Public, Protected & Private Inheritance – we can declare a derived class from a base class with different access control, i.e. public inheritance, private inheritance, or protected inheritance.

