Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»
Факультет: «Прикладная математика и физика»

Отчет по лабораторным работам
по курсу
«Объектно-ориентированное программирование»

Группа: М8О – 204Б-16
Студент: Девяткина Д.В.
Преподаватель: Поповкин А.В.
Вариант: №4
Дата:
Оценка:
Подпись:

Москва
2017

Лабораторная работа №1

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.

2. Задание

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 общий виртуальный метод *Print*, печатающий параметры фигуры и ее тип в стандартный поток вывода *cout*.
- Должный иметь общий виртуальный метод расчета площади фигуры – *Square*.
- Должны иметь конструктор, считывающий значения основных параметров фигуры из стандартного потока *cin*.
- Должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Вариант фигур: Трапеция, Ромб, Пятиугольник.

3. Описание

Появление **объектно-ориентированного программирования** стало результатом возросших требований к функционалу программ, когда описывать объект приходилось раз за разом в разных участках кода.

Класс — это шаблон, описание ещё не созданного объекта. Класс содержит данные, которые описывают строение объекта и его возможности, методы работы с ним; **Объект** — экземпляр класса. То, что «рождено» по «чертежу», то есть по описанию из класса. В качестве примера объекта и класса можно привести технический чертёж для изготовления детали — это класс. Выточенная же на станке по размерам и указаниям из чертежа деталь - объект.

Абстракция — способ выделения самых значимых характеристик объекта, при этом менее значимые отбрасываются. В ООП абстракция - работа только со значимыми характеристиками.

Инкапсуляция — принцип **объектно-ориентированного программирования**, позволяющий собрать объект в пределах одной структуры или массива, убрав способ обработки данных и сами данные от «чужих глаз».

Наследование — способность в **объектно-ориентированном программировании** построить новый класс на основе уже заданного. При этом функционал может как полностью совпадать, так и отличаться.

Полиморфизм — способность объектов самим определять, какие методы они должны применить в зависимости от того, где именно в коде они находятся.

Конструктор — это метод, имеющий имя, совпадающее с именем класса. Две основные задачи конструктора заключаются в **выделении памяти** под вновь создаваемый объект и его **инициализация**. Могут существовать и переменные класса — это такие переменные, которые всегда имеются ровно в одном экземпляре, независимо от того как много имеется объектов данного класса.

Виртуальная функция — это функция, которая определяется в базовом классе, а любой порожденный класс может ее переопределить.

Для использования объектно-ориентированного консольного ввода-вывода с помощью потоков (stream) STL в программу необходимо включить заголовочный файл `<iostream>`. Потоки *cin*, *cout* и *cerr* соответствуют потокам *stdin*, *stdout* и *stderr* соответственно.

4. Исходный код

Pentagon.h

```
#pragma once
#ifndef Pentagon_H
#define Pentagon_H

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

class Pentagon : public Figure
{
public:
	Pentagon();
	Pentagon(std::istream &is);
	Pentagon(size_t i, size_t j, size_t k, size_t l, size_t m, size_t a);
	Pentagon(const Pentagon& orig);

	double Square() override;
	void Print() override;
	virtual ~Pentagon();

private:
	size_t side1;
	size_t side2;
	size_t side3;
	size_t side4;
	size_t side5;
	size_t apothem;
};

#endif
```

Pentagon.cpp

```
#include "Pentagon.h"

#include <iostream>
#include <cmath>
```

```
Pentagon::Pentagon() : Pentagon(0, 0, 0, 0, 0, 0)
{
}

Pentagon::Pentagon(size_t i, size_t j, size_t k, size_t l, size_t m, size_t a) :
side1(i), side2(j), side3(k), side4(l), side5(m), apothem(a)
{
	std::cout << "=========================" << std::endl;
	std::cout << "Pentagon created: " << std::endl;
	std::cout << "side 1 = " << side1 << std::endl;
	std::cout << "side 2 = " << side2 << std::endl;
	std::cout << "side 3 = " << side3 << std::endl;
	std::cout << "side 4 = " << side4 << std::endl;
	std::cout << "side 5 = " << side5 << std::endl;
	std::cout << "apothem = " << apothem << std::endl;
}

Pentagon::Pentagon(std::istream &is)
{
	std::cout << "=========================" << std::endl;
	std::cout << "Enter side 1: ";
	is >> side1;
	std::cout << "Enter side 2: ";
	is >> side2;
	std::cout << "Enter side 3: ";
	is >> side3;
	std::cout << "Enter side 4: ";
	is >> side4;
	std::cout << "Enter side 5: ";
	is >> side5;
	std::cout << "Enter apothem: ";
	is >> apothem;
}

Pentagon::Pentagon(const Pentagon& orig)
{
	std::cout << "Pentagon copy created" << std::endl;
	side1 = orig.side1;
	side2 = orig.side2;
	side3 = orig.side3;
	side4 = orig.side4;
	side5 = orig.side5;
	apothem = orig.apothem;
}

double Pentagon::Square()
{
	return (double(side1 + side2 + side3 + side4 + side5) / 2.0) * double(apothem);
}

void Pentagon::Print()
{
	std::cout << "=========================" << std::endl;
	std::cout << "Figure type - Pentagon " << std::endl;
	std::cout << "Size of side 1: " << side1 << std::endl;
	std::cout << "Size of side 2: " << side2 << std::endl;
	std::cout << "Size of side 3: " << side3 << std::endl;
	std::cout << "Size of side 4: " << side4 << std::endl;
	std::cout << "Size of side 5: " << side5 << std::endl;
	std::cout << "Size of apothem: " << apothem << std::endl;
}

Pentagon::~Pentagon()
{
	std::cout << "=========================" << std::endl;
```

```
	std::cout << "Pentagon deleted" << std::endl;
}
```

Rhomb.cpp

```
#include "Rhomb.h"

#include <iostream>
#include <cmath>

Rhomb::Rhomb() : Rhomb(0, 0)
{
}

Rhomb::Rhomb(size_t i, size_t j) : side(i), height(j)
{
	std::cout << "=========================" << std::endl;
	std::cout << "Rhomb created: " << std::endl;
	std::cout << "side = " << side << std::endl;
	std::cout << "height = " << height << std::endl;
}

Rhomb::Rhomb(std::istream &is)
{
	std::cout << "=========================" << std::endl;
	std::cout << "Enter side: ";
	is >> side;
	std::cout << "Enter height: ";
	is >> height;
}

Rhomb::Rhomb(const Rhomb& orig)
{
	std::cout << "Rhomb copy created" << std::endl;
	side = orig.side;
	height = orig.height;
}

double Rhomb::Square()
{
	return double(side) * double(height);
}

void Rhomb::Print()
{
	std::cout << "=========================" << std::endl;
	std::cout << "Figure type - Rhomb " << std::endl;
	std::cout << "Size of side: " << side << std::endl;
	std::cout << "Height: " << height << std::endl;
}

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

Rhomb.h

```
#pragma once
#ifndef Rhomb_H
#define Rhomb_H

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

class Rhomb : public Figure
```

```
{
public:
	Rhomb();
	Rhomb(std::istream &is);
	Rhomb(size_t i, size_t j);
	Rhomb(const Rhomb& orig);

	double Square() override;
	void Print() override;
	virtual ~Rhomb();

private:
	size_t side;
	size_t height;
};
#endif
```

Trapeze.cpp

```
#include "Trapeze.h"

#include <iostream>
#include <cmath>

Trapeze::Trapeze() : Trapeze(0, 0, 0)
{
}

Trapeze::Trapeze(size_t i, size_t j, size_t k) : side_a(i), side_b(j), height_h(k)
{
	std::cout << "=========================" << std::endl;
	std::cout << "Trapeze created: " << std::endl;
	std::cout << "side a = " << side_a << std::endl;
	std::cout << "side b = " << side_b << std::endl;
	std::cout << "height_h = " << height_h << std::endl;
}

Trapeze::Trapeze(std::istream &is)
{
	std::cout << "=========================" << std::endl;
	std::cout << "Enter side a: ";
	is >> side_a;
	std::cout << "Enter side b: ";
	is >> side_b;
	std::cout << "Enter height h: ";
	is >> height_h;
}

Trapeze::Trapeze(const Trapeze& orig)
{
	std::cout << "Trapeze copy created" << std::endl;
	side_a = orig.side_a;
	side_b = orig.side_b;
	height_h = orig.height_h;
}

double Trapeze::Square()
{
	return double(side_a + side_b) * double(height_h) / 2.0;
}

void Trapeze::Print()
{
	std::cout << "=========================" << std::endl;
	std::cout << "Figure type - trapeze " << std::endl;
	std::cout << "Size of side a: " << side_a << std::endl;
	std::cout << "Size of side b: " << side_b << std::endl;
```

```
	std::cout << "Height: " << height_h << std::endl;
}

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

Trapeze.h

```
#pragma once
#ifndef Trapeze_H
#define Trapeze_H
#include <cstdlib>
#include <iostream>
#include "Figure.h"

class Trapeze : public Figure
{
public:
	Trapeze();
	Trapeze(std::istream &is);
	Trapeze(size_t i, size_t j, size_t k);
	Trapeze(const Trapeze& orig);

	double Square() override;
	void Print() override;
	virtual ~Trapeze();
private:
	size_t side_a;
	size_t side_b;
	size_t height_h;
};
#endif
```

