

# Lecture 18

## C++ Function Overriding

The derived classes inherit features of the base class.

Suppose, the same function is defined in both the derived class and the based class. Now if we call this function using the object of the derived class, the function of the derived class is executed.

This is known as **function overriding** in C++. The function in derived class overrides the function in base class.

### Example 1: C++ Function Overriding

```
// C++ program to demonstrate function overriding

#include <iostream>
using namespace std;

class Base {
public:
    void print() {
        cout << "Base Function" << endl;
    }
};

class Derived : public Base {
public:
    void print() {
        cout << "Derived Function" << endl;
    }
}
```

```
};

int main() {
    Derived derived1;
    derived1.print();
    return 0;
}
```

## Output

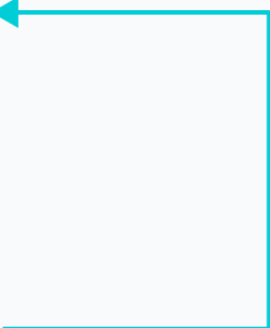
Derived Function

Here, the same function `print()` is defined in both `Base` and `Derived` classes. So, when we call `print()` from the `Derived` object `derived1`, the `print()` from `Derived` is executed by overriding the function in `Base`.

```
class Base {
public:
    void print() {
        // code
    }
};

class Derived : public Base {
public:
    void print() {
        // code
    }
};

int main() {
    Derived derived1;
    derived1.print();
    return 0;
}
```



As we can see, the function was overridden because we called the function from an object of the `Derived` class.

Had we called the `print()` function from an object of the `Base` class, the function would not have been overridden.

```
// Call function of Base class
Base base1;
base1.print(); // Output: Base Function
```

## Access Overridden Function in C++

To access the overridden function of the base class, we use the scope resolution operator `::`.

We can also access the overridden function by using a pointer of the base class to point to an object of the derived class and then calling the function from that pointer.

### Example 2: C++ Access Overridden Function to the Base Class

```
// C++ program to access overridden function
// in main() using the scope resolution operator ::

#include <iostream>
using namespace std;

class Base {
public:
    void print() {
        cout << "Base Function" << endl;
    }
};

class Derived : public Base {
public:
    void print() {
```

```
        cout << "Derived Function" << endl;
    }
};

int main() {
    Derived derived1, derived2;
    derived1.print();

    // access print() function of the Base class
    derived2.Base::print();

    return 0;
}
```

## Output

```
Derived Function
Base Function
```

Here, this statement

```
derived2.Base::print();
```

accesses the `print()` function of the Base class.

```
class Base {
public:
    void print() {
        // code
    }
};

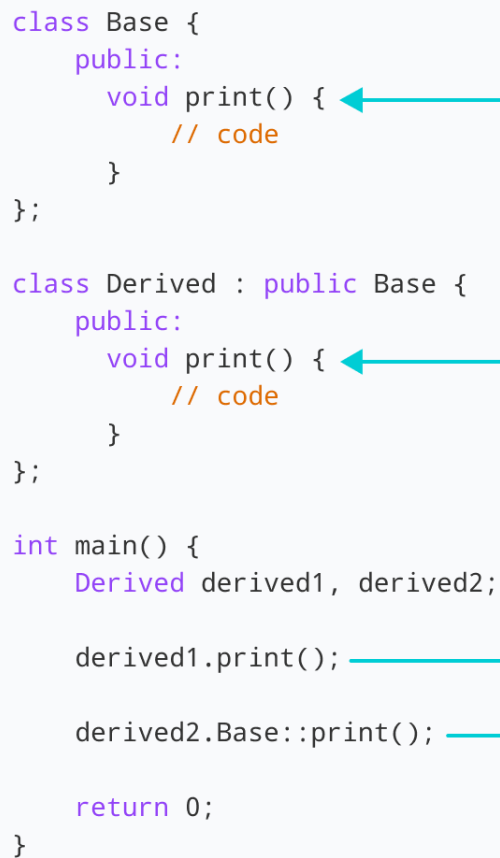
class Derived : public Base {
public:
    void print() {
        // code
    }
};

int main() {
    Derived derived1, derived2;

    derived1.print();

    derived2.Base::print();

    return 0;
}
```



### Example 3: C++ Call Overridden Function From Derived Class

```
// C++ program to call the overridden function
// from a member function of the derived class

#include <iostream>
using namespace std;

class Base {
public:
    void print() {
        cout << "Base Function" << endl;
    }
}
```

```
};

class Derived : public Base {
public:
    void print() {
        cout << "Derived Function" << endl;

        // call overridden function
        Base::print();
    }
};

int main() {
    Derived derived1;
    derived1.print();
    return 0;
}
```

## Output

```
Derived Function
Base Function
```

In this program, we have called the overridden function inside the `Derived` class itself.

