Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы

по курсу «Объектно-ориентированное программирование»

III семестр

|  |  |
| --- | --- |
| Студент: | Гринин В.В. |
| Группа: | 8О-208Б, №1 |
| Преподаватель: | Поповкин А.В. |
|  |  |
| Оценка: |  |
| Дата: |  |

Москва 2018

Содержание

[Лабораторная работа №1 3](#_Toc533283844)

[Цель работы: 4](#_Toc533283845)

[Задание: 4](#_Toc533283846)

[Код программы: 4](#_Toc533283847)

[Тесты: 8](#_Toc533283848)

[Лабораторная работа №2 9](#_Toc533283849)

[Цель работы: 10](#_Toc533283850)

[Задание: 10](#_Toc533283851)

[Код программы: 11](#_Toc533283852)

[Тесты: 16](#_Toc533283853)

[Лабораторная работа №3 17](#_Toc533283854)

[Цель работы: 18](#_Toc533283855)

[Задание: 18](#_Toc533283856)

[Код программы: 19](#_Toc533283857)

[Тесты: 26](#_Toc533283858)

[Лабораторная работа №4 27](#_Toc533283859)

[Цель работы: 28](#_Toc533283860)

[Задание: 28](#_Toc533283861)

[Код программы: 29](#_Toc533283862)

[Тесты: 30](#_Toc533283863)

[Лабораторная работа №5 31](#_Toc533283864)

[Цель работы: 32](#_Toc533283865)

[Задание: 32](#_Toc533283866)

[Код программы: 33](#_Toc533283867)

[Тесты: 37](#_Toc533283868)

[Лабораторная работа №6 38](#_Toc533283869)

[Цель работы: 39](#_Toc533283870)

[Задание: 39](#_Toc533283871)

[Код программы: 40](#_Toc533283872)

[Тесты: 44](#_Toc533283873)

[Лабораторная работа №7 45](#_Toc533283874)

[Цель работы: 46](#_Toc533283875)

[Задание: 46](#_Toc533283876)

[Код программы: 47](#_Toc533283877)

[Тесты: 51](#_Toc533283878)

[Лабораторная работа №8 52](#_Toc533283879)

[Цель работы: 53](#_Toc533283880)

[Задание: 53](#_Toc533283881)

[Код программы: 54](#_Toc533283882)

[Тесты: 54](#_Toc533283883)

[Заключение 54](#_Toc533283884)

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №1.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Гринин В.В. |
| Группа: | 8О-208Б, №1 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №5 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Программирование классов на языке С++; управление памятью в языке С++; 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.

## Задание:

Необходимо спроектировать и запрограммировать на языке C++ классы фигур, согласно [вариантов задания](#_Варианты_задания_(фигуры)).

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.

Должны 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.

Вариант:

Фигура 1: ромб

Фигура 2: 5-угольник

Фигура 3: 6-угольник

## Код программы:

**Figure.h:**

#ifndef FIGURE\_H

#define FIGURE\_H

class Figure {

public:

virtual double Square() = 0;

virtual void Print() = 0;

virtual ~Figure() {};

};

#endif /\* FIGURE\_H \*/

**Hexagon.h:**

#ifndef HEXAGON\_H

#define HEXAGON\_H

#include <cstdlib>

#include <iostream>

#include "Figure.h"

class Hexagon : public Figure{

public:

Hexagon();

Hexagon(std::istream &is);

Hexagon(size\_t a);

Hexagon(const Hexagon& orig);

double Square() override;

void Print() override;

virtual ~Hexagon();

private:

size\_t side\_a;

};

#endif /\* HEXAGON\_H \*/

**Hexagon.cpp:**

#include "Hexagon.h"

#include <iostream>

#include <cmath>

Hexagon::Hexagon() : Hexagon(0) {

}

Hexagon::Hexagon(size\_t i) : side\_a(i) {

std::cout << "Hexagon created: " << side\_a << std::endl;

}

Hexagon::Hexagon(std::istream &is) {

is >> side\_a;

}

Hexagon::Hexagon(const Hexagon& orig) {

std::cout << "Hexagon copy created" << std::endl;

side\_a = orig.side\_a;

}

double Hexagon::Square() {

double tmp = ((3\*sqrt(3))/2) \* pow(side\_a, 2);

std::cout << "Square = " << tmp << std::endl;

return tmp;

}

void Hexagon::Print() {

std::cout << "a = " << side\_a << std::endl;

}

Hexagon::~Hexagon() {

std::cout << "Hexagon deleted" << std::endl;

}

**Pentagon.h:**

#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 a);

Pentagon(const Pentagon& orig);

double Square() override;

void Print() override;

virtual ~Pentagon();

private:

size\_t side\_a;

};

#endif /\* PENTAGON\_H \*/

**Pentagon.cpp:**

#include "Pentagon.h"

#include <iostream>

#include <cmath>

const double pi = 3.141592;

const double k = 0.688191;

Pentagon::Pentagon() : Pentagon(0) {

}

Pentagon::Pentagon(size\_t i) : side\_a(i) {

std::cout << "Pentagon created: " << side\_a << std::endl;

}

Pentagon::Pentagon(std::istream &is) {

is >> side\_a;

}

Pentagon::Pentagon(const Pentagon& orig) {

std::cout << "Pentagon copy created" << std::endl;

side\_a = orig.side\_a;

}

double Pentagon::Square() {

double r = k \* side\_a;

double tmp = 5 \* pow(r, 2) \* tan(pi/5);

std::cout << "Square = " << tmp << std::endl;

return tmp;

}

void Pentagon::Print() {

std::cout << "a = " << side\_a << std::endl;

}

Pentagon::~Pentagon() {

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

}

**Rhombus.h:**

#ifndef RHOMBUS\_H

#define RHOMBUS\_H

#include <cstdlib>

#include <iostream>

#include "Figure.h"

class Rhombus : public Figure{

public:

Rhombus();

Rhombus(std::istream &is);

Rhombus(size\_t d1, size\_t d2);

Rhombus(const Rhombus& orig);

double Square() override;

void Print() override;

virtual ~Rhombus();

private:

size\_t side\_d1;

size\_t side\_d2;

};

#endif /\* RHOMBUS\_H \*/

**Rhombus.cpp:**

#include "Rhombus.h"

#include <iostream>

#include <cmath>

Rhombus::Rhombus() : Rhombus(0, 0) {

}

Rhombus::Rhombus(size\_t i, size\_t j) : side\_d1(i), side\_d2(j) {

std::cout << "Rhombus created: " << side\_d1 << ", " << side\_d2 << std::endl;

}

Rhombus::Rhombus(std::istream &is) {

is >> side\_d2;

is >> side\_d1;

}

Rhombus::Rhombus(const Rhombus& orig) {

std::cout << "Rhombus copy created" << std::endl;

side\_d1 = orig.side\_d1;

side\_d2 = orig.side\_d2;

}

double Rhombus::Square() {

double tmp = (side\_d1 \* side\_d2)/2;

std::cout << "Square = " << tmp << std::endl;

return tmp;

}

void Rhombus::Print() {

std::cout << "d1 = " << side\_d1 << ", d2 = " << side\_d2 << std::endl;

}

Rhombus::~Rhombus() {

std::cout << "Rhombus deleted" << std::endl;

}

**main.cpp:**

#include <cstdlib>

#include "Hexagon.h"

#include "Pentagon.h"

#include "Rhombus.h"

int main(int argc, char\*\* argv) {

std::cout << "Choose the Figure:\n1. Hexagon\n2. Pentagon\n3. Rhombus" << std::endl;

char choice;

Figure \*ptr;

for(;;) {

std::cin >> choice;

switch(choice) {

case '1':

std::cout << "Enter the side: ";

ptr = new Hexagon(std::cin);

Hexagon test(\*ptr);

ptr->Print();

ptr->Square();

delete ptr;

break;

case '2':

std::cout << "Enter the side: ";

ptr = new Pentagon(std::cin);

ptr->Print();

ptr->Square();

delete ptr;

break;

case '3':

std::cout << "Enter 2 diagonals: ";

ptr = new Rhombus(std::cin);

ptr->Print();

ptr->Square();

delete ptr;

case 'e':

std::cout << "Exiting the program" << std::endl;

return 0;

case 'h':

std::cout << "Choose the Figure:\n1. Hexagon\n2. Pentagon\n3. Rhombus" << std::endl;

break;

default:

std::cout << "Error! For help type 'h'!" << std::endl;

break;

}

}

return 0;

}

**Makefile:**

all: main.exe

rm \*.o

main.exe: main.o Hexagon.o Pentagon.o Rhombus.o

g++ -Wall main.o Hexagon.o Pentagon.o Rhombus.o -o main.exe

main.o: main.cpp

g++ -Wall -c main.cpp

Hexagon.o: Hexagon.cpp

g++ -Wall -c Hexagon.cpp

Pentagon.o: Pentagon.cpp

g++ -Wall -c Pentagon.cpp

Rhombus.o: Rhombus.cpp

g++ -Wall -c Rhombus.cpp

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 1 3 | 3 23.3827 |
| 2 3 | 3 6.88191 |
| 3 2 7 | 2 7 7 |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №2.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Гринин В.В. |
| Группа: | 8О-208Б, №1 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №5 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Закрепление навыков работы с классами; создание простых динамических структур данных; работа с объектами, передаваемыми «по значению».