5. Консоль

```
==========================
Menu:
1) Trapeze
2) Rhomb
3) Pentagon
4) Exit
==========================
Choose action:
1
==========================
You chose 1) Trapeze
==========================
Enter side a: 8
Enter side b: 7
Enter height h: 6
==========================
Figure type - trapeze
Size of side a: 8
Size of side b: 7
Height: 6
Square = 45
==========================
Trapeze deleted
==========================
Menu:
1) Trapeze
2) Rhomb
3) Pentagon
4) Exit
==========================
Choose action:
2
==========================
You chose 2) Rhomb
==========================
Enter side: 8
Enter height: 4
==========================
Figure type - Rhomb
Size of side: 8
Height: 4
Square = 32
==========================
Rhomb deleted
==========================
Menu:
1) Trapeze
2) Rhomb
3) Pentagon
4) Exit
==========================
Choose action:
3
==========================
You chose 3) Pentagon
==========================
Enter side 1: 8
Enter side 2: 7
Enter side 3: 6
Enter side 4: 9
Enter side 5: 8
Enter apothem: 3
==========================
Figure type - Pentagon
Size of side 1: 8
Size of side 2: 7
Size of side 3: 6
Size of side 4: 9
Size of side 5: 8
Size of apothem: 3
Square = 57
==========================
Pentagon deleted
==========================
Menu:
1) Trapeze
2) Rhomb
3) Pentagon
4) Exit
==========================
Choose action: 4
```

6. Вывод

В этой лабораторной работе у меня был первый опят с языком C++ и объектно-ориентированным программированием в целом. Я изучила основные, фундаментальные понятия, как наследование, полиморфизм, инкапсуляция и т.д. В данной работе я научилась использовать классы, виртуальные функции. Класс это основной элемент, в рамках которого осуществляется конструирование программ. Класс содержит в себе данные и код, который управляет этими данными. Все эти понятия применимы ко всем объектно-ориентированным языкам программирования, так что данный опыт поможет мне не только в изучении C++, но и других языков.

Лабораторная работа №2

1. **Цель работы**

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

2. **Задание**

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру (колонка фигура 1), согласно вариантов задания (реализованную в ЛР1).
Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Классы фигур должны иметь переопределенный оператор вывода в поток std::ostream (<<).
- Оператор должен распечатывать параметры фигуры (тип фигуры, длины сторон, радиус и т.д).
- Классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream (>>).
- Оператор должен вводить основные параметры фигуры (длины сторон, радиус и т.д).
- Классы фигур должны иметь операторы копирования (=).
- Классы фигур должны иметь операторы сравнения с такими же фигурами (==).
- Класс-контейнер должен соджержать 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 (опередеяется структурой контейнера).

3. **Описание**

При использовании многих структур данных достаточно часто бывает, что они должны иметь переменный размер во время выполнения программы. В этих случаях необходимо применять динамическое выделение памяти. Одной из самых распространенных таких структур данных являются массивы, в которых изначально размер не определен и не зафиксирован.

Динамический массив – это массив, размер которого заранее не фиксирован и может меняться во время исполнения программы. Для изменения размера динамического массива язык программирования C++, поддерживающий такие массивы, предоставляет специальные встроенные функции или операции. Динамические массивы дают возможность более гибкой работы с данными, так как позволяют не прогнозировать хранимые объемы данных, а регулировать размер массива в соответствии с реально необходимыми объемами.

Дружественная функция — это функция, которая не являясь частью класса имеет доступ ко всем элементам из дружественного себе класса.

Ключевое слово `operator` объявляет функцию, которая указывает, что означает `operator-symbol` при применении к экземпляру класса. Это дает оператору более одного значения — "перегружает" его. Компилятор различает разные значения оператора, проверяя типы его операндов.

- с помощью перегрузки невозможно создавать новые символы для операций;
- перегрузка операторов не изменяет порядок выполнения операций и их приоритет;
- унарный оператор не может использоваться для переопределения бинарной операции так же, как и бинарный оператор не переопределит унарную операцию.

4. Код программы

Классы:

```
class Array {
public:
	Array(int);
	Array();
	~Array();

	Trapeze& operator[](int);
	void push(Trapeze &kv);
	void del(int i);
	int isize();

	friend ostream& operator<<(ostream& os, Array &re);
	bool check();
	void resize();

private:
	Trapeze *arr;
	int size;
	int real_size;
};

class Trapeze {
public:
	Trapeze();
	Trapeze(size_t i, size_t j, size_t k);
	~Trapeze();

	double Square();
	void print();
	bool prov();
	Trapeze& operator = (Trapeze &add);

	friend bool operator == (Trapeze &k1, Trapeze &k2);
	friend ostream& operator << (ostream& os, Trapeze &pt);
	friend istream& operator >> (istream& os, Trapeze &pr);

private:
	size_t a;
	size_t b;
	size_t h;
};
```

Array.cpp

```
#include "Array.h"

Array::Array(int a)
{
	real_size = a;
	size = 0;
	arr = new Trapeze[a];
}

Array::Array()
{
	real_size = 10;
	size = 0;
	arr = new Trapeze[10];
}

Array::~Array()
{
	cout << "Array delete." << endl;
	delete[] arr;
}

Trapeze& Array::operator[](int i)
{
	return arr[i];
}

void Array::push(Trapeze & kv)
{
	if (check()) {
		resize();
	}
	arr[size] = kv;
	size++;
}

void Array::del(int i)
{
	if (i >= size || i < 0) {
		cout << "Error: element number " << i << " does not exist in array." <<
endl;
	}
	else {
		Trapeze *ne;
		ne = new Trapeze[size + (real_size - size) / 2];
		int k = 0;
		for (int j = 0; j < size; j++) {
			if (j != i) {
				ne[k] = arr[j];
				k++;
			}
		}
		real_size = size + (real_size - size) / 2;
		size--;
		delete[] arr;
		arr = ne;
	}
}

int Array::isize()
{
	return size;
```

```
}

bool Array::check()
{
	if (size == real_size - 1) return 1;
	else return 0;
}

void Array::resize()
{
	Trapeze *ne;
	real_size *= 2;
	ne = new Trapeze[real_size];
	for (int i = 0; i < size; i++) {
		ne[i] = arr[i];
	}
	delete[] arr;
	arr = ne;
}

ostream & operator<<(ostream & os, Array &re)
{
	for (int i = 0; i < re.isize(); i++) {
		if (re[i].prov()) {
			os << "Figure number " << i << endl << "    " << re[i];
		}
	}
	return os;
}
```

5. Консоль

```
==========================
Menu:
1) Add trapeze
2) Delete trapeze
3) Print
4) Exit
==========================
Choose action:
1
==========================
You chose 1) Add trapeze
Enter side a =
8
Enter side b =
7
Enter height =
5
==========================
Menu:
1) Add trapeze
2) Delete trapeze
3) Print
4) Exit
==========================
Choose action:
1
==========================
You chose 1) Add trapeze
Enter side a =
8
Enter side b =
7
```

```
==========================
Menu:
1) Add trapeze
2) Delete trapeze
3) Print
4) Exit
==========================
Choose action:
3
==========================
Figure number 0
    Side a = 8
Side b = 7
Height = 5
Figure number 1
    Side a = 8
Side b = 7
Height = 5
Figure number 2
    Side a = 9
Side b = 5
Height = 7

==========================
Menu:
1) Add trapeze
2) Delete trapeze
3) Print
4) Exit
==========================
Choose action:
2
```

```
Enter height =
5
==========================
Menu:
1) Add trapeze
2) Delete trapeze
3) Print
4) Exit
==========================
Choose action:
1
==========================
You chose 1) Add trapeze
Enter side a =
9
Enter side b =
5
Enter height =
7
```

```
==========================
You chose 2) Delete
Enter trapeze number = 2
==========================
Menu:
1) Add trapeze
2) Delete trapeze
3) Print
4) Exit
==========================
Choose action:
3
==========================
Figure number 0
    Side a = 8
Side b = 7
Height = 5
Figure number 1
    Side a = 8
Side b = 7
Height = 5

==========================
Menu:
1) Add trapeze
2) Delete trapeze
3) Print
4) Exit
==========================
Choose action:
```

6. Вывод

В этой работе я научилась организовывать динамическую структуру – массив – в языке C++. Для этого также можно использовать средства языка – стандартную библиотеку с шаблонами. Динамические массивы очень удобны для работы, так как их размеры определяются на момент компиляции программы.

Лабораторная работа №3

1. **Цель работы**
   - Закрепление навыков работы с классами.
   - Знакомство с умными указателями.
2. **Задание**

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий все три фигуры класса фигуры, согласно варианту задания (реализованную в ЛР1).
Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Класс-контейнер должен соджержать 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 (опередеяется
- структурой контейнера).
- Класс-контейнер должен иметь метод по удалению фигуры из контейнера (опередеяется
- структурой контейнера).
- Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток
- std::ostream (<<).
- Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.
- 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.

Фигуры: Ромб, Параллелограмм, Пятиугольник

Контейнер: Массив

3. **Описание**

Smart pointer — это объект, работать с которым можно как с обычным указателем, но при этом, в отличии от последнего, он предоставляет некоторый дополнительный функционал (например, автоматическое освобождение закрепленной за указателем области памяти).

Умные указатели призваны для борьбы с утечками памяти, которые сложно избежать в больших проектах. Они особенно удобны в местах, где возникают исключения, так как при последних происходит процесс раскрутки стека и уничтожаются локальные объекты. В случае обычного указателя — уничтожится переменная-указатель, при этом ресурс останется не освобожденным. В случае умного указателя — вызовется деструктор, который и освободит выделенный ресурс.

В новом стандарте появились следующие умные указатели: unique_ptr, shared_ptr и weak_ptr. Все они объявлены в заголовочном файле <memory>. `shared_ptr` реализует подсчет ссылок на ресурс. Ресурс освободится тогда, когда счетчик ссылок на него будет равен 0.

