Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Уфимский университет науки и технологий»

Факультет информатики и робототехники

Кафедра вычислительной математики и кибернетики

Лабораторная работа №5

“Жизненный цикл объектов C++ и Виртуальность”

По дисциплине “Объектно-ориентированное программирование”

Выполнил:

Студент группы ПРО-232Б

Степанов К.А.

Проверил:

доцент кафедры ВМиК

Котельников В. А.

Уфа-2023

# Цель работы

Изучение основ полиморфизма, приведения типов и умных указателей.

# Задание

• Реализация

o проверки на принадлежность некоторому классу

o безопасного приведения типов (dynamic\_cast)

o безопасного приведения типов (вручную)

• Передача объектов как параметров в функции и возвращения объектов как результата из функции, контроль их жизненного цикла

• Умные указатели unique\_ptr и shared\_ptr, их влияние на жизненный цикл объектов

# Ход работы

Лабораторная работа разделена на 5 .cpp файлов, содержащих ответы на вопросы лабораторной работы.

**.cpp:**

OOP5.cpp:

Создать в классе-предке виртуальный конструктор и виртуальный деструктор, зачем нужен виртуальный деструктор и как он работает?

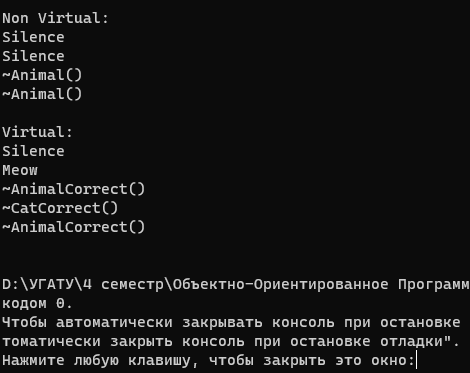


Рис. 1. Работа OOP5.cpp

Tasks.cpp:

• В методе1 базового класса вызывается метод2, который определен в этом же классе как невиртуальный, у класса-потомка метод2 переопределен: что происходит при вызове метода1 у класса-потомка?

• В методе1 базового класса вызывается метод2, который определен в этом же классе как виртуальный, у класса-потомка метод2 переопределен: что происходит при вызове метода1 у класса-потомка?

• В базовом классе объявить метод невиртуальный, а в классе-потомке объявить метод с таким же именем: какой метод будет вызываться при обращении к объекту через указатель на базовый класс, через указатель на класс-потомок?

• В базовом классе объявить метод виртуальный, а в классе-потомке объявить метод с таким же именем: какой метод будет вызываться при обращении к объекту через указатель на базовый класс, через указатель на класс-потомок?

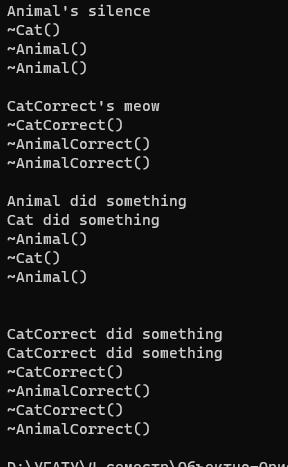


Рис. 2. Работа Tasks.cpp

Conversion.cpp:

• Зачем нужна проверка на принадлежность некоторому классу?

• Зачем и в каких случаях требуется производить безопасное приведение типов? В каком случае может понадобиться вызвать метод потомка для объекта, который лежит в переменной предка?

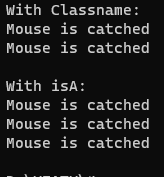


Рис. 3. Работа Conversion.cpp

ObjectsAsParameter.cpp:

Для проверки механизма передачи объектов как параметров в функции необходимо написать три функции (отдельные функции, а не методы класса):

void func1(Base obj) { ... };

void func2(Base \*obj) { ... };

void func3(Base &obj) { ... };

Далее необходимо создать простой класс Base и его потомок Desc и реализовать в каждом из них три конструктора и по одному деструктору (в каждой из трех функций и во всех конструкторах и деструкторе должен быть отладочный вывод в консоль!):

Base() { ... };

Base(Base \*obj) { ... };

Base(Base &obj) { ... };

~Base() { ... };

Desc() { ... };

Desc(Desc \*obj) { ... };

Desc(Desc &obj) { ... };

~Desc() { ... };

После этого необходимо создавать объекты классов Base и Desc, и передавать их в каждую из трех функций, объясняя при этом, когда и почему вызывается какой конструктор или деструктор. Можно ли внутри функции привести переданный Base к Desc с помощью приведения типов? Необходимо уметь объяснять достоинства и недостатки каждого из вариантов функции.

