

-
-
-
-
-
-
-

Hiding & Overriding



C++ Object Oriented Programming

Pei-yih Ting

NTOU CS

Overloading, Overriding, Hiding

- ✧ **Overloading**: two functions in the same scope, have the same name, **different signatures** (virtual is not required)

Overloading, Overriding, Hiding

- ✧ **Overloading**: two functions in the same scope, have the same name, **different signatures** (virtual is not required)

- `service(int)`
- `service(double, int)`

Overloading, Overriding, Hiding

- ✧ **Overloading**: two functions in the same scope, have the same name, **different signatures** (virtual is not required)

- `service(int)`
- `service(double, int)`

- ✧ **Overriding**: two functions in **different scopes** (parent vs **child**), have the same name. same signatures (**virtual** is required)

Overloading, Overriding, Hiding

- ✧ **Overloading**: two functions in the same scope, have the same name, **different signatures** (virtual is not required)

- `service(int)`
- `service(double, int)`

- ✧ **Overriding**: two functions in **different scopes** (parent vs **child**), have the same name. same signatures (**virtual** is required)

- `virtual service(int,int)`
- `virtual service(int,int)`



Overloading, Overriding, Hiding

- ❖ **Overloading**: two functions in the same scope, have the same name, **different signatures** (virtual is not required)

• service(int)
• service(double, int)

The diagram shows two function signatures, `service(int)` and `service(double, int)`, each preceded by a bullet point. They are enclosed in a single rectangular box, indicating they exist in the same scope.

- ❖ **Overriding**: two functions in **different scopes** (parent vs **child**), have the same name. same signatures (**virtual** is required)

• virtual service(int,int)
• virtual service(int,int)

The diagram shows two function signatures, `virtual service(int,int)`, each preceded by a bullet point. They are enclosed in separate rectangular boxes. An upward-pointing arrow is positioned between the two boxes, indicating that the bottom box (child scope) overrides the top box (parent scope).

- ❖ **Hiding**: base class member function is hidden

Overloading, Overriding, Hiding

- ❖ **Overloading**: two functions in the same scope, have the same name, **different signatures** (virtual is not required)

- service(int)
- service(double, int)

- ❖ **Overriding**: two functions in **different scopes** (parent vs **child**), have the same name. same signatures (**virtual** is required)

- virtual service(int,int)
- virtual service(int,int)

- ❖ **Hiding**: base class member function is hidden

1. When a base class and a derived class declare virtual member functions with different signatures but **with** the same name.

Overloading, Overriding, Hiding

- ❖ **Overloading**: two functions in the same scope, have the same name, **different signatures** (virtual is not required)

- service(int)
- service(double, int)

- ❖ **Overriding**: two functions in **different scopes** (parent vs **child**), have the same name. same signatures (**virtual** is required)

- virtual service(int,int)
- virtual service(int,int)

- ❖ **Hiding**: base class member function is hidden

1. When a base class and a derived class declare virtual member functions with different signatures but **with** the same name.

- virtual service(double)
- virtual service(int,int)

Overloading, Overriding, Hiding

- ❖ **Overloading**: two functions in the same scope, have the same name, **different signatures** (virtual is not required)

- service(int)
- service(double, int)

- ❖ **Overriding**: two functions in **different scopes** (parent vs **child**), have the same name. same signatures (**virtual** is required)

- virtual service(int,int)



- virtual service(int,int)

- ❖ **Hiding**: base class member function is hidden

1. When a base class and a derived class declare virtual member functions with different signatures but **with** the same name.

- virtual service(double)



- virtual service(int,int)

2. When a base class declares a **non-virtual** member function and a derived class declares a member function **with** the same name but **with** or **without** the same signature.

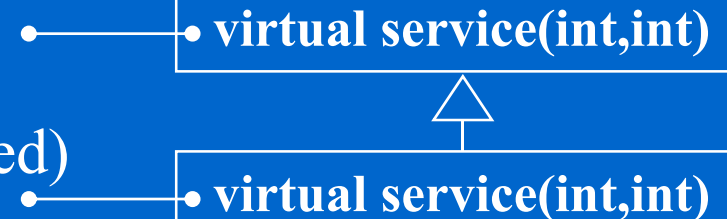
Overloading, Overriding, Hiding

- ❖ **Overloading**: two functions in the same scope, have the same name, **different signatures** (virtual is not required)

• service(int)
• service(double, int)

- ❖ **Overriding**: two functions in **different scopes** (parent vs **child**), have the same name. same signatures (**virtual** is required)

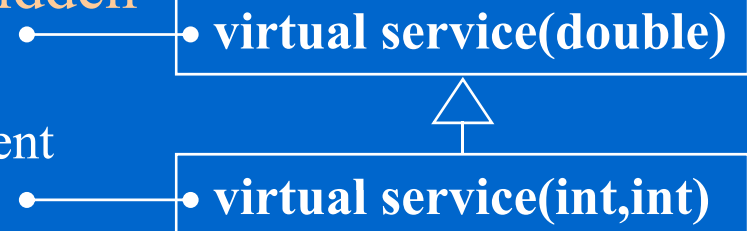
• virtual service(int,int)
• virtual service(int,int)



- ❖ **Hiding**: base class member function is hidden

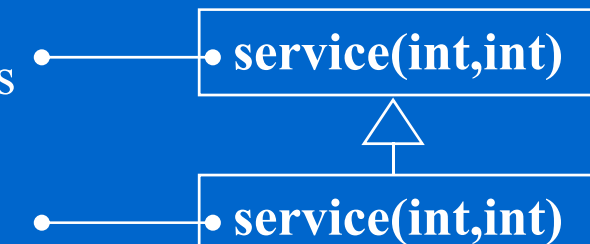
1. When a base class and a derived class declare virtual member functions with different signatures but **with** the same name.