4. Код программы

ArrayItem.h

```
#include "ArrayItem.h"

ArrayItem::ArrayItem() : pentagon(nullptr), rhombus(nullptr), trapeze(nullptr) {}
ArrayItem::ArrayItem(std::shared_ptr<Pentagon> &pentagon) : pentagon(pentagon),
rhombus(nullptr), trapeze(nullptr) {}
ArrayItem::ArrayItem(std::shared_ptr<Rhombus> &rhombus) : pentagon(nullptr),
rhombus(rhombus), trapeze(nullptr) {}
ArrayItem::ArrayItem(std::shared_ptr<Trapeze> &trapeze) : pentagon(nullptr),
rhombus(nullptr), trapeze(trapeze) {}
ArrayItem::~ArrayItem() {}

bool ArrayItem::IsPentagon()
{
	if (pentagon != nullptr) return true;
	else return false;
}

bool ArrayItem::IsRhombus()
{
	if (rhombus != nullptr) return true;
	else return false;
}

bool ArrayItem::IsTrapeze()
{
	if (trapeze != nullptr) return true;
	else return false;
}

std::shared_ptr<Pentagon> ArrayItem::GetPentagon()
{
	return this->pentagon;

}

std::shared_ptr<Rhombus> ArrayItem::GetRhombus()
{
	return this->rhombus;
}

std::shared_ptr<Trapeze> ArrayItem::GetTrapeze()
{
	return this->trapeze;
```

```
}

std::ostream& operator << (std::ostream &os, ArrayItem &item)
{
	if (item.IsPentagon())
		os << *item.pentagon << " (I am pentagon)";
	else if (item.IsRhombus())
		os << *item.rhombus << " (I am rhombus)";
	else if (item.IsTrapeze())
		os << *item.trapeze << " (I am trapeze)";
	else
		os << "empty";
	return os;
}
```

ArrayItem.h

```
#ifndef ARRAYITEM_H
#define ARRAYITEM_H

#include "Pentagon.h"
#include "Rhombus.h"
#include "Trapeze.h"
#include <memory>

class ArrayItem
{
public:
	ArrayItem();
	ArrayItem(std::shared_ptr<Pentagon> &pentagon);
	ArrayItem(std::shared_ptr<Rhombus> &rhombus);
	ArrayItem(std::shared_ptr<Trapeze> &trapeze);

	bool IsPentagon();
	bool IsRhombus();
	bool IsTrapeze();

	std::shared_ptr<Pentagon> GetPentagon();
	std::shared_ptr<Rhombus> GetRhombus();
	std::shared_ptr<Trapeze> GetTrapeze();

	friend std::ostream& operator << (std::ostream &os, ArrayItem &item);
	virtual ~ArrayItem();

private:
	std::shared_ptr<Pentagon> pentagon;
	std::shared_ptr<Rhombus> rhombus;
	std::shared_ptr<Trapeze> trapeze;
};
#endif
```

FArray.cpp

```
#include "FArray.h"

FArray::FArray()
{
	for (int i = 0; i < size; i++)
	{
		a[i] = ArrayItem();
	}
}

void FArray::Insert(std::shared_ptr<Pentagon> &pentagon, int index)
```

```
{
	a[index] = ArrayItem(pentagon);
}

void FArray::Insert(std::shared_ptr<Rhombus> &rhombus, int index)
{
	a[index] = ArrayItem(rhombus);
}

void FArray::Insert(std::shared_ptr<Trapeze> &trapeze, int index)
{
	a[index] = ArrayItem(trapeze);
}

bool FArray::IsPentagon(int index)
{
	return a[index].IsPentagon();
}

bool FArray::IsRhombus(int index)
{
	return a[index].IsRhombus();
}

bool FArray::IsTrapeze(int index)
{
	return a[index].IsRhombus();
}

std::shared_ptr<Pentagon> FArray::GetPentagon(int index)
{
	return a[index].GetPentagon();
}

std::shared_ptr<Rhombus> FArray::GetRhombus(int index)
{
	return a[index].GetRhombus();
}

std::shared_ptr<Trapeze> FArray::GetTrapeze(int index)
{
	return a[index].GetTrapeze();
}

void FArray::Delete(int index)
{
	a[index] = ArrayItem();
}

std::ostream& operator << (std::ostream &os, FArray &array)
{
	for (int i = 0; i < size; i++)
	{
		os << "[" << i << "] " << array.a[i] << std::endl;
	}
	return os;
}

FArray::~FArray()
{
	for (int i = 0; i < size; i++)
	{
		a[i] = ArrayItem();
	}
}
```

FArray.h

```
#ifndef FArray_H
#define FArray_H

#include "Trapeze.h"
#include "Pentagon.h"
#include "Rhombus.h"
#include "ArrayItem.h"
#include <memory>

const int size = 10;

class FArray
{
public:
	FArray();
	void Insert(std::shared_ptr<Pentagon> &pentagon, int index);
	void Insert(std::shared_ptr<Rhombus> &rhombus, int index);
	void Insert(std::shared_ptr<Trapeze> &trapeze, int index);

	bool IsPentagon(int index);
	bool IsRhombus(int index);
	bool IsTrapeze(int index);

	std::shared_ptr<Pentagon> GetPentagon(int index);
	std::shared_ptr<Rhombus> GetRhombus(int index);
	std::shared_ptr<Trapeze> GetTrapeze(int index);

	void Delete(int index);
	friend std::ostream& operator << (std::ostream &os, FArray &array);
	virtual ~FArray();

private:
	ArrayItem a[size];
};
#endif
```

main.cpp

```
#include <iostream>
#include <cstdlib>
#include "Trapeze.h"
#include "FArray.h"


int main()
{
	int x = 6;
	int i, num;
	FArray figure_array;

	while (true)
	{
		std::cout << "=======================" << std::endl;
		std::cout << "Menu:" << std::endl;
		std::cout << "1) Add figure" << std::endl;
		std::cout << "2) Print figure" << std::endl;
		std::cout << "3) Delete figure" << std::endl;
		std::cout << "4) Print array" << std::endl;
		std::cout << "5) Exit" << std::endl;
		std::cout << "=======================" << std::endl;

		std::cout << "Choose action: ";
```

```
		std::cin >> num;

		if (num > 5)
		{
			std::cout << "Error, put another number " << std::endl;
			continue;
		}

		if (num == 5)
			break;

		switch (num)
		{

		case 1:
		{
			std::cout << "=======================" << std::endl;
			std::cout << "You chose 1) Add figure" << std::endl;
			char figure_name;
			std::cout << "Enter figure name ([p]-pentagon, [r]-rhombus, [t]-
trapeze): ";
			std::cin >> figure_name;
			std::cout << "Enter index: ";
			std::cin >> i;
			if (figure_name == 'p')
				figure_array.Insert(std::shared_ptr<Pentagon>(new
Pentagon(std::cin)), i);
			else if (figure_name == 'r')
				figure_array.Insert(std::shared_ptr<Rhombus>(new
Rhombus(std::cin)), i);
			else if (figure_name == 't')
				figure_array.Insert(std::shared_ptr<Trapeze>(new
Trapeze(std::cin)), i);

			break;
		}

		case 2:
		{
			std::cout << "=======================" << std::endl;
			std::cout << "You chose 2) Print figure" << std::endl;
			std::cout << "Enter index: ";
			std::cin >> i;

			if (figure_array.IsPentagon(i))
				std::cout << *figure_array.GetPentagon(i) << std::endl;
			else if (figure_array.IsRhombus(i))
				std::cout << *figure_array.GetRhombus(i) << std::endl;
			else if (figure_array.IsTrapeze(i))
				std::cout << *figure_array.GetTrapeze(i) << std::endl;
			else
				std::cout << "Empty element" << std::endl;

			break;
		}

		case 3:
		{
			std::cout << "=======================" << std::endl;
			std::cout << "You chose 3) Delete figure" << std::endl;
			std::cout << "enter index: ";

			std::cin >> i;
			figure_array.Delete(i);
```

```
                break;
        }

        case 4:
        {
                std::cout << "=======================" << std::endl;
                std::cout << "You chose 4) Print array" << std::endl;
                std::cout << "Figure array:\n" << figure_array;

                break;
        }
        }
    }

    return 0;
}
```

5. Консоль

```
=======================
Menu:
1) Add figure
2) Print figure
3) Delete figure
4) Print array
5) Exit
=======================
Choose action: 1
=======================
You chose 1) Add figure
Enter figure name ([p]-pentagon, [r]-
rhombus, [t]-trapeze): p
Enter index: 0
Enter side a: 6
Enter side b: 4
Enter height 1: 5
Enter height 2: 3
Pentagon created
=======================
Menu:
1) Add figure
2) Print figure
3) Delete figure
4) Print array
5) Exit
=======================
Choose action: 1
=======================
You chose 1) Add figure
Enter figure name ([p]-pentagon, [r]-
rhombus, [t]-trapeze): r
Enter index: 1
Enter diagonal 1: 2
Enter diagonal 2: 3
Rhombus created
=======================
Menu:
1) Add figure
2) Print figure
3) Delete figure
4) Print array
5) Exit
=======================
Choose action: 1
```

```
=======================
Menu:
1) Add figure
2) Print figure
3) Delete figure
4) Print array
5) Exit
=======================
Choose action: 4
=======================
You chose 4) Print array
Figure array:
[0] Side = 6
Diagonal = 4
Height 1 = 5
Height 2 = 3 (I am pentagon)
[1] Diagonal 1 = 2
Diagonal 2 = 3 (I am rhombus)
[2] Side a = 4
Side b = 5
Height = 6 (I am trapeze)
[3] Empty
[4] Empty
[5] Empty
[6] Empty
[7] Empty
[8] Empty
[9] Empty
=======================
Menu:
1) Add figure
2) Print figure
3) Delete figure
4) Print array
5) Exit
=======================
Choose action: 3
=======================
You chose 3) Delete figure
enter index: 1
Rhombus deleted
=======================
Menu:
1) Add figure
2) Print figure
```

```
=======================
You chose 1) Add figure
Enter figure name ([p]-pentagon, [r]-
rhombus, [t]-trapeze): t
Enter index: 2
Enter side a: 4
Enter side b: 5
Enter height: 6
Trapeze created
```

```
3) Delete figure
4) Print array
5) Exit
=======================
Choose action: 4
=======================
You chose 4) Print array
Figure array:
[0] Side = 6
Diagonal = 4
Height 1 = 5
Height 2 = 3 (I am pentagon)
[1] Empty
[2] Side a = 4
Side b = 5
Height = 6 (I am trapeze)
[3] Empty
[4] Empty
[5] Empty
[6] Empty
[7] Empty
[8] Empty
[9] Empty
=======================
Menu:
1) Add figure
2) Print figure
3) Delete figure
4) Print array
5) Exit
=======================
Choose action:
```