## Задание:

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер [первого уровня](#_Варианты_задания_(структуры), содержащий одну фигуру (колонка фигура 1), согласно [вариантов задания](#_Варианты_задания_(фигуры)) (реализованную в [ЛР1](#_Лабораторная_работа_№1)).

Классы должны удовлетворять следующим правилам:

Требования к классу фигуры аналогичны требованиям из [лабораторной работы 1](#_Задание).

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 содержать объекты фигур “по значению” (не по ссылке).

Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.

Класс-контейнер должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера).

Класс-контейнер должен иметь метод по удалению фигуры из контейнера (определяется 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 расположены 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 и удалять фигуры из контейнера.

Вариант:

Фигура 1: ромб

Фигура 2: 5-угольник

Фигура 3: 6-угольник

Контейнер первого уровня: массив

## Код программы:

Файлы, не претерпевшие изменений: Figure.h, Hexagon.h, Pentagon.h, Rhombus.h, Hexagon.cpp, Pentagon.cpp, Rhombus.cpp

**TArray.h:**

#ifndef TARRAY\_H

#define TARRAY\_H

#include "Rhombus.h"

class TArray {

public:

TArray();

void Add(Rhombus rhombus);

void Remove(Rhombus rhombus);

Rhombus\* GetRhombus(int pos) const;

unsigned int GetAmount() const;

friend std::ostream& operator<<(std::ostream& os, TArray& array);

virtual ~TArray();

private:

unsigned int amount;

Rhombus\* array;

};

#endif /\* TARRAY\_H \*/

**TArray.cpp:**

#include "TArray.h"

#include <iostream>

#include <cmath>

TArray::TArray() {

amount = 0;

array = NULL;

}

void TArray::Add(Rhombus rhombus) {

if(amount == 0) {

array = new Rhombus(rhombus);

if(array == nullptr) {

std::cout << "Ошибка - нехватка памяти!" << std::endl;

exit(1);

}

amount++;

return;

}

else {

array = (Rhombus\*) realloc(array, (amount + 1) \* sizeof(Rhombus));

/\*if(array == nullptr) {

std::cout << "Ошибка - нехватка памяти!" << std::endl;

exit(1);

}\*/

array[amount++] = rhombus;

return;

}

}

void TArray::Remove(Rhombus rhombus) {

Rhombus tmp;

for(unsigned int step = 0; step < amount; step++) {

if(rhombus == array[step]) {

for(unsigned int push = step; push < (amount - 1); push++) {

tmp = array[push];

array[push] = array[push + 1];

array[push + 1] = tmp;

}

array = (Rhombus\*) realloc(array, (--amount) \* sizeof(Rhombus));

break;

}

}

return;

}

unsigned int TArray::GetAmount() const {

return amount;

}

Rhombus\* TArray::GetRhombus(int pos) const {

return &array[pos];

}

std::ostream& operator<<(std::ostream& os, TArray& array) {

for(unsigned int step = 0; step < array.GetAmount(); step++) {

os << step + 1 << ". " << \*array.GetRhombus(step);

}

return os;

}

TArray::~TArray() {

delete array;

std::cout << "Array destroyed!" << std::endl;

}

**main.cpp:**

Полностью переписан main:

#include <cstdlib>

#include "Rhombus.h"

#include "TArray.h"

int main(int argc, char\*\* argv) {

char choice;

TArray array;

Rhombus\* tmp = NULL;

std::cout << "Choose the Function:\n1. Add\n2. Delete\n3. Print\ne. Close" << std::endl;

for(;;) {

std::cin >> choice;

switch(choice) {

case '1':

std::cout << "Enter the diagonals of rhombus:";

tmp = new Rhombus(std::cin);

array.Add(\*tmp);

delete tmp;

break;

case '2':

std::cout << "Enter the diagonals of rhombus:";

tmp = new Rhombus(std::cin);

array.Remove(\*tmp);

delete tmp;

break;

case '3':

std::cout << array << std::endl;

break;

case 'e':

std::cout << "Exiting the program" << std::endl;

return 0;

case 'h':cd

std::cout << "Choose the Function:\n1. Hexagon\n2. Pentagon\n3. Rhombus\n4. Use Stack for rhombus" << std::endl;

break;

default:

std::cout << "Error! For help type 'h'!" << std::endl;

break;

}

}

return 0;

}

**Makefile:**

all: main.exe

rm \*.o

main.exe: main.o Rhombus.o TArray.o

g++ -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -g main.o TArray.o Rhombus.o -o main.exe

main.o: main.cpp

g++ -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -c -g main.cpp

Rhombus.o: Rhombus.cpp

g++ -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -c -g Rhombus.cpp

TArray.o: TArray.cpp

g++ -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -c -g TArray.cpp

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 1 2 5 1 3 7 3 e | 2 5 3 7 |
| 1 2 7 1 2 1 2 2 1 3 e | 2 7 |
| 1 1 1 1 1 1 1 2 4 1 1 1 2 1 1 3 e | 1 1 2 4 1 1 |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №3.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Гринин В.В. |
| Группа: | 8О-208Б, №1 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №5 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Закрепление навыков работы с классами; знакомство с умными указателями.

## Задание:

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер [первого уровня](#_Варианты_задания_(структуры), содержащий все три фигуры класса фигуры, согласно [вариантов задания](#_Варианты_задания_(фигуры)) (реализованную в [ЛР1](#_Лабораторная_работа_№1)).

Классы должны удовлетворять следующим правилам:

Требования к классу фигуры аналогичны требованиям из [лабораторной работы 1](#_Задание).

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

Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.

Класс-контейнер должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера).

Класс-контейнер должен иметь метод по удалению фигуры из контейнера (определяется 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 расположены 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.

Вариант:

Фигура 1: ромб

Фигура 2: 5-угольник

Фигура 3: 6-угольник

Контейнер первого уровня: массив

## Код программы:

Файлы, не претерпевшие изменений: Figure.h

**Hexagon.h:**

Добавлено:

friend bool operator==(const Hexagon& left,const Hexagon& right);

Изменено:

Тип перемнной size\_a с size\_t в int.

**Hexagon.cpp:**

Добавлено:

bool operator==(const Hexagon& left,const Hexagon& right) {

if(left.side\_a == right.side\_a)

return true;

else

return false;

}

Изменёно:

Hexagon::Hexagon(std::istream &is) {

do {

is >> side\_a;

if(side\_a < 0) {

std::cout << "Number is < 0. Enter another number: ";

}

} while(side\_a < 0);

std::cout << "Hexagon created!" << std::endl;

}

Hexagon::Hexagon(int i) : side\_a(i) {

std::cout << "Hexagon created: " << side\_a << std::endl;

}

void Hexagon::Print() {

std::cout << "Hexagon: a = " << side\_a << std::endl;

}

**Pentagon.h:**

Добавлено:

friend bool operator==(const Pentagon& left,const Pentagon& right);

Изменено:

Тип переменной side\_a с size\_t в int.

