Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Уфимский университет науки и технологий»

Кафедра ВМиК

Отчёт по лабораторной работе № 1

по дисциплине: «Объектно-ориентированное программирование»

на тему: «Кнопки и формы»

Выполнил:

Студент группы: ПРО-231б Блинов Д. В.

Проверил:

Преподаватель: Котельников В. А.

Уфа – 2022

**Цель**

Определение и реализация тестовых классов, и написание программы, иллюстрирующей их использование (код, делающий нечто, и при этом **обязательный отладочный вывод** в консоль, для контроля того, какие методы вызываются – особенно важен отладочный вывод во всех конструкторах и деструкторах).

Лабораторная работа может включать несколько программ, не стоит пытаться писать все сразу в одну программу и строго последовательно. Например, сначала стоит написать определение и реализацию простого объекта, создать его статически и динамически, проверить вызов атрибутов и методов. Затем определить конструкторы и деструкторы и проверить их работу. Затем научиться определять классы-наследники. Затем научиться переопределять методы и создавать виртуальные методы (не требуется в этой лабораторной, но вдруг?) и проверить их использование, и так далее. Настоятельно **не** рекомендуется в качестве методов использовать переопределяемые операции (++, <<, >> и т.д.).

Рекомендуемые объекты для манипуляций: геометрические фигуры (точки, линии, квадраты, круги, эллипсы), математические объекты (вектора, матрицы) и т.д.

**Не**рекомендуемые объекты для манипуляций (преподаватель плохо разбирается в этой предметной области, да и надоело): Warrior, Weapon, HP, Damage, и т.д.

* Определения
  + классов
  + атрибутов, методов
  + классов-наследников
  + конструкторов (обязательно несколько: без параметров, с параметрами, с параметром-объектом того же класса), деструкторов
  + доступности атрибутов и методов (показать, как влияют модификаторы private, protected, public)
* Реализация
  + методов объектов
  + реализация методов сразу в определении или после определения
  + конструкторов, деструкторов
* Создание и использование (обращение к атрибутам, вызов методов)
  + статически создаваемых объектов («MyClass obj;»)
  + динамически создаваемых объектов («MyClass \*obj = new MyClass();»)
  + объектов с помощью различных конструкторов (**у каждого** создаваемого объекта должны быть: конструктор без параметров, с параметрами, с параметром-объектом того же класса – конструктор копирования)
  + инициализация полей класса в списке инициализации конструктора
  + помещение объектов в переменные различных типов (объяснять, чем отличается MyBase \* obj = new MyBase() от MyBase \* obj = new MyDeriv())
  + объектов классов-наследников (проверить и продемонстрировать, какие конструкторы классов при этом вызываются)
  + композиция объектов: атрибутом одного объекта класса A является другой объект класса B (не указатель!), инициализируемый в списке инициализации конструктора класса A; показать, когда этот объект класса B удаляется
  + композиция объектов: атрибутом одного объекта класса A является указатель на другой объект класса B, создаваемый в конструкторе класса A и уничтожаемый в деструкторе класса A; композируемые классы должны определяться отдельно (не один в другом); при композиции показать, в чем разница, если объект класса А хранит прямо объект класса В или указатель на объект класса В.
* Уничтожение
  + статически созданных объектов
  + динамически созданных объектов
  + объектов классов-наследников (проверить и продемонстрировать, какие деструкторы классов при этом вызываются)

Лабораторная работа требует уверенного понимания и работы с указателями. Студенты должны четко понимать всё, что происходит при выполнении строки MyBase \* obj = new MyDeriv() (создается переменная-указатель, создается объект в памяти и возвращается его адрес, этот адрес записывается в созданную переменную-указатель).

В рамках лабораторной работы необходимо написать достаточно примеров, чтобы разобраться и понимать, например, суть того, что будет, если:

* в классе-предке есть конструктор и деструктор, а в классе-потомке нет ни того, ни того; что происходит при создании предка, при создании потомка?
* в классе-предке и в классе потомке есть и конструкторы и деструкторы: в каком порядке они вызываются при создании и удалении объекта?
* в переменную-указатель на базовый класс поместить объект-потомок: какие методы можно вызывать, какие нет?

**Ход работы**

1. Приводится отладочный вывод статического создания объектов Line по умолчанию, с одним параметром, с двумя параметрами, конструктор копирования, а также отладочный вывод метода Line, изменяющий свойство этого класса.