6. Вывод

Умные указатели — очень удобная и полезная вещь. Стоит отметить, что умные указатели (кроме `unique_ptr`) не предназначены для владения массивами. Это связано с тем, что деструктор вызывает именно `delete`, а не `delete[]` (что требуется для массивов). Для `unique_ptr` мы имеем дело с предопределенной специализацией для массивов. Для ее использования необходимо указать `[]` возле параметра шаблона.

Лабораторная работа № 4

1. **Цель работы**

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

2. **Задание**

Необходимо спроектировать и запрограммировать на языке C++ шаблон класса-контейнера первого уровня, содержащий все три фигуры класса фигуры, согласно вариантов задания (реализованную в ЛР1).
Классы должны удовлетворять следующим правилам:

- Требования к классам фигуры аналогичны требованиям из лабораторной работы 1.
- Шаблон класса-контейнера должен соджержать объекты используя std:shared_ptr<…>.
- Шаблон класса-контейнера должен иметь метод по добавлению фигуры в контейнер.
- Шаблон класса-контейнера должен иметь методы по получению фигуры из контейнера
- (опередеяется структурой контейнера).
- Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера
- (опередеяется структурой контейнера).
- Шаблон класса-контейнера должен иметь перегруженный оператор по выводу контейнера в поток
- std::ostream (<<).
- Шаблон класса-контейнера должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположенны 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 контейнера.
- Удалять фигуры из контейнера.

3. **Описание**

Любой шаблон языка c++ начинается со слова `template`, будь то шаблон функции или шаблон класса. После ключевого слова `template` идут угловые скобки — `< >`, в которых перечисляется список параметров шаблона. Каждому параметру должно предшествовать зарезервированное слово `class` или `typename`.

Ключевое слово `typename` говорит о том, что в шаблоне будет использоваться встроенный тип данных, такой как: `int`, `double`,`float`, `char` и т. д. А ключевое

слово `class` сообщает компилятору, что в шаблоне функции в качестве параметра будут использоваться пользовательские типы данных, то есть классы.

4. Код программы

FigureArray.h

```
#ifndef FIGUREARRAY_H
#define FIGUREARRAY_H

#include "Trapeze.h"
#include "Pentagon.h"
#include "Rhomb.h"
#include "ArrayItem.cpp"
#include <memory>

template <class T1, class T2, class T3>
class FigureArray
{
public:
	FigureArray(int size);

	void Insert(std::shared_ptr<T1> &pentagon, int index);
	void Insert(std::shared_ptr<T2> &rhomb, int index);
	void Insert(std::shared_ptr<T3> &trapeze, int index);

	bool IsPentagon(int index);
	bool IsRhomb(int index);
	bool IsTrapeze(int index);

	std::shared_ptr<T1> GetPentagon(int index);
	std::shared_ptr<T2> GetRhomb(int index);
	std::shared_ptr<T3> GetTrapeze(int index);

	void Delete(int index);

	template <class T1, class T2, class T3>
	friend std::ostream& operator << (std::ostream &os, FigureArray<T1, T2, T3>
&array);

	virtual ~FigureArray();

private:
	ArrayItem<T1, T2, T3> *data;
	int size;
};

#endif
```

FigureArray.cpp

```
#include "FigureArray.h"

template <class T1, class T2, class T3>
FigureArray<T1, T2, T3>::FigureArray(int size)
{
	data = new ArrayItem<T1, T2, T3>[size];
	FigureArray<T1, T2, T3>::size = size;
}

template <class T1, class T2, class T3>
void FigureArray<T1, T2, T3>::Insert(std::shared_ptr<T1> &pentagon, int index)
```

```
{
	data[index] = ArrayItem<T1, T2, T3>(pentagon);
}

template <class T1, class T2, class T3>
void FigureArray<T1, T2, T3>::Insert(std::shared_ptr<T2> &rhomb, int index)
{
	data[index] = ArrayItem<T1, T2, T3>(rhomb);
}

template <class T1, class T2, class T3>
void FigureArray<T1, T2, T3>::Insert(std::shared_ptr<T3> &trapeze, int index)
{
	data[index] = ArrayItem<T1, T2, T3>(trapeze);
}

template <class T1, class T2, class T3>
bool FigureArray<T1, T2, T3>::IsPentagon(int index)
{
	return data[index].IsPentagon();
}

template <class T1, class T2, class T3>
bool FigureArray<T1, T2, T3>::IsRhomb(int index)
{
	return data[index].IsRhomb();
}

template <class T1, class T2, class T3>
bool FigureArray<T1, T2, T3>::IsTrapeze(int index)
{
	return data[index].IsTrapeze();
}

template <class T1, class T2, class T3>
std::shared_ptr<T1> FigureArray<T1, T2, T3>::GetPentagon(int index)
{
	return data[index].GetPentagon();
}

template <class T1, class T2, class T3>
std::shared_ptr<T2> FigureArray<T1, T2, T3>::GetRhomb(int index)
{
	return data[index].GetRhomb();
}

template <class T1, class T2, class T3>
std::shared_ptr<T3> FigureArray<T1, T2, T3>::GetTrapeze(int index)
{
	return data[index].GetTrapeze();
}

template <class T1, class T2, class T3>
void FigureArray<T1, T2, T3>::Delete(int index)
{
	data[index] = ArrayItem<T1, T2, T3>();
}

template <class T1, class T2, class T3>
std::ostream& operator << (std::ostream &os, FigureArray<T1, T2, T3> &array)
{
	for (int i = 0; i < array.size; i++)
	{
		os << "[" << i << "] " << array.data[i] << std::endl;
	}
```

```
        return os;
}

template <class T1, class T2, class T3>
FigureArray<T1, T2, T3>::~FigureArray() {}
```

5. Консоль

```
=======================
Menu:
1) Add figure
2) Print figure
3) Delete figure
4) Print array
5) Exit
=======================
Choose action: 4
=======================
You chose 4) Print array
Figure array:
[0] Side = 6
Diagonal = 4
Height 1 = 5
Height 2 = 3 (I am pentagon)
[1] Diagonal 1 = 2
Diagonal 2 = 3 (I am rhombus)
[2] Side a = 4
Side b = 5
Height = 6 (I am trapeze)
[3] Empty
[4] Empty
[5] Empty
[6] Empty
[7] Empty
[8] Empty
[9] Empty
=======================
Menu:
1) Add figure
2) Print figure
3) Delete figure
4) Print array
5) Exit
=======================
Choose action: 3
=======================
You chose 3) Delete figure
enter index: 1
Rhombus deleted
```

```
=======================
Menu:
1) Add figure
2) Print figure
3) Delete figure
4) Print array
5) Exit
=======================
Choose action: 4
=======================
You chose 4) Print array
Figure array:
[0] Side = 6
Diagonal = 4
Height 1 = 5
Height 2 = 3 (I am pentagon)
[1] Empty
[2] Side a = 4
Side b = 5
Height = 6 (I am trapeze)
[3] Empty
[4] Empty
[5] Empty
[6] Empty
[7] Empty
[8] Empty
[9] Empty
=======================
Menu:
1) Add figure
2) Print figure
3) Delete figure
4) Print array
5) Exit
=======================
Choose action:
```

6. Вывод

Шаблоны оказываются очень удобными, когда нужно сократить количество одинакового кода. Они также являются безопасными с точки зрения типов. Шаблоны используются, когда код является аналогичным для различных данных. Также шаблоны класса делают возможным обернуть любой тип данных вокруг специфичной реализации.

Лабораторная работа № 5

1. **Цель работы**

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

2. **Задание**

Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛР№4) спроектировать и разработать Итератор для динамической 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 контейнеры std.

Программа должна позволять:

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

3. **Описание**

Итератор — это такая структура данных, которая используется для обращения к определенному элементу в контейнерах STL. Обычно из используют с контейнерами `set`, `list` , а у вектора для этого применяют индексы. Для их использования можно подключить библиотеку <iterator>, но мы будем писать его сами.

Для динамического массива понадобится выполнять **операцию разыменования** (обращаться к значению элемента на которое указывает итератор), сравнивать на равенства, использовать инкремент (++) и сравнение на неравенство.