**Pentagon.cpp:**

Добавлено:

bool operator==(const Pentagon& left,const Pentagon& right) {

if(left.side\_a == right.side\_a)

return true;

else

return false;

}

Изменёно:

Pentagon::Pentagon(int i) : side\_a(i) {

std::cout << "Pentagon created: " << side\_a << std::endl;

}

Pentagon::Pentagon(std::istream &is) {

do {

is >> side\_a;

if(side\_a < 0) {

std::cout << "Number is < 0. Enter another number: ";

}

} while(side\_a < 0);

std::cout << "Pentagon created!" << std::endl;

}

bool operator==(const Pentagon& left,const Pentagon& right) {

if(left.side\_a == right.side\_a)

return true;

else

return false;

}

void Pentagon::Print() {

std::cout << "Pentagon: a = " << side\_a << std::endl;

}

**Rhombus.h:**

Изменено:

Тип переменных side\_d1, side\_d2 с size\_t в int.

**Rhombus.cpp:**

Изменёно:

Rhombus::Rhombus(int i, int j) : side\_d1(i), side\_d2(j) {

std::cout << "Rhombus created: " << side\_d1 << ", " << side\_d2 << std::endl;

}

Rhombus::Rhombus(std::istream &is) {

do {

is >> side\_d1;

is >> side\_d2;

if(side\_d1 < 0 || side\_d2 < 0) {

std::cout << "Number is < 0. Enter another numbers: ";

}

} while(side\_d1 < 0 || side\_d2 < 0);

std::cout << "Rhombus created!" << std::endl;

}

void Rhombus::Print() {

std::cout << "Rhombus: d1 = " << side\_d1 << ", d2 = " << side\_d2 << std::endl;

}

**TArray.h:**

Добавлено:

#include "Figure.h"

#include <memory>

void Print(int step);

Изменено:

void Add(std::shared\_ptr<Figure> figure);

std::shared\_ptr<Figure> Remove(int i);

std::shared\_ptr<Figure>\* array;

**TArray.cpp:**

Добавлено:

void TArray::Print(int step) {

array[step]->Print();

return;

}

Изменёно:

void TArray::Add(std::shared\_ptr<Figure> figure) {

if(amount == 0) {

array = (std::shared\_ptr<Figure>\*) malloc((++amount) \* sizeof(std::shared\_ptr<Figure>));

array[0] = figure;

return;

}

else {

array = (std::shared\_ptr<Figure>\*) realloc(array, (++amount) \* sizeof(std::shared\_ptr<Figure>));

array[amount - 1] = figure;

return;

}

}

std::shared\_ptr<Figure> TArray::Remove(int i) {

std::shared\_ptr<Figure> tmp;

tmp = array[i - 1];

array[i - 1] = array[amount - 1];

array[amount - 1] = nullptr;

array = (std::shared\_ptr<Figure>\*) realloc(array, (--amount) \* sizeof(std::shared\_ptr<Figure>));

return tmp;

}

**main.cpp:**

#include <cstdlib>

#include "Rhombus.h"

#include "Pentagon.h"

#include "Hexagon.h"

#include "TArray.h"

std::shared\_ptr<Figure> Choise() {

std::shared\_ptr<Figure> ptr;

char choice;

std::cout << "==================\nChoose the Figure:\n1. Hexagon\n2. Pentagon\n3. Rhombus\ne. Exit" << std::endl;

for(;;) {

std::cout << "> ";

std::cin >> choice;

switch(choice) {

case '1':

std::cout << "Enter the side of Hexagon: ";

ptr = std::make\_shared<Hexagon> (Hexagon(std::cin));

return ptr;

case '2':

std::cout << "Enter the side of Pentagon: ";

ptr = std::make\_shared<Pentagon> (Pentagon(std::cin));

return ptr;

case '3':

std::cout << "Enter the diagonals of Rhombus: ";

ptr = std::make\_shared<Rhombus> (Rhombus(std::cin));

return ptr;

case 'e':

return nullptr;

}

}

}

int main(int argc, char\*\* argv) {

char choice;

TArray array;

std::shared\_ptr<Figure> tmp = NULL;

for(;;) {

std::cout << "==================\nChoose the Function:\n1. Add\n2. Delete\n3. Print\ne. Close" << std::endl;

std::cout << "> ";

std::cin >> choice;

switch(choice) {

case '1':

array.Add(Choise());

break;

case '2':

int i;

std::cout << "Enter the number of element: ";

std::cin >> i;

tmp = array.Remove(i);

tmp = nullptr;

break;

case '3':

std::cout << "==================" << std::endl;

std::cout << array;

break;

case 'e':

std::cout << "Exiting the program" << std::endl;

return 0;

case 'h':

std::cout << "=\nChoose the Function:\n1. Hexagon\n2. Pentagon\n3. Rhombus\n4. Use Stack for rhombus" << std::endl;

break;

default:

std::cout << "Error! For help type 'h'!" << std::endl;

break;

}

}

return 0;

}

**Makefile:**

all: main.exe

rm \*.o

main.exe: main.o Rhombus.o Hexagon.o Pentagon.o TArray.o

g++ -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -g main.o TArray.o Rhombus.o -o main.exe

main.o: main.cpp

g++ -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -c -g main.cpp

Hexagon.o: Hexagon.cpp

g++ -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -c -g Hexagon.cpp

Pentagon.o: Pentagon.cpp

g++ -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -c -g Pentagon.cpp

Rhombus.o: Rhombus.cpp

g++ -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -c -g Rhombus.cpp

TArray.o: TArray.cpp

g++ -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -c -g TArray.cpp

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 1 1 1 1 2 3 1 3 2 5 3 2 2 3 e | 1 3 2 5 1 2 5 |
| 1 1 2 1 1 1 1 3 2 5 3 e | 2 1 2 5 |
| 1 2 4 2 1 e | Ничего |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №4.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Гринин В.В. |
| Группа: | 8О-208Б, №1 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №5 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Знакомство с шаблонами классов; построение шаблонов динамических структур данных.

## Задание:

Необходимо спроектировать и запрограммировать на языке C++ шаблон класса-контейнера [первого уровня](#_Варианты_задания_(структуры), содержащий все три фигуры класса фигуры, согласно [вариантов задания](#_Варианты_задания_(фигуры)) (реализованную в [ЛР1](#_Лабораторная_работа_№1)).

Классы должны удовлетворять следующим правилам:

Требования к классам фигуры аналогичны требованиям из [лабораторной работы 1](#_Задание);

Шаблон класса-контейнера должен содержать 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 (определяется структурой контейнера);

Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера (определяется 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.

Вариант:

Фигура 1: ромб

Фигура 2: 5-угольник

Фигура 3: 6-угольник

Контейнер первого уровня: массив

## Код программы:

Файлы, не претерпевшие изменений: Figure.h, Hexagon.h, Hexagon.cpp, Pentagon.h, Pentagon.cpp, Rhombus.h, Rhombus.cpp, Makefile

**TArray.h:**

Класс TArray теперь использует шаблоны.

**TArray.cpp:**

Реализации также в соответствии с header’ом используют шаблоны.

Добавлено:

#include "Figure.h"

template class TArray<Figure>;

template std::ostream& operator<<(std::ostream& os, TArray<Figure>& array);

**main.cpp:**

Изменена строка:

TArray<Figure> array;

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 1 1 1 1 2 3 1 3 2 5 3 e | 1. Hexagon: a = 1  2. Pentagon: a = 3  3. Rhombus: d1 = 2, d2 = 5 |
| 1 1 1 1 2 3 1 3 2 5 3 2 1 2 1 2 1 3 e | Ничего |
| 1 3 2 6 1 1 2 3 2 2 3 2 1 3 e | 1. Rhombus: d1 = 2, d2 = 6  2. Hexagon: a = 2  1. Rhombus: d1 = 2, d2 = 6  Ничего |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №5.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Гринин В. В. |
| Группа: | 8О-208Б, №1 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №5 |
| Оценка: |  |
| Дата: |  |

Москва 2018

Цель работы:

Закрепление навыков работы с шаблонами классов; построение итераторов для динамических структур данных.

## Задание:

Используя структуры данных, разработанные для предыдущей лабораторной работы ([ЛР№4](#_Лабораторная_работа_№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](#_Варианты_задания_(фигуры)_1). Итератор должен позволять использовать структуру данных в операторах типа for.

Нельзя использовать:

Стандартные контейнеры std.

Программа должна позволять:

Вводить произвольное количество фигур и добавлять их в контейнер;

Распечатывать содержимое контейнера; удалять фигуры из контейнера.