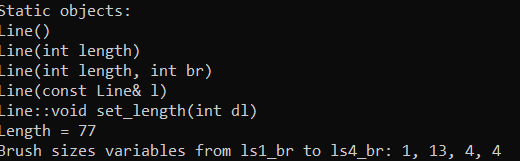


Рис 1. Статическое создание

1. Приводится отладочный вывод динамического создания объектов Line по умолчанию, с одним параметром, с двумя параметрами, конструктор копирования, а также отладочный вывод метода Line, изменяющий свойство этого класса.

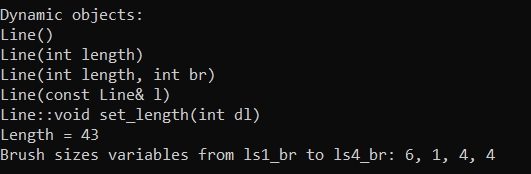


Рис 2. Динамическое создание

1. Приводится вызов методов для объектов Line: метод pull\_out, который выводит свойства length и brush\_size, а также деструкторы, которые вызываются при удалении объектов. Отладочный вывод методов и деструкторов статических объектов выглядит точно так же, как и динамических, однако есть между ними разница: статические объекты удаляются, когда программы за блок кода, в котором находится объект(фигурные скобки), а динамические вручную при помощи оператора delete.

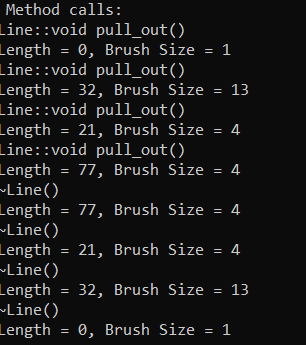


Рис 3. Вызовы методов

1. Приводятся родительский класс Car и его дочерний класс Truck. Мы создаем объект Truck в переменной класса Truck. Сначала идут отладочные выводы конструкторов: первым идет Car, так как он является базовым классом, затем конструктор дочернего класса Truck. Затем отладочный вывод метода demonstrate у Truck, который в свою очередь вызывает в коде demonstrate класса Car. Далее выводятся отладочные свойства методов и значения. Далее мы создаем объект Truck в переменной класса Car. Вызывается только конструктор Car, далее также идет вызов методов demonstrate. Деструкторы вызываются в обратном порядке. Это пример наследования

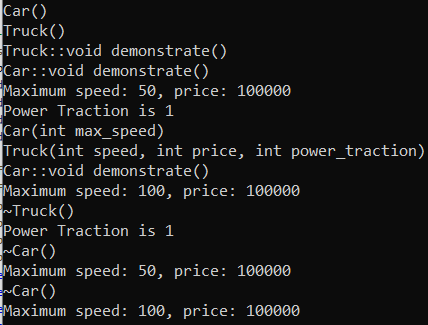


Рис 4. Реализация наследования. Отладочный вывод

1. Приводятся классы Spoon, Knife и Utensils для реализации композиции. У нас есть объект Utensils, в котором хранятся объекты Knife и Spoon. Сначала вызывается конструктор Utensils, затем по порядку объявления Knife и Spoon. После этого идут отладочные выводы методов Knife и Spoon, вызываемые через Utensils. Далее вызываются деструкторы по порядку: Spoon, Knife, Utensils.

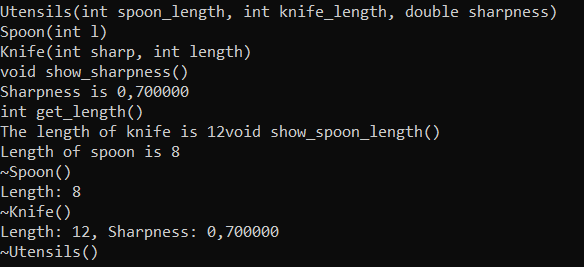


Рис 5. Реализация композиции. Отладочный вывод.

**Вывод**

На этой лабораторной работе я научился использовать простейшие основы ООП, такие как объявление класса, создание объекта динамически и статически, а также основам наследования и композиции.

