**Программирование на С++**

**9-11 классы**

**Занятие 9**

**Тема: Наследование в С++.**

**Цели и задачи:**

* Рассказать о принципе наследования
* Рассказать о модификаторах доступа в случае наследования от родительского класса

**По результатам занятия слушатель будет знать:**

* Принцип наследования в ООП
* Как использовать полиморфизм для взаимодействия объектов класса между собой и не только

**Теоретическая часть**

**Наследование**

Наследование является одной из наиболее важных концепций объектно-ориентированного программирования.

Наследование позволяет нам определять класс, основанный на другом классе. Это облегчает удобство создания и управления приложением. Класс, чьи свойства наследованы другим классом, называется Базовым классом. Класс, который наследует свойства, называется Производным. Например, класс Daughter (производный) может быть наследован от класса Mother (базовый).

Производный класс наследует все особенности базового класса, и может иметь свои дополнительные особенности.

Идея наследования заключается в отношении это. Например, млекопитающее ЭТО животное, собака ЭТО млекопитающее, следовательно, собака ЭТО также животное.

Базовый класс указывается с помощью использования двоеточия и спецификатора доступа: public означает, что все публичные элементы базового класса являются публичными в производном классе.

Другими словами, все публичные элементы класса Mother становятся публичными элементами класса Daughter. Производный класс наследует все базовые методы класса, кроме следующего:

- Конструкторы, деструкторы

- Перегруженные операторы

- Дружественные(friend) функции

Класс может быть наследован от множества классов, с помощью указания базовых классов, разделенных запятыми. Например: class Daughter: public Mother, public Father

**Спецификаторы Доступа**

До этого времени, мы работали только со спецификаторами доступа public и private.

Доступ к публичным элементам может быть получен снаружи класса, в то время как доступ к приватным элементам имеют только их класс и дружественные функции.

Protected

Существует еще один спецификатор доступа - protected.

Protected переменная или функция очень похожа на приватный элемент, с одним отличием – она может быть доступна в производном классе.

**Типы наследования**

Спецификаторы доступа также используются для описания типа наследования.

Наследование с ключом доступа Public: public элементы базового класса становятся public элементами производного класса, protected элементы базового класса становятся protected элементами производного класса. Private элементы никогда не бывают доступны из производного класса, но могут быть доступны, с помощью вызова public и protected элементов базового класса.

Наследование с ключом доступа Protected: public и protected элементы базового класса становятся protected элементами производного класса.

Наследование с ключом доступа Private: public и protected элементы базового private класса становятся private элементами производного класса.

При наследовании классов, конструктор и деструктор базового класса не наследуется.

Однако, они вызываются, когда создается или удаляется объект производного класса.

**Порядок вызовов конструктора и деструктора**

Конструктор базового класса вызывается первым.

Деструктор производного класса будет вызван первым, затем будет вызван деструктор базового класса.

**Полиморфизм**

Слово полиморфизм означает "имеет много форм".

Обычно, полиморфизм используется там, где присутствует иерархия классов, связанная наследованием.

В C++ полиморфизм означает, что вызов функций-членов будет причиной выполнения различного исполнения, в зависимости от типа объекта, который вызывает функцию.

Проще говоря, полиморфизм означает, что одна функция может иметь множество различных действий.

Предположим, вы хотите сделать простую игру, которая включает различных врагов: монстры, ниндзя и др. Все враги имеют одну общую функцию: функцию attack(атаковать). Однако все они атакуют различным образом. В этой ситуации полиморфизм позволяет вызывать одну и ту же функцию attack в разных объектах, но результатом будет различное поведение.

**Работа с укзателями на объект класса (см пример на слайде 12)**

Мы получили бы такой же результат, если бы вызвали эти функции прямо из объектов. Однако, быстрее и намного эффективнее использовать указатели.

Также, указатели демонстрируют, что вы можете использовать указатель Enemy не зная точно, что он содержит объект подкласса.

**Практическая часть**

**Задание 1**

Дополнить последний пример. Создать класс defender, sniper и superman.

Полями класса являются нанесение урона врагу. Связать все три класса с помощью наследования.

\*

Использовать указатели для работы с объектами класса

(Файл с кодом лежит в папке code/)