Вариант:

Фигура 1: ромб

Фигура 2: 5-угольник

Фигура 3: 6-угольник

Контейнер первого уровня: массив

## Код программы:

Файлы, не претерпевшие изменений: Hexagon.h, Hexagon.cpp, Pentagon.h, Pentagon.cpp, Rhombus.h, Rhombus.cpp

**Figure.h:**

Добавлено:

friend std::ostream& operator<<(std::ostream &os, Figure& a){

return a.Print(os);

}

**TIterator.h:**

#ifndef TITERATOR\_H

#define TITERATOR\_H

#include <memory>

#include <iostream>

template <class node, class T>

class TIterator {

public:

TIterator(std::shared\_ptr<T>\* n) {

node\_ptr = n;

}

T\* operator \* () {

return &(\*\*node\_ptr);

}

T\* operator -> () {

return &(\*\*node\_ptr);

}

void operator ++ () {

++node\_ptr;

}

TIterator operator ++ (int) {

TIterator iter(\*this);

++(\*this);

return iter;

}

bool operator == (TIterator const& i) {

return node\_ptr == i.node\_ptr;

}

bool operator != (TIterator const& i) {

return !(\*this == i);

}

private:

std::shared\_ptr<T>\* node\_ptr;

};

#endif

**TArray.h:**

Добавлено:

#include "Titerator.h"

bool Empty();

TIterator<TArrayItem<T>,T> begin();

TIterator<TArrayItem<T>,T> end();

std::shared\_ptr<T>\* GetArray();

Изменено:

Класс TArray теперь имеет подкласс TarrayItem.

**TArray.cpp:**

Добавлено:

template <class T>

std::shared\_ptr<T>\* TArray<T>::GetArray() {

if(amount != 0) {

return array;

}

else {

return nullptr;

}

}

template <class T>

TIterator<TArrayItem<T>,T> TArray<T>::begin() {

if(amount != 0) {

return TIterator<TArrayItem<T>,T>(array);

}

else {

return TIterator<TArrayItem<T>,T>(nullptr);

}

}

template <class T>

Titerator<TArrayItem<T>,T> TArray<T>::end() {

if(amount != 0) {

return TIterator<TArrayItem<T>,T>(array + amount);

}

else {

return TIterator<TArrayItem<T>,T>(nullptr);

}

}

Изменёны названия методов:

Remove → Pop

Add → Push

Класс TArray теперь имеет подкласс TarrayItem.

**TArrayItem.h:**

#ifndef TARRAYITEM\_H

#define TARRAYITEM\_H

#include "TAllocationBlock.h"

#include <memory>

template<class T> class TArrayItem {

public:

TArrayItem(const std::shared\_ptr<T>& figure);

std::shared\_ptr<TArrayItem<T>> SetNext(std::shared\_ptr<TArrayItem<T>>& next);

std::shared\_ptr<TArrayItem<T>> GetNext() const;

std::shared\_ptr<T> GetValue() const;

void Print() const;

std::ostream& Print(std::ostream& os) const;

template<class A>

friend std::ostream& operator<<(std::ostream& os, const TArrayItem<A>& obj);

virtual ~TArrayItem();

private:

std::shared\_ptr<T> item;

std::shared\_ptr<TArrayItem<T>> next;

};

#endif /\* TARRAYITEM\_H \*/

**TArrayItem.cpp:**

#include "TArrayItem.h"

#include <iostream>

template <class T>

TArrayItem<T>::TArrayItem(const std::shared\_ptr<T>& item) {

this->item = item;

this->next = nullptr;

}

template <class T>

std::shared\_ptr<T> TArrayItem<T>::GetValue() const {

return this->item;

}

template <class T>

std::shared\_ptr<TArrayItem<T>> TArrayItem<T>::GetNext() const {

return this->next;

}

template <class T>

std::shared\_ptr<TArrayItem<T>> TArrayItem<T>::SetNext(std::shared\_ptr<TArrayItem<T>>& next) {

std::shared\_ptr<TArrayItem <T>> old = this->next;

this->next = next;

return old;

}

template <class T>

void TArrayItem<T>::Print() const {

std::cout << \*(this->item) << std::endl;

return;

}

template <class T>

std::ostream& TArrayItem<T>::Print(std::ostream& os) const {

os << \*(this->item) << std::endl;

return os;

}

template <class T>

TArrayItem<T>::~TArrayItem() {

}

template <class A>

std::ostream& operator<<(std::ostream& os, const TArrayItem<A>& obj) {

os << \*obj.item << std::endl;

return os;

}

#include "Figure.h"

template class TArrayItem<Figure>;

template std::ostream& operator<<(std::ostream& os, const TArrayItem<Figure>& obj);

**main.cpp:**

Добавлено:

for(auto i : array)

std::cout << \*i << std::endl;

Изменено:

ptr = std::shared\_ptr<Pentagon> (new Pentagon(std::cin));

ptr = std::shared\_ptr<Hexagon>(new Hexagon(std::cin));

ptr = std::shared\_ptr<Rhombus> (new Rhombus(std::cin));

Убрано:

std::cout << array;

**Makefile:**

Добавлено:

TIterator.o:

$(CC) $(CFLAGS) TIterator.h

Изменено:

main.exe: main.o Rhombus.o Hexagon.o Pentagon.o TArray.o Titerator.o

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 1 2 62 1 2 12 1 1 86 1 3 32 65 1 3 221 643 4 e | 1. Pentagon: a = 62  2. Pentagon: a = 12  3. Hexagon: a = 86  4. Rhombus: d1 = 32, d2 = 65  5. Rhombus: d1 = 221, d2 = 643 |
| 1 1 2 1 1 1 1 3 2 5 3 e | 1. Hexagon: a = 2  2. Hexagon: a = 1  3. Rhombus: d1 = 2, d2 = 5 |
| 1 1 1 1 1 3 1 3 2 5 3 2 2 3 e | 1. Hexagon: a = 1  2. Hexagon: a = 3  3. Pentagon: a = 5  1. Hexagon: a = 1 |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №6.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Гринин В.В. |
| Группа: | 8О-208Б, №1 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №5 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Закрепление навыков по работе с памятью в C++.

Создание аллокаторов памяти для динамических структур данных.

## Задание:

Используя структуры данных, разработанные для предыдущей лабораторной работы ([ЛР№5](#_Лабораторная_работа_№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](#_Варианты_задания_(структуры_1)). 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.

Вариант:

Фигура 1: ромб

Фигура 2: 5-угольник

Фигура 3: 6-угольник

Контейнер первого уровня: массив

Контейнер второго уровня: очередь

## Код программы:

Файлы, не претерпевшие изменений: Hexagon.h, Hexagon.cpp, Pentagon.h, Pentagon.cpp, Rhombus.h, Rhombus.cpp, TIterator.h

**TArray.h:**

Добавлено:

unsigned int current;

unsigned int amount;

**TArray.cpp:**

Изменено:

#include "TArray.h"

#include <memory>

#include <iostream>

#include <cmath>

template <class T>

TArray<T>::TArray() {

amount = 0;

current = 0;

array = NULL;

}

template <class T>

bool TArray<T>::Empty() {

return amount == 0;

}

template <class T>

bool TArray<T>::Push(std::shared\_ptr<T> &&figure) {

std::shared\_ptr<TArrayItem<T>> other(new TArrayItem<T>(figure));

if(figure == nullptr) {

return true;

}

if(amount == 0) {

array = (std::shared\_ptr<TArrayItem<T>>\*) malloc(sizeof(std::shared\_ptr<T>));

amount = 1;

current = 0;

array[current] = other;

return true;

}

else {

if(current == amount - 1) {

amount \*= 2;

array = (std::shared\_ptr<TArrayItem<T>>\*) realloc(array, amount\*sizeof(std::shared\_ptr<T>));

if(array == nullptr) {

puts("Error in realloc!");

return false;

}

std::cout << "Size increased to " << amount << "!" << std::endl;

}

array[current]->SetNext(other);

++current;

array[current] = other;

return true;

}

}

template <class T>

std::shared\_ptr<T> TArray<T>::Pop(int i) {

std::shared\_ptr<TArrayItem<T>> tmp;

if(i > 1 && i < current) {

array[i - 2]->SetNext(array[i]);

}

else if(i >= current) {

std::cout << "Warning: Index out of array borders" << std::endl;

return nullptr;

}

for(int step = i - 1; step < current; ++step) {

tmp = array[step];

array[step] = array[step + 1];

array[step + 1] = tmp;

