Inheritance

Inheritance is one of the important features of object oriented programming language. Inheritance is the capability of getting the properties and characteristics of one class to other class. The C++ allows user to create a new class from an existing class. The class whose properties are inherited by other class is called base class or parent class or super class and the class which inherits properties of other class is called derived class or child class or sub class. The derived class inherits all the features from a base class and it can have additional features of its own. All the members of a class except private members are inherited to derived class.

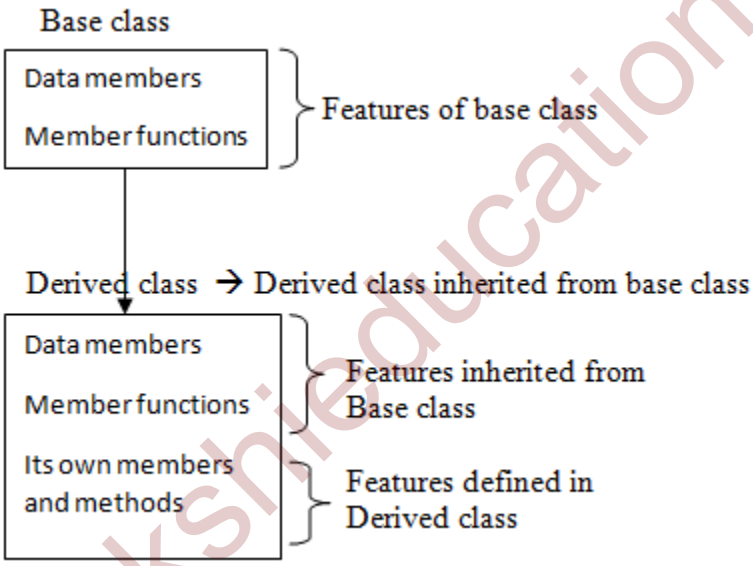

Figure1: Inheritance

Syntax:

class Derivedclass_Name : access_mode/visibility_mode Baseclass_Name

{

};

Inheritance visibility mode

The visibility mode in inheritance indicates the availability of class member of base class in the derived class; the access modifiers in the class can be private, protected or public.

Public Inheritance

This is the most used inheritance visibility mode. Here the protected member of base class becomes protected members of derived class and a public member becomes public member.

Syntax:

```
class BaseClass : public DerivedClass
{
        statements;
};
```

Private Inheritance

In this mode of inheritance the protected and public members of base class becomes private members of derived class. By default every class is private inheritance.

Syntax:

```
class BaseClass : private DerivedClass
{
        statements;
};
 or
class BaseClass : DerivedClass
{
        statements;
};
```

Protected Inheritance

In this mode of inheritance the public and protected members of base class becomes a protected member of derived class.

Syntax:

```
class BaseClass : protected DeivedClass
{
         statements;
};
```

Base Class	Derived Class		
	Private mode	**Protected mode**	**Public mode**
Private mode	Not inherited	Not inherited	Not inherited
Protected mode	Private	Protected	Protected
Public mode	Private	Protected	Public

Table1: Visibility modes

Accessibility	Private	Protected	Pubic
Accessible from same class	Yes	Yes	Yes
Accessible from derived class	No	Yes	Yes
Accessible from outside	No	No	Yes

Table2: Access level from different classes

Example:

```
#include <iostream>

using namespace std;

class rectangle
{
	public:
		float l,b;
	public:
		void get_sides()
		{
			cout << "Enter the Lenght: ";
			cin >> l;
			cout << "Enter the Breadth: ";
			cin >> b;
```

```
		}
};

class area : public rectangle  // derived class from base class rectangle
{
	public:
		float area_cal()
		{
			return (l*b);
		}
};

class perimeter : public rectangle  // derived class from base class rectangle
{
	public:
		float perimeter_cal()
		{
			return (2*(l+b));
		}
};


int main()
{

	area a;
	a.get_sides();
	cout << "Area of Rectagle is: " << a.area_cal() << endl;
	perimeter p;
	p.get_sides();
	cout << "Perimeter of Rectangle is: " << p.perimeter_cal() << endl;
	system("pause");
	return 0;

}
```