• virtual service(double)
• virtual service(int,int)



2. When a base class declares a **non-virtual** member function and a derived class declares a member function **with** the same name but **with** or **without** the same signature.

• service(int,int)
• service(int,int)



Hiding

```
class Person {  
public:  
    virtual void display() const;  
};
```

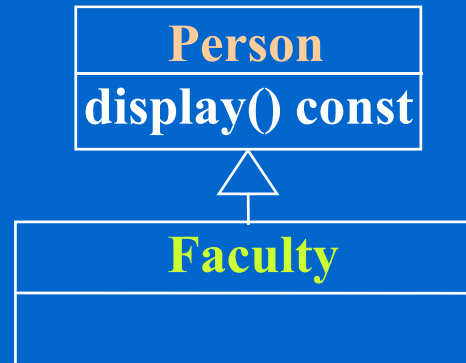
Person
display() const

```
Person p;  
p.display();
```

Hiding

```
class Person {  
public:  
    virtual void display() const;  
};
```

```
class Faculty: public Person {  
public:  
  
};
```

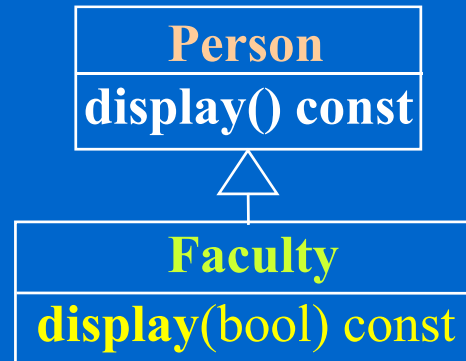


```
Person p;  
p.display();  
  
Faculty f;  
f.display();
```

Hiding

```
class Person {  
public:  
    virtual void display() const;  
};
```

```
class Faculty: public Person {  
public:  
    virtual void display(bool showDetail) const;  
};
```



```
Person p;  
p.display();  
Faculty f;  
f.display();
```

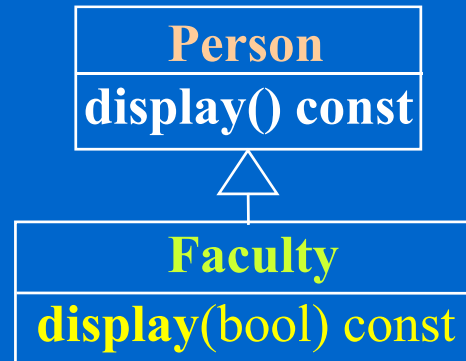
different signature

- ★ In **Faculty** class, **display(bool)** does **NOT override** **Person::display()**
⇒ different entries in virtual table

Hiding

```
class Person {  
public:  
    virtual void display() const;  
};
```

```
class Faculty: public Person {  
public:  
    virtual void display(bool showDetail) const;  
};
```



```
Person p;  
p.display();
```

```
Faculty f;  
f.display();
```

different signature

- ★ In **Faculty** class, **display(bool)** does **NOT** *override* **Person::display()**
⇒ different entries in virtual table

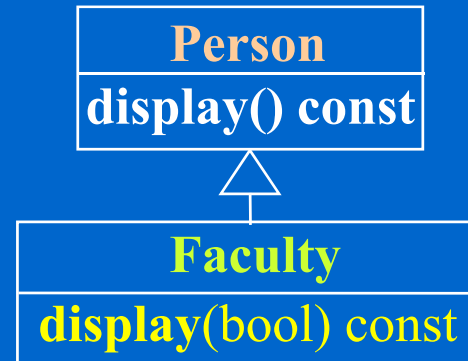
- ★ **display(bool)** does **NOT** *overload* **Person::display()**

different scope

Hiding

```
class Person {  
public:  
    virtual void display() const;  
};
```

```
class Faculty: public Person {  
public:  
    virtual void display(bool showDetail) const;  
};
```



```
Person p;  
p.display();  
Faculty f;  
f.display();  
f.display(true);
```

different signature

- ★ In **Faculty** class, **display(bool)** does **NOT** *override* **Person::display()**
⇒ different entries in virtual table

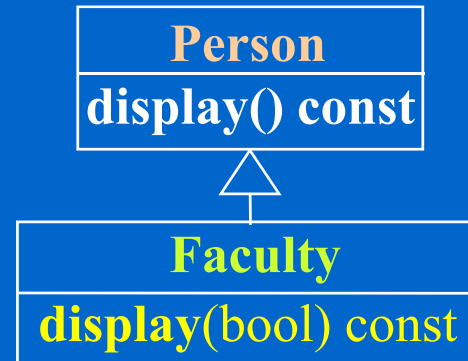
- ★ **display(bool)** does **NOT** *overload* **Person::display()** different scope

- ★ Only **display(bool)** can be called with a **Faculty** object, reference, or pointer, cannot see **display()**, although **Person::display()** can be called.