Ссылка на код:

[OOP-LABA2/LABA2\_OOP.cpp at main · greenkristal1/OOP-LABA2 (github.com)](https://github.com/greenkristal1/OOP-LABA2/blob/main/LABA2_OOP/LABA2_OOP/LABA2_OOP.cpp)

**Приложение 1**

// LABA2\_OOP.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <string>

#include <stdio.h>

using namespace std;

//Part 1

class Line {//класс Line, содержащий одно приватное свойство length и публичное свойство length

private:

int length;

public:

int brush\_size;

Line() : length(0), brush\_size(1) {//конструктор по-умолчанию

printf\_s("Line()\n");

}

Line(int length) : length(length), brush\_size(1) {//конструктор с одним параметром

printf\_s("Line(int length)\n");

}

Line(int length, int br) : length(length), brush\_size(br) {//конструктор с двумя параметрами

printf\_s("Line(int length, int br)\n");

}

Line(const Line& l) : length(l.length), brush\_size(l.brush\_size) {//конструктор копирования

printf\_s("Line(const Line& l)\n");

}

~Line() {//деструктор

printf\_s("~Line()\n");

printf\_s("Length = %d, Brush Size = %d\n", length, brush\_size);

}

void pull\_out() {//метод, который выводмт значения свойств length и sharpness

printf\_s("Line::void pull\_out()\n");

printf\_s("Length = %d, Brush Size = %d\n", length, brush\_size);

}

void set\_length(int dl);//объявление метода без реализации внутри класса

};

void Line::set\_length(int dl) {//реализация метода вне класса, изменяющее длину линии

printf\_s("Line::void set\_length(int dl)\n");

if (dl > 4) {

length = dl;

}

printf\_s("Length = %d\n", length);

}

//part 2 --------------------------------------------------------------------------------------------------

class Car {//класса Car, имеющее одно приватное поле max\_speed, одно протектед поле price

private:

int max\_speed;

protected:

int price;

public:

Car() { //конструктор по-умолчанию

printf("Car()\n");

max\_speed = 50;

price = 100'000;

}

Car(int max\_speed) {//конструктор с одним параметром

printf("Car(int max\_speed)\n");

this->max\_speed = max\_speed;

this->price = 50000;

}

Car(int max\_speed, int price) : Car(max\_speed) {//конструктор с двумя параметрами

printf("Car(int max\_speed, int price)\n");

this->price = price;

}

Car(const Car& c) {//конструктор копирования

printf("Car(const Car& c)\n");

max\_speed = c.max\_speed;

price = c.price;

}

void demonstrate() {//метод, выыводящий поля max\_speed, price

printf("Car::void demonstrate()\n");

printf("Maximum speed: %d, price: %d\n", max\_speed, price);

}

~Car() {

printf("~Car()\n");

printf("Maximum speed: %d, price: %d\n", max\_speed, price);

}

};

class Truck : public Car {//класс, имеющий приватное поле power\_traction

private:

int power\_traction;

public:

Truck() : Car() {//консрукор по умолчанию, вызывающий конструктор по-умолчанию предка Car

printf("Truck()\n");

power\_traction = 1;

}

Truck(int speed, int price, int power\_traction) : Car(speed) {//конструктор с параметрами, вызывающий конструктор предка Car

printf("Truck(int speed, int price, int power\_traction)\n");

//this->speed = speed - Ошибка. Дочерний класс не может использовать свойства private. Поэтому делегируем в конструктор с параметром базового класса

this->price = price;

this->power\_traction = power\_traction;

}

Truck(const Truck& t) :Car(t) {//конструктор копирования, вызывающий конструктор копирования Car

printf("Car(const Truck& t)\n");

this->power\_traction = t.power\_traction;

}

~Truck() {//деструктор

printf("~Truck()\n");

printf("Power Traction is %d\n", power\_traction);

}

void demonstrate() {//метод, выводящий поле power\_traction

printf("Truck::void demonstrate()\n");

Car::demonstrate();

printf("Power Traction is %d\n", power\_traction);

};

};

//part 3

class Spoon {//класс, имеющий одно протектед поле length, используется для реализации композиции

protected:

int length;

public:

Spoon() {//конструкор по умолчанию

printf\_s("Spoon()\n");

length = 10;

}

Spoon(int l) {//конструктор с параметрами

printf\_s("Spoon(int l)\n");

length = l;

}

Spoon(const Spoon& s) {//конструктор копирования

printf\_s("Spoon(const Spoon& s)\n");

length = s.length;

}

~Spoon() {//деструктор

printf\_s("~Spoon()\n");

printf\_s("Length: %d\n", length);

}

void show\_spoon\_length() {//метод, выводящий свойтство length

printf\_s("void show\_spoon\_length()\n");

printf\_s("Length of spoon is %d\n", length);

}

};

class Knife {////класс, имеющий 2 протектед поля length и sharpness, используется для реализации композиции

protected:

double sharpness;

int length;

public:

Knife() {//конструктор по-умолчанию

printf("Knife()\n");

sharpness = 1;

length = 10;