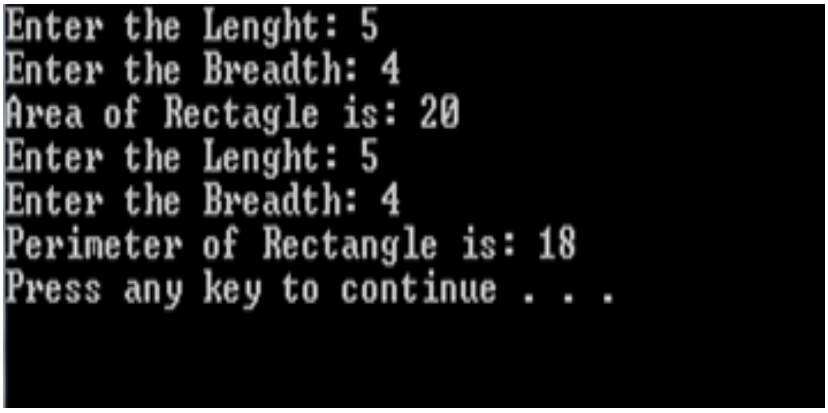

Figure2: Output of the program

In the above example the 'l' and 'b' variables in the base class are public member variables, so these variables can be accesses from both derived class and from main function. If these variables are protected, then these data members can be accessed from derived class but not accessible from outside the class and if these are private members, then these members are not accessible to derived class also.

The purpose of Inheritance

- Code reusability
- Use of virtual keyword
- Method overriding

Types of Inheritance levels

There are five types of inheritance level and they are as follows

1. Single level Inheritance
2. Multi level Inheritance
3. Multiple Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

Single level Inheritance

In single level inheritance one derived class inherits from only one base class.

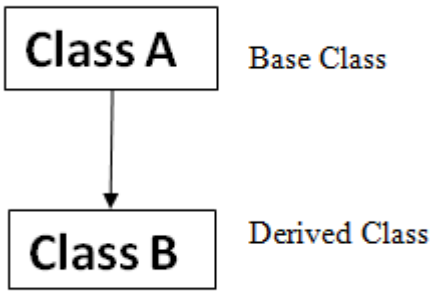

Figure3: Single Level Inheritance

Example program for single inheritance:

```
#include <iostream>

using namespace std;

class Base
{
private:
	int val1;
public:
	void get_value1(int x) // function definition inside the class
	{
		val1 = x;
	}
	int get_val1();  // function declaration

	/*int print_value1()
	{
		return val1;
	}*/

};

int Base :: get_val1() // function definition outside class
	{
		return val1;
	}
```

```
class Derived : public Base  // Single level inheritance
{

private:
	int val2, sum;

public:
	void get_value2(int y)
	{
		val2 = y;
	}

	void addition()
	{
		sum = get_val1()+val2;
	}

	void print()
	{
		cout << "Value1 is:" << get_val1() << endl;
		cout << "Value2 is:" << val2 << endl;
		cout << "Addition is:" << sum << endl;
	}
};

int main()
{
	int a,b;
	Derived obj;
	cout << "Enter Value of a:" ;
	cin >> a;
	cout << "Enter Value of b:" ;
	cin >>b;
	obj.get_value1(a);   // public mode
	obj.get_value2(b);   // public mode
	obj.addition();      // public mode
	obj.print();         // public mode
	system("pause");
	return 0;
```

```
}
```

```
Enter Value of a:12
Enter Value of b:54
Value1 is:12
Value2 is:54
Addition is:66
Press any key to continue . . .
```

Figure4: Output of the program

Multi level Inheritance

In multi level inheritance the derived class inherits from a base class, which in turn inherits from some other class.