}

tmp = array[current];

array[current] = nullptr;

--current;

if(current == (amount / 2) - 1) {

amount /= 2;

array = (std::shared\_ptr<TArrayItem<T>>\*) realloc(array, amount \* sizeof(std::shared\_ptr<T>));

std::cout << "Size decreased to " << amount << "!" << std::endl;

}

return tmp->GetValue();

}

template <class A>

std::ostream& operator<<(std::ostream& os, TArray<A>& obj) {

for(int step = 0; step < obj.amount; step++) {

os << step + 1 << ". " << \*obj.array[step];

}

return os;

}

**TArrayItem.h:**

Добавлено:

void\* operator new (size\_t size);

void operator delete(void \*p);

static TAllocationBlock arrayitem\_allocator;

**TArrayItem.cpp:**

Добавлено:

template <class T>

void\* TArrayItem<T>::operator new (size\_t size) {

return arrayitem\_allocator.allocate();

}

template <class T>

void TArrayItem<T>::operator delete(void \*p) {

arrayitem\_allocator.deallocate(p);

}

template <class T>

TAllocationBlock TarrayItem<T>::arrayitem\_allocator(sizeof(TArrayItem<T>),100);

**TAllocation.h:**

#ifndef TALLOCATIONBLOCK\_H

#define TALLOCATIONBLOCK\_H

#include "TQueue.h"

#include <cstdlib>

class TAllocationBlock {

public:

TAllocationBlock(size\_t size, size\_t count);

void\* allocate();

void deallocate(void \*pointer);

bool has\_free\_blocks();

virtual ~TAllocationBlock();

private:

size\_t \_size;

size\_t \_count;

char\* \_used\_blocks;

void\*\* \_free\_blocks;

TQueue<char> \_queue\_free\_blocks;

size\_t \_free\_count;

};

#endif /\* TALLOCATIONBLOCK\_H \*/

**TAllocation.cpp:**

#include "TAllocationBlock.h"

#include <memory>

#include <iostream>

TAllocationBlock::TAllocationBlock(size\_t size,size\_t count): \_size(size),\_count(count) {

\_used\_blocks = (char\*)malloc(\_size\*\_count);

for(size\_t i = 0; i < \_count; ++i) {

\_queue\_free\_blocks.push(std::shared\_ptr<char>(\_used\_blocks + (i\*\_size)));

}

\_free\_count = \_count;

std::cout << "TAllocationBlock: Memory init" << std::endl;

}

void\* TAllocationBlock::allocate() {

void\* result = nullptr;

if(\_free\_count > 0) {

result = (\_queue\_free\_blocks.pop()).get();

--\_free\_count;

std::cout << "TAllocationBlock: Allocate " << (\_count - \_free\_count) << " of " << \_count << std::endl;

}

else {

std::cout << "TAllocationBlock: No memory exception :-)" << std::endl;

}

return result;

}

void TAllocationBlock::deallocate(void\* pointer) {

std::cout << "TAllocationBlock: Deallocate block "<< std::endl;

\_queue\_free\_blocks.push(std::shared\_ptr<char>((char\*)pointer));

\_free\_count++;

}

bool TAllocationBlock::has\_free\_blocks() {

return \_free\_count > 0;

}

TAllocationBlock::~TAllocationBlock() {

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;

}

**TQueueSpec.h:**

#ifndef TQUEUESPEC\_H

#define TQUEUESPEC\_H

#include "TQueueItemSpec.h"

#include <memory>

class TQueueSpec {

public:

TQueueSpec();

void push(char\* item);

bool empty();

char\* pop();

virtual ~TQueueSpec();

private:

size\_t size;

std::shared\_ptr<TQueueItemSpec> tail;

std::shared\_ptr<TQueueItemSpec> head;

};

#endif

**TQueueSpec.cpp:**

#include "TQueueSpec.h"

TQueueSpec::TQueueSpec() {

head = nullptr;

tail = nullptr;

size = 0;

}

void TQueueSpec::push(char\* item) {

std::shared\_ptr<TQueueItemSpec> other(new TQueueItemSpec(item));

if(tail == nullptr) {

head = other;

tail = other;

size = 1;

}

else {

tail->SetNext(other);

tail = tail->GetNext();

++size;

}

}

bool TQueueSpec::empty() {

return head == nullptr;

}

char\* TQueueSpec::pop() {

char\* result = nullptr;

if (head != nullptr) {

result = head->GetValue();

head = head->GetNext();

}

--size;

return result;

}

TQueueSpec::~TQueueSpec() {

head = nullptr;

}

**TQueueItemSpec.h:**

#ifndef TQUEUEITEMSPEC\_H

#define TQUEUEITEMSPEC\_H

#include <memory>

class TQueueItemSpec {

public:

TQueueItemSpec(char\* figure);

std::shared\_ptr<TQueueItemSpec> SetNext(std::shared\_ptr<TQueueItemSpec> next);

std::shared\_ptr<TQueueItemSpec> GetNext();

char\* GetValue() const;

virtual ~TQueueItemSpec();

private:

char\* item;

std::shared\_ptr<TQueueItemSpec> next;

};

#endif

**TQueueItemSpec.cpp:**

#include "TQueueItemSpec.h"

#include <iostream>

TQueueItemSpec::TQueueItemSpec(char\* item) {

this->item = item;

this->next = nullptr;

std::cout << "Stack item: created" << std::endl;

}

std::shared\_ptr<TQueueItemSpec> TQueueItemSpec::SetNext(std::shared\_ptr<TQueueItemSpec> next) {

std::shared\_ptr<TQueueItemSpec> old = this->next;

this->next = next;

return old;

}

char\* TQueueItemSpec::GetValue() const {

return this->item;

}

std::shared\_ptr<TQueueItemSpec> TQueueItemSpec::GetNext() {

return this->next;

}

TQueueItemSpec::~TQueueItemSpec() {

next = nullptr;

std::cout << "Memory get" << std::endl;

}

**main.cpp:**

Добавлено:

#include "allocation.h"

Изменено:

int number = 0;

number = 0;

for(auto i : array) {

++number;

std::cout << number << ". " << \*i << std::endl;

}

**Makefile:**

Изменены действия:

CC=g++

CFLAGS=-pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -c -g

all: main.exe

rm \*.o

main.exe: main.o Rhombus.o Hexagon.o Pentagon.o TArray.o TArrayItem.o TQueue.o TQueueItem.o TAllocationBlock.o

$(CC) Rhombus.o Hexagon.o Pentagon.o TArray.o TArrayItem.o TQueue.o TQueueItem.o TAllocationBlock.o main.o -g -o main.exe

main.o: main.cpp

$(CC) $(CFLAGS) main.cpp

Hexagon.o: Hexagon.cpp

$(CC) $(CFLAGS) Hexagon.cpp

Pentagon.o: Pentagon.cpp

$(CC) $(CFLAGS) Pentagon.cpp

Rhombus.o: Rhombus.cpp

$(CC) $(CFLAGS) Rhombus.cpp

TArray.o: TArray.cpp

$(CC) $(CFLAGS) TArray.cpp

TArrayItem.o: TArrayItem.cpp

$(CC) $(CFLAGS) TArrayItem.cpp

TQueue.o: TQueue.cpp

$(CC) $(CFLAGS) TQueue.cpp

TQueueItem.o: TQueueItem.cpp

$(CC) $(CFLAGS) TQueueItem.cpp

TAllocationBlock.o: TAllocationBlock.cpp

$(CC) $(CFLAGS) TallocationBlock.cpp

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 1 3 132 132  1 1 459  1 3 679 679  1 2 935  2 1  e | Memory slot created!  TAllocationBlock: Memory init  Rhombus created: 132, 132  Memory slot removed!  TAllocationBlock: Allocate 1 of 100  Hexagon created: 459  Memory slot removed!  TAllocationBlock: Allocate 2 of 100  Size increased to 2!  Rhombus created: 679, 679  Memory slot removed!  TAllocationBlock: Allocate 3 of 100  Size increased to 4!  Pentagon created: 935  Memory slot removed!  TAllocationBlock: Allocate 4 of 100  Rhombus deleted  TAllocationBlock: Deallocate block  Memory slot created!  Array destroyed!  TAllocationBlock: Memory freed  Memory slot removed! |
| 1 1 831  1 1 530  1 2 384  2 1  2 2  e | Memory slot created!  TAllocationBlock: Memory init  Hexagon created: 831  Memory slot removed!  TAllocationBlock: Allocate 1 of 100  Size increased to 8!  Hexagon created: 530  Memory slot removed!  TAllocationBlock: Allocate 2 of 100  Pentagon created: 384  Memory slot removed!  TAllocationBlock: Allocate 3 of 100  Rhombus created: 589, 589  Memory slot removed!  Memory slot created!  Hexagon deleted  TAllocationBlock: Deallocate block  Memory slot created!  Pentagon deleted  TAllocationBlock: Deallocate block  Memory slot created!  Array destroyed!  TAllocationBlock: Memory freed  Memory slot removed! |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №7.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Гринин В.В. |
| Группа: | 8О-208Б, №1 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №5 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Создание сложных динамических структур данных; закрепление принципа OCP.

