Лабораторная работа №1

❖ ЗАДАНИЕ

Данный курс построен таким образом, что каждая новая лабораторная – это продолжение старой, поэтому в отчете приведены только основные фрагменты проекта, дабы избежать лишних повторений одинаковых программ.
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).
Программа должна позволять вводить фигуру каждого типа с клавиатуры, выводить параметры фигур на
экран и их площадь.

Фигуры: Пятиугольник, трапеция, ромб.

❖ ОПИСАНИЕ

Классы и объекты в C++ являются основными концепциями объектно-ориентированного программирования — ООП.
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.

Для разграничения содержимого класса, например которое пользователю лучше не трогать, были добавлены ***спецификаторы доступа*** public, private. Это и есть инкапсуляция, которую мы упоминали выше.
• ***public*** — дает публичный доступ, содержимому, которое в нем указано. Так можно обратится к любой переменной или функции из любой части программы.

• ***private*** — запрещает обращаться к свойствам вне класса. Поэтому под крылом этого доступа часто находятся именно объявления переменных, массивов, а также прототипов функций.

Перегрузка операторов в программировании — 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.

Виртуальная функция — это функция, которая определяется в базовом классе, а любой порожденный класс может ее переопределить. Виртуальная функция вызывается только через указатель или ссылку на базовый класс.

Конструктор – это специальный метод класса, который предназначен для инициализации элементов класса некоторыми начальными значениями.

В отличие от конструктора, ***деструктор*** — специальный метод класса, который служит для уничтожения элементов класса. Чаще всего его используют тогда, когда в конструкторе при создании объекта класса динамически был выделен участок памяти и необходимо эту память очистить, если эти значения уже не нужны для дальнейшей работы программы.

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.

❖ 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
```

- **ВЫВОД КОНСОЛИ**

```
==========================
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
```

- **ВЫВОДЫ**

Данная лабораторная работа является своеобразным вводным уроком в ООП, знакомя меня с основными принципами данной парадигмы. Так же происходит знакомство и завязывается тесная дружба с классами, перегрузками, деструкторами. В результате выполнения задания были спроектированы классы фигур, в которых использовались операции ввода-вывода из стандартных библиотек.

Лабораторная работа №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).
• 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).
• 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.
Контейнер первого уровня: список.

- **ОПИСАНИЕ**

Если до начала работы с данными невозможно определить, сколько памяти потребуется для их хранения, память следует распределять во время выполнения программы по мере необходимости отдельными блоками. Блоки связываются друг с другом с помощью указателей. Такой способ организации данных называется ***динамической структурой данных***, поскольку она размещается в динамической памяти и ее размер изменяется во время выполнения программы.
Динамические структуры данных в процессе существования в памяти могут изменять не только число составляющих их элементов, но и характер связей между элементами. При этом не учитывается изменение содержимого самих элементов данных. Такая особенность динамических структур, как непостоянство их размера и характера отношений между элементами, приводит к тому, что на этапе создания машинного кода программа-компилятор не может выделить для всей структуры в целом участок памяти фиксированного размера, а также не может сопоставить с отдельными компонентами структуры конкретные адреса. Для решения проблемы адресации динамических структур данных используется метод, называемый динамическим распределением памяти, то есть память под отдельные элементы выделяется в момент, когда они "начинают существовать" в процессе выполнения программы, а не во время компиляции. Компилятор в этом случае выделяет фиксированный объем памяти для хранения адреса динамически размещаемого элемента, а не самого элемента.

Динамическая структура данных характеризуется тем что:

1. Она не имеет имени;
2. Ей выделяется память в процессе выполнения программы;
3. Количество элементов структуры может не фиксироваться;
4. Размерность структуры может меняться в процессе выполнения программы;
5. В процессе выполнения программы может меняться характер взаимосвязи между элементами структуры.

❖ ИСХОДНЫЙ КОД

```
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;
}
```

- **ВЫВОД КОНСОЛИ**

