ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ

Кафедра ВМиК

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

по предмету **«Объектно-Ориентированное Программирование»**

Выполнил: студент группы МО-204Б

Исламов Ильнур Фандасович.

Проверил:
доцент каф. ВМиК

Котельников В.А.

Уфа 2025г.

Цель лабораторной работы

- В рамках лабораторной работы необходимо разобраться:
- каким образом работают виртуальные методы и механизм позднего связывания;
- каким образом передавать и возвращать объекты из функций без потери информации о типе;
- каким образом безопасно выполнять приведение типов во время выполнения;
- каким образом управлять временем жизни объектов с помощью умных указателей.

- **Задание**

- **Часть 1. Исследование виртуальных методов**
- **Базовые классы:**
    - Создать класс Base с виртуальными и невиртуальными методами
    - Создать класс Desc, унаследованный от Base, с переопределением методов
    - Реализовать отладочный вывод во всех конструкторах и деструкторах
- **Эксперименты с виртуальностью:**
    - Показать разницу между вызовами виртуальных и невиртуальных методов
    - Продемонстрировать необходимость виртуального деструктора
    - Исследовать вызовы методов при обращении через указатели разных типов
- **Часть 2. Идентификация типов и безопасное приведение**
- **Методы идентификации:**
    - Реализовать виртуальный метод string classname()
    - Реализовать виртуальный метод bool isA(string classname)
    - Показать преимущества и недостатки каждого подхода
- **Безопасное приведение типов:**
    - Продемонстрировать опасное приведение через static_cast
    - Реализовать безопасное приведение с помощью isA()

    - Использовать dynamic_cast для безопасного приведения
- **Часть 3. Передача объектов в функции**
- **Три способа передачи параметров:**

```
void func1(Base obj);   // По значению
void func2(Base *obj);  // По указателю
void func3(Base &obj);  // По ссылке
```

    - Создать объекты Base и Desc
    - Передать их всеми тремя способами в функции
    - Проанализировать вызовы конструкторов и деструкторов
    - Показать возможность приведения типов внутри функций
- **Часть 4. Возврат объектов из функций**
- **Шесть способов возврата:**

```
// Возврат локальных объектов
Base func1();      // По значению
Base* func2();     // По указателю
Base& func3();     // По ссылке

// Возврат динамических объектов
Base func4();      // По значению
Base* func5();     // По указателю
Base& func6();     // По ссылке
```

    - Проанализировать корректность каждого способа
    - Выявить проблемные случаи (утечки памяти, висячие ссылки)
- **Часть 5. Умные указатели**
- **Использование** unique_ptr **и** shared_ptr**:**
    - Создание объектов через make_unique и make_shared
    - Передача умных указателей в функции
    - Возврат умных указателей из функций
    - Сравнение времени жизни с обычными указателями

Ход выполнения лабораторной работы

```
=== Эксперимент 1: Виртуальные vs перекрываемые методы ===
Base constructor
Base constructor
Derived constructor

1. Прямой вызов:
Base method
Derived method

2. Через указатель на базовый класс:
Base method
Derived method

3. Невиртуальные методы:
Base non-virtual method
Base non-virtual method

4. Вызов из других методов:
Base object:
Base::testMethod1 calls: Base method
Base::testMethod2 calls: Base non-virtual method
Derived object:
Base::testMethod1 calls: Derived method
Base::testMethod2 calls: Base non-virtual method
Derived destructor
Base destructor
Base destructor
```

Рис. 1 Виртуальные и перекрываемые методы

```
=== Эксперимент 2: Проверка типов ===
Base constructor
Base constructor
Derived constructor

1. classname() method:
basePtr->classname(): Base
derivedPtr->classname(): Derived

2. isA() method:
basePtr->isA('Base'): 1
basePtr->isA('Derived'): 0
derivedPtr->isA('Base'): 1
derivedPtr->isA('Derived'): 1

3. dynamic_cast:
dynamic_cast<Derived*>(basePtr): FAILED
dynamic_cast<Derived*>(derivedPtr): SUCCESS

4. Безопасное приведение с проверкой isA:
Safe to cast - object is Derived
Derived method
Base destructor
Derived destructor
Base destructor
```

Рис. 2 Определение типов. Проверка.