## 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](#_Варианты_задания_(структуры_2)). Каждым элементом контейнера, в свою, является динамической структурой данных одного из следующих видов ([Контейнер 2-го уровня](#_Варианты_задания_(структуры_3)). Таким образом у нас получается контейнер в контейнере. Элементом второго контейнера является объект-фигура, [определенная вариантом задания](#_Варианты_задания_(фигуры)_2). При этом должно выполняться правило, что количество объектов в контейнере второго уровня не больше 5. Т.е. если нужно хранить больше 5 объектов, то создается еще один контейнер второго уровня. Объекты в контейнерах второго уровня должны быть отсортированы по возрастанию площади объекта (в том числе и для деревьев). При удалении объектов должно выполняться правило, что контейнер второго уровня не должен быть пустым. Т.е. если он становится пустым, то он должен удалится.

Нельзя использовать:

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 (типу и площади).

Вариант:

Фигура 1: ромб

Фигура 2: 5-угольник

Фигура 3: 6-угольник

Контейнер первого уровня: массив

Контейнер второго уровня: очередь

## Код программы:

Файлы, не претерпевшие изменений: Figure.h, Hexagon.h, Hexagon.cpp, Pentagon.h, Pentagon.cpp, Rhombus.h, Rhombus.cpp, Allocation.h, Allocation.cpp, TQueueSpec.h, TQueueItemSpec.cpp, main.cpp

**TArray.h:**

Изменено:

Контейнер адаптирован под хранение очередей.

Убрано:

Временно закомментированы итераторы

**TArray.cpp:**

Изменено:

Удаление и вставка адаптированы под условия лабораторной работы.

template <class T>

bool TArray<T>::Push(std::shared\_ptr<T> &&figure) {

std::shared\_ptr<TQueue<T>> tmp;

if(figure == nullptr) {

return true;

}

if(amount == 0) {

++amount;

array = (std::shared\_ptr<TArrayItem<TQueue<T>>>\*) malloc(sizeof(std::shared\_ptr<TArrayItem<TQueue<T>>>));

std::shared\_ptr<TArrayItem<TQueue<T>>> smth(new TArrayItem<TQueue<T>>());

array[current] = smth;

array[current]->GetValue()->push(figure);

return true;

}

else {

if(current == amount - 1) {

amount \*= 2;

array = (std::shared\_ptr<TArrayItem<TQueue<T>>>\*) realloc(array, amount\*sizeof(std::shared\_ptr<TArrayItem<TQueue<T>>>));

if(array == nullptr) {

puts("Error in realloc!");

return false;

}

std::cout << "Size increased to " << amount << "!" << std::endl;

}

int step = 0;

while(array[step]->GetValue()->GetSize() == 5) {

++step;

}

array[step]->GetValue()->push(figure);

if(array[step]->GetValue()->GetSize() == 5) {

++current;

std::shared\_ptr<TArrayItem<TQueue<T>>> smth(new TArrayItem<TQueue<T>>);

array[current] = smth;

}

return true;

}

}

template <class T>

std::shared\_ptr<T> TArray<T>::Pop(int i) {

std::shared\_ptr<T> tmp = nullptr;

int step = (i - 1) / 5;

int index = (i - 1) % 5;

if(array[step]->GetValue()->GetSize() == 1) {

std::shared\_ptr<TArrayItem<TQueue<T>>> smth;

tmp = array[step]->GetValue()->pop(index);

if(!tmp) {

return nullptr;

}

if(step != 0) {

array[step - 1]->SetNext(array[step + 1]);

}

for(int i = step; i < current; ++i) {

smth = array[i + 1];

array[i + 1] = array[i];

array[i] = smth;

}

--current;

}

if(array[step]->GetValue()->GetSize() > 1) {

tmp = array[step]->GetValue()->pop(index);

}

if(current == (amount / 2) - 1) {

amount /= 2;

array = (std::shared\_ptr<TArrayItem<TQueue<T>>>\*) realloc(array, amount \* sizeof(std::shared\_ptr<TArrayItem<TQueue<T>>>));

std::cout << "Size decreased to " << amount << "!" << std::endl;

}

return tmp;

}

**TQueue.h:**

#include "TIterator.h"

#include "TQueueItem.h"

#include <memory>

#include <iostream>

template <class T>

class TQueue {

public:

TQueue();

void push(std::shared\_ptr<T> item);

bool empty();

size\_t GetSize();

//TIterator<TQueueItem<T>,T> begin();

//TIterator<TQueueItem<T>,T> end();

std::shared\_ptr<T> pop(int step);

template <class A>

friend std::ostream& operator<<(std::ostream& os,const TQueue<A>& queue);

virtual ~TQueue();

private:

size\_t size;

std::shared\_ptr<TQueueItem<T>> head;

};

#endif /\* TQUEUE\_H \*/

**TQueue.cpp:**

#include "TQueue.h"

template <class T>

TQueue<T>::TQueue() {

head = nullptr;

size = 0;

}

template <class T>

std::ostream& operator<<(std::ostream& os, const TQueue<T>& queue) {

std::shared\_ptr<TQueueItem<T>> item = queue.head;

while(item != nullptr) {

os << \*item;

item = item->GetNext();

}

return os;

}

template <class T>

void TQueue<T>::push(std::shared\_ptr<T> item) {

std::shared\_ptr<TQueueItem<T>> tmp = head;

std::shared\_ptr<TQueueItem<T>> other(new TQueueItem<T>(item));

if(head == nullptr) {

head = other;

}

else {

if(other->GetValue()->Square() <= head->GetValue()->Square()) {

head = other;

other->SetNext(tmp);

++size;

return;

}

while(other->GetValue()->Square() >= tmp->GetValue()->Square()) {

tmp = tmp->GetNext();

}

}

++size;

}

template <class T>

size\_t TQueue<T>::GetSize() {

return size;

}

template <class T>

bool TQueue<T>::empty() {

return head == nullptr;

}

template <class T>

std::shared\_ptr<T> TQueue<T>::pop(int index) {

std::shared\_ptr<T> result = nullptr;

std::shared\_ptr<TQueueItem<T>> tmp = head;

if(index == 0) {

result = head->GetValue();

head = head->GetNext();

}

else if(index == size - 1) {

for(int i = 1; i < index; ++i)

tmp = tmp->GetNext();

result = tmp->GetNext()->GetValue();

tmp->SetNext(nullptr);

}

else if(index >= 0 && index < size) {

for(int i = 1; i < index; ++i) {

tmp = tmp->GetNext();

}

result = tmp->GetNext()->GetValue();

tmp->SetNext(tmp->GetNext()->GetNext());

}

else {

std::cout << "Out of queue borders" << std::endl;

return nullptr;

}

--size;

return result;

}

/\*

template <class T>

TIterator<TQueueItem<T>,T> TQueue<T>::begin() {

return TIterator<TQueueItem<T>,T>(head);

}

template <class T>

TIterator<TQueueItem<T>,T> TQueue<T>::end() {

return TIterator<TQueueItem<T>,T>(nullptr);

}

\*/

template <class T>

TQueue<T>::~TQueue() {

head = nullptr;

}

#include "Figure.h"

template class TQueue<Figure>;

template std::ostream& operator<<(std::ostream& os, const TQueue<Figure>& obj);

**TQueueItem.h:**

#ifndef TQUEUEITEM\_H

#define TQUEUEITEM\_H

#include <memory>

template<class T>

class TQueueItem {

public:

