CSE-4301
Object Oriented Programming
2022-2023

Week-8-9

Inheritance

-Faisal Hussain
Lecturer, Department Of Computer Science And Engineering, IUT

Contents

- Accessing member using pointer to object
- Virtual Function
- Virtual destructor
- Virtual Base class
- Friend Function
- Friend Class
- Assignment and Copy Initialization
- The this pointer
- Dynamic cast

Accessing member data of object using pointer

- Base class pointer can point to base class object
- Derived class pointer can point to derived class object

- Base class pointer can point to derived class object (Derived from the mentioned base class)

The rule is that pointers to objects of a derived class are type compatible with pointers to objects of the base class

- Derive class pointer **can not** point to base class object

Virtual Function

- **A virtual function is a function whose behavior can be overridden within an inheriting class by a function with the same signature.**
- How to declare a virtual function?
- How to **override** a virtual function?

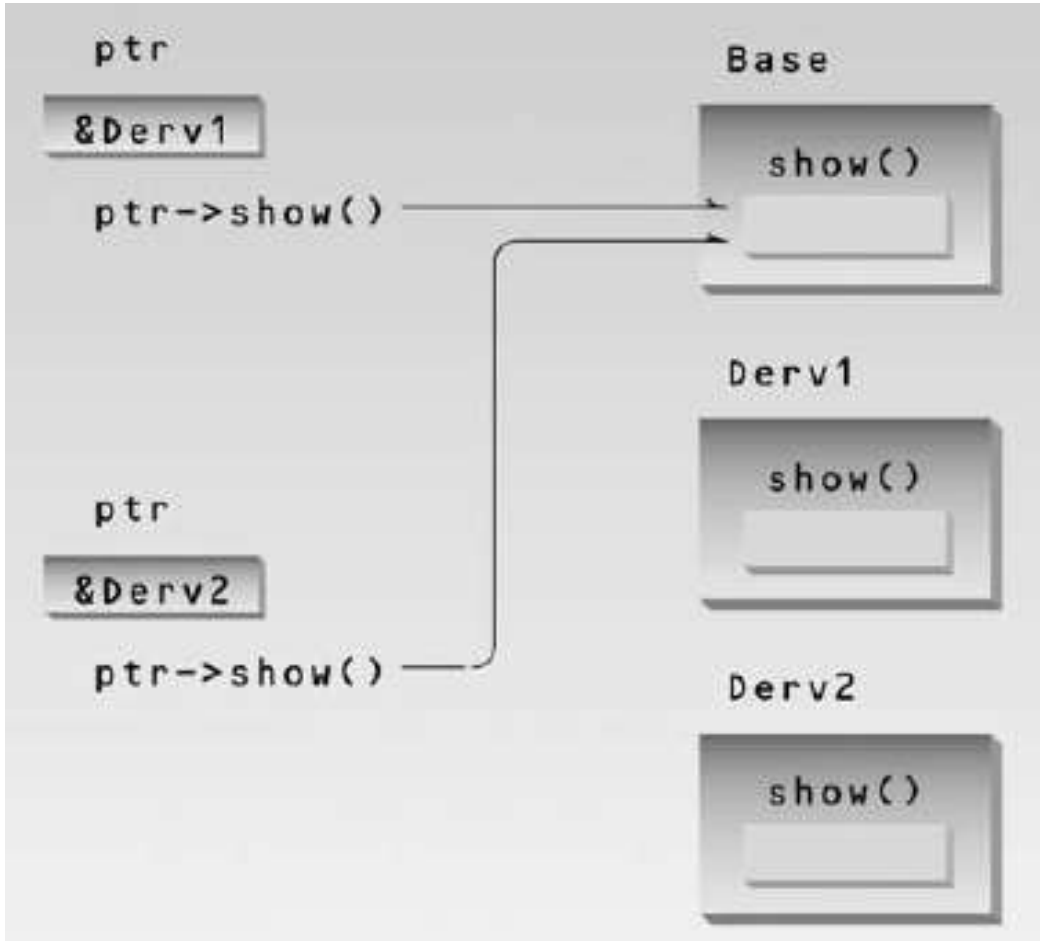
- **Virtual function implements: "One interface multiple Methods"**

Virtual Function : Polymorphism

- Virtual function supports **run time polymorphism** –when a **virtual function is called through a pointer.**

- **Base class pointer point -> derived objects**, that **contains a virtual function** and the **function is called through that pointer**:
  - C++ **compiler determine which version of that function will be executed** based upon the type of object being pointed by the pointer
  - Check the Sample Example