Hiding

```
class Person {  
public:  
    virtual void display() const;  
};
```

```
class Faculty: public Person {  
public:  
    virtual void display(bool showDetail) const;  
};
```



```
Person p;  
p.display();
```

```
Faculty f;  
f.display();  
f.display(true);
```

```
Person *ptr = &f;  
ptr->display();
```

different signature

- ★ In **Faculty** class, **display(bool)** does **NOT** *override* **Person::display()**

⇒ different entries in virtual table

- ★ **display(bool)** does **NOT** *overload* **Person::display()**

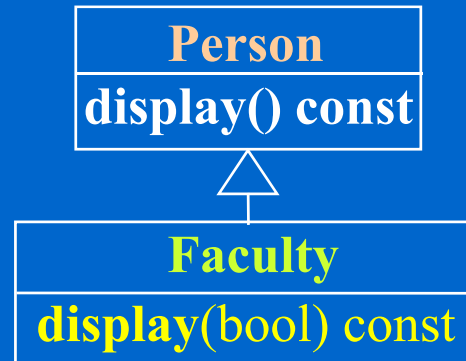
different scope

- ★ Only **display(bool)** can be called with a **Faculty** object, reference, or pointer, cannot see **display()**, although **Person::display()** can be called.

Hiding

```
class Person {  
public:  
    virtual void display() const;  
};
```

```
class Faculty: public Person {  
public:  
    virtual void display(bool showDetail) const;  
};
```



```
Person p;  
p.display();
```

```
Faculty f;  
f.display();  
f.display(true);
```

```
Person *ptr = &f;  
ptr->display();  
ptr->display(true);
```

different signature

- ★ In **Faculty** class, **display(bool)** does **NOT** *override* **Person::display()**

⇒ different entries in virtual table

- ★ **display(bool)** does **NOT** *overload* **Person::display()**

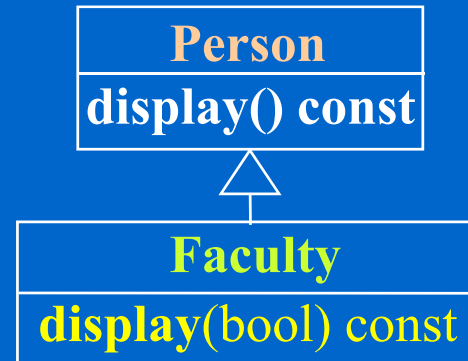
different scope

- ★ Only **display(bool)** can be called with a **Faculty** object, reference, or pointer, cannot see **display()**, although **Person::display()** can be called.

Hiding

```
class Person {  
public:  
    virtual void display() const;  
};
```

```
class Faculty: public Person {  
public:  
    virtual void display(bool showDetail) const;  
};
```



```
Person p;  
p.display();
```

```
Faculty f;  
f.display();  
f.display(true);
```

```
Person *ptr = &f;  
ptr->display();  
ptr->display(true);
```

```
Faculty *fptr = &f;  
fptr->display();
```

different signature

- ★ In **Faculty** class, **display**(bool) does **NOT** *override* **Person::display()**
⇒ different entries in virtual table

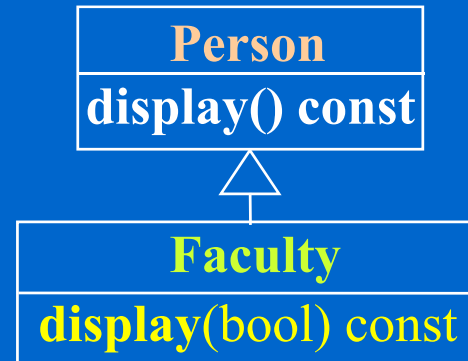
different scope

- ★ **display**(bool) does **NOT** *overload* **Person::display()**
- ★ Only **display**(bool) can be called with a **Faculty** object, reference, or pointer, cannot see **display()**, although **Person::display()** can be called.

Hiding

```
class Person {  
public:  
    virtual void display() const;  
};
```

```
class Faculty: public Person {  
public:  
    virtual void display(bool showDetail) const;  
};
```



```
Person p;  
p.display();
```

```
Faculty f;  
f.display();  
f.display(true);
```

```
Person *ptr = &f;  
ptr->display();  
ptr->display(true);
```

```
Faculty *fptr = &f;  
fptr->display();  
fptr->display(true);
```

different signature

- ★ In **Faculty** class, **display(bool)** does **NOT** *override* **Person::display()**

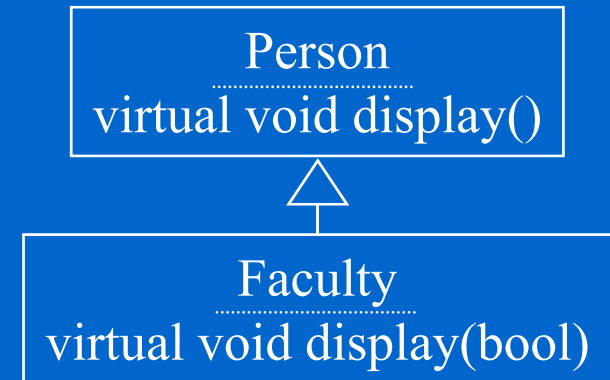
⇒ different entries in virtual table

- ★ **display(bool)** does **NOT** *overload* **Person::display()**

different scope

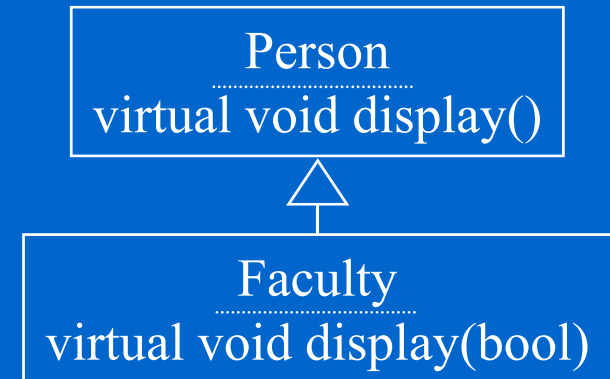
- ★ Only **display(bool)** can be called with a **Faculty** object, reference, or pointer, cannot see **display()**, although **Person::display()** can be called.

