

Member Function Inheritance

Member function inheritance

- A child class inherits the non-private member functions of its parent class

```
class Vehicle {  
    public:  
        void start() { cout << "Engine started: "; }  
};  
class Aeroplane : public Vehicle {  
    public:  
        /// void start(); Inherited from Vehicle ///  
};
```

```
plane.start(); // Calls start() on an instance of Aeroplane. Prints out "Engine started: "
```

Re-implementing Base Class Member Functions

- We can use child member functions to re-implement base class member functions:

```
class Aeroplane : public Vehicle {  
    ...  
public:  
    void start() {  
        cout << "Carrying out checks... ";  
        cout << "Ready for take-off!";  
    }  
};
```

```
plane.start();
```

```
// Prints out "Carrying out checks... Ready for take-off!"
```

Calling Base Class Member Functions

- We can also use child member functions to extend base class member functions:

```
class Aeroplane : public Vehicle {  
    ...  
public:  
    void start() {  
        cout << "Carrying out checks... ";  
        Vehicle::start();           // Call base class member function  
        cout << "Ready for take-off!";  
    }  
};
```

Overloading member functions

- We can overload inherited member functions

```
class Vehicle {  
public:  
    void accelerate() { ... }  
};
```

```
class Aeroplane : public Vehicle {  
public:  
    void accelerate(int height);  
};
```

// Overload of accelerate()

Hidden member functions

- If we define an overloaded function in the derived class, it will "hide" all the inherited member functions of that name

```
class Aeroplane : public Vehicle {  
    public:  
    /// void accelerate(); Inherited from Base - or is it? ///  
    void accelerate(int height); // Overload of accelerate() - hides the inherited version  
};  
  
plane.accelerate(); // Error! No matching function for Aeroplane::accelerate()
```

- We could solve this by defining the inherited functions in the derived class

```
class Aeroplane : public Vehicle {  
    void accelerate() { Vehicle::accelerate(); } // Call parent version of accelerate()  
    void accelerate(int height);                // Overload of accelerate()  
};
```

- A better solution is to put the parent class version in the derived class with "using" in front of it

```
using Vehicle::accelerate;           // Make parent version of accelerate() available  
void accelerate(int height);         // Overload of accelerate()
```

- Note that there are no brackets after the parent version
- If the parent has more than one version of accelerate (overloads), this will make all of them available in the child