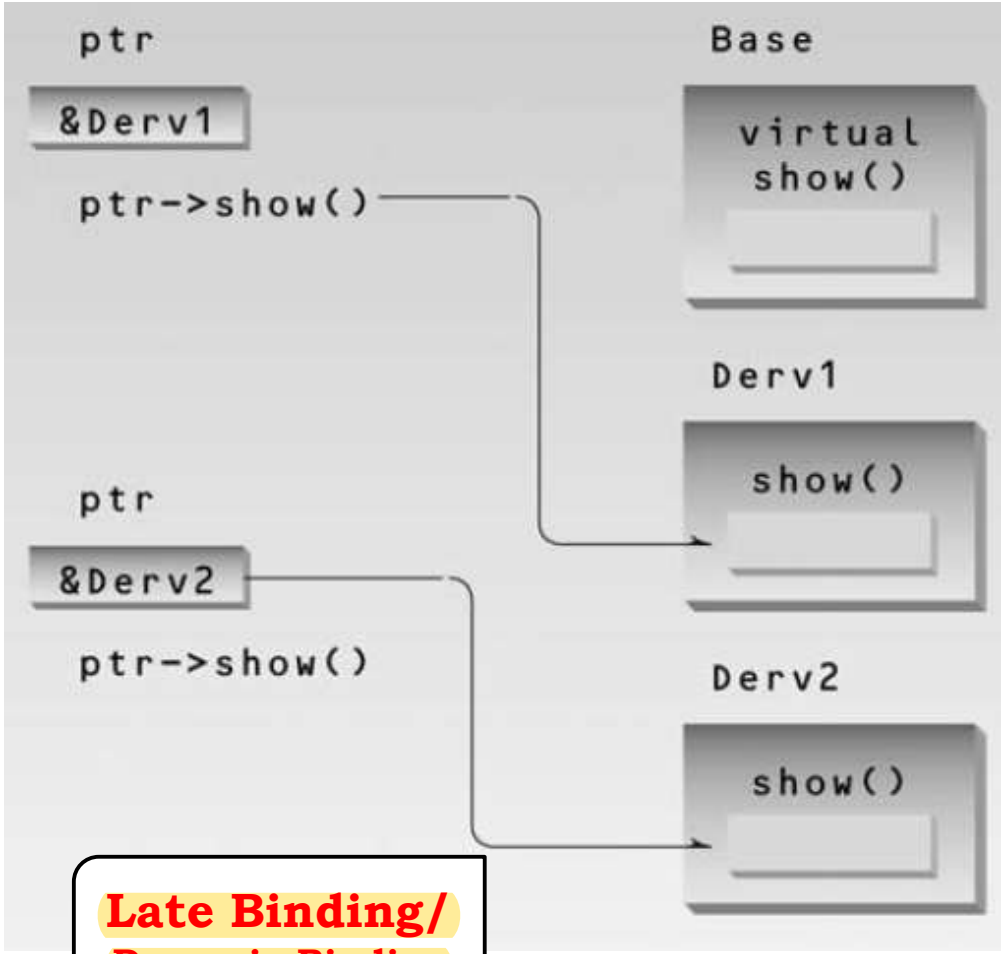
Non Virtual Pointer Access




```
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
class Base                              //base class
   {
   public:
      void show()                       //normal function
         { cout << "Base\n"; }
   };
////////////////////////////////////////////////////////////
class Derv1 : public Base               //derived class 1
   {
   public:
      void show()
         { cout << "Derv1\n"; }
   };
////////////////////////////////////////////////////////////
class Derv2 : public Base               //derived class 2
   {
   public:
      void show()
         { cout << "Derv2\n"; }
   };
////////////////////////////////////////////////////////////
int main()
   {
   Derv1 dv1;              //object of derived class 1
   Derv2 dv2;              //object of derived class 2
   Base* ptr;              //pointer to base class

   ptr = &dv1;             //put address of dv1 in pointer
   ptr->show();            //execute show()
   ptr = &dv2;             //put address of dv2 in pointer
   ptr->show();            //execute show()
   return 0;
   }
```

Virtual function




```
class Base                                  //base class
   {
   public:
      virtual void show()                   //virtual function
         { cout << "Base\n"; }
   };
//////////////////////////////////////////////////////////
class Derv1 : public Base                   //derived class 1
   {
   public:
      void show()
         { cout << "Derv1\n"; }
   };
//////////////////////////////////////////////////////////
class Derv2 : public Base                   //derived class 2
   {
   public:
      void show()
         { cout << "Derv2\n"; }
   };
//////////////////////////////////////////////////////////
int main()
   {
   Derv1 dv1;              //object of derived class 1
   Derv2 dv2;              //object of derived class 2
   Base* ptr;              //pointer to base class

   ptr = &dv1;             //put address of dv1 in pointer
   ptr->show();            //execute show()

   ptr = &dv2;             //put address of dv2 in pointer
   ptr->show();            //execute show()
   return 0;
   }
```

Virtual Function

- When does a derived class not override a virtual function?
- Virtual functions are hierarchical.

