Установка итератора на конец
int GetSize(); // получение размера класса
void * operator new (size_t size); // переопределенный оператор выделения памяти из кучи
void operator delete(void *p); // возвращение памяти в кучу
int MySort(int l, int r);
void Quicksort_parallel(int l, int r);

private:
void ResizeVector(std::shared_ptr<T> *&array); // изменение размера вектора
int size; // текущее количетсво элементов
int capacity; // выделенное место
std::shared_ptr<T> *array; // массив

static TAllocationBlock vector_allocator;
};

#include "Vector.h"
#include <stdlib.h>

const int INCREASE = 2;
// конструктор класса
template <class T> TMyVector<T>::TMyVector(int vCap) {
array = new std::shared_ptr<T> [vCap];
size = 0;
capacity = vCap;
}

template <class T> TAllocationBlock TMyVector<T>::vector_allocator(sizeof(TMyVector<T>), 100);

template <class T> void TMyVector<T>::Quicksort_parallel(int l, int r) {
int i = l, j = r;
int x = l + (r - l) / 2;
while (i <= j) {
```

```
while (array[i]->Square() < array[x]->Square()) {
i++;
}
while (array[j]->Square() > array[x]->Square()) {
j--;
}
if (i <= j) {
std::shared_ptr<Figure> tmp = array[i];
array[i] = array[j];
array[j] = tmp;
i++;
j--;
}
}
if (i < r) {
std::packaged_task<void(void)> task(bind(std::mem_fn(&TMyVector<T>::Quicksort_parallel), this, i, r));
std::thread tht(move(task));
tht.detach();
}
if (l < j) {
std::packaged_task<void(void)> task(bind(std::mem_fn(&TMyVector<T>::Quicksort_parallel), this, l, j));
std::thread thtw(move(task));
thtw.detach();
}
}

template <class T> int TMyVector<T>::MySort(int l, int r) {
int x = l + (r - l) / 2;
int i = l;
int j = r;

while(i <= j) {
while(array[i]->Square() < array[x]->Square()) {
i++;
}
while(array[j]->Square() > array[x]->Square()) {
j--;
}
if(i <= j) {
std::shared_ptr<Figure> tmp = array[i];
array[i] = array[j];
array[j] = tmp;
i++;
j--;
}
}
if (i < r) {
MySort(i, r);
}

if (l < j) {
MySort(l, j);
}
return 0;
}

// изменение размера вектора
template <class T> void TMyVector<T>::ResizeVector(std::shared_ptr<T> *&array) {
capacity *= INCREASE;
std::shared_ptr<T> *tmp = new std::shared_ptr<T> [capacity];
for(int i = 0; i < size; ++i) {
tmp[i] = array[i];
array[i] = nullptr;
}
delete [] array;
array = nullptr;
array = tmp;
tmp = nullptr;
```

```
}
// добавление элемента в класс
template <class T> void TMyVector<T>::Push(std::shared_ptr<T>&temp) {
if(size == capacity) {
ResizeVector(array);
}
array[size] = temp;
++size;
}
// получение элемента класса
template <class T> void TMyVector<T>::Get(int index) {
array[index]->Print();
}
// удаление последнего элемента класса
template <class T> void TMyVector<T>::Delete() {
array[--size]->~T();
}
// new для своего аллокатора
template <class T> void * TMyVector<T>::operator new (size_t size) {
return vector_allocator.allocate();
}
// delete для своего аллокатора
template <class T> void TMyVector<T>::operator delete(void *p) {
vector_allocator.deallocate(p);
}

// void TMyVector::Delete(int index) {
//     int i;
//     int j;
//     T *temp = (T *)malloc(sizeof(T) * (size - 1));
//     for(j = 0, i = 0; i < size; ++i) {
//         if(i != index) {
//             temp[j] = array[i];
//             ++j;
//         }
//     }
//     free(array);
//     array = temp;
//     --size;
// }