Member Function Calling Rule



Member Function Calling Rule

referrer.function()
referrer->function()

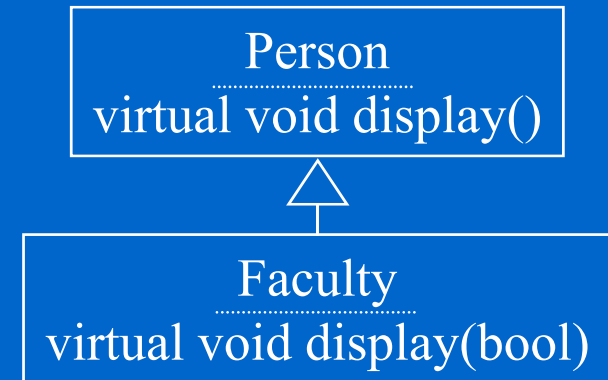


Member Function Calling Rule

referrer.function()

referrer->function()

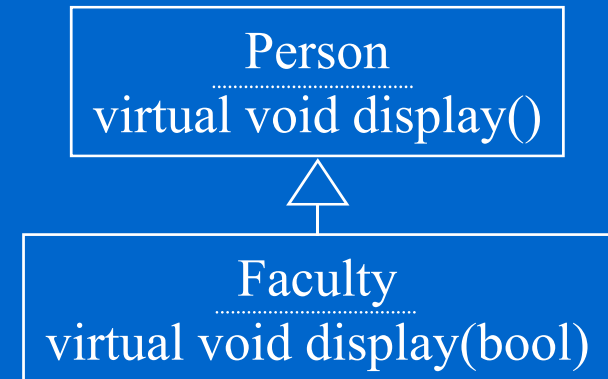
1. Search in the scope of the static type of the referrer pointer/reference/object to find the specified function in its **explicitly** defined functions



Member Function Calling Rule

referrer.function()
referrer->function()

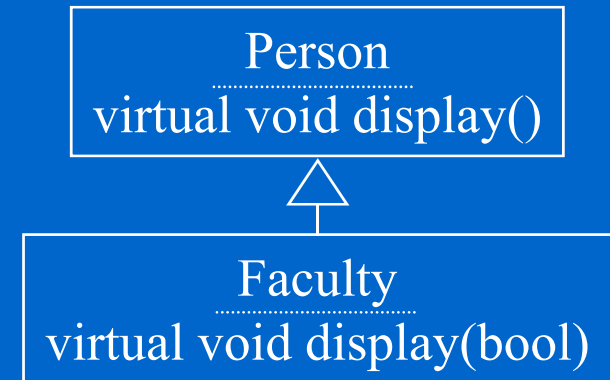
1. Search in the scope of the static type of the referrer pointer/reference/object to find the specified function in its **explicitly** defined functions
2. If it is a virtual function and referrer is a **pointer** (including **this** pointer) or **reference**, use dynamic binding otherwise use static one



Member Function Calling Rule

referrer.function()
referrer->function()

1. Search in the scope of the static type of the referrer pointer/reference/object to find the specified function in its **explicitly** defined functions
2. If it is a virtual function and referrer is a **pointer** (including **this** pointer) or **reference**, use dynamic binding otherwise use static one

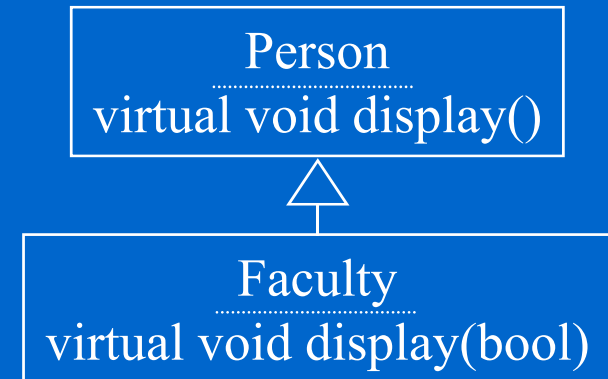


What functions are **explicit** in the scope of a class?

Member Function Calling Rule

referrer.function()
referrer->function()

1. Search in the scope of the static type of the referrer pointer/reference/object to find the specified function in its **explicitly** defined functions
2. If it is a virtual function and referrer is a **pointer** (including **this** pointer) or **reference**, use dynamic binding otherwise use static one



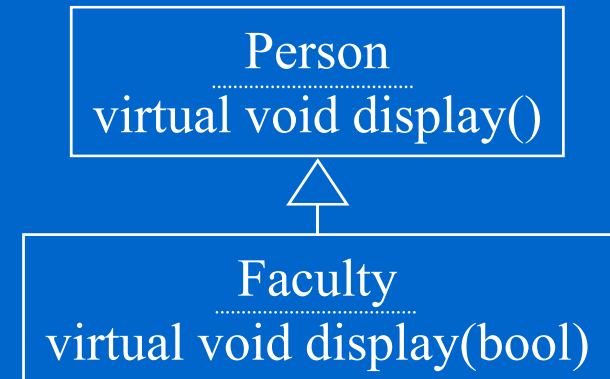
What functions are **explicit** in the scope of a class?