- Check the Sample Example 1,2

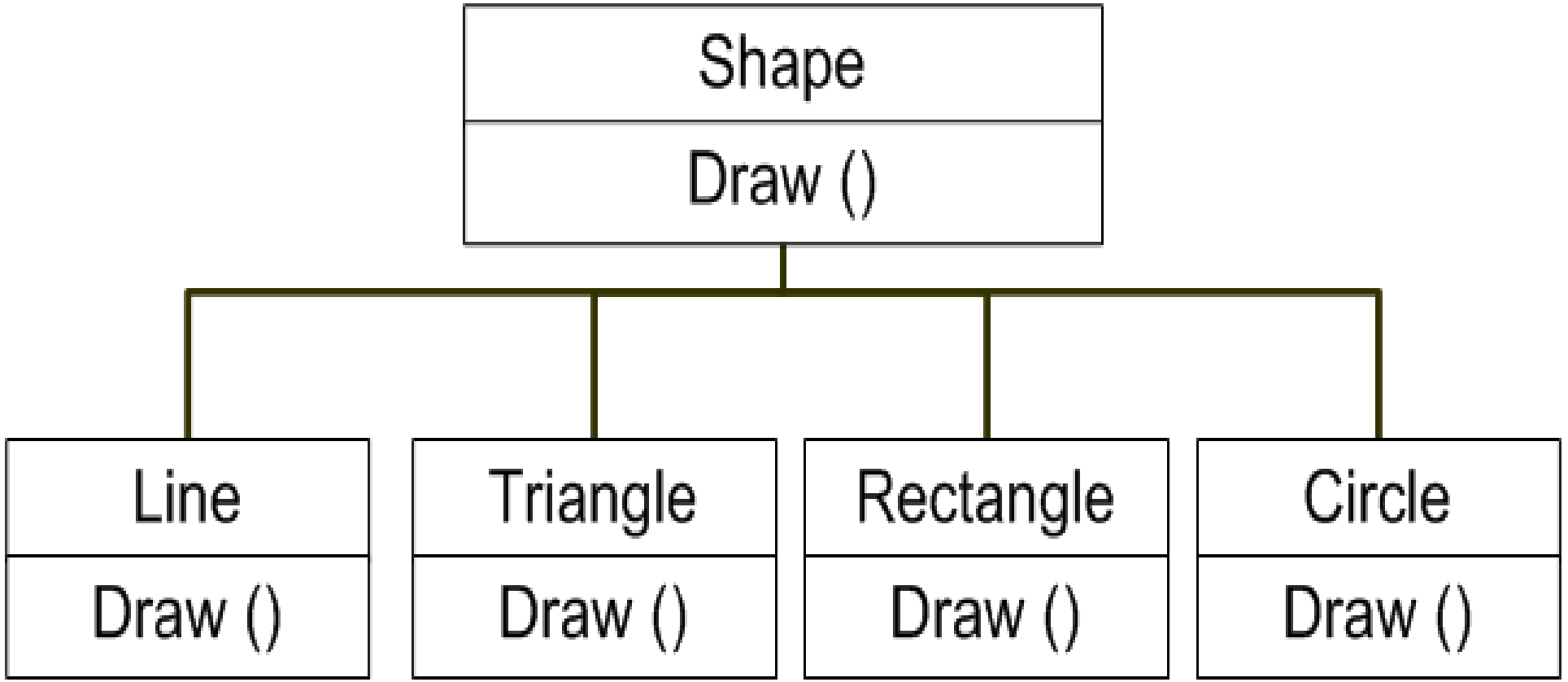
Why use **Virtual Function** ?

- when you want to **override a certain behavior** **for your derived class** than the one implemented for the Base class.

  And

- want to do so **at run-time through a pointer to Base class**.

Why use Virtual Function ?



Check the Sample Example 3

pure virtual function

- A **pure virtual function** has no difference with **virtual function except:**
  - **Only functions prototype is included (no body)**
  - **Derived class Must override this function**

  - How to declare a pure virtual function?

Check the Sample Example 4

Abstract Classes :

- Abstract Classes :

containing pure virtual methods/functions are termed "abstract" and they cannot be instantiated directly.

A subclass of an abstract class can only be instantiated directly if all inherited pure virtual methods have been implemented by that class or a parent class.