// Установка итератора на начало
template <class T> TIterator<T> TMyVector<T>::begin() {
return TIterator<T>(array);
}
// Установка итератора на конец
template <class T> TIterator<T> TMyVector<T>::end() {
return TIterator<T>(array + size);
}
// деструктор класса
template <class T> TMyVector<T>::~TMyVector() {
for(int i = 0; i < size; ++i) {
array[i] = nullptr;
}
delete [] array;
array = nullptr;
capacity = size = 0;
}
// вывод в стандартный поток
template <class T> std::ostream& operator<<(std::ostream& os, const TMyVector<T>& vector) {
for(int i = 0; i < vector.size; ++i) {
vector.array[i]->Print();
}
return os;
}
// получение размера класса
template <class T> int TMyVector<T>::GetSize() {
return size;
```

```
}

#endif
```

**Все остальные методы и классы остаются прежними. Однако, при компиляции необходим новый дополнительный ключ - «-pthread».**

ТЕСТЫ:

```
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab8$ ./lab8
TAllocationBlock: Память инициализирована
TAllocationBlock: Веделено 1 of 100
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Напечатать массив при помощи итератора и быстрой сортировки
9) Напечатать массив при помощи итератора и параллельной сортировки
0) Выход
1
Введите стороны треугольника:
3 4 5
Треугольник создан: 3, 4, 5
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Напечатать массив при помощи итератора и быстрой сортировки
9) Напечатать массив при помощи итератора и параллельной сортировки
0) Выход
1
Введите стороны треугольника:
13 15 17
Треугольник создан: 13, 15, 17
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Напечатать массив при помощи итератора и быстрой сортировки
9) Напечатать массив при помощи итератора и параллельной сортировки
0) Выход
1
Введите стороны треугольника:
10 9 12
Треугольник создан: 10, 9, 12
Введите
```

1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Напечатать массив при помощи итератора и быстрой сортировки
9) Напечатать массив при помощи итератора и параллельной сортировки
0) Выход
7
a=3, b=4, c=5
a=13, b=15, c=17
a=10, b=9, c=12
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Напечатать массив при помощи итератора и быстрой сортировки
9) Напечатать массив при помощи итератора и параллельной сортировки
0) Выход
9
a=3, b=4, c=5
a=10, b=9, c=12
a=13, b=15, c=17
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Напечатать массив при помощи итератора и быстрой сортировки
9) Напечатать массив при помощи итератора и параллельной сортировки
0) Выход
1
Введите стороны треугольника:
1 2 3
Треугольник создан: 1, 2, 3
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.

7) Чтобы напечать массив.
8) Напечатать массив при помощи итератора и быстрой сортировки
9) Напечатать массив при помощи итератора и параллельной сортировки
0) Выход
7
a=3, b=4, c=5
a=10, b=9, c=12
a=13, b=15, c=17
a=1, b=2, c=3
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Напечатать массив при помощи итератора и быстрой сортировки
9) Напечатать массив при помощи итератора и параллельной сортировки
0) Выход
8
a=1, b=2, c=3
a=3, b=4, c=5
a=10, b=9, c=12
a=13, b=15, c=17
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Напечатать массив при помощи итератора и быстрой сортировки
9) Напечатать массив при помощи итератора и параллельной сортировки
0) Выход
5
Треугольник удалён
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Напечатать массив при помощи итератора и быстрой сортировки
9) Напечатать массив при помощи итератора и параллельной сортировки
0) Выход
9
a=1, b=2, c=3

a=3, b=4, c=5
a=10, b=9, c=12
Введите
1) Чтобы добавить треугольник.
2) Чтобы добавить прямоугольник.
3) Чтобы добавить квадрат.
4) Чтобы получить фигуру.
5) Чтобы удалить фигуру.
6) Чтобы удалить все фигуры.
7) Чтобы напечать массив.
8) Напечатать массив при помощи итератора и быстрой сортировки
9) Напечатать массив при помощи итератора и параллельной сортировки
0) Выход
6
Треугольник удалён
Треугольник удалён
Треугольник удалён
TAllocationBlock: Блок был возвращён
TAllocationBlock: Память освобождена
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab8$

# ЛАБОРАТОРНАЯ РАБОТА №9

## ЦЕЛЬ РАБОТЫ:

- Знакомство с лямбда-выражениями

## ПОСТАНОВКА ЗАДАЧИ:

Используя структуры данных, разработанные для лабораторной работы №6(контейнер первого уровня и классы-фигуры) необходимо разработать:

- Контейнер второго уровня с использованием шаблонов.
- Реализовать с помощью лямбда-выражений набор команд, совершающих операции над контейнером 1-го уровня:
  - Генерация фигур со случайным значением параметров;
  - Печать контейнера на экран;
  - Удаление элементов со значением площади меньше определенного числа;
- В контейнер второго уровня поместить цепочку команд.
- Реализовать цикл, который проходит по всем командам в контейнере второго уровня и выполняет их, применяя к контейнеру первого уровня.

Для создания потоков использовать механизмы:

- future
- packaged_task/async

Для обеспечения потоко-безопасности структур данных использовать:

- mutex
- lock_guard

Нельзя использовать:

- Стандартные контейнеры std.

## ВАРИАНТ ЗАДАНИЯ(№1):

- Контейнер 1-ого уровня – массив.
  - Фигура №1 – треугольник.
  - Фигура №2 – прямоугольник.
  - Фигура №3 – квадрат.
- Контейнер 2-ого уровня – массив.

## ТЕОРЕТИЧЕСКАЯ ЧАСТЬ:

Лямбда-выражения в C++ – это краткая форма записи анонимных функторов. Они позволяют определить анонимный объект-функцию прямо внутри кода или передать функцию в качестве аргумента.

КОД ПРОГРАММЫ:

Для того, чтобы хранить команды и передавать и потом выполнять их в цикле над контейнером первого уровня, мне пришлось реализовать второй массив в файлах: TTVectorItem.h , TTVectorItem.cpp и сами методы в TTVector.cpp.

## TTVector.cpp