```
==========================
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: 4
```

❖ **ВЫВОД**

Данная лабораторная работа обеспечивает навыки работы с динамическими структурами, которые применимы, когда используются переменные, имеющие довольно большой размер (например, массивы большой размерности), необходимые в

одних частях программы и совершенно не нужные в других; в процессе работы программы нужен массив, список или иная структура, размер которой изменяется в широких пределах и трудно предсказуем; когда размер данных, обрабатываемых в программе, превышает объем сегмента данных.

Лабораторная работа №3

- **ЗАДАНИЕ**

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня,
содержащий все три фигуры класса фигуры, согласно вариантов 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

- **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).
В новом стандарте появились следующие умные указатели: unique_ptr, shared_ptr и weak_ptr. Все они объявлены в заголовочном файле <memory>.

1. unique_ptr

Этот указатель пришел на смену старому и проблематичному auto_ptr. Основная проблема последнего заключается в правах владения. Объект этого класса теряет права владения ресурсом при копировании (присваивании, использовании в конструкторе копий, передаче в функцию по значению).Это очень неудобно, при работе с контейнером из умных указателей.В отличии от auto_ptr, unique_ptr запрещает копирование.

2. shared_ptr

Это самый популярный и самый широкоиспользуемый 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.

3. weak_ptr

Этот указатель также, как и shared_ptr начал свое рождение в проекте boost, затем был включен в C++ Technical Report 1 и, наконец, пришел в новый стандарт.
Данный класс позволяет разрушить циклическую зависимость, которая, несомненно, может образоваться при использовании shared_ptr.

❖ ИСХОДНЫЙ КОД

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;
}
```

❖ ВЫВОД КОНСОЛИ

```
=======================
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: 5
```

- **ВЫВОД**

Умные указатели — очень удобная и полезная вещь. 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.

Лабораторная работа №4

- **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
(опередеряется структурой контейнера).
• Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера (опередеряется 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.

- **ОПИСАНИЕ**

Шаблон класса начинается с ключевого слова template. В угловых скобках записывают параметры шаблона. При использовании шаблона на место этих параметров шаблону передаются аргументы: типы и константы, перечисленные через запятую.
Типы могут быть как стандартными, так и определенными пользователем. Для их описания в списке параметров используется ключевое слово class.
Описание параметров шаблона в заголовке функции должно соответствовать шаблону класса.
Локальные классы не могут иметь шаблоны в качестве своих элементов.
Шаблоны методов не могут быть виртуальными.
Шаблоны классов могут содержать статические элементы, дружественные функции и классы.

Шаблоны могут быть производными как от шаблонов, так и от обычных классов, а также являться базовыми и для шаблонов, и для обычных классов.

❖ ИСХОДНЫЙ КОД

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() {}
```

❖ ВЫВОД КОНСОЛИ

```
========================
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
```

❖ **ВЫВОД**

Шаблон класса позволяет задать класс, параметризованный типом данных. Передача классу различных типов данных в качестве параметра создает семейство родственных классов. Наиболее широкое применение шаблоны находят при создании контейнерных классов. Контейнерным называется класс, который предназначен для хранения каким-либо образом организованных данных и работы с ними. Преимущество использования шаблонов состоит в том, что как только алгоритм работы с данными определен и отлажен, он может применяться к любым типам данных без переписывания кода.

Лабораторная работа №5

❖ **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.

❖ ОПИСАНИЕ

Итератор — это объект, который может выполнять итерацию элементов в контейнере STL и предоставлять доступ к отдельным элементам. Все контейнеры STL предоставляют итераторы, чтобы алгоритмы могли получить доступ к своим элементам стандартным способом, независимо от типа контейнера, в котором сохранены элементы.
Для получения итераторов контейнеры в C++ обладают такими функциями, как begin() и end(). Функция begin() возвращает итератор, который указывает на первый элемент контейнера (при наличии в контейнере элементов). Функция end() возвращает итератор, который указывает на следующую позицию после последнего элемента, то есть по сути на конец контейнера. Если контейнер пуст, то итераторы, возвращаемые обоими методами begin и end совпадают. Если итератор begin не равен итератору end, то между ними есть как минимум один элемент.
Принцип работы итераторов очень похожий на работу указателей: для получения значения также используется оператор разыменования, операции инкремента и декремента обеспечивают доступ в прямом и обратном направлении соответственно.
Для динамического массива понадобится выполнять **операцию разыменования** (обращаться к значению на которое указывает итератор), сравнивать на равенства, использовать инкремент(++) и сравнение на неравенство.

❖ ИСХОДНЫЙ КОД

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++;
}
```