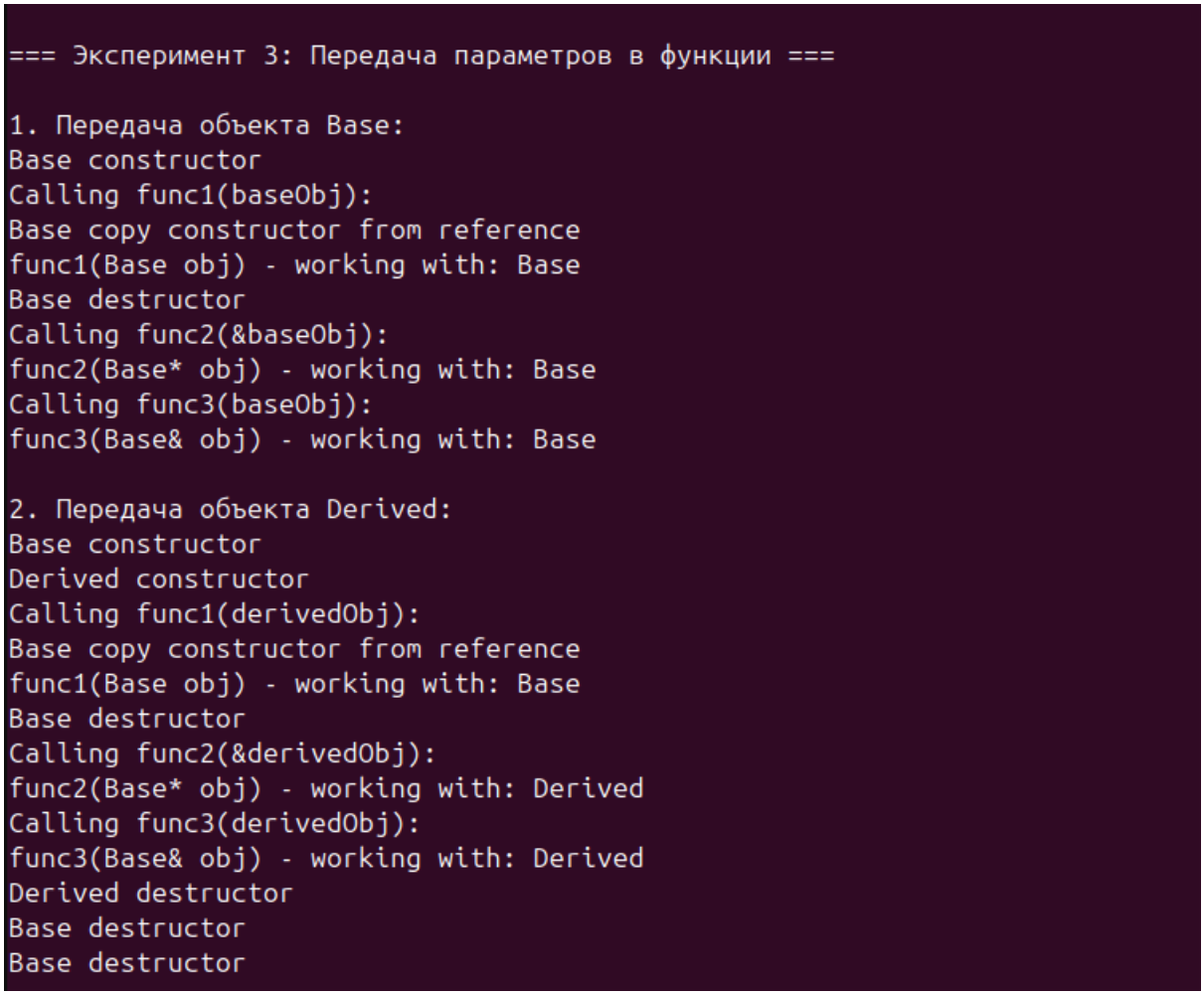

Рис. 3 Проверка конструкторов. Проверка передачи данных.

```
=== Эксперимент 4: Возврат объектов из функций ===

1. Безопасные способы возврата:
func1_ret() - возврат по значению:
Base constructor
func1_ret - returning local object by value
func2_ret() - возврат указателя на динамический объект:
Base constructor
func2_ret - returning dynamically allocated object by pointer
func3_ret() - возврат ссылки на статический объект:
Base constructor
func3_ret - returning static object by reference

2. Опасные способы возврата:
func4_ret() - утечка памяти:
Base constructor
func4_ret - returning dynamically allocated object by value (PROBLEMATIC!)
Base copy constructor from reference
func5_ret() - висячий указатель:
Base constructor
func5_ret - returning pointer to local object (DANGEROUS!)
Base destructor
func6_ret() - висячая ссылка:
Base constructor
func6_ret - returning reference to local object (DANGEROUS!)
Base destructor
Base destructor
Base destructor
Base destructor
```

Рис. 4 Возврат объектов. Опасные и безопасные возвраты объектов.

```
=== Эксперимент 5: Умные указатели ===

1. unique_ptr - эксклюзивное владение:
Base constructor
ptr1 created, use count: 1 (implicit)
After move: ptr1 is empty
After move: ptr2 is valid
Base constructor
Derived constructor
Polymorphism with unique_ptr works!
Derived destructor
Base destructor
Base destructor

2. shared_ptr - разделяемое владение:
Base constructor
Derived constructor
ptr1 use count: 1
After copy - ptr1 use count: 2
After copy - ptr2 use count: 2
In inner scope - use count: 3
After inner scope - use count: 2
Derived destructor
Base destructor

3. Передача умных указателей в функции:
Base constructor
Derived constructor
Before function call, use count: 1
Processing object in function, use count: 2
After function call, use count: 1
Derived destructor
Base destructor
```

Рис. 5 Использование умных указателей.