4. **Код программы**

Iterator.h

```
#ifndef ITERATOR_H
#define ITERATOR_H

#include <memory>
#include <iostream>
#include "ArrayItem.cpp"

template<class T1, class T2, class T3>
class Iterator
{
public:

	Iterator();
	Iterator(ArrayItem<T1, T2, T3>* p);

	ArrayItem<T1, T2, T3>* operator *();
```

```
        bool operator == (Iterator i);
        bool operator != (Iterator i);

        void operator ++ ();

private:
        ArrayItem<T1, T2, T3>* element;
};

#endif
```

Iterator.cpp

```
#include "Iterator.h"

template<class T1, class T2, class T3>
Iterator<T1, T2, T3>::Iterator()
{
        element = nullptr;
}

template<class T1, class T2, class T3>
Iterator<T1, T2, T3>::Iterator(ArrayItem<T1, T2, T3>* p)
{
        element = p;
}

template<class T1, class T2, class T3>
ArrayItem<T1, T2, T3>* Iterator<T1, T2, T3>::operator * ()
{
        return element;
}

template<class T1, class T2, class T3>
bool Iterator<T1, T2, T3>::operator ==(Iterator i)
{
        return element == i.element;
}

template<class T1, class T2, class T3>
bool Iterator<T1, T2, T3>::operator !=(Iterator i)
{
        return !(*this == i);
}

template<class T1, class T2, class T3>
void Iterator<T1, T2, T3>::operator ++ ()
{
        element++;
}
```

5. Консоль

```
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command: 1
============================
Enter figure name (p - pentagon, r -
rhombus, t - trapeze): t
Enter index: 0
Enter base 1: 543
Enter base 2: 234
Enter height: 222
Trapeze created
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command: 1
============================
Enter figure name (p - pentagon, r -
rhombus, t - trapeze): r
Enter index: 1
Enter diagonal 1: 63
Enter diagonal 2: 54
Rhombus created
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command: 1
============================
Enter figure name (p - pentagon, r -
rhombus, t - trapeze): p
Enter index: 2
Enter side a: 45
Enter side b: 75
Enter height 1: 4
Enter height 2: 3
Pentagon created
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command: 2
============================
Enter index: 0
```

```
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command: 4
============================
base 1 = 543
base 2 = 234
height = 222 (I am trapeze)
diagonal 1 = 63
diagonal 2 = 54 (I am rhombus)
side = 45
diagonal = 75
height 1 = 4
height 2 = 3 (I am pentagon)
Empty
Empty
Empty
Empty
Empty
Empty
Empty
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command: 3
============================
Enter index: 1
Rhombus deleted
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command: 4
============================
base 1 = 543
base 2 = 234
height = 222 (I am trapeze)
Empty
side = 45
diagonal = 75
height 1 = 4
height 2 = 3 (I am pentagon)
Empty
Empty
Empty
Empty
Empty
```

```
base 1 = 543
base 2 = 234
height = 222
```

```
Empty
Empty
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command:
```

6. Вывод

Итератор - это объект, который позволяет перемещаться (итерироваться) по элементам некоторой последовательности. Итератор это по сути расширенный указатель и с его помощью очень удобно перебирать элементы. Он удобен для всех видов контейнеров, ведь тогда не придется ссылаться на весь контейнер, а на его часть.

Лабораторная работа № 6

1. **Цель работы**

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

2. **Задание**

Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛР№5) 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.
Аллокатор 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:

- Стандартные контейнеры std.

Программа должна позволять:

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

3. **Описание**

Работа с динамической памятью зачастую является узким местом во многих алгоритмах, если не применять специальные ухищрения. Для этого у каждого контейнера есть свой аллокатор, который отвечает за выделение памяти. Умный аллокатор может ускорить работу с памятью. Можно забрать у ОС большой блок памяти и разбить его на равные блоки размера malloc(Node), при выделении памяти брать блок из пула, при освобождении — возвращать в пул.

4. **Код программы**

AllocationBlock.h

```
#ifndef ALLOCATIONBLOCK_H
#define ALLOCATIONBLOCK_H
class AllocationBlock
{
public:
	AllocationBlock();
	AllocationBlock(size_t s, size_t c);
	void *allocate();
	void deallocate(void *pointer);
	bool HasFreeBlocks();
	virtual ~AllocationBlock();
private:
	size_t size;
	size_t count;
```

```
	char* used_blocks;
	void** free_blocks;
	size_t free_count;
};
#endif
```

AllocationBlock.cpp

```
#include "AllocationBlock.h"
#include <iostream>
AllocationBlock::AllocationBlock() : size(0), count(0), used_blocks(0), free_blocks(0),
free_count(0) {}

AllocationBlock::AllocationBlock(size_t s, size_t c) : size(s), count(c)
{
	used_blocks = (char*)malloc(size*count); // выделили память под указанное число
блоков размера size
	free_blocks = (void**)malloc(sizeof(void*)*count); // создали указатели
	for (int i = 0; i < count; i++)
		free_blocks[i] = used_blocks + i*size; // расставили указатели
	free_count = count; // запомнили номер последнего свободного блока
	std::cout << "TAllocationBlock: Memory init" << std::endl;
}
void *AllocationBlock::allocate()
{
	void *result = nullptr;
	if (free_count > 0)
	{
		result = free_blocks[free_count - 1];
		free_count--;
		std::cout << "AllocationBlock: Allocate " << (count - free_count) << " of "
<< count << std::endl;
	}
	else
	{
		std::cout << "TAllocationBlock: No memory exception :-)" << std::endl;
	}
	return result;
}
void AllocationBlock::deallocate(void *pointer)
{
	std::cout << "TAllocationBlock: Deallocate block " << std::endl;
	free_blocks[free_count] = pointer;
	free_count++;
}
bool AllocationBlock::HasFreeBlocks()
{
	return free_count>0;
}
AllocationBlock::~AllocationBlock()
{
	if (free_count<count)
		std::cout << "TAllocationBlock: Memory leak?" << std::endl;
	else
		std::cout << "TAllocationBlock: Memory freed" << std::endl;
	delete free_blocks;
	delete used_blocks;
}
```

5. Консоль

```
TAllocationBlock: Memory init
============================
Menu:
============================
Enter command: 2
============================
```

```
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command: 1
============================
Enter figure name (p - pentagon, r -
rhombus, t - trapeze): p
Enter index: 0
Enter side a: 3
Enter side b: 4
Enter height 1: 5
Enter height 2: 2
Pentagon created
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command: 1
============================
Enter figure name (p - pentagon, r -
rhombus, t - trapeze): r
Enter index: 4
Enter diagonal 1: 3
Enter diagonal 2: 2
Rhombus created
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command: 1
============================
Enter figure name (p - pentagon, r -
rhombus, t - trapeze): t
Enter index: 2
Enter base 1: 5
Enter base 2: 4
Enter height: 5
Trapeze created
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 – Exit
```

```
Enter index: 0
side = 3
diagonal = 4
height 1 = 5
height 2 = 2
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command: 4
============================
side = 3
diagonal = 4
height 1 = 5
height 2 = 2 (I am pentagon)
Empty
base 1 = 5
base 2 = 4
height = 5 (I am trapeze)
Empty
diagonal 1 = 3
diagonal 2 = 2 (I am rhombus)
Empty
Empty
Empty
Empty
Empty
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command: 3
============================
Enter index: 0
Pentagon deleted
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print array,
0 - Exit
============================
Enter command: 2
============================
Enter index: 0
Empty element
```

6. Вывод

Экономия времени и памяти важна нам, как программистам, всегда. Поэтому правильное использование аллокаторов, отслеживание использования malloc, new и free очень важно при написании программы, так как в более масштабных проектах потеря памяти и использованное время ощущается намного сильнее, чем в нашей лабораторной.

Лабораторная работа № 7

1. **Цель работы**

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

2. **Задание**

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):
1. Массив
2. Связанный список
3. Бинарное- Дерево.
4. N-Дерево (с ограничением не больше 4 элементов на одном уровне).
5. Очередь
6. Стек

Каждым элементом контейнера, в свою, является динамической структурой данных одного из следующих видов (Контейнер 2-го уровня):
1. Массив
2. Связанный список
3. Бинарное- Дерево
4. N-Дерево (с ограничением не больше 4 элементов на одном уровне).
5. Очередь
6. Стек

Таким образом у нас получается контейнер в контейнере. Т.е. для варианта (1,2) это будет массив, каждый из элементов которого – связанный список. А для варианта (5,3) – это очередь из бинарных деревьев.
Элементом второго контейнера является объект-фигура, определенная вариантом задания. 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. Например, для варианта (1,2) добавление объектов будет выглядеть следующим образом:
1. Вначале массив пустой.
2. Добавляем Объект1: В массиве по индексу 0 создается элемент с типом список, в список
добавляется Объект 1.
3. Добавляем Объект2: Объект добавляется в список, находящийся в массиве по индекс 0.
4. Добавляем Объект3: Объект добавляется в список, находящийся в массиве по индекс 0.
5. Добавляем Объект4: Объект добавляется в список, находящийся в массиве по индекс 0.
6. Добавляем Объект5: Объект добавляется в список, находящийся в массиве по индекс 0.
7. Добавляем Объект6: В массиве по индексу 1 создается элемент с типом список, в список добавляется Объект 6.

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:

- Стандартные контейнеры std.

Программа должна позволять:

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

Фигуры: Пятиугольник, Ромб, Параллелограмм

Контейнеры: Н-Дерево, Массив

3. Описание

Контейнер – объект, который может содержать в себе другие объекты. Существует несколько категорий таких контейнеров – последовательные, ассоциативные, контейнеры-адаптеры и псевдоконтейнеры. В этой лабораторной необходимо составить контейнер из двух, вложенных в друг друга.

НДерево это дерево узлов, каждое из которых содержит тип (T в template) и максимум Н-детей. Как и любой другой STL контейнер, ндерево использует итератор как псевдо указатель на узел. Ндерево сохраняется в массиве детей-указателей на каждый узел.