TQueueItem(const std::shared\_ptr<T>& figure);

template<class A>

friend std::ostream& operator<<(std::ostream& os, const TQueueItem<A>& obj);

std::shared\_ptr<TQueueItem<T>> SetNext(std::shared\_ptr<TQueueItem> next);

std::shared\_ptr<TQueueItem<T>> GetNext();

std::shared\_ptr<T> GetValue() const;

virtual ~TQueueItem();

private:

std::shared\_ptr<T> item;

std::shared\_ptr<TQueueItem<T>> next;

};

#endif /\* TQUEUEITEM\_H \*/

**TQueueItem.cpp:**

#include "TQueueItem.h"

#include <iostream>

template <class T>

TQueueItem<T>::TQueueItem(const std::shared\_ptr<T>& item) {

this->item = item;

this->next = nullptr;

std::cout << "Stack item: created" << std::endl;

}

template <class T>

std::shared\_ptr<TQueueItem<T>> TQueueItem<T>::SetNext(std::shared\_ptr<TQueueItem<T>> next) {

std::shared\_ptr<TQueueItem <T>> old = this->next;

this->next = next;

return old;

}

template <class T>

std::shared\_ptr<T> TQueueItem<T>::GetValue() const {

return this->item;

}

template <class T>

std::shared\_ptr<TQueueItem<T>> TQueueItem<T>::GetNext() {

return this->next;

}

template <class T>

TQueueItem<T>::~TQueueItem() {

next = nullptr;

std::cout << "Memory get" << std::endl;

}

template <class A>

std::ostream& operator<<(std::ostream& os, const TQueueItem<A>& obj) {

os << \*obj.item << std::endl;

return os;

}

#include "Figure.h"

template class TQueueItem<Figure>;

template std::ostream& operator<<(std::ostream& os, const TQueueItem<Figure>& obj);

**Makefile:**

Добавлено:

TQueueSpec.o: TQueueSpec.cpp

$(CC) $(CFLAGS) TQueueSpec.cpp

TQueueItemSpec.o: TQueueItemSpec.cpp

$(CC) $(CFLAGS) TQueueItemSpec.cpp

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Результат |
| 1 1 2 1 2 4 1 3 2 5 1 2 7 1 1 8 1 2 23 3 e | 1:  Hexagon: a = 2  Pentagon: a = 4  Rhombus: d1 = 2 d2 = 5  Pentagon: a = 7  Hexagon: a = 8  2:  Pentagon: a = 23 |
| 1 3 6 7 1 1 4 1 3 2 5 1 2 7 1 1 8 1 2 23 2 6 3 e | 1:  Rhombus: d1 = 6 d2 = 7  Hexagon: a = 4  Rhombus: d1 = 2 d2 = 5  Pentagon: a = 7  Hexagon: a = 8 |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №8.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Гринин В.В. |
| Группа: | 8О-208Б, №1 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №5 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Знакомство с параллельным программированием в C++.

## Задание:

Используя структуры данных, разработанные для [лабораторной работы №6](#_Лабораторная_работа_№6_1) (контейнер первого уровня и классы-фигуры) разработать алгоритм быстрой сортировки для класса-контейнера.

Необходимо разработать два вида алгоритма:

1) Обычный, без параллельных вызовов.

2) 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;

Вариант:

Фигура 1: ромб

Фигура 2: 5-угольник

Фигура 3: 6-угольник

Контейнер первого уровня: массив

Контейнер второго уровня: очередь

## Код программы:

Изменения не затронули файлы: Figure.h, Hexagon.h, Hexagon.cpp, Pentagon.h, Pentagon.cpp, Rhombus.h, Rhombus.cpp, Allocation.h, Allocation.cpp, TQueueSpec.h, TQueueItemSpec.cpp, TQueue.h, TQueueItem.cpp.

**main.cpp:**

Добавлено:

#include <future>

#include <random>

#include <thread>

void ParallelSort(TArray<Figure>\* array) {

if (array->Size() > 1) {

std::shared\_ptr<Figure> middle = array->Pop(0);

TArray<Figure> left, right;

while (!array->Empty()) {

std::shared\_ptr<Figure> item = array->Pop(0);

if (\*item < \*middle) {

left.Push(item);

} else {

right.Push(item);

}

}

std::future<void> left\_res = std::async(std::launch::async, ParallelSort, &left);

std::future<void> right\_res = std::async(std::launch::async, ParallelSort, &right);

left\_res.get();

while (!left.Empty()) {

array->Push(left.Pop(0));

}

array->Push(middle);

right\_res.get();

while (!right.Empty()) {

array->Push(right.Pop(0));

}

}

return;

}

std::cout << "Parallel sort? (1 or 0): ";

std::cin >> i;

if(i == 1) {

ParallelSort(&array);

}

else if (i == 0) {

array.Sort();

}

else {

std::cout << "Error: bad format!" << std::endl;

}

break;

Изменено:

Вместо самостоятельного добавления элементов стоит генератор 20 разных элементов:

//\*

for (int i = 0; i < 20; i++) {

side1 = distribution(generator);

side2 = distribution(generator);

fig = param(generator);

switch(fig) {

case 1:

array.Push(std::shared\_ptr<Hexagon> (new Hexagon(side1)));

break;

case 2:

array.Push(std::shared\_ptr<Pentagon> (new Pentagon(side2)));

break;

case 3:

array.Push(std::shared\_ptr<Rhombus> (new Rhombus(side1, side2)));

break;

}

}//\*/

//(array.Push(Choise())) ? puts("Figure succesfully added in array!") : puts("Failed!");

**TArray.h:**

Добавлено:

void Sort();

**TArray.cpp:**

Добавлено:

template <class T>

void TArray<T>::Sort() {

int l, r;

for(l = 0; l < current - 1; ++l) {

r = l + 1;

while(r > 1 && (\*array[r - 1]) > (\*array[r])) {

std::shared\_ptr<TArrayItem<T>> tmp = array[r - 1];

array[r - 1] = array[r];

array[r] = tmp;

--r;

}

}

return;

}

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 1 3 1 4 e | 1. Rhombus: d1 = 1, d2 = 132  2. Hexagon: a = 48  3. Pentagon: a = 67  4. Rhombus: d1 = 48, d2 = 679  5. Rhombus: d1 = 73, d2 = 632  6. Rhombus: d1 = 366, d2 = 248  7. Pentagon: a = 167  8. Rhombus: d1 = 273, d2 = 437  9. Rhombus: d1 = 654, d2 = 416  10. Rhombus: d1 = 687, d2 = 589  11. Rhombus: d1 = 633, d2 = 757  12. Pentagon: a = 384  13. Hexagon: a = 459  14. Hexagon: a = 478  15. Hexagon: a = 530  16. Pentagon: a = 754  17. Hexagon: a = 831  18. Hexagon: a = 847  19. Hexagon: a = 898  20. Hexagon: a = 911 |
| 1 3 0 4 e | 1. Rhombus: d1 = 1, d2 = 12  2. Hexagon: a = 52  3. Pentagon: a = 79  4. Rhombus: d1 = 48, d2 = 679  5. Rhombus: d1 = 73, d2 = 632  6. Rhombus: d1 = 366, d2 = 248  7. Pentagon: a = 167  8. Rhombus: d1 = 273, d2 = 437  9. Rhombus: d1 = 654, d2 = 416  10. Rhombus: d1 = 687, d2 = 589  11. Rhombus: d1 = 633, d2 = 757  12. Pentagon: a = 384  13. Hexagon: a = 447  14. Hexagon: a = 470  15. Hexagon: a = 502  16. Pentagon: a = 751  17. Hexagon: a = 831  18. Hexagon: a = 835  19. Hexagon: a = 842  20. Hexagon: a = 992 |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №9.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Гринин В.В. |
| Группа: | 8О-208Б, №1 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №5 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Знакомство с лямбда-выражениями

## 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 контенйром 1-го уровня:
  + Генерация фигур со случайным значением параметров;
  + Печать контейнера на экран;
  + Удаление элементов со значением площади меньше определенного числа;
* В контенер 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.

