Inheritance
Lecture -1

Definition: Creating or deriving a new class using another class as a base is called inheritance in C++. The new class created is called a **Derived class** and the old class used as a base is called a **Base class** in C++ inheritance terminology.

The derived class will inherit all the features of the base class in C++ inheritance. The derived class can also add its own features, data etc., It can also override some of the features (functions) of the base class, if the function is declared as **virtual** in base class.

C++ inheritance is very similar to a parent-child relationship. When a class is inherited all the functions and data member are inherited, although not all of them will be accessible by the member functions of the derived class. But there are some exceptions to it too.

Some of the exceptions to be noted in C++ inheritance are as follows.

- The constructor and destructor of a base class are not inherited
- the assignment operator is not inherited
- the friend functions and friend classes of the base class are also not inherited.

Let us see a piece of sample code for C++ inheritance. The sample code considers a class named vehicle with two properties to it, namely color and the number of wheels. A vehicle is a generic term and it can later be extended to any moving vehicles like car, bike, bus etc.,

```
class vehicle //Sample base class for c++ inheritance tutorial
{
 protected:
   char colorname[20];
   int number_of_wheels;
 public:
   vehicle();
   ~vehicle();
   void start();
   void stop();
   void run();
 };
class Car: public vehicle //Sample derived class for C++ inheritance tutorial
{
  protected:
     char type_of_fuel;
  public:
     Car();
};
```

The derived class Car will have access to the protected members of the base class. It can also use the functions start, stop and run provided the functionalities remain the same.

Need for Inheritance

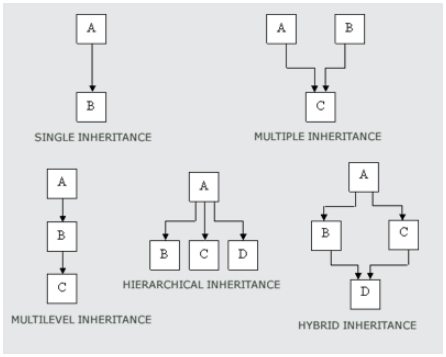
1. Reusability – Inheritance allows to add more features to an existing class without modifying it , and in this way provides reusability of attributes defined resulting into faster development time, easier maintenance and easier control.
2. Transitive nature – if a class B is derived from class A then all the classes derived from class B would automatically inherit the properties of class A. This is called Transitive nature of Inheritance.
3. One major reason behind this is the capability to express the Inheritance relationship. Inheritance is considered to be as **Extended classes** because a class can extend its feature in the form of derived class without modifying the existing class.

Base Class/ Super Class - A class from which another class is inheriting its properties is called base class.

Derived /Sub Class – The class inheriting properties is known as a sub class.

Type of Inheritance

1. Single Inheritance – When a sub class inherits only from a single base class.
2. Multiple Inheritance – when a sub class inherits from multiple base classes, it is known as multiple inheritance.
3. Hierarchical Inheritance – When many sub classes inherit from a single base class, it is known as hierarchical Inheritance.
4. Multilevel Inheritance – The transitive nature of Inheritance is reflected by this form of Inheritance. When a subclass inherits from a class that itself inherits from another class, is known as Multilevel inheritance.
5. Hybrid Inheritance- This type of Inheritance combines two or more forms of Inheritance eg. when a sub class inherits from multiple base classes and all of its base classes inherit from a single base class is known as Hybrid Inheritance.



Declaration and Syntax

class Base { …. };

class Sub : visibility mode Base

{……};

Visibility Modes – the visibility modes in the definition of the derived class specifies whether the features of the base class are privately derived or publicly derived or protected derived. The visibility mode basically control the access specifier to be for inheritable members of base class, in the derived class.

Visibility Mode is	Inheritable public member becomes (in derived class)	Inheritable protected member becomes (in derived class)	Private members of base class are not directly accessible to derived class
Public	Public	Protected	
Protected	Protected	Protected	
Private	Private	Private	

Significance of Visibility Modes

Public Visibility Mode

- Derived class can access the public and protected members of the base class, but not the private members of the base class.
- The public members of the base class become public members of the derived class and protected members of the base class become protected members of the base class.
- In public derivation access specifiers for inherited members in the derived class.
- Derive class publicly when the situation wants the derived class to have all the attributes of the base class, plus some extra features.