1. Defined in the class declaration

Member Function Calling Rule

referrer.function()
referrer->function()

1. Search in the scope of the static type of the referrer pointer/reference/object to find the specified function in its **explicitly** defined functions
2. If it is a virtual function and referrer is a **pointer** (including **this** pointer) or **reference**, use dynamic binding otherwise use static one



What functions are **explicit** in the scope of a class?

1. Defined in the class declaration
2. Search upward the inheritance tree, match all functions not hidden/overridden previously (by any function having the same name)

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Base b;

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

```
Base b;  
b.funcA(); // #1  
b.funcB(); // #2  
b.funcC(); // #3  
b.funcD(); // #4  
b.funcE(); // #5  
b.funcE(1,2); // #6
```

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Derived d;

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
Derived d;  
d.funcA(); // #1  
d.funcB(); // #2
```

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
Derived d;  
d.funcA(); // #1  
d.funcB(); // #2  
d.funcC(); // #7
```

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
Derived d;  
d.funcA(); // #1  
d.funcB(); // #2  
d.funcC(); // #7  
d.Base::funcC(); // hidden #3
```

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
Derived d;  
d.funcA(); // #1  
d.funcB(); // #2  
d.funcC(); // #7  
d.Base::funcC(); // hidden #3  
d.funcD(); // #8
```

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
Derived d;  
d.funcA(); // #1  
d.funcB(); // #2  
d.funcC(); // #7  
d.Base::funcC(); // hidden #3  
d.funcD(); // #8  
d.Base::funcD(); // overridden #4
```

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
Derived d;  
d.funcA(); // #1  
d.funcB(); // #2  
d.funcC(); // #7  
d.Base::funcC(); // hidden #3  
d.funcD(); // #8  
d.Base::funcD(); // overridden #4  
//d.funcE(); error 'funcE' does not take 0 param
```

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
Derived d;  
d.funcA(); // #1  
d.funcB(); // #2  
d.funcC(); // #7  
d.Base::funcC(); // hidden #3  
d.funcD(); // #8  
d.Base::funcD(); // overridden #4  
//d.funcE(); error 'funcE' does not take 0 param  
d.Base::funcE(); // hidden #5
```

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
Derived d;  
d.funcA(); // #1  
d.funcB(); // #2  
d.funcC(); // #7  
d.Base::funcC(); // hidden #3  
d.funcD(); // #8  
d.Base::funcD(); // overridden #4  
//d.funcE(); error 'funcE' does not take 0 parameters  
d.Base::funcE(); // hidden #5  
//d.funcE(1,2); error 'funcE' does not take 2 parameters
```

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
Derived d;  
d.funcA(); // #1  
d.funcB(); // #2  
d.funcC(); // #7  
d.Base::funcC(); // hidden #3  
d.funcD(); // #8  
d.Base::funcD(); // overridden #4  
//d.funcE(); error 'funcE' does not take 0 parameters  
d.Base::funcE(); // hidden #5  
//d.funcE(1,2); error 'funcE' does not take 2 parameters  
d.Base::funcE(1,2); // hidden #6
```

Calling Member Functions

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
Derived d;  
d.funcA(); // #1  
d.funcB(); // #2  
d.funcC(); // #7  
d.Base::funcC(); // hidden #3  
d.funcD(); // #8  
d.Base::funcD(); // overridden #4  
//d.funcE(); error 'funcE' does not take 0 parameters  
d.Base::funcE(); // hidden #5  
//d.funcE(1,2); error 'funcE' does not take 2 parameters  
d.Base::funcE(1,2); // hidden #6  
d.funcE(3); // #9
```

Calling Member Functions (cont'd)

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Calling Member Functions (cont'd)

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {  
};
```

Calling Member Functions (cont'd)

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {  
};  
FDerived1 fd1;
```

Virtual table: 2, 8, 5, 6, 9

Calling Member Functions (cont'd)

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {  
};  
FDerived1 fd1;  
fd1.funcA(); // #1  
fd1.funcB(); // #2
```

Virtual table: 2, 8, 5, 6, 9

Calling Member Functions (cont'd)

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {  
};  
FDerived1 fd1;  
fd1.funcA(); // #1  
fd1.funcB(); // #2  
fd1.funcC(); // #7
```

Virtual table: 2, 8, 5, 6, 9

Calling Member Functions (cont'd)

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {  
};  
FDerived1 fd1;  
fd1.funcA(); // #1  
fd1.funcB(); // #2  
fd1.funcC(); // #7  
fd1.Base::funcC(); // hidden #3
```

Virtual table: 2, 8, 5, 6, 9

Calling Member Functions (cont'd)

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {  
};  
FDerived1 fd1;  
fd1.funcA(); // #1  
fd1.funcB(); // #2  
fd1.funcC(); // #7  
fd1.Base::funcC(); // hidden #3  
fd1.funcD(); // #8
```