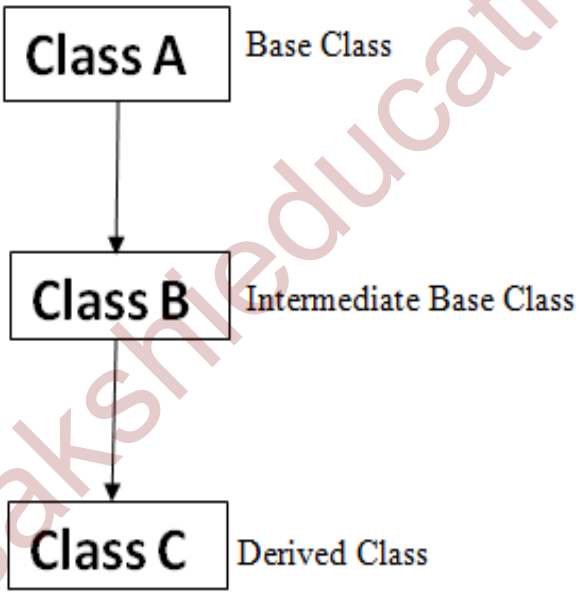

Figure5: Multi Level Inheritance

Example program for multi level inheritance:

```
#include <iostream>

using namespace std;

class A       // base class
{
public:
       void print_classA()  // base class function
```

```
	{
		cout << "Base Class  --- Class A" << endl;
		cout << "------------------------" << endl;
	}

};

class B : public A  // derived class from class A
{
public:
	void print_classB()
	{
		cout <<"Derived class B of class A calling class A function using singlelevelInheritance";
	}

};

class C : public B  // derived class from class B
{
public:
	void print_classC()
	{
		cout <<"Derived class C of class B calling class A function using multilevel Inheritance";
	}

};

int main()
{
	C obj_C;  // class C object
	obj_C.print_classC();
	obj_C.print_classA();  //calling class A function using derived class of class C using Multi level Inheritance

	//--------------------
	B obj_B; // class B object
	obj_B.print_classB();
```

```
        obj_B.print_classA();  //calling class A function using derived class of class C
using single level Inheritance

        //----------------------
        A obj_A;  //class A object
        obj_A.print_classA();  // calling class A function using class A object
        system("pause");
        return 0;
}
```

In the above example program class A is the base class and class B is derived class of class A and class C is the derived class of B. In this program we are calling the "print_classA" function using object of class C using multi level inheritance and using object of class B using single level inheritance.

```
Derived class C of class B calling class A function using multilevel Inheritance
Base Class  --- Class A
-------------------------
Derived class B of class A calling class A function using singlelevelInheritance
Base Class  --- Class A
-------------------------
Base Class  --- Class A
-------------------------
Press any key to continue . . .
```

Figure6: Output of the program

Multiple Inheritance

In multiple Inheritances, a single derived class may inherit from two or more the two base classes.

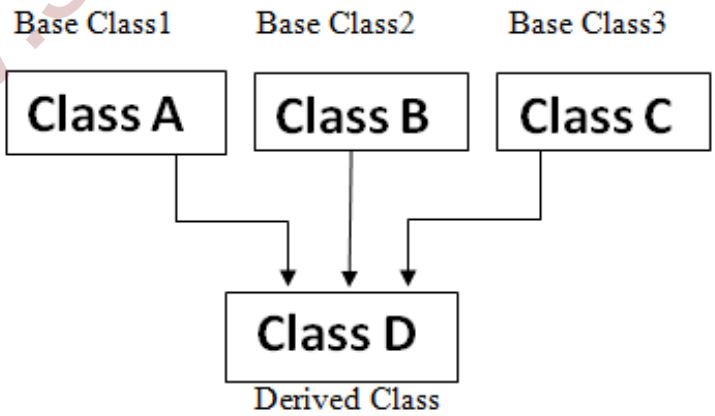

Figure7: Multiple Inheritance

Example program for multiple inheritance:

```
#include <iostream>

using namespace std;

class A  // class A
{
protected:
	int val1;
public:
	void get_value1(int x)
	{
		val1 = x;
	}
};

class B  // class B
{
protected:
	int val2;
public:
	void get_value2(int y)
	{
		val2 = y;
	}
};

class C : public A, public B // class C and multiple inheritance from class A and class B
{
private:
	int sum;

public:
	void addition()
	{
		sum = val1+val2;   // val1 is from class A and val2 is from class B  --
members from different classes
	}
```

```
        void print()
        {
                cout << "Value1 is:" << val1 << endl;
                cout << "Value2 is:" << val2 << endl;
                cout << "Addition is:" << sum << endl;
        }
};

int main()
{
        C obj;
        obj.get_value1(100);
        obj.get_value2(100);
        obj.addition();
        obj.print();
        system("pause");
        return 0;
}
```

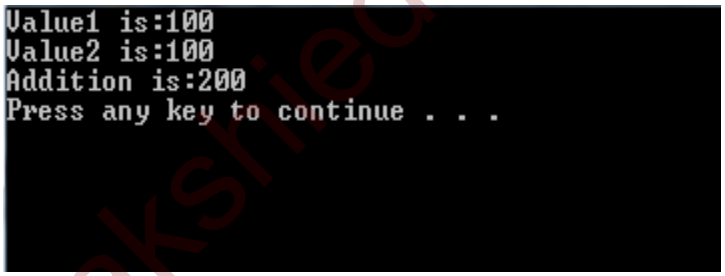

Figure8: Output of the program

Ambiguity in multiple Inheritance:

Multiple inheritance is useful in some cases but sometimes it will give problems like for example If two base classes having same function name which is not overridden in derived class and if you program to access that function using derived class object then the compiler shows error because it is unable to determine which function to be called.