4. Код программы

NTree.h

```
#ifndef NTREE_H
#define NTREE_H

#include "TreeItem.cpp"

const int max_nodes = 5;

template <class T1, class T2, class T3>
class NTree
{
public:
	NTree();
	TreeItem<T1, T2, T3>* Insert(std::shared_ptr<T1> &pentagon);
	TreeItem<T1, T2, T3>* Insert(std::shared_ptr<T2> &rhomb);
	TreeItem<T1, T2, T3>* Insert(std::shared_ptr<T3> &trapeze);

	bool IsFull();
	bool IsEmpty();

	TreeItem<T1, T2, T3>* GetRoot();

	TreeItem<T1, T2, T3>* FindMin(TreeItem<T1, T2, T3>* p);
	void Display(TreeItem<T1, T2, T3>* root);

	void Sort(TreeItem<T1, T2, T3>* root);
```

```
	void QuickSort(TreeItem<T1, T2, T3>* array[], int L, int R);

	TreeItem<T1, T2, T3>* DeleteBySquare(TreeItem<T1, T2, T3>* root, double square);

	TreeItem<T1, T2, T3>* DeletePentagon(TreeItem<T1, T2, T3>* root);
	TreeItem<T1, T2, T3>* DeleteRhomb(TreeItem<T1, T2, T3>* root);
	TreeItem<T1, T2, T3>* DeleteTrapeze(TreeItem<T1, T2, T3>* root);

	TreeItem<T1, T2, T3>* DeleteRecPentagon(TreeItem<T1, T2, T3>* root);
	TreeItem<T1, T2, T3>* DeleteRecRhomb(TreeItem<T1, T2, T3>* root);
	TreeItem<T1, T2, T3>* DeleteRecTrapeze(TreeItem<T1, T2, T3>* root);

	TreeItem<T1, T2, T3>* root;
	int size;
private:

};

#endif
```

NTreeArray.h

```
#ifndef NTREEARRAY_H
#define NTREEARRAY_H
#include "NTree.cpp"

template <class T1, class T2, class T3>
class NTreeArray
{
public:
	NTreeArray(int size);

	void Insert(std::shared_ptr<T1> &pentagon);
	void Insert(std::shared_ptr<T2> &rhomb);
	void Insert(std::shared_ptr<T3> &trapeze);

	void DeleteByType(char type);
	void DeleteBySquare(double square);

	void Display();

private:
	NTree<T1, T2, T3>* data;
	int size;
};

#endif
```

TreeItem.cpp

```
#ifndef TREEITEM_H
#define TREEITEM_H

#include "Pentagon.h"
#include "Rhomb.h"
#include "Trapeze.h"
#include <memory>

const int max_children_size = 4;
template <class T1, class T2, class T3>
class TreeItem
{
public:
	TreeItem();
```

```
        TreeItem(std::shared_ptr<T1> &pentagon);
        TreeItem(std::shared_ptr<T2> &rhomb);
        TreeItem(std::shared_ptr<T3> &trapeze);

        bool IsPentagon();
        bool IsRhomb();
        bool IsTrapeze();
        bool IsVisited();
        bool IsLeaf();

        std::shared_ptr<T1> GetPentagon();
        std::shared_ptr<T2> GetRhomb();
        std::shared_ptr<T3> GetTrapeze();
        double GetSquare();

        TreeItem* GetChild(int index);
        void SetChild(std::shared_ptr<T1> &pentagon, int index);
        void SetChild(std::shared_ptr<T2> &rhomb, int index);
        void SetChild(std::shared_ptr<T3> &trapeze, int index);

        bool operator < (TreeItem &rhs);
        bool operator > (TreeItem &rhs);

        bool visited = 0;
        TreeItem<T1, T2, T3>* children[max_children_size];
        template <class T1, class T2, class T3> friend std::ostream& operator <<
(std::ostream &os, TreeItem<T1, T2, T3> item);
        double square;
        std::shared_ptr<T1> pentagon;
        std::shared_ptr<T2> rhomb;
        std::shared_ptr<T3> trapeze;

private:

};
```

NTreeArray.cpp

```
#include "NTreeArray.h"

template <class T1, class T2, class T3> NTreeArray<T1, T2, T3>::NTreeArray(int size)
{
        data = new NTree<T1, T2, T3>[size];
        NTreeArray<T1, T2, T3>::size = size;
}

template <class T1, class T2, class T3> void NTreeArray<T1, T2,
T3>::Insert(std::shared_ptr<T1> &pentagon)
{
        for (int i = 0; i < size; i++)
        {
                if (!data[i].IsFull())
                {
                        data[i].Insert(pentagon);
                        return;
                }
        }
}

template <class T1, class T2, class T3> void NTreeArray<T1, T2,
T3>::Insert(std::shared_ptr<T2> &rhomb)
{
        for (int i = 0; i < size; i++)
        {
```

```
			if (!data[i].IsFull())
			{
				data[i].Insert(rhomb);
				return;
			}
		}
}

template <class T1, class T2, class T3> void NTreeArray<T1, T2,
T3>::Insert(std::shared_ptr<T3> &trapeze)
{
		for (int i = 0; i < size; i++)
		{
			if (!data[i].IsFull())
			{
				data[i].Insert(trapeze);
				return;
			}
		}
}

template <class T1, class T2, class T3> void NTreeArray<T1, T2, T3>::DeleteByType(char
type)
{
		for (int i = 0; i < size; i++)
		{
			if (type == 'p') data[i].root = data[i].DeletePentagon(data[i].root);
			else if (type == 'r') data[i].root = data[i].DeleteRhomb(data[i].root);
			else if (type == 't') data[i].root = data[i].DeleteTrapeze(data[i].root);
		}
}

template <class T1, class T2, class T3> void NTreeArray<T1, T2,
T3>::DeleteBySquare(double square)
{
		for (int i = 0; i < size; i++)
		{
			data[i].root = data[i].DeleteBySquare(data[i].root, square);
		}
}
template <class T1, class T2, class T3> void NTreeArray<T1, T2, T3>::Display()
{
		for (int i = 0; i < size; i++)
		{
			std::cout << "[" << i << "] " << "size = " << data[i].size << std::endl;

			data[i].Display(data[i].GetRoot());

		}
}
```

5. Консоль

```
============================
Menu:
1 - Add figure,
2 - Print all,
3 - Delete,
0 - Exit
============================
Enter command: 1
============================
enter figure name (p - pentagon, r -
rhomb, t - trapeze): r
```

```
Menu:
1 - Add figure,
2 - Print all,
3 - Delete,
0 - Exit
============================
Enter command: 2
============================
[0] size = 5
\__ diagonal 1 = 8
diagonal 2 = 4 Square = 16 (I am rhomb)
```

```
Enter diagonal 1: 8
Enter diagonal 2: 4
Rhomb created
============================
Menu:
1 - Add figure,
2 - Print all,
3 - Delete,
0 - Exit
============================
Enter command: 1
============================
enter figure name (p - pentagon, r -
rhomb, t - trapeze): p
Enter side a: 5
Enter side b: 49
Enter height 1: 6
Enter height 2: 3
Pentagon created
============================
Menu:
1 - Add figure,
2 - Print all,
3 - Delete,
0 - Exit
============================
Enter command: 1
============================
enter figure name (p - pentagon, r -
rhomb, t - trapeze): t
Enter base 1: 43
Enter base 2: 5
Enter height: 4
Trapeze created
============================
Menu:
1 - Add figure,
2 - Print all,
3 - Delete,
0 - Exit
============================
Enter command: 1
============================
enter figure name (p - pentagon, r -
rhomb, t - trapeze): r
Enter diagonal 1: 43
Enter diagonal 2: 95
Rhomb created
============================
Menu:
1 - Add figure,
2 - Print all,
3 - Delete,
0 - Exit
============================
Enter command: 2
============================
[0] size = 4
\__ diagonal 1 = 8
diagonal 2 = 4 Square = 16 (I am rhomb)
   \__ base 1 = 43
base 2 = 5
height = 4 Square = 96 (I am trapeze)
   \__ side = 5
diagonal = 49
   \__ base 1 = 43
base 2 = 5
height = 4 Square = 96 (I am trapeze)
   \__ base 1 = 93
base 2 = 4
height = 2 Square = 97 (I am trapeze)
   \__ side = 5
diagonal = 49
height 1 = 6
height 2 = 3 Square = 235 (I am pentagon)
   \__ diagonal 1 = 43
diagonal 2 = 95 Square = 2042 (I am rhomb)
[1] size = 0
[2] size = 0
[3] size = 0
[4] size = 0
[5] size = 0
[6] size = 0
[7] size = 0
[8] size = 0
[9] size = 0
============================
Menu:
1 - Add figure,
2 - Print all,
3 - Delete,
0 - Exit
============================
Enter command: 3
============================
[s]-by square, [t]-by type: s
enter square: 2042
Rhomb deleted
============================
Menu:
1 - Add figure,
2 - Print all,
3 - Delete,
0 - Exit
============================
Enter command: 2
============================
[0] size = 4
\__ diagonal 1 = 8
diagonal 2 = 4 Square = 16 (I am rhomb)
   \__ base 1 = 43
base 2 = 5
height = 4 Square = 96 (I am trapeze)
   \__ base 1 = 93
base 2 = 4
height = 2 Square = 97 (I am trapeze)
   \__ side = 5
diagonal = 49
height 1 = 6
height 2 = 3 Square = 235 (I am pentagon)
[1] size = 0
[2] size = 0
[3] size = 0
[4] size = 0
[5] size = 0
[6] size = 0
[7] size = 0
[8] size = 0
[9] size = 0
============================
```

```
height 1 = 6
height 2 = 3 Square = 235 (I am pentagon)
   \__ diagonal 1 = 43
diagonal 2 = 95 Square = 2042 (I am rhomb)
[1] size = 0
[2] size = 0
[3] size = 0
[4] size = 0
[5] size = 0
[6] size = 0
[7] size = 0
[8] size = 0
[9] size = 0
============================
Menu:
1 - Add figure,
2 - Print all,
3 - Delete,
0 - Exit
============================
Enter command: 1
============================
enter figure name (p - pentagon, r -
rhomb, t - trapeze): t
Enter base 1: 93
Enter base 2: 4
Enter height: 2
Trapeze created
============================
```

```
Menu:
1 - Add figure,
2 - Print all,
3 - Delete,
0 - Exit
============================
Enter command: 3
============================
[s]-by square, [t]-by type: t
enter figure type (p - pentagon, r -
rhomb, t - trapeze): t
Trapeze deleted
Trapeze deleted
============================
Menu:
1 - Add figure,
2 - Print all,
3 - Delete,
0 - Exit
============================
Enter command: 2
============================
[0] size = 2
\__ diagonal 1 = 8
diagonal 2 = 4 Square = 16 (I am rhomb)
   \__ side = 5
diagonal = 49
height 1 = 6
height 2 = 3 Square = 235 (I am pentagon)
[1] size = 0
[2] size = 0
[3] size = 0
[4] size = 0
[5] size = 0
[6] size = 0
[7] size = 0
[8] size = 0
[9] size = 0
```