Virtual table: 2, 8, 5, 6, 9

Calling Member Functions (cont'd)

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {  
};  
FDerived1 fd1;  
fd1.funcA(); // #1  
fd1.funcB(); // #2  
fd1.funcC(); // #7  
fd1.Base::funcC(); // hidden #3  
fd1.funcD(); // #8  
fd1.Base::funcD(); // overridden #4
```

Virtual table: 2, 8, 5, 6, 9

Calling Member Functions (cont'd)

```
class Base {
public:
    void funcA() { cout << "Base::funcA() #1\n"; }
    virtual void funcB() { cout << "Base::funcB() #2\n"; }
    void funcC() { cout << "Base::funcC() #3\n"; }
    virtual void funcD() { cout << "Base::funcD() #4\n"; }
    virtual void funcE() { cout << "Base::funcE() #5\n"; }
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {
public:
    void funcC() {
        cout << "Derived::funcC() #7\n";
    }
    void funcD() {
        cout << "Derived::funcD() #8\n";
    }
    virtual void funcE(int) {
        cout << "Derived::funcE(int) #9\n";
    }
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {
};
FDerived1 fd1;
fd1.funcA(); // #1
fd1.funcB(); // #2
fd1.funcC(); // #7
fd1.Base::funcC(); // hidden #3
fd1.funcD(); // #8
fd1.Base::funcD(); // overridden #4
//fd1.funcE(); error 'funcE' does not take 0 param
```

Virtual table: 2, 8, 5, 6, 9

Calling Member Functions (cont'd)

```
class Base {
public:
    void funcA() { cout << "Base::funcA() #1\n"; }
    virtual void funcB() { cout << "Base::funcB() #2\n"; }
    void funcC() { cout << "Base::funcC() #3\n"; }
    virtual void funcD() { cout << "Base::funcD() #4\n"; }
    virtual void funcE() { cout << "Base::funcE() #5\n"; }
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {
public:
    void funcC() {
        cout << "Derived::funcC() #7\n";
    }
    void funcD() {
        cout << "Derived::funcD() #8\n";
    }
    virtual void funcE(int) {
        cout << "Derived::funcE(int) #9\n";
    }
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {
};
FDerived1 fd1;
fd1.funcA(); // #1
fd1.funcB(); // #2
fd1.funcC(); // #7
fd1.Base::funcC(); // hidden #3
fd1.funcD(); // #8
fd1.Base::funcD(); // overridden #4
//fd1.funcE(); error 'funcE' does not take 0 param
fd1.Base::funcE(); // hidden #5
```

Virtual table: 2, 8, 5, 6, 9

Calling Member Functions (cont'd)

```
class Base {
public:
    void funcA() { cout << "Base::funcA() #1\n"; }
    virtual void funcB() { cout << "Base::funcB() #2\n"; }
    void funcC() { cout << "Base::funcC() #3\n"; }
    virtual void funcD() { cout << "Base::funcD() #4\n"; }
    virtual void funcE() { cout << "Base::funcE() #5\n"; }
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {
public:
    void funcC() {
        cout << "Derived::funcC() #7\n";
    }
    void funcD() {
        cout << "Derived::funcD() #8\n";
    }
    virtual void funcE(int) {
        cout << "Derived::funcE(int) #9\n";
    }
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {
};
FDerived1 fd1;
fd1.funcA(); // #1
fd1.funcB(); // #2
fd1.funcC(); // #7
fd1.Base::funcC(); // hidden #3
fd1.funcD(); // #8
fd1.Base::funcD(); // overridden #4
//fd1.funcE(); error 'funcE' does not take 0 param
fd1.Base::funcE(); // hidden #5
//fd1.funcE(1,2); error 'funcE' does not take 2 pa
```

Virtual table: 2, 8, 5, 6, 9

Calling Member Functions (cont'd)

```
class Base {
public:
    void funcA() { cout << "Base::funcA() #1\n"; }
    virtual void funcB() { cout << "Base::funcB() #2\n"; }
    void funcC() { cout << "Base::funcC() #3\n"; }
    virtual void funcD() { cout << "Base::funcD() #4\n"; }
    virtual void funcE() { cout << "Base::funcE() #5\n"; }
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {
public:
    void funcC() {
        cout << "Derived::funcC() #7\n";
    }
    void funcD() {
        cout << "Derived::funcD() #8\n";
    }
    virtual void funcE(int) {
        cout << "Derived::funcE(int) #9\n";
    }
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {
};
FDerived1 fd1;
fd1.funcA(); // #1
fd1.funcB(); // #2
fd1.funcC(); // #7
fd1.Base::funcC(); // hidden #3
fd1.funcD(); // #8
fd1.Base::funcD(); // overridden #4
//fd1.funcE(); error 'funcE' does not take 0 param
fd1.Base::funcE(); // hidden #5
//fd1.funcE(1,2); error 'funcE' does not take 2 param
fd1.Base::funcE(1,2); // hidden #6
```

Virtual table: 2, 8, 5, 6, 9

Calling Member Functions (cont'd)

