Module 22: Programming in C++

Inheritence (Part 2): Override and Overload

Intructors: Abir Das and Sourangshu Bhattacharya

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

{*abir, sourangshu*}*@cse.iitkgp.ac.in*

Slides taken from NPTEL course on Programming in Modern 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

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*
  - Derived class *may add* new member functions
- **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

Data Members

- 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

Object Layout

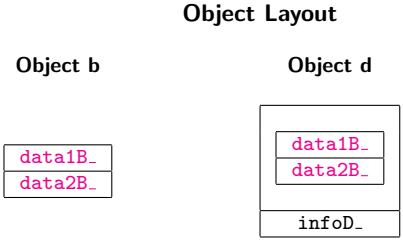
```
class B { // Base Class
    int data1B_;
public:
    int data2B_;
    // ...
};

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

B b; // Base Class Object

D d; // Derived Class Object
```

Object Layout



- d *cannot access* data1B_ even though is a part of d!
- d *can access* data2B_

Member Functions

- Derived **ISA** Base
- **Member Functions**
  - Derived class *inherits* all member functions of Base class
    - **Note**: Derived class *does not inherit* the Constructors and Destructor of Base class but *must have access to them*
  - 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

Overrides and Overloads

Inheritance

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

Comparison of Overloading vis-a-vis Overriding

Basis	Function Overloading	Function Overriding
Name of Function	• All overloads have the same function name	• All overrides have the same function name
Signature	• Function signatures must be different	• Function signatures are same
Type of Function	• Can be global, friend, static or non-static member function	• Must be a non-static member function - non-virtual or virtual
Inheritance	• Can happen with or without inheritance	• Happens only with inheritance
Polymorphism	• Static (Compile time)	• Static (Compile time) or Dynamic (Runtime)
Scope	• Overloaded functions are in the same scope	• Functions are in different scopes (base clase and derived class)
Purpose	• To have multiple functions with same name that act differently depending on parameters	• To perform additional or different tasks than the base class function
Constructor	• Constructors can be overloaded	• Constructors cannot be overridden
Destructor	• The destructor cannot be overloaded	• The destructor cannot be overridden
Usage	• Can be overloaded multiple times	• Can be overridden once in the derived class

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