Protected Visibility Mode

- Derived class can access the public and protected members of the base class protectedly, but not the private members of the base class.
- The public and protected members of the base class become protected members of the derived class.
- Derive protectedly when the features are required to be hidden from the outside world and at the same time required to be inheritable.

Private Visibility Mode

- Derived class can access the public and protected members of the base class privately, but not the private members of the base class.
- The public and protected members of the base class become private members of the derived class.

- Derive class privately when the derived class requires to use some attributes of the base class and these inherited features can not be inherited further.

Inheritance and the Base class

- Members intended to be inherited and the same time intended to be available to every function, even to non members, should be declared as public members in the base class.
- Members intended to be Inherited but at the same time hidden from outside world, should be declared under protected section.
- Members which are not to be inherited further should be declared under private section.
- Public members of base class can be accessed from its own class, derived class and from objects outside class.
- Protected members of base class can be accessed from its own class, derived class but not from objects outside class.
- Private members of base class are only accessible within its class only.

Examples of different types of Inheritance

Single Inheritance	Multiple Inheritance	Multilevel Inheritance

class person {
char name[20];
int age;
public :
void read_data();
void disp_data( );
};
class student : public person
{
int roll;
int marks;
public:
void getdata();
void showdata( );
};
	class person {
	char name[20];
	int age;
	public :
	void read_data();
	void disp_data( );
	};
	class info
	{
	char address[40];
	char state[30];
	public:
	void read(); void show();};
	class student : public person,public info
	{
	int roll;
	int marks;
	public:
	void getdata();
	void showdata( );
	};
		class person {
		char name[20];
		int age;
		public :
		void read_data();
		void disp_data( );
		};
		class info:public person
		{
		char address[40];
		char state[30];
		public:
		void read(); void show();};
		class student :public info
		{
		int roll;
		int marks;
		public:
		void getdata();
		void showdata( );
		};

Inheritance and constructor and Destrucors

- As we know the constructors are special functions which are invoked automatically at the time of object creation so the base class constructor gets invoked followed by base class constructor. The derived class constructors shouldn't dupilcate the base class constructor work, rather it should provide additional details that the derived class requires.
- The constructor functions are not inherited rather they are called using derived class constructor(s).
- If the constructor is default constructor then it will automatically called in the order of inheritance i,e, Base Class constructor followed by derived class constructor. eg

```
class A
{
private : int x;
public : A( );
void getA( );  void putA( ); ~A( ); };
class B : public A
{
private : int Z;
public : B( ); void getB( ); ~B( ); };
```

```
class C : public B
{
private : int W;
public : C( );
void getC( );
~C( );
};
C C1;
```

in the above example the constructor A( ) will invoked followed by B ( ) and followed by C( ) and the destructor behaves in reverse manner i.e. ~C( ) , ~B( ), ~A( ) in the case when an object expires.(example related to multilevel Inheritance)

- If the Base class has parameterized constructors also, then this is the responsibility of derived class to provide the parameters to the Base class constructor so it is mandatory for the derived class to have a constructor and pass the arguments to the base class constructor. The derived class takes the responsibility of supplying the value to the parameters of the base class constructor.

few examples are given:

Case - I Multilevel Inheritance , Only base class has parameterised constructor.

```
class A
{
private : int x;
public : A( int a) { x=a;}
void getA( );
void putA( );
~A( );
};
class B : public A
{
private : int Z;
public : B( int c,int d) : A(d) { Z=c; }
void getB( );
~B( );
};
class C : public B
{
private : int W;
public : C( int a,int b) : B(a,b)
 {   };
void getC( );
~C( );
};
C C1(13,5);
```

In the above example value of x will be 13 and Z will 5. The number of parameters ithe derived class object requires are to be passed as parameters.

Constructors in Multiple Inheritance -

If this is the case of multiple Inheritance The construcors are called in the order of their Inheritance eg. Class C: public A,private B , then first A class consrtuctor will invoke then B( ) and then C( ) will invoke. in eg. C(int a, Int b) : B(a,b) , the part immmediately following the colon is called initialisation and the list containing values for base class initialisation is called Mem-Initialisation.

Inheritance and access control - Read from Book Pgno. 279 - 283.

Containership

When a class contains objects of other class types as its members, it is referred to as CONTAINERSHIP or CONTAINMENT or AGGREGATION.