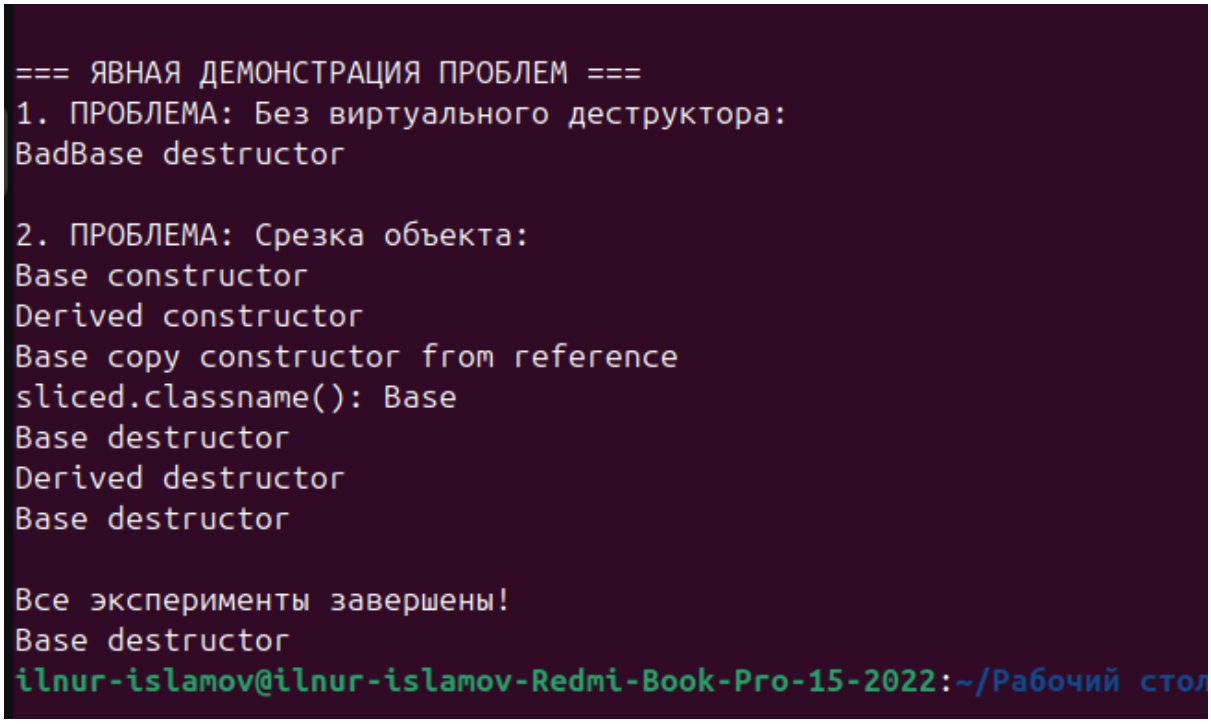

Рис. 6 Демонстрация проблем.

Выводы по лабораторной работе

В результате выполнения лабораторной работы были освоены ключевые аспекты объектно-ориентированного программирования в C++. Были изучены и практически применены механизмы виртуальности, полиморфизма и управления жизненным циклом объектов. В ходе работы были реализованы и проанализированы различные способы передачи и возврата объектов, изучены особенности работы с умными указателями, а также продемонстрированы методы безопасного приведения типов. Особое внимание было уделено пониманию различий между виртуальными и перекрываемыми методами, а также важности виртуальных деструкторов для корректного освобождения ресурсов. Полученные знания заложили фундамент для понимания принципов управления памятью и полиморфного поведения объектов в сложных иерархиях наследования.

Приложение №1

```
#pragma once
```

```
#include <iostream>

#include <string>

#include <memory>

// Базовый класс

class Base { public:

// Конструкторы и деструктор

Base();

Base(Base* obj);

Base(Base& obj);

virtual ~Base();

// Методы
virtual void method();
void nonVirtualMethod();

// Методы для проверки типа
virtual std::string classname();
virtual bool isA(const std::string& className);

// Методы для демонстрации вызовов
virtual void testMethod1();
virtual void testMethod2();


};

// Производный класс

class Derived : public Base { public:

// Конструкторы и деструктор

Derived();
```

```
Derived(Derived* obj);

Derived(Derived& obj);

~Derived();

// Переопределенные методы
void method() override;
void nonVirtualMethod(); // Перекрытие, не переопределение

// Методы для проверки типа
std::string classname() override;
bool isA(const std::string& className) override;


};


// Функции для экспериментов с передачей параметров

void func1(Base obj);

void func2(Base* obj);

void func3(Base& obj);

// Функции для экспериментов с возвратом объектов

Base func1_ret();

Base* func2_ret();

Base& func3_ret();

Base func4_ret();

Base* func5_ret();

Base& func6_ret();

// Функции экспериментов void experiment1_virtual_vs_override();
```

```
void experiment2_type_checking();

void experiment3_parameter_passing();

void experiment4_object_return();

void experiment5_smart_pointers();

void demonstrate_problems();
```