Constructor and inheritance

The compiler automatically calls a base class constructor before executing the derived class constructor. The compiler's default action is to call the default constructor in the base class. You can specify which of several base class constructors should be called during the creation of a derived class object.

This is done by specifying the arguments to the selected base class constructor in the definition of the derived class constructor.

```
class Rectangle
{
private :
	float length;
	float width;
public:
	Rectangle ()
	{
		length = 0;
		width = 0;
	}

	Rectangle (float len, float wid)
	{
		length = len;
		width = wid;
	}

	float area()
	{
		return length * width ;
	}
};

class Box : public Rectangle
{
private :
	float height;
public:
	Box ()
	{
		height = 0;
	}
```

```
        Box (float len, float wid, float ht) : Rectangle(len, wid)
        {
                height = ht;
        }

        float volume()
        {
                return area() * height;
        }
};

int main ()
{
        Box bx;
        Box cx(4,8,5);
        cout << bx.volume() << endl;
        cout << cx.volume() << endl;
        return 0;
}
```

output :

0
160

Overriding Base Class Functions

A derived class can override a member function of its base class by defining a derived class member function with the same name and parameter list.
It is often useful for a derived class to define its own version of a member function inherited from its base class. This may be done to specialize the member function to the needs of the derived class. When this happens, the base class member function is said to be overridden by the derived class.

```
class mother
{
public:
	void display ()
	{
		cout << "mother: display function\n";
	}
};

class daughter : public mother
{
public:
	void display ()
	{
		cout << "daughter: display function\n\n";
	}
};

int main ()
{
	daughter rita;
	rita.display();
	return 0;
}
```

output:
daughter: display function

Gaining Access to an Overridden Function

It is occasionally useful to be able to call the overridden version. This is done by using the scope resolution operator to specify the class of the overridden member function being accessed.

```
class daughter : public mother
{
public:
	void display ()
	{
		cout << "daughter: display function\n\n";
		mother::display();
	}
};
```

output:

daughter: display function

mother: display function

Virtual Base Class

Multipath inheritance may lead to duplication of inherited members from a grandparent base class. This may be avoided by making the common base class a virtual base class. When a class is made a virtual base class, C++ takes necessary care to see that only one copy of that class is inherited.

```
class A
{
       .....
       .....
};

class B1 : virtual public A
{
       .....
       .....
};

class B2 : virtual public A
{
       .....
       .....
};

class C : public B1, public B2
{
       .....// only one copy of A
       .....// will be inherited
};
```