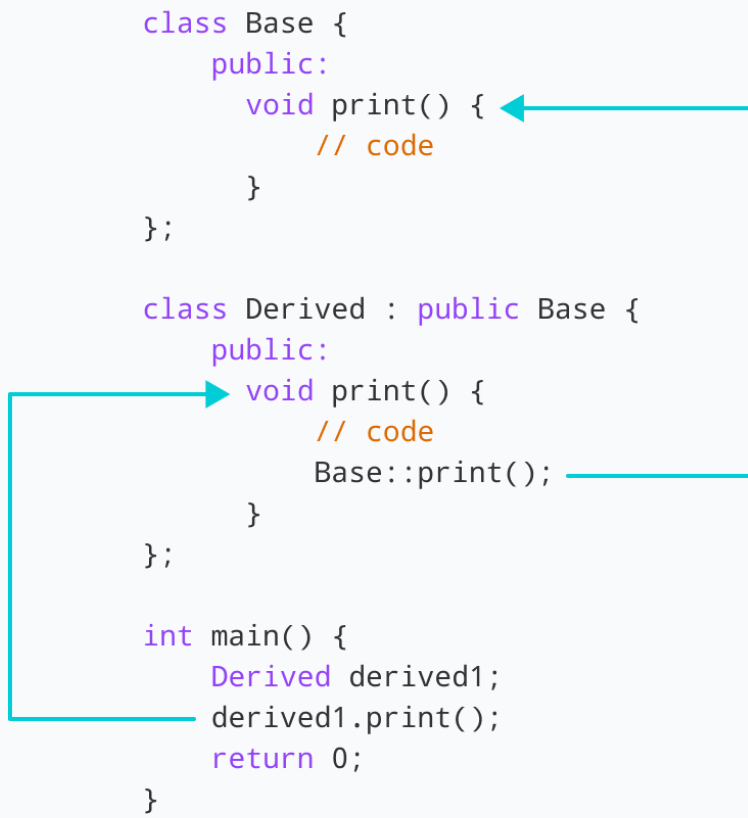
```
class Derived : public Base {
public:
    void print() {
        cout << "Derived Function" << endl;
        Base::print();
    }
};
```

Notice the code `Base::print();`, which calls the overridden function inside the `Derived` class.

```
class Base {
public:
    void print() {
        // code
    }
};

class Derived : public Base {
public:
    void print() {
        // code
        Base::print();
    }
};

int main() {
    Derived derived1;
    derived1.print();
    return 0;
}
```



## Example 4: C++ Call Overridden Function Using Pointer

```
// C++ program to access overridden function using pointer
// of Base type that points to an object of Derived class

#include <iostream>
using namespace std;

class Base {
public:
    void print() {
        cout << "Base Function" << endl;
    }
};
```

```

class Derived : public Base {
public:
    void print() {
        cout << "Derived Function" << endl;
    }
};

int main() {
    Derived derived1;

    // pointer of Base type that points to derived1
    Base* ptr = &derived1;

    // call function of Base class using ptr
    ptr->print();

    return 0;
}

```

## Output

```
Base Function
```

In this program, we have created a pointer of `Base` type named `ptr`. This pointer points to the `Derived` object `derived1`.

```

// pointer of Base type that points to derived1
Base* ptr = &derived1;

```

When we call the `print()` function using `ptr`, it calls the overridden function from `Base`.

```

// call function of Base class using ptr
ptr->print();

```

This is because even though `ptr` points to a `Derived` object, it is actually of `Base` type. So, it calls the member function of `Base`.



In order to override the `Base` function instead of accessing it, we need to use virtual functions in the `Base` class.