```
#include <iostream>
#include <memory>
#include "TTVector.h"

template <class T>
TTVector<T>::TTVector()
{
head = nullptr;
count = 0;
}



template <class T>
void TTVector<T>::Push(const T &item)
{
if(count==0) {
        TTVectorItem<T> *tmp = new TTVectorItem<T>(item, head);
          head = tmp;
        } else {
          TTVectorItem<T> *el = new TTVectorItem<T>(item, head);
          auto tmp = head;
          while (tmp->next) {
                tmp =tmp->next;
          }
          tmp->next = el;
        }

++count;
}

template <class T>
bool TTVector<T>::IsEmpty() const
{
return !count;
}

template <class T>
uint32_t TTVector<T>::GetSize() const
{
return count;
}

template <class T>
void TTVector<T>::Pop()
{
if(head) {
TTVectorItem<T> *tmp = &head->GetNext();
delete head;
head = tmp;
--count;
}
}
```

```
template <class T>
T &TTVector<T>::Top()
{
return head->Pop();
}

template <class T>
TTVector<T>::~TTVector()
{
for(TTVectorItem<T> *tmp = head, *tmp2; tmp; tmp = tmp2) {
tmp2 = &tmp->GetNext();
delete tmp;
}
}


typedef unsigned char Byte;

template class TTVector<void *>;
template class TTVector<std::shared_ptr<std::function<void(void)>>>;
```

## main.cpp

В функции main.cpp мы изменили достаточно много: генирируя случайные значения для фигур (т.е. тип самой фигуры и длины её сторон) выполняется перечень команд. Добавив несколько фигур в массив мы печатаем его и пробуем удалить фигуры у которых площадь меньше чем сгенирированный параметр sqr. Затем снова пытаемся напечатать массив.