- **ВЫВОД КОНСОЛИ**

```
============================
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:
```

❖ ВЫВОД

Главное предназначение итераторов заключается в предоставлении возможности пользователю обращаться к любому элементу контейнера при сокрытии внутренней структуры контейнера от пользователя. Это позволяет контейнеру хранить элементы любым способом при допустимости работы пользователя с ним как с простой последовательностью или списком. Проектирование класса итератора обычно тесно связано с соответствующим классом контейнера. Обычно контейнер предоставляет методы создания итераторов. Итератор похож на указатель своими основными операциями: он указывает на отдельный элемент коллекции объектов (предоставляет доступ к элементу) и содержит функции для перехода к другому элементу списка (следующему или предыдущему). Контейнер, который реализует поддержку итераторов, должен предоставлять первый элемент списка, а также возможность проверить, перебраны ли все элементы контейнера (является ли

итератор конечным). В зависимости от используемого языка и цели, итераторы могут поддерживать дополнительные операции или определять различные варианты поведения.

Лабораторная работа №6

- **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.
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:
• Стандартные контейнеры std.
Программа должна позволять:
• Вводить произвольное количество фигур и добавлять их в контейнер.
• Распечатывать содержимое контейнера.
• Удалять фигуры из контейнера.

Контейнер второго уровня: очередь.

- **ОПИСАНИЕ**

"Allocator" - переводится как "распределитель" - попросту говоря аллокатор - это действующая по определённой логике высокоуровневая прослойка между запросами памяти под динамические объекты и стандартными сервисами выделения памяти (new/malloc или другими (например запросами напрямую к ядру о.с.)), конечно же прослойка берет на себя и вопросы управлением отдачей уже ненужной памяти назад. По другому можно сказать - что аллокатор это реализация стратегии управления памятью.
Естественно что епархия аллокаторов - в основном объекты-контейнеры, т.к. именно там остро проявляется необходимость в некоторой стратегии, но так же можно использовать и для одиночных динамических объектов, на протяжении всего времени жизни программы размещая эти объекты в заранее выделенном бассейне "pool" памяти. Кстати обычно pool переводят как пул, но бассейн вроде звучит - так что бассейне памяти. Так же обычно pool аллокаторы - понятие, применяемое к определенной категории аллокаторов, работающих с объектами одного размера. Использование стратегии будет не выгодно по памяти, но очень выгодно по времени выполнения.

- **ИСХОДНЫЙ КОД**

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);
free_blocks = (void**)malloc(sizeof(void*)*count);
for (int i = 0; i < count; i++)
free_blocks[i] = used_blocks + i*size;
free_count = count;
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;
}
```

❖ ВЫВОД КОНСОЛИ

```
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
```

❖ **ВЫВОД**

Использование аллокаторов позволяет добиться существенного повышения производительности в работе с динамическими объектами (особенно с объектами-контейнерами). Если в коде много работы по созданию/уничтожению динамических объектов, и нет никакой стратегии управления памятью, а только использование стандартных сервисов new/malloc - то весьма вероятно, что не смотря на то, что программа написанна на Си++ (или даже чистом Си) - она окажется более медленной чем схожая программа написанная на Java или C#.

Ссылка на GitHub: `https://github.com/pavels-k/OOP.git`