Example:

```
class A
{
        public:
         void function()  // same name function in class A
```

```
                {
                statements;
        }
};

class B
{
        public:
         void function()  // same name function in class B
                {
                statements;
         }

};

class C : public A, public B
        {
        statements;
};

int main()
{
        C obj;
        obj.function();  // it will through a compile time error that is unable to decide which
                            function to call.
}
```

This problem can be overcome using scope resolution function to specify which function of the class to be called that is either class A function or class B function as shown in below.

```
int main()
{
```

```
        obj.A::function();      // class A function
        obj.B::function;        //class B function
}
```

Hierarchical Inheritance

In Hierarchical Inheritance multiple derived classes inherits from a single base class.

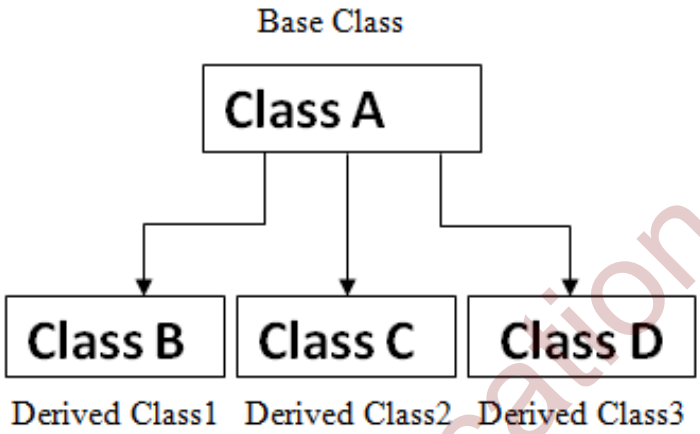

Figure8: Hierarchical Inheritance

Syntax:

```
class Base_Class
        {
        statements;
};

class Derived_Class1
        {
        statements;
};

class Derived_Class2
        {
        statements;
};
```

```
class Derived_Class3
        {
        statements;
};

int main()
{
        statements;
}
```

Hybrid Inheritance

The Hybrid Inheritance is combination of Hierarchical Inheritance and Multilevel Inheritance.

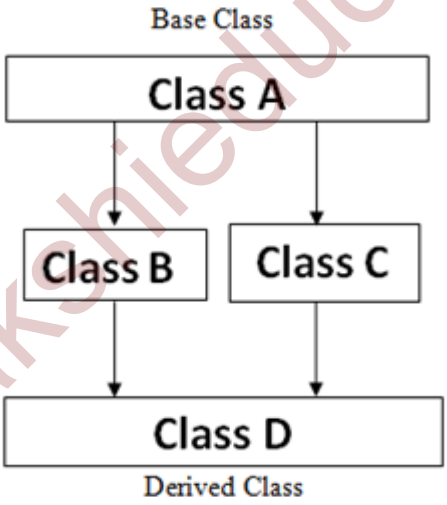

Figure9: Hybrid Inheritance

When we have hybrid inheritance then a diamond problem may arise, in this inheritance a derived class will have multiple paths to a base class. This will results in duplicate inherited members of the base class.

Example for diamond problem:

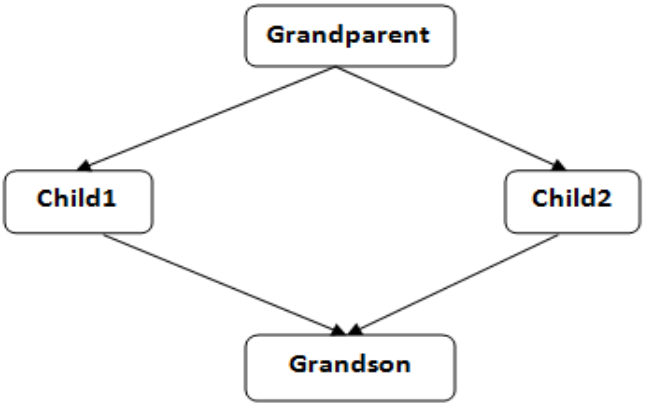

Figure10: Diamond problem

We can avoid Diamond problem with using virtual Inheritance. In the above diamond example the child classes in this case should inherit grandparent class by using virtual inheritance.

```
class GrandParent
{
        statements;
};
class Child1 : public virtual GrandParent
{
         statements;
};
class Child2 : public virtual GrandParent
{
         statements;
};
class Grandson : public Child1, public Child2
{
         statements;
};
```