Reading Assignment

- **Multiple Inheritance**
- **Virtual member function and Virtual base class**
- **Working with array of Objects (Finding most attractive object)**
- **Passing object as argument to any function defined outside of the class.**
- **Passing array of Object as argument**
- **Pointer of an object & How to handle those.**

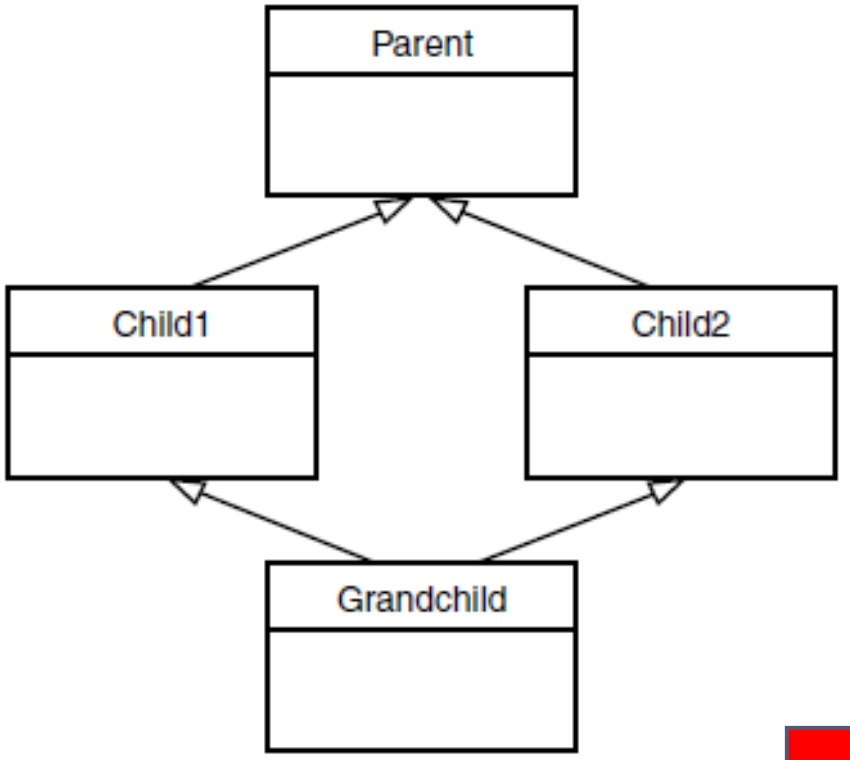
Virtual Destructor

```
int main()
{
    Base* pBase = new Derv;
    delete pBase;
    return 0;
}
```

- Check the sample code 5

- If non virtual destructor is present in Base class then delete of Base class pointer triggers only Base class destructor.
- If Base class destructor is a virtual function and Base class pointer is pointing a Derived class object then delete of Base class pointer triggers the Derv class destructor then Base class destructor.

Virtual Base Class




```
class Parent
   {
   protected:
      int basedata;
   };
class Child1 : public Parent
   { };
class Child2 : public Parent
   { };
class Grandchild : public Child1, public Child2
   {
   public:
      int getdata()
         { return basedata; }   // ERROR: ambiguous
   };
```

Virtual Base Class

```
class Parent
   {
   protected:
      int basedata;
   };
class Child1 : virtual public Parent    // shares copy of Parent
   { };
class Child2 : virtual public Parent    // shares copy of Parent
   { };
class Grandchild : public Child1, public Child2
   {
   public:
      int getdata()
         { return basedata; }     // OK: only one copy of Parent
   };
```

- **Virtual base classes**, used in virtual inheritance, is a way of preventing multiple "instances" of a given class appearing in an inheritance hierarchy when using multiple inheritance

Check the Sample Example 6

Friend Functions

- Non-member functions should not be able to access an object's private or protected data
- a function to operate on objects of two different classes
- Check the sample code 4

- A programmer who does not have access to the source code for the class cannot make a function into a friend

- If many friends functions are needed, you may need to rethink the design of the program.

Check the Sample Example 7,8,9

Friend Class

- The member functions of a class can all be made friends at the same time when you make the entire class a friend.
- Check the sample code 10

The this pointer

- The member functions of every object have access to a special pointer named **this** which points to the object itself.
- Check the sample code 11

RTTI (Run-time type Information)

- RTTI (Run-time type information) is a mechanism that exposes information about an object's data type at runtime and is available only for the classes which have at least one virtual function.
- The **RTTI** is available only when there is a polymorphic behaviour

dynamic_cast

- #include <typeinfo> for dynamic_cast
- Base class pointer may hold different derived class pointer (that inherits the same base class), how to check a certain derive class pointer or not.
- Check sample code 7
- Check typeid operator