}

Knife(double sharp, int length) {//конструктор с параметрами

printf("Knife(int sharp, int length)\n");

if (sharp <= 1 && sharp > 0) {

sharpness = sharp;

}

else {//если sharpness в пределах от 0 до 1, идет присваивание

printf("Incoreect sharpness. Set to 1 by default\n");

sharpness = 1;

}

this->length = length;

}

Knife(const Knife& k) {//конструктор копирования

printf("Knife(const Knife& k)\n");

sharpness = k.sharpness;

length = k.length;

}

~Knife() {//деструктор

printf("~Knife()\n");

printf("Length: %d, Sharpness: %f\n", length, sharpness);

}

void show\_sharpness() {//метод, выводящий sharpness

printf("void show\_sharpness()\n");

printf("Sharpness is %f\n", this->sharpness);

}

double get\_length() {//метод, возращающй length

printf("int get\_length()\n");

return length;

}

};

class Utensils {

public:

Spoon\* s;//реалиация композиции

Knife\* k;

Utensils() {//конструктор по-умолчанию

printf("Utensils()\n");

s = new Spoon();

k = new Knife();

}

Utensils(int spoon\_length, int knife\_length, double sharpness) {//конструктор с параметрами

printf("Utensils(int spoon\_length, int knife\_length, double sharpness)\n");

s = new Spoon(spoon\_length);

k = new Knife(sharpness, knife\_length);

}

Utensils(const Utensils& u) {//конструктор копирования

printf("Utensils(const Utensils& u)\n");

s = new Spoon(\*(u.s));

k = new Knife(\*(u.k));

}

~Utensils() {//деструктор, удаляет объекты композиции

delete s;

delete k;

printf("~Utensils()\n");

}

};

int main()

{

setlocale(LC\_ALL, "ru");

//part 1--------------------------------------------------------------------------------------------------------------------

printf\_s("Static objects: \n");

{//создание статических объектов

Line ls1;//создание без параметров

Line ls2(32);//создание с парамиетрами

Line ls3(21, 4);//создание с парамиетрами

Line ls4(ls3);//создание с помощью конструктора копирования

//ls2.length = 77 - Ошибка. Нелья обратиться к length напрямую , так как это поле не public

ls4.set\_length(77);

ls2.brush\_size = 13;

int ls1\_br = ls1.brush\_size;

int ls2\_br = ls2.brush\_size;

int ls3\_br = ls3.brush\_size;

int ls4\_br = ls4.brush\_size;

printf\_s("Brush sizes variables from ls1\_br to ls4\_br: %d, %d, %d, %d\n", ls1\_br, ls2\_br, ls3\_br, ls4\_br);

printf\_s("\n Method calls:\n");

ls1.pull\_out();

ls2.pull\_out();

ls3.pull\_out();

ls4.pull\_out();

}

printf\_s("\n\n\nDynamic objects:\n");

//динамическое создание

Line\* ld1 = new Line;//создание без параметров

Line\* ld2 = new Line(6);//создание с парамиетрами

Line\* ld3 = new Line(17, 4);//создание с парамиетрами

Line\* ld4 = new Line(\*ld3);//создание с помощью конструктора копирования

//ld3->length = 43 - Ошибка. Нелья обратиться к length напрямую , так как это поле не public

ld3->set\_length(43);

ld1->brush\_size = 6;

int ld1\_br = ld1->brush\_size;

int ld2\_br = ld2->brush\_size;

int ld3\_br = ld3->brush\_size;

int ld4\_br = ld4->brush\_size;

printf\_s("Brush sizes variables from ls1\_br to ls4\_br: %d, %d, %d, %d\n", ld1\_br, ld2\_br, ld3\_br, ld4\_br);

printf\_s("\n Method calls:\n");

ld1->pull\_out();

ld2->pull\_out();

ld3->pull\_out();

ld4->pull\_out();

delete ld1;

delete ld2;

delete ld3;

delete ld4;

printf\_s("-----------------------------------------------------------------------------------\n");

printf\_s("Часть вторая\n");

Truck\* t1 = new Truck;

t1->demonstrate();

Car\* t2 = new Truck(100, 100'000, 56);

t2->demonstrate();

delete t1;

delete t2;

printf\_s("-------------------------------------------------------------------------------------------------\n");

//part 3

Utensils\* u = new Utensils(8, 12, 0.7);

u->k->show\_sharpness();

int l = u->k->get\_length();

printf\_s("The length of knife is %d", l);

u->s->show\_spoon\_length();

delete u;

}