```
class Base {
public:
    void funcA() { cout << "Base::funcA() #1\n"; }
    virtual void funcB() { cout << "Base::funcB() #2\n"; }
    void funcC() { cout << "Base::funcC() #3\n"; }
    virtual void funcD() { cout << "Base::funcD() #4\n"; }
    virtual void funcE() { cout << "Base::funcE() #5\n"; }
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {
public:
    void funcC() {
        cout << "Derived::funcC() #7\n";
    }
    void funcD() {
        cout << "Derived::funcD() #8\n";
    }
    virtual void funcE(int) {
        cout << "Derived::funcE(int) #9\n";
    }
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {
};
FDerived1 fd1;
fd1.funcA(); // #1
fd1.funcB(); // #2
fd1.funcC(); // #7
fd1.Base::funcC(); // hidden #3
fd1.funcD(); // #8
fd1.Base::funcD(); // overridden #4
//fd1.funcE(); error 'funcE' does not take 0 param
fd1.Base::funcE(); // hidden #5
//fd1.funcE(1,2); error 'funcE' does not take 2 param
fd1.Base::funcE(1,2); // hidden #6
fd1.funcE(3); // #9
```

Virtual table: 2, 8, 5, 6, 9

Calling Member Functions (cont'd)

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {  
};
```

Virtual table: 2, 8, 5, 6, 9

Calling Member Functions (cont'd)

```
class Base {  
public:  
    void funcA() { cout << "Base::funcA() #1\n"; }  
    virtual void funcB() { cout << "Base::funcB() #2\n"; }  
    void funcC() { cout << "Base::funcC() #3\n"; }  
    virtual void funcD() { cout << "Base::funcD() #4\n"; }  
    virtual void funcE() { cout << "Base::funcE() #5\n"; }  
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }  
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {  
public:  
    void funcC() {  
        cout << "Derived::funcC() #7\n";  
    }  
    void funcD() {  
        cout << "Derived::funcD() #8\n";  
    }  
    virtual void funcE(int) {  
        cout << "Derived::funcE(int) #9\n";  
    }  
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {  
};
```

Virtual table: 2, 8, 5, 6, 9

Virtual table: 2, 8, 10, 11, 9

```
class FDerived2: public Derived {  
public:  
    void funcE() {  
        cout << "FDerived2::funcE() #10\n";  
    }  
    void funcE(int, int) {  
        cout << "FDerived2::funcE(int, int) #11\n";  
    }  
};
```

Calling Member Functions (cont'd)

```
class Base {
public:
    void funcA() { cout << "Base::funcA() #1\n"; }
    virtual void funcB() { cout << "Base::funcB() #2\n"; }
    void funcC() { cout << "Base::funcC() #3\n"; }
    virtual void funcD() { cout << "Base::funcD() #4\n"; }
    virtual void funcE() { cout << "Base::funcE() #5\n"; }
    virtual void funcE(int, int) { cout << "Base::funcE(int,int) #6\n"; }
};
```

Virtual table: 2, 4, 5, 6

Explicit: 1,2,3,4,5,6

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

Virtual table: 2, 8, 5, 6, 9

```
class Derived: public Base {
public:
    void funcC() {
        cout << "Derived::funcC() #7\n";
    }
    void funcD() {
        cout << "Derived::funcD() #8\n";
    }
    virtual void funcE(int) {
        cout << "Derived::funcE(int) #9\n";
    }
};
```

Explicit: 1,2,7,8,9
Implicit: 3,4,5,6

```
class FDerived1: public Derived {
};
```

Virtual table: 2, 8, 5, 6, 9

Virtual table: 2, 8, 10, 11, 9

```
class FDerived2: public Derived {
public:
    void funcE() {
        cout << "FDerived2::funcE() #10\n";
    }
    void funcE(int, int) {
        cout << "FDerived2::funcE(int, int) #11\n";
    }
};
```

Explicit: 1,2,7,8,10,11
Implicit: 3,4,5,6,9

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4
```

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10
```

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5
```

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11
```

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6
```

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6  
//fd2.funcE(3); // error 'funcE' does not take 1 params
```

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6  
//fd2.funcE(3); // error 'funcE' does not take 1 params  
fd2.Derived::funcE(3); // hidden #9
```

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6  
//fd2.funcE(3); // error 'funcE' does not take 1 params  
fd2.Derived::funcE(3); // hidden #9
```

```
Base *bp=&fd2;  
bp->funcB(); // #2
```

Virtual table: 2, 8, 10, 11, 9

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6  
//fd2.funcE(3); // error 'funcE' does not take 1 params  
fd2.Derived::funcE(3); // hidden #9
```

```
Base *bp=&fd2;  
bp->funcB(); // #2  
bp->funcD(); // overriding Base::funcD() #8
```

Virtual table: 2, 8, 10, 11, 9

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6  
//fd2.funcE(3); // error 'funcE' does not take 1 params  
fd2.Derived::funcE(3); // hidden #9
```

```
Base *bp=&fd2;  
bp->funcB(); // #2  
bp->funcD(); // overriding Base::funcD() #8  
bp->funcE(); // overriding Base::funcE() #10
```

Virtual table: 2, 8, 10, 11, 9

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6  
//fd2.funcE(3); // error 'funcE' does not take 1 params  
fd2.Derived::funcE(3); // hidden #9
```

```
Base *bp=&fd2;  
bp->funcB(); // #2  
bp->funcD(); // overriding Base::funcD() #8  
bp->funcE(); // overriding Base::funcE() #10  
bp->funcE(1,2); // overriding Base::funcE(1,2) #11
```

Virtual table: 2, 8, 10, 11, 9

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6  
//fd2.funcE(3); // error 'funcE' does not take 1 params  
fd2.Derived::funcE(3); // hidden #9
```