```
#include <iostream>
#include <memory>
#include <cstdlib>
#include <cstring>
#include <random>
#include "Triangle.h"
#include "Foursquare.h"
#include "Rectangle.h"
#include "TVector.h"
#include "TTVector.h"


int main(void)
{
TVector<Figure> vector;
typedef std::function<void(void)> Command;
TTVector<std::shared_ptr<Command>> myvector;
std::mutex mtx;

Command cmdInsert = [&]() {
std::lock_guard<std::mutex> guard(mtx);

uint32_t seed = std::chrono::system_clock::now().time_since_epoch().count();

std::default_random_engine generator(seed);
std::uniform_int_distribution<int> distFigureType(1, 3);
std::uniform_int_distribution<int> distFigureParam(1, 10);
for (int i = 0; i < 5; ++ i) {
std::cout <<"Команда: Вставка"<< std::endl;
```

```
switch(distFigureType(generator)) {
case 1: {
std::cout <<"Треугольник добавлен"<< std::endl;

int side_a = distFigureParam(generator);
int side_b = distFigureParam(generator);
int side_c = distFigureParam(generator);

std::shared_ptr<Figure> ptr = std::make_shared<Triangle>(Triangle(side_a, side_b, side_c));
vector.PushFirst(ptr);
break;
}

case 2: {
std::cout <<"Прямоугольник добавлен"<< std::endl;

int side = distFigureParam(generator);
std::shared_ptr<Figure> ptr = std::make_shared<Rectangle>(Rectangle(side_a, side_b));
vector.PushFirst(ptr);

break;
}

case 3: {
std::cout <<"Квадрат добавлен"<< std::endl;
int side = distFigureParam(generator);
std::shared_ptr<Figure> ptr = std::make_shared<Foursquare>(Foursquare(side));

vector.PushFirst(ptr);

break;
}
}
}
};



Command cmdRemove = [&]() {
std::lock_guard<std::mutex> guard(mtx);

std::cout <<"Команда: Удаление"<< std::endl;

if (vector.IsEmpty()) {
std::cout <<"Массив пуст"<< std::endl;
} else {
uint32_t seed = std::chrono::system_clock::now().time_since_epoch().count();

std::default_random_engine generator(seed);
std::uniform_int_distribution<int> distSquare(1, 150);
double sqr = distSquare(generator);
std::cout <<"Меньше чем "<< sqr << std::endl;

for (int32_t i = 0; i < 5; ++i) {
auto iter = vector.begin();
for (int32_t k = 0; k < vector.GetLength(); ++k) {
if (iter->Square() < sqr) {
vector.Pop(k);
break;
}
++iter;
}
}
}
```

```
};

Command cmdPrint = [&]() {
std::lock_guard<std::mutex> guard(mtx);

std::cout <<"Команда: Печать"<< std::endl;
if(!vector.IsEmpty()) {
std::cout << vector << std::endl;
} else {
std::cout <<"Массив пуст"<< std::endl;
}
};
myvector.Push(std::shared_ptr<Command>(&cmdInsert, [](Command*){}));
myvector.Push(std::shared_ptr<Command>(&cmdPrint, [](Command*){}));
myvector.Push(std::shared_ptr<Command>(&cmdRemove, [](Command*){}));
myvector.Push(std::shared_ptr<Command>(&cmdPrint, [](Command*){}));


while (!myvector.IsEmpty()) {
std::shared_ptr<Command> cmd = myvector.Top();
std::future<void> ft = std::async(*cmd);
ft.get();
myvector.Pop();
}

return 0;
}
```

# ТЕСТЫ:

```
dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab9$ ./run
Команда: Вставка
Треугольник добавлен
Команда: Вставка
Прямоугольник добавлен
Команда: Вставка
Квадрат добавлен
Команда: Вставка
Прямоугольник добавлен
Команда: Вставка
Квадрат добавлен
Команда: Печать
Индекс: 0   Сторона= 9, Тип: квадрат