6. Вывод

Данная работа была довольно большой по объему и трудоемкости, так как понять как один контейнер вкладывается в другой требует много времени. Так как материала в интернете не очень много, то приходится искать информацию в самых неожиданных местах: индийские туториалы на ютубе и англоязычные любительские сайты.

Лабораторная работа № 8

1. **Цель работы**

- Знакомство с параллельным программированием в C++.

2. **Задание**

Используя структуры данных, разработанные для лабораторной работы №6 (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:

- mutex
- lock_guard

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

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

3. **Описание**

Класс `mutex` является примитивом синхронизации, который может использоваться для защиты разделяемых данных от одновременного доступа нескольких потоков. Вызывающий поток владеет мьютексом со времени успешного вызова `lock` или `try_lock`, и до момента вызова `unlock`.

Пока поток владеет мьютексом, все остальные потоки при попытке завладения им блокируются на вызове `lock` или получают значение false при вызове `try_lock`. Вызывающий поток не должен владеть мьютексом до вызова `lock` или `try_lock`.

Шаблонный класс `std::future` обеспечивает механизм доступа к результатам асинхронных операций. Асинхронные операции (созданные с помощью std::async, std::packaged_task, или std::promise) могут вернуть объект типа `std::future` создателю этой операции.

Создатель асинхронной операции может использовать различные методы запроса, ожидания или получения значения из `std::future`. Этим методы могут заблокировать выполнение до получения результата асинхронной операции.

Когда асинхронная операция готова к отправке результата её создателю, она может сделать это, изменив shared state (например, std::promise::set_value), которое связано с `std::future` создателя.

4. Код программы

FigureArray.cpp

```
#include "FigureArray.h"
#include <algorithm>

template <class T1, class T2, class T3> FigureArray<T1, T2, T3>::FigureArray(int size)
{
	data = new ArrayItem<T1, T2, T3>[size];
	FigureArray<T1, T2, T3>::size = size;
	for (int i = 0; i < size; i++)
	{
		data[i].SetIndex(i);
	}
	first = Iterator<T1, T2, T3>(data);
	last = Iterator<T1, T2, T3>(&data[size]);
}

template <class T1, class T2, class T3> void FigureArray<T1, T2,
T3>::Insert(std::shared_ptr<T1> &pentagon, int index)
{
	data[index] = ArrayItem<T1, T2, T3>(pentagon, index);
}
template <class T1, class T2, class T3> void FigureArray<T1, T2,
T3>::Insert(std::shared_ptr<T2> &rhomb, int index)
{
	data[index] = ArrayItem<T1, T2, T3>(rhomb, index);
}
template <class T1, class T2, class T3> void FigureArray<T1, T2,
T3>::Insert(std::shared_ptr<T3> &trapeze, int index)
{
	data[index] = ArrayItem<T1, T2, T3>(trapeze, index);
}

template <class T1, class T2, class T3> bool FigureArray<T1, T2, T3>::IsPentagon(int
index)
{
	return data[index].IsPentagon();
}
template <class T1, class T2, class T3> bool FigureArray<T1, T2, T3>::IsRhomb(int index)
{
	return data[index].IsRhomb();
}
template <class T1, class T2, class T3> bool FigureArray<T1, T2, T3>::IsTrapeze(int
index)
{
	return data[index].IsTrapeze();
}

template <class T1, class T2, class T3> std::shared_ptr<T1> FigureArray<T1, T2,
T3>::GetPentagon(int index)
{
	return data[index].GetPentagon();
}
template <class T1, class T2, class T3> std::shared_ptr<T2> FigureArray<T1, T2,
T3>::GetRhomb(int index)
{
	return data[index].GetRhomb();
}
template <class T1, class T2, class T3> std::shared_ptr<T3> FigureArray<T1, T2,
T3>::GetTrapeze(int index)
{
	return data[index].GetTrapeze();
```

```
}

template <class T1, class T2, class T3> void FigureArray<T1, T2, T3>::Delete(int index)
{
	data[index] = ArrayItem<T1, T2, T3>();
}

template <class T1, class T2, class T3> std::ostream& operator << (std::ostream &os,
FigureArray<T1, T2, T3> &array)
{
	for (int i = 0; i < array.size; i++)
	{
		os << "[" << i << "] " << array.data[i] << std::endl;
	}
	return os;
}

template <class T1, class T2, class T3> Iterator<T1, T2, T3> FigureArray<T1, T2,
T3>::begin()
{
	return Iterator<T1, T2, T3>(data);
}

template <class T1, class T2, class T3> Iterator<T1, T2, T3> FigureArray<T1, T2,
T3>::end()
{
	return Iterator<T1, T2, T3>(&data[size]);
}

template <class T1, class T2, class T3>
FigureArray<T1, T2, T3>::~FigureArray() {}

template <class T1, class T2, class T3>
void FigureArray<T1, T2, T3>::QuickSort(int L, int R)
{
	int left = L;
	int right = R;

	ArrayItem<T1, T2, T3> middle = data[(right + left) / 2];

	do
	{
		while (data[left] < middle) left++;
		while (data[right] > middle) right--;

		if (left <= right)
		{
			if (data[left]>data[right])
				std::swap(data[left], data[right]);
			left++;
			right--;
		}
	} while (left <= right);

	if (right > L) QuickSort(L, right);
	if (left < R) QuickSort(left, R);
}

template <class T1, class T2, class T3>
std::future<void> FigureArray<T1, T2, T3>::QuickSortInBackground(int L, int R)
{
	std::packaged_task<void(int, int)> task
	(
		std::bind
		(
```

```
                    &FigureArray<T1, T2, T3>::QuickSortParallel,
                    this,
                    std::placeholders::_1,
                    std::placeholders::_2
                )
        );
        std::future<void> res(task.get_future());
        std::thread th(std::move(task), L, R);
        th.detach();
        return res;
}

template <class T1, class T2, class T3>
void FigureArray<T1, T2, T3>::QuickSortParallel(int L, int R)
{
        int left = L;
        int right = R;

        ArrayItem<T1, T2, T3> middle = data[(right + left) / 2];

        do
        {
                while (data[left] < middle) left++;
                while (data[right] > middle) right--;

                if (left <= right)
                {
                        if (data[left]>data[right]) std::swap(data[left], data[right]);
                        left++;
                        right--;
                }
        } while (left <= right);

        std::future<void> left_res;
        std::future<void> right_res;

        if (right > L)
        {
                left_res = QuickSortInBackground(L, right);
        }
        if (left < R)
        {
                right_res = QuickSortInBackground(left, R);
        }
        if (right > L) left_res.get();
        if (left < R) right_res.get();

}
```

5. Консоль

```
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print all,
5 - Sort,
0 - Exit
============================
Enter command: 1
============================
Enter figure name (p - pentagon, r - rhomb,
t - trapeze): t
Enter index: 0
0 - Exit
============================
Enter command: 2
============================
Enter index: 0
base 1 = 32
base 2 = 35
height = 4
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print all,
```

```
Enter base 1: 32
Enter base 2: 35
Enter height: 4
Trapeze created
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print all,
5 - Sort,
0 - Exit
============================
Enter command: 1
============================
Enter figure name (p - pentagon, r - rhomb,
t - trapeze): p
Enter index: 4
Enter side a: 2
Enter side b: 46
Enter height 1: 34
Enter height 2: 30
Pentagon created
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print all,
5 - Sort,
0 - Exit
============================
Enter command: 1
============================
Enter figure name (p - pentagon, r - rhomb,
t - trapeze): r
Enter index: 3
Enter diagonal 1: 54
Enter diagonal 2: 54
Rhomb created
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print all,
5 - Sort,
0 - Exit
============================
Enter command: 1
============================
Enter figure name (p - pentagon, r - rhomb,
t - trapeze): t
Enter index: 2
Enter base 1: 43
Enter base 2: 5
Enter height: 4
Trapeze created
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print all,
5 - Sort,
0 - Exit
============================
Enter command: 4
============================
[0] base 1 = 32
base 2 = 35
height = 4 Square = 134 (I am trapeze)
[1] Empty
[2] base 1 = 43
base 2 = 5
height = 4 Square = 96 (I am trapeze)
[3] diagonal 1 = 54
diagonal 2 = 54 Square = 1458 (I am rhomb)
[4] side = 2
diagonal = 46
height 1 = 34
height 2 = 30 Square = 1506 (I am pentagon)
[5] Empty
[6] Empty
[7] Empty
[8] Empty
[9] Empty
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print all,
5 - Sort,
0 - Exit
============================
Enter command: 5
============================
Sorted
============================
Menu:
1 - Add figure,
2 - Print figure,
3 - Delete,
4 - Print all,
5 - Sort,
0 - Exit
============================
Enter command: 4
============================
[0] Empty
[1] Empty
[2] Empty
[3] Empty
[4] Empty
[5] Empty
[6] base 1 = 43
base 2 = 5
height = 4 Square = 96 (I am trapeze)
[7] base 1 = 32
base 2 = 35
height = 4 Square = 134 (I am trapeze)
[8] diagonal 1 = 54
diagonal 2 = 54 Square = 1458 (I am rhomb)
[9] side = 2
diagonal = 46
height 1 = 34
height 2 = 30 Square = 1506 (I am pentagon)
```

6. **Вывод**

Как уже было сказано, для программиста важным объектом наблюдений является время и память. При использовании потоков, распараллеленные программы работают быстрее, что очень важно. Рекомендуется разделять даже не связанные куски программы, для того, чтобы они выполнялись одновременно.