Virtual table: 2, 8, 10, 11, 9

```
Base *bp=&fd2;  
bp->funcB(); // #2  
bp->funcD(); // overriding Base::funcD() #8  
bp->funcE(); // overriding Base::funcE() #10  
bp->funcE(1,2); // overriding Base::funcE(1,2) #11
```

Virtual table: 2, 8, 10, 11, 9

```
Derived *dp=&fd2;  
dp->funcB(); // #2
```

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6  
//fd2.funcE(3); // error 'funcE' does not take 1 params  
fd2.Derived::funcE(3); // hidden #9
```

Virtual table: 2, 8, 10, 11, 9

```
Base *bp=&fd2;  
bp->funcB(); // #2  
bp->funcD(); // overriding Base::funcD() #8  
bp->funcE(); // overriding Base::funcE() #10  
bp->funcE(1,2); // overriding Base::funcE(1,2) #11
```

Virtual table: 2, 8, 10, 11, 9

```
Derived *dp=&fd2;  
dp->funcB(); // #2  
dp->funcD(); // #8
```

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6  
//fd2.funcE(3); // error 'funcE' does not take 1 params  
fd2.Derived::funcE(3); // hidden #9
```

```
Base *bp=&fd2;  
bp->funcB(); // #2  
bp->funcD(); // overriding Base::funcD() #8  
bp->funcE(); // overriding Base::funcE() #10  
bp->funcE(1,2); // overriding Base::funcE(1,2) #11
```

Virtual table: 2, 8, 10, 11, 9

Virtual table: 2, 8, 10, 11, 9

```
Derived *dp=&fd2;  
dp->funcB(); // #2  
dp->funcD(); // #8  
// dp->funcE(); // hidden
```

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6  
//fd2.funcE(3); // error 'funcE' does not take 1 params  
fd2.Derived::funcE(3); // hidden #9
```

```
Base *bp=&fd2;  
bp->funcB(); // #2  
bp->funcD(); // overriding Base::funcD() #8  
bp->funcE(); // overriding Base::funcE() #10  
bp->funcE(1,2); // overriding Base::funcE(1,2) #11
```

Virtual table: 2, 8, 10, 11, 9

Virtual table: 2, 8, 10, 11, 9

```
Derived *dp=&fd2;  
dp->funcB(); // #2  
dp->funcD(); // #8  
// dp->funcE(); // hidden  
dp->Base::funcE(); // #5, static
```

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6  
//fd2.funcE(3); // error 'funcE' does not take 1 params  
fd2.Derived::funcE(3); // hidden #9
```

```
Base *bp=&fd2;  
bp->funcB(); // #2  
bp->funcD(); // overriding Base::funcD() #8  
bp->funcE(); // overriding Base::funcE() #10  
bp->funcE(1,2); // overriding Base::funcE(1,2) #11
```

Virtual table: 2, 8, 10, 11, 9

Virtual table: 2, 8, 10, 11, 9

```
Derived *dp=&fd2;  
dp->funcB(); // #2  
dp->funcD(); // #8  
// dp->funcE(); // hidden  
dp->Base::funcE(); // #5, static  
// dp->funcE(1,2); // hidden
```

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6  
//fd2.funcE(3); // error 'funcE' does not take 1 params  
fd2.Derived::funcE(3); // hidden #9
```

```
Base *bp=&fd2;  
bp->funcB(); // #2  
bp->funcD(); // overriding Base::funcD() #8  
bp->funcE(); // overriding Base::funcE() #10  
bp->funcE(1,2); // overriding Base::funcE(1,2) #11
```

Virtual table: 2, 8, 10, 11, 9

Virtual table: 2, 8, 10, 11, 9

```
Derived *dp=&fd2;  
dp->funcB(); // #2  
dp->funcD(); // #8  
// dp->funcE(); // hidden  
dp->Base::funcE(); // #5, static  
// dp->funcE(1,2); // hidden  
dp->Base::funcE(1,2); // #6, static
```

Calling Member Functions (cont'd)

```
FDerived2 fd2;  
fd2.funcA(); // #1  
fd2.funcB(); // #2  
fd2.funcC(); // Derived::funcC() #7  
fd2.Base::funcC(); // hidden by Derived::funcC() #3  
fd2.funcD(); // Derived::funcD() #8  
fd2.Base::funcD(); // overridden by Derived::funcD() #4  
fd2.funcE(); // overriding Base::funcE() #10  
fd2.Base::funcE(); // overridden #5  
fd2.funcE(1,2); // overriding Base::funcE(int,int) #11  
fd2.Base::funcE(1,2); // overridden #6  
//fd2.funcE(3); // error 'funcE' does not take 1 params  
fd2.Derived::funcE(3); // hidden #9
```

```
Base *bp=&fd2;  
bp->funcB(); // #2  
bp->funcD(); // overriding Base::funcD() #8  
bp->funcE(); // overriding Base::funcE() #10  
bp->funcE(1,2); // overriding Base::funcE(1,2) #11
```

Virtual table: 2, 8, 10, 11, 9

Virtual table: 2, 8, 10, 11, 9

```
Derived *dp=&fd2;  
dp->funcB(); // #2  
dp->funcD(); // #8  
// dp->funcE(); // hidden  
dp->Base::funcE(); // #5, static  
// dp->funcE(1,2); // hidden  
dp->Base::funcE(1,2); // #6, static  
dp->funcE(3); // #9
```