Индекс: 1   Сторона a = 4 Сторона b =3, Тип: прямоугольник

Индекс: 2   Сторона= 6, Тип: квадрат

Индекс: 3   Сторона a = 5 Сторона b =4, Тип: прямоугольник

Индекс: 4   Сторона a = 10, Сторона b = 8, Сторона c = 8, Тип: треугольник


Команда: Удаление
Меньше чем 94
Команда: Печать

dom@dom-HP-250-G3-Notebook-PC:~/Desktop/myOOP/lab9$ ./run
Команда: Вставка
Треугольник добавлен
Команда: Вставка
Квадрат добавлен
Команда: Вставка
Квадрат добавлен
Команда: Вставка
Прямоугольник добавлен
Команда: Вставка
Прямоугольник добавлен
Команда: Печать
Индекс: 0   Сторона a = 2 Сторона b =1, Тип: прямоугольник

Индекс: 1   Сторона a = 10 Сторона b =9, Тип: прямоугольник

Индекс: 2   Сторона= 5, Тип: квадрат

Индекс: 3   Сторона= 10, Тип: квадрат

Индекс: 4   Сторона a = 3, Сторона b = 10, Сторона c = 8, Тип: треугольник


Команда: Удаление
Меньше чем 77
Команда: Печать
Индекс: 0   Сторона a = 10 Сторона b =9, Тип: прямоугольник

Индекс: 1   Сторона= 10, Тип: квадрат
```

В первом случае были удалены все фигуры, так как параметр sqr = 94, и все фигуры, чья площадь меньше чем 94 должны быть удалены. Во втором случае есть фигуры с площадью больше чем 77.

Выводы по курсу “Объектно-ориентированное программирование”

Познакомившись с ОО парадигмой программирования я узнал, что основной метод в этой области – это представление программы в виде некоторой совокупности объектов одного вида, типа, которые обладают одними и теми же свойствами, операциями и функциями. Эти объекты образуют семейства – так называемые “Классы”, которые могут быть наследуемые. Это означает, что их некоторые свойства могут перенимать объекты другого определенного класса.

Данный курс являлся крайне интересным, продолжительным проектом, в котором, на протяжении всего семестра мне приходилось создавать новые “вещи”, дополнять старые и улучшать работу этой программы. Такая практика очень интересна и позволяет студенту ближе понять принцип написания большого проекта. Благодаря этому проекту я научился применять новые средства, работать с классами и достаточно близко познакомился с новым для меня языком программирования – C++. В этом семестре мне необходимо было выполнить 9 лабораторных работ и познакомится определенными вещами:

I. Лабораторная работа
   - Изучение базовых понятий ООП.
   - Знакомство с классами в C++.
   - Знакомство с перегрузкой операторов.
   - Знакомство с дружественными функциями.
   - Знакомство с операциями ввода-вывода из стандартных библиотек.

II. Лабораторная работа
   - Создание простых динамических структур данных.
   - Закрепление навыков работы с классами.
   - Работа с объектами, передаваемыми «по значению».

III. Лабораторная работа
   - Закрепление навыков работы с классами.
   - Знакомство с умными указателями

IV. Лабораторная работа
   - Построение шаблонов динамических структур данных.
   - Знакомство с шаблонами классов.

V. Лабораторная работа
   - Построение итераторов для динамических структур данных.
   - Закрепление навыков работы с шаблонами классов.

VI. Лабораторная работа
   - Создание аллокаторов памяти для динамических структур данных.
   - Закрепление навыков по работе с памятью в C++.

VII. Лабораторная работа
   - Создание сложных динамических структур данных.
   - Закрепление принципа OCP.

VIII. Лабораторная работа
   - Знакомство с параллельным программированием в C++.

IX. Лабораторная работа

- Знакомство с лямбда-выражениями

В итоге выполнения этого проекта, я получил хорошие навыки программирования и проектирования на C++. Освоил ряд возможностей языка, которые уже широко использую. Я узнал, как создавать свой аллокатор памяти, как работать с многопоточностью и шаблонами. Думаю, что буду интересоваться и изучать эту парадигму и дальше.