Лабораторная работа № 9

1. **Цель работы**

- Знакомство с лямбда-выражениями

2. **Задание**

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 контенере 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.

3. **Описание**

Лямбда-выражение в C++11 — это удобный способ определения анонимного объекта-функции непосредственно в месте его вызова или передачи в функцию в качестве аргумента. Обычно лямбда-выражения используются для инкапсуляции нескольких строк кода, передаваемых алгоритмам или асинхронным методам.

В теле лямбда-выражения могут вводиться новые переменные (в **C ++14**). Кроме того, лямбда-выражения могут использовать, или *фиксировать*, переменные из окружающей области видимости. Лямбда-выражение начинается с предложения фиксации (в синтаксисе стандарта — *lambda-introducer*), в котором указываются фиксируемые переменные, а также способ фиксации: по значению или по ссылке. Доступ к переменным

с префиксом с амперсандом (`&`) осуществляется по ссылке, а к переменным без префикса — по значению.

Пустое предложение фиксации (`[ ]`) показывает, что тело лямбда-выражения не осуществляет доступ к переменным во внешней области видимости.

Чтобы задать способ фиксации внешних переменных, на которые ссылается лямбда-выражение, вы можете использовать режим фиксации по умолчанию (в синтаксисе стандарта — `capture-default`): [&] — все переменные фиксируются по ссылке, [=] — по значению.

4. Код программы

NTree.h

```
#ifndef NTREE_H
#define NTREE_H

#include <memory>
#include <future>
#include <mutex>
#include <thread>
#include "TreeItem.h"

template <class T>
class NTree
{
public:
	NTree();
	TreeItem<T>* Insert(TreeItem<T>* root, T value);
	TreeItem<T>* root;
	int size;
private:
};
#endif
```

TreeItem.h

```
#ifndef TREEITEM_H
#define TREEITEM_H

#include <thread>

const int max_children_size = 4;

template <class T>
class TreeItem
{
public:
	// конструкторы
	TreeItem();
	TreeItem(T v);
	bool IsLeaf(); // проверка, является ли листом
	T value; // значение
	TreeItem<T>* children[max_children_size]; // массив указателей на детей
```

```
private:
};
#endif
```

NTree.cpp

```
#include "NTree.h"
#include <queue>
#include <iostream>

template <class T>
NTree<T>::NTree() :
	root(nullptr),
	size(0) {}

template <class T>
TreeItem<T>* NTree<T>::Insert(TreeItem<T>* root, T value)
{
	std::queue <TreeItem<T>*> nodes_to_visit;
	TreeItem<T>* current_node;
	if (!root)
	{
		root = new TreeItem<T>(value);
		return root;
	}
	else
	{
		nodes_to_visit.push(root);
	}
	while (!nodes_to_visit.empty())
	{
		current_node = nodes_to_visit.front();
		nodes_to_visit.pop();

		for (int i = 0; i<max_children_size; i++)
		{
			if (!current_node->children[i])
			{
				current_node->children[i] = new TreeItem<T>(value);
				return root;
			}
			else nodes_to_visit.push(current_node->children[i]);
		}
	}
}

#include <functional>
template class NTree<std::function<void(void)>>;
```

TreeItem.cpp

```
#include "TreeItem.h"
#include <functional>

template <class T>
TreeItem<T>::TreeItem() : value(NULL)
{
	for (int i = 0; i < max_children_size; i++) children[i] = nullptr;
}

template <class T>
TreeItem<T>::TreeItem(T v) : value(v)
{
	for (int i = 0; i < max_children_size; i++) children[i] = nullptr;
```

```
}

template <class T>
bool TreeItem<T>::IsLeaf()
{
	bool flag = 1;
	for (int i = 0; i < max_children_size; i++)
	{
		if (children[i]) flag = 0;
	}
	return flag;
}

template class TreeItem<std::function<void(void)>>;
template std::ostream& operator << (std::ostream &os, TreeItem<std::function<void(void)>>
item);
```

main.cpp

```
#include <iostream>
#include <cstdlib>
#include <random>
#include <functional>
#include <future>
#include <thread>
#include <queue>

#include "FigureArray.h"
#include "NTree.h"
#include "Pentagon.h"
#include "Rhomb.h"
#include "Trapeze.h"

const int array_size = 10;

typedef std::function<void(void)> command;
FigureArray <Pentagon, Rhomb, Trapeze> figure_array(array_size); // создание массива
фигур
NTree<command> tree; // создание дерева команд

//обход в ширину дерева команд и их выполнение
void Executing(TreeItem<command> *root)
{
	std::queue <TreeItem<command>*> nodes_to_visit;
	TreeItem<command>* current_node;
	nodes_to_visit.push(root);
	while (!nodes_to_visit.empty())
	{
		current_node = nodes_to_visit.front();
		nodes_to_visit.pop();
		command cmd = current_node->value;
		cmd();

		for (int i = 0; i < max_children_size; i++)
		{
			if (current_node->children[i])
			{
				nodes_to_visit.push(current_node->children[i]);
			}
		}
	}
}

int main()
```

```
{
	int pos = 0;
	// лямбда-функция вставки
	command cmd_insert = [&]()
	{
		int n;
		std::cout << "number of figures: ";
		std::cin >> n;
		for (int i = 0; i < n; i++)
		{
			int figure_number = rand() % 3;
			int a = rand() % 10;
			int b = rand() % 10;
			int c = rand() % 10;
			int d = rand() % 10;
			if (figure_number == 0)
				figure_array.Insert(std::shared_ptr<Pentagon>(new Pentagon(a,
b, c, d)), pos);
			else if (figure_number == 1)
				figure_array.Insert(std::shared_ptr<Rhomb>(new Rhomb(a, b)),
pos);
			else if (figure_number == 2)
				figure_array.Insert(std::shared_ptr<Trapeze>(new Trapeze(a, b,
c)), pos);
			pos++;
		}
	};

	// лямбда-функция удаления по площади
	command cmd_delete_by_square = [&]()
	{
		int square;
		std::cout << "Enter square: ";
		std::cin >> square;
		figure_array.DeleteBySquare(square);
	};
	// лямбда-функция печати
	command cmd_print = [&]()
	{
		std::cout << "Figure array:\n" << figure_array;
	};

	//вставка команд в дерево команд
	tree.root = tree.Insert(tree.root, cmd_insert);
	tree.root = tree.Insert(tree.root, cmd_insert);
	tree.root = tree.Insert(tree.root, cmd_insert);
	tree.root = tree.Insert(tree.root, cmd_print);
	tree.root = tree.Insert(tree.root, cmd_delete_by_square);
	tree.root = tree.Insert(tree.root, cmd_print);

	//обход дерева и выполнение
	Executing(tree.root);
	system("pause");
	return 0;
}
```

5. Консоль

```
number of figures: 2
Trapeze created
Rhomb created
number of figures: 2
Trapeze created
Enter square: 28
Trapeze deleted
Trapeze deleted
Trapeze deleted
Pentagon deleted
```

```
Trapeze created
number of figures: 3
Pentagon created
Rhomb created
Trapeze created
Figure array:
size:10
[0] base 1 = 7
base 2 = 4
height = 0
Square = 0 (I am trapeze)
[1] diagonal 1 = 8
diagonal 2 = 8
Square = 32 (I am rhomb)
[2] base 1 = 5
base 2 = 1
height = 7
Square = 21 (I am trapeze)
[3] base 1 = 5
base 2 = 2
height = 7
Square = 24 (I am trapeze)
[4] side = 4
diagonal = 2
height 1 = 3
height 2 = 2
Square = 11 (I am pentagon)
[5] diagonal 1 = 1
diagonal 2 = 6
Square = 3 (I am rhomb)
[6] base 1 = 6
base 2 = 1
height = 8
Square = 28 (I am trapeze)
[7] Empty
[8] Empty
[9] Empty
```

```
Rhomb deleted
Figure array:
size:10
[0] Empty
[1] diagonal 1 = 8
diagonal 2 = 8
Square = 32 (I am rhomb)
[2] Empty
[3] Empty
[4] Empty
[5] Empty
[6] base 1 = 6
base 2 = 1
height = 8
Square = 28 (I am trapeze)
[7] Empty
[8] Empty
[9] Empty
```

6. Вывод

Лямбда выражения представляют упрощенную запись анонимных методов, позволяют создать емкие методы, которые могут возвращать некоторое значение и которые можно передать в качестве параметров другим методам. В отличие от функций они компактны и не требуют объявления класса.

GitHub: https://github.com/9largefries/MAI/tree/master/3sem/OOP