Программа должна позволять:

Вводить произвольное количество фигур и добавлять их в контейнер;

Распечатывать содержимое контейнера;

Удалять фигуры из контейнера;

Проводить сортировку контейнера;

Вариант:

Фигура 1: ромб

Фигура 2: 5-угольник

Фигура 3: 6-угольник

Контейнер первого уровня: массив

Контейнер второго уровня: очередь

## Код программы:

Изменения не затронули все файлы, кроме main.cpp

**main.cpp:**

Добавлено:

typedef std::function<void(void)> command;

TQueue<command> queue;

command cmd\_insert = [&]() {

std::cout << "Command: Create triangles" << std::endl;

std::default\_random\_engine generator;

std::uniform\_int\_distribution<int> distribution(1, 1000);

std::uniform\_int\_distribution<int> param(1, 3);

for (int i = 0; i < 10; i++) {

int side1 = distribution(generator);

int side2 = distribution(generator);

int fig = param(generator);

switch(fig) {

case 1:

array.Push(std::shared\_ptr<Hexagon> (new Hexagon(side1)));

break;

case 2:

array.Push(std::shared\_ptr<Pentagon> (new Pentagon(side1)));

break;

case 3:

array.Push(std::shared\_ptr<Rhombus> (new Rhombus(side2, side2)));

break;

}

}

};

command cmd\_print = [&]() {

std::cout << "Command: Print array" << std::endl;

std::cout << array;

};

command cmd\_reverse = [&]() {

std::cout << "Command: Reverse array" << std::endl;

array.Reverse();

};

queue.push(&cmd\_print);

queue.push(&cmd\_reverse);

queue.push(&cmd\_print);

queue.push(&cmd\_insert);

if(!queue.empty()) {

command\* cmd = queue.pop();

std::future<void> ft = std::async(\*cmd);

ft.get();

//std::thread(\*cmd).detach();

}

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 3 3 3 3 e | Command: Print array  Command: Reverse array  Command: Print Array  Command: Create Figures  Rhombus created: 132, 132  Memory slot removed!  TAllocationBlock: Allocate 1 of 100  Hexagon created: 459  Memory slot removed!  TAllocationBlock: Allocate 2 of 100  Size increased to 2!  Rhombus created: 679, 679  Memory slot removed!  TAllocationBlock: Allocate 3 of 100  Size increased to 4!  Pentagon created: 935  Memory slot removed!  TAllocationBlock: Allocate 4 of 100  Hexagon created: 831  Memory slot removed!  TAllocationBlock: Allocate 5 of 100  Size increased to 8!  Hexagon created: 530  Memory slot removed!  TAllocationBlock: Allocate 6 of 100  Pentagon created: 384  Memory slot removed!  TAllocationBlock: Allocate 7 of 100  Rhombus created: 589, 589  Memory slot removed!  TAllocationBlock: Allocate 8 of 100  Hexagon created: 847  Memory slot removed!  TAllocationBlock: Allocate 9 of 100  Size increased to 16!  Rhombus created: 416, 416  Memory slot removed!  TAllocationBlock: Allocate 10 of 100 |
| 3 3 3 3 3 3 4 e | 1. Rhombus: d1 = 132, d2 = 132  2. Hexagon: a = 459  3. Rhombus: d1 = 679, d2 = 679  4. Pentagon: a = 935  5. Hexagon: a = 831  6. Hexagon: a = 530  7. Pentagon: a = 384  8. Rhombus: d1 = 589, d2 = 589  9. Hexagon: a = 847  10. Rhombus: d1 = 416, d2 = 416 |
| 3 3 3 3 1 2 5 4 2 2 2 2 2 2 2 2 2 2 4 | 1. Rhombus: d1 = 132, d2 = 132  2. Pentagon: a = 384  3. Rhombus: d1 = 589, d2 = 589  4. Hexagon: a = 847  5. Rhombus: d1 = 416, d2 = 416  6. Pentagon: a = 5 |

Заключение:

1) Классы в С++ являются замечательной абстракцией. Мы можем создавать какой-либо класс объектов, к примеру класс треугольников, прописывать там необходимые свойства. И уже в случае необходимости можем создавать уже объекты этого класса и взаимодействовать с ними. Так же хороший инструмент — уровни доступа к методам и переменным. А именно инкапсуляция — одно из свойств классов в С++. Позволяет в явном виде указать, что лучше не стоит выставлять на показ, а что можно выставить как интерфейс, с которым в последствии можно работать. Из всего этого заметил недостаток. Даже если нам нужен какой-то один объект с какими-либо свойствами — нам нужно всё равно прописывать отдельный класс.

2) Объекты разных классов в зависимости от реализации могут выполнять разные роли. В случае с данной лабораторной работой — есть роль контейнера и есть роль какого-либо абстрактного объекта с определёнными свойствами. С++ неплохо подходит для проектирования разной сложности структур. Самое главное, чтобы программист смог всё правильно разобрать на составные части и правильно выстроить между этими частями связи. Иначе есть риск, что можно элементарно запутаться.

3) В этой лабораторной работе мы познакомились с ещё одним свойством классов — наследование. Это свойство позволяет нам создавать супер-классы, от которых наследуются дочерние классы. Зачем это нужно? Ну хорошим примером является данная лабораторная работа, где у нас имелся контейнер, который может хранить фигуры разных классов, но при этом наследующие общий супер-класс. Благодаря этому мы можем при необходимости обобщать какие-либо свойства объектов, тем самым создавая супер-класс.

4) Шаблоны — хороший, но не очень приятный и понятный инструмент в С++. Концепция максимально простая — мы можем создавать шаблон какого-либо класса и потом в при необходимости вставлять в этот шаблон другие необходимые классы. То есть мы можем при необходимости создать один шаблон какого-либо контейнера, и уже при необходимости генерировать контейнеры для определённых типов данных, например, для int и double. При этом нам не придётся писать отдельные классы контейнеров специально для int и double.

5) Итераторы — не очень приятная для реализации штука. Идея итераторов проста — они позволяют проходить по какой-либо динамической структуре, при этом не вникая в особенности этой самой структуры. Поэтому, когда мы имеем дело с незнакомой нам структурой и нам повезло, что для этой структуры есть итератор, то можем использовать итератор для своих целей, при этом не зная вообще, что там происходит. Хорошая абстракция.

6) Аллокаторы — это прикольная штука. Она позволяет нам создавать свои ***new*** и ***delete,*** которые будут лучше подходить для какой-либо структуры данных. Стандартные ***new*** и ***delete*** сконструированы специально так, чтобы удовлетворять потребностям как можно более широкому кругу разработчиков. Однако иногда, когда мы имеем дело со сложной или, наоборот, слишком простой структурой, они по скорости могут проигрывать, что будет плохо сказываться на производительности программы. Именно для этого нам позволяют создавать свои аллокаторы.

7) Вот эта лабораторная может оказаться лютым кошмаром для студента, если ему не повезёт столкнуться с вариантом, где у него будет бинарное дерево внутри другого бинарного дерева. Мне к счастью повезло и я не столкнулся с этим, но боль можно почувствовать уже прочитав такое. В процессе работы с этой лабораторной пришлось переделать контейнер первого уровня. Зачастую происходила путаница, и я порой вообще переставал понимать, что у меня происходит в коде. Но посыл этой лабораторной такой — мы можем создавать разного рода сложные структуры, которые подобно матрёшке содержат в себе другие сложные структуры, прибегая к не сильно большим изменениям в плане переписывания кода.

8) Многопоточное программирование в С++ вышло на иной уровень. Уже достаточно посмотреть то количество классов, что применяются для генерации потоков и их менеджмента. Если сравнивать с обычным POSIX, который также предоставляет возможность для создания потоков, то в случае с С++ мы получаю красивую обёртку, которая позволяет нам более безопасно и красиво работать с потоками. Стоит отметить любимые мьютексы, семафоры, условные переменные, которые помогают ограничить доступ к какому-либо критическому участку кода нескольким потокам, чтобы там работал только один поток (ну или несколько потоков, если код позволяет).

9) Лямба-выражения оказались тёмной лошадкой. Их идея в том, что мы можем внутри функции с генерировать что-то вроде скрипта или безымянной функции, и эту функцию уже где-то применять. На мой взгляд, данная лабораторная работа не позволила реализовать одну фишку лямба-выражений — мы можем передать функции выражение при помощи обычного указателя. Таким образом одна функция в зависимости от полученного указателя будет делать разные вещи. Конечно, мы там передавали этот же указатель при создании нового потока, но это всё ещё не то. Ведь при создании потока мы можем также передать указатель и на обычную функцию.