Module 22: Programming in C++

Inheritance: Part 2 (Data Member & Member Function - Override)

Sourangshu Bhattacharya

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

sourangshu@cse.iitkgp.ac.in

Slides taken from NPTEL course on Programming in C++

by **Prof. Partha Pratim Das**

Module Objectives

- Understand how inheritance impacts data members and member functions
- Introduce overriding of member function and its interactions with overloading

Module Outline

- ISA Relationship
- Inheritance in C++
    - Semantics
    - Data Members and Object Layout
    - Member Functions
        - Overriding
        - Overloading
    - protected Access
    - Constructor & Destructor
    - Object Lifetime
- Example – Phone Hierarchy
- Inheritance in C++ (private)
    - Implemented-As Semantics

Inheritance in C++: Semantics

- Derived **ISA** Base
- Data Members
  - Derived class *inherits* all data members of Base class
  - Derived class may *add* data members of its own
- Member Functions
  - Derived class *inherits* all member functions of Base class
  - Derived class may *override* a member function of Base class by *redefining* it with the same signature
  - Derived class may *overload* a member function of Base class by *redefining* it with the *same name*; but *different signature*
- Access Specification
  - Derived class *cannot access* private members of Base class
  - Derived class *can access* protected members of Base class
- Construction-Destruction
  - A *constructor* of the Derived class *must first* call a *constructor* of the Base class to construct the Base class instance of the Derived class
  - The *destructor* of the Derived class *must* call the *destructor* of the Base class to destruct the Base class instance of the Derived class

Inheritance in C++: Data Members and Object Layout

- Derived **ISA** Base
- Data Members
  - Derived class *inherits* all data members of Base class
  - Derived class may *add* data members of its own
- Object Layout
  - Derived class *layout* contains an instance of the Base class
  - Further, Derived class layout will have data members of its own
  - C++ does not guarantee the relative position of the Base class instance and Derived class members

Inheritance in C++: Data Members and Object Layout

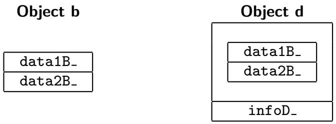
```
class B { // Base Class
    int data1B_;
public:
    int data2B_;
    // ...
};

class D: public B { // Derived Class
    // Inherits B::data1B_
    // Inherits B::data2B_
    int infoD_; // Adds D::infoD_
public:
    / ...
};

B b;

D d;
```

Object Layout



d cannot access data1B_ even though data_ is a part of it!

d can access data2B_

Inheritance in C++: Member Functions – Overrides and Overloads

- Derived **ISA** Base
- Member Functions
  - Derived class *inherits* all member functions of Base class
  - Derived class may *override* a member function of Base class by *redefining* it with the same signature
  - Derived class may *overload* a member function of Base class by *redefining* it with the *same name*; but *different signature*
  - Derived class *may add* new member functions
- Static Member Functions
  - Derived class *does not inherit* the static member functions of Base class
- Friend Functions
  - Derived class *does not inherit* the friend functions of Base class

Inheritance in C++: Member Functions – Overrides and Overloads

Inheritance

```
class B { // Base Class
public:
    void f(int i);
    void g(int i);
};
class D: public B { // Derived Class
public:
    // Inherits B::f(int)
    // Inherits B::g(int)




};

B b;
D d;

b.f(1); // Calls B::f(int)
b.g(2); // Calls B::g(int)

d.f(3); // Calls B::f(int)
d.g(4); // Calls B::g(int)
```

Override & Overload

```
class B { // Base Class
public:
    void f(int);
    void g(int i);
};
class D: public B { // Derived Class
public:
    // Inherits B::f(int)
    void f(int);      // Overrides B::f(int)
    void f(string&); // Overloads B::f(int)
    // Inherits B::g(int)
    void h(int i);    // Adds D::h(int)
};

B b;
D d;

b.f(1);     // Calls B::f(int)
b.g(2);     // Calls B::g(int)

d.f(3);     // Calls D::f(int)
d.g(4);     // Calls B::g(int)

d.f("red"); // Calls D::f(string&)
d.h(5);     // Calls D::h(int)
```

- D::f(int) overrides B::f(int)
- D::f(string) overloads B::f(int)

Module Summary

- Discussed the effect of inheritance on Data Members and Object Layout
- Discussed the effect of inheritance on Member Functions with special reference to Overriding and Overloading