Виртуальные функции и полиморфизм

Виртуальные функции

- Функция-член, объявленная в базовом классе и переопределенная в производном
- То есть виртуальный означает видимый, но не существующий в реальности
- Программа, которая, казалось бы, вызывает функцию одного класса, может вызывать функцию совсем другого класса

Зачем нужны виртуальные функции

- Пусть имеется набор объектов разных классов
  - Например, есть разные геометрические фигуры: треугольник, шар и квадрат. В каждом классе есть метод draw(), который прорисовывает на экране соответствующие фигуры
- Задача: нарисовать картинку, сгруппировав эти элементы, без дополнительных сложностей
- Решение: создать массив указателей на все неповторяющиеся элементы картинки, т.е. указатель на объект шарик, указатель на квадрат и т.п.
- Обращаясь к разным элементам массива можно рисовать разные фигуры

Причем здесь полиморфизм

- То есть имеем «один интерфейс (функции называются одинаково draw()) и несколько методов (реально вызываются разные функции, рисующие разные фигуры)»
- Это есть – полиморфизм
- Перегрузка функций – статический полиморфизм
- Наследование и виртуальные функции – динамический полиморфизм

Как создаются виртуальные функции

- В базовом классе перед объявлением виртуальной функции указывается ключевое слово: virtual
- В производном классе функция переопределяется – то есть создается конкретная реализация функции
- Для примера рассмотренного выше:
  - Все классы (шар, треугольник, квадрат) должны быть наследниками одного и того же базового класса
  - Функция draw() должна быть объявлена как virtual

Доступ к обычным методам через указатели: базовый и производные классы содержат функции с одним и тем же именем, к ним обращаются с помощью указателей, но без виртуальных функций

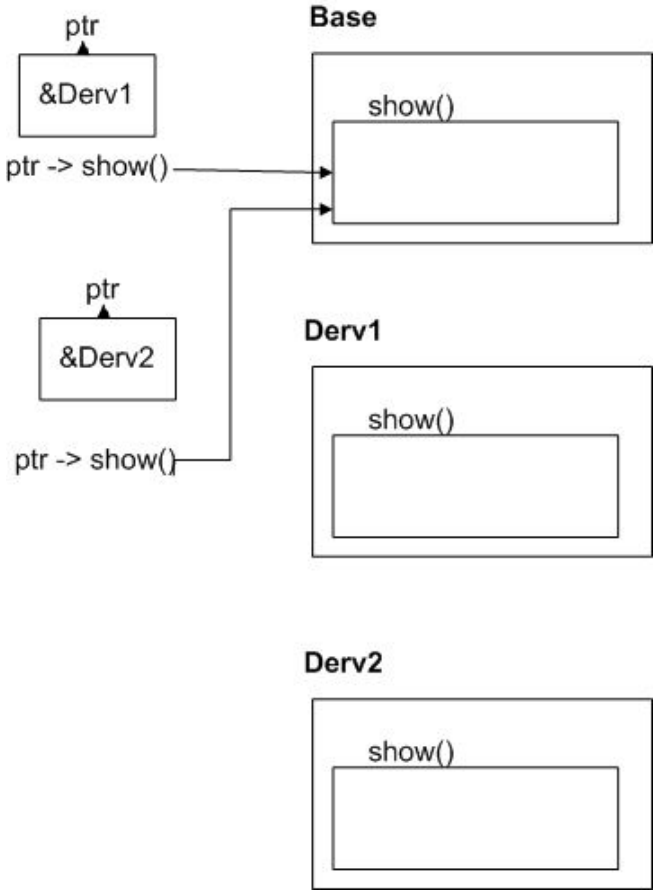
```
class base
{
public:
      void show() { cout <<“Base \n”;}
};
Class Derv1:public Base
{
      void show() { cout <<“Derv1\n”;}
};
Class Derv2:public Base
{
      void show() { cout <<“Derv2\n”;}
};

void main ()
{
      Derv1 dv1;
      Derv2 dv2;
      Base* ptr;

      ptr = &dv1;
      ptr ->show();

      ptr = &dv2;
      ptr ->show();
}
```

Вывод: ?



Доступ к виртуальным методам через указатели

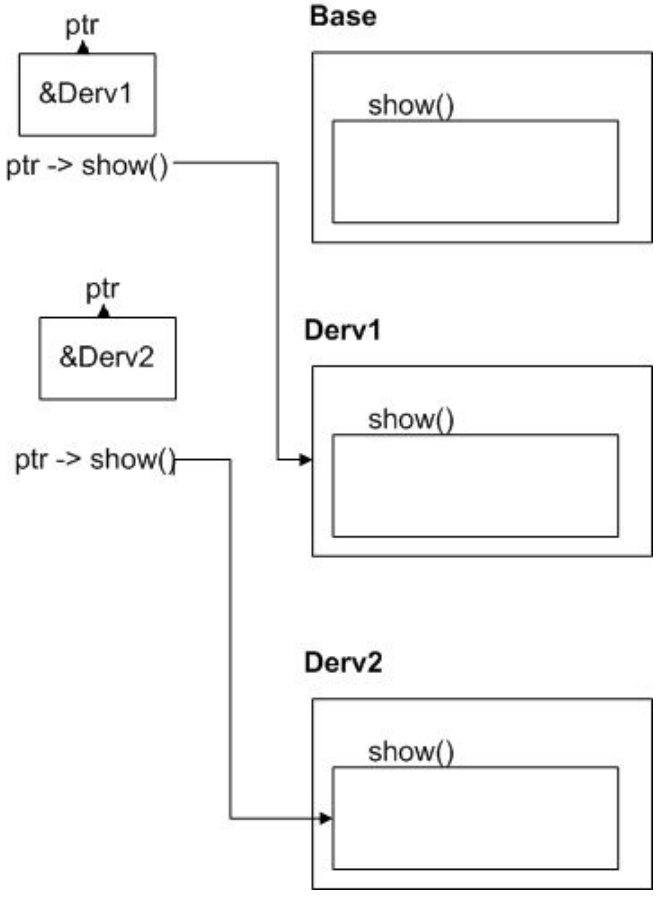
```
class base
{
public:
        virtual void show() { cout <<“Base \n”;}
};
Class Derv1:public Base
{
        void show() { cout <<“Derv1\n”;}
};
Class Derv2:public Base
{
        void show() { cout <<“Derv2\n”;}
};

void main ()
{
        Derv1 dv1;
        Derv2 dv2;
        Base* ptr;

        ptr = &dv1;
        ptr ->show();

        ptr = &dv2;
        ptr ->show();
}
```

Вывод: ?



Позднее или динамическое связывание

Какая функция компилируется при компиляции выражения:

ptr ->show(); ?

- Всегда компилируется функция из базового класса
- Однако в последней программе компилятор не знает к какому классу относится содержимое ptr.
- Когда программа поставлена на выполнение, когда известно, на что указывает ptr, тогда запускается соответствующая версия show().
- Выбор функции во время компиляции называется ранним или статическим связыванием
- Позднее связывание требует больше ресурсов, но дает выигрыш в возможностях и гибкости

Задание

- Создайте программу, реализующую кошелек, используя виртуальные функции

Помните:

- Виртуальные функции позволяют решать прямо в процессе выполнения программы, какую именно функцию вызывать
- Виртуальные функции дают большую гибкость при выполнении одинаковых действий над разнородными объектами