Containership establishes “has-a” relationship among classes. This relationship is different from “a kind of” relationship of Inheritance. For example, a microcomputer has a microprocessor whereas it is a kind of digital computer. Thus, a microcomputer can inherit the properties of a digital computer but it must contain a microprocessor. For eg.

```
class address{ int hno;
                char addr_line[40];
                public :
                void getdata();
                void putdata();
                };
class person{
```

```
                char name[20];
                int age;
                address add; // CONTAINERSHIP
                public :
                void read_data();                void display_data();       };
```

Constructors in Containership

For the creation of nested object , first the object s contained inside other objects are constructed by invoking constructor of their classes in the same order of their declaration .In case of Default constructors , constructors are implicitly invoked, But if the constructor requires arguments, this can be done by using an initialization list in the constructor of the enclosing class. For eg.

```
class address{ int hno;
                char addr_line[40];
                public :
                address(int a);
                void getdata();
                void putdata();
                };
class person{           char name[20];
                int age;
                address add; // CONTAINERSHIP
                public :
                person(int x, int y) : add(x) //object name is used i.e. implicit call
                {
                age=y; }
                void read_data();                void display_data();       };
```

We can use as many member objects as are required in a class. For each member object we add a constructor call in the initialization list. If a base class contains an object of some other class then, a class deriving from this base class needs to pass all the arguments required by the base class which in turn to its member objects. Consider the following class definition where student class is deriving from person as base class.

```
class student : public person
{
…… // data members
public:
student( int a, int b, int c) : person(a,b) // class name is written , i.e. explicit call
{….// constructor body of class student
} };
```

The constructor for a member object of a class is invoked implicitly (through the object name) and the constructor for the base class is invoked explicitly (through the class name).

Relationship between classes

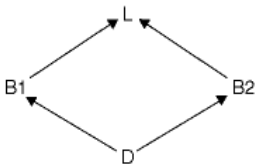
IS – A / A-kind-of relationship - when a class inherits from another class , the derived class has is-a relationship and to provide the values to the base class is the responsibility of derived class.

HAS-A relationship – When a class contains object of another class then enclosing class has has-a relationship with enclosed class and has the ownership of the contained object i.e. the responsibility for the creation and destruction of the object.

HOLDS – A relationship – it is similar to HAS-A relationship but ownership is missing in HOLDS-A relationship. A class thet indirectly contains other objects via pointer or reference, is said to have HOLDS-A relationship with class whose object is indirect member.

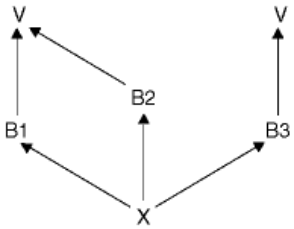
VIRTUAL BASE CLASS –

Suppose you have two derived classes B and C that have a common base class A, and you also have another class D that inherits from B and C. You can declare the base class A as virtual to ensure that B and C share the same subobject of A. In the following example, an object of class D has two distinct subobjects of class L, one through class B1 and another through class B2. You can use the keyword virtual in front of the base class specifiers in the base lists of classes B1 and B2 to indicate that only one sub object of type L, shared by class B1 and class B2, exists. For example:




```
class L { /* ... */ }; // indirect base class
class B1 : virtual public L { /* ... */ };
class B2 : virtual public L { /* ... */ };
class D : public B1, public B2 { /* ... */ }; // valid
```

Using the keyword virtual in this example ensures that an object of class D inherits only one subobject of class L. A derived class can have both virtual and non virtual base classes. For example:




```
class V { /* ... */ };
class B1 : virtual public V { /* ... */ }; class B2 : virtual public V { /* ... */ };
class B3 : public V { /* ... */ };
class X : public B1, public B2, public B3 { /* ... */
};
```

In the above example, class X has two sub objects of class V, one that is shared by classes B1 and B2 and one through class B3.

Point to remember

- If a class has no data members its size will be 1 byte.
- The total size of the object of the class is the total size of its data members.
- The private data members of the base class are visible in the derived class but they are not directly accessible so their size will be added to the size of derived class object.
- A class that serves only as a base class and no objects are declared for that class is called ABSTRACT class.
- An object can only **public Section** of the class but a member function can access all the sections of the class.
- Members means data members (variables) and member functions (functions) both.
- Private Members can be made Inheritable by changing the visibility mode of the private members as public or private.
- The derived class can use all the INHERITABLE members in the same way it uses its own members.
- In case of containership , the embedded object can only access the public section of that class, but in case of inheritance the inherited members are used like own members.