Для проверки механизма возврата объектов из функции необходимо написать шесть функций (отдельных функций, а не методов класса):

Base func1() { ... };

Base\* func2() { ... };

Base& func3() { ... };

Base func4() { ... };

Base\* func5() { ... };

Base& func6() { ... };

Внутри функций 1,2,3 необходимо создавать и возвращать статический (локальный) объект, внутри функций 4,5,6 возвращаемый объект необходимо создавать динамически с помощью new. Затем необходимо помещать результат вызова каждой из этих трёх функций в локальную переменную и объяснять наблюдаемый отладочный вывод (показывающий последовательность создания и удаления объектов). Необходимо уметь объяснять достоинства и недостатки каждого из вариантов, в том числе уметь идентифицировать неправильную работу с памятью.

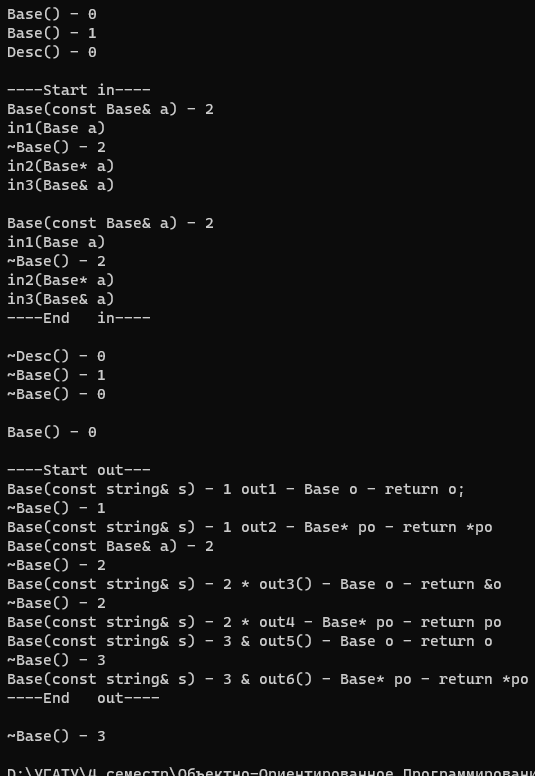


Рис. 4. Работа ObjectsAsParameter.cpp

Smart\_pointers.cpp:

Для изучения умных указателей необходимо создать объекты, управляемые с помощью unique\_ptr и shared\_ptr (с помощью make\_unique и make\_shared и/или без них), помещать их в переменные, передавать их в функции, возвращать их из функций и демонстрировать, как они влияют на время жизни объекта, которым управляют.

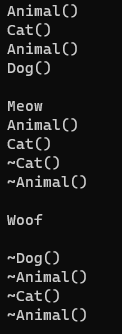


Рис. 5. Работа Smart\_pointers.cpp

# Выводы

По ходу выполнения работы были освоены основы полиморфизма, работа с умными указателями, проверки на принадлежность к классам и другое.

Ссылка на GitHub: <https://github.com/stkirill15/OOP-5>.

# Приложение 1

**OOP5.cpp**

#include <vector>

#include <iostream>

#include <string>

using namespace std;

class Animal

{

public:

string voice()

{

return "Silence";

}

~Animal()

{

cout << "~Animal()" << endl;

}

};

class Cat : public Animal

{

public:

int\* tmp;

Cat()

{

tmp = new int[8];

}

string voice()

{

return "Meow";

}

~Cat()

{

cout << "~Cat()" << endl;

delete[] tmp;

}

};

class AnimalCorrect

{

public:

virtual string voice()

{

return "Silence";

}

virtual ~AnimalCorrect()

{

cout << "~AnimalCorrect()" << endl;

}

};

class CatCorrect : public AnimalCorrect

{

public:

int\* tmp;

CatCorrect()

{

tmp = new int[8];

}

string voice() override

{

return "Meow";

}

~CatCorrect() override

{

cout << "~CatCorrect()" << endl;

delete[] tmp;

}

};

int main()

{

vector<Animal\*> animals(2);

animals[0] = new Animal();

animals[1] = new Cat();

cout << "Non Virtual:" << endl;

for (auto someAnimal : animals) {

cout << someAnimal->voice() << endl;

}

delete animals[0];

delete animals[1];

cout << endl;

vector<AnimalCorrect\*> correctAnimals(2);

correctAnimals[0] = new AnimalCorrect();

correctAnimals[1] = new CatCorrect();

cout << "Virtual:" << endl;

for (auto someAnimal : correctAnimals) {

cout << someAnimal->voice() << endl;

}

delete correctAnimals[0];

delete correctAnimals[1];

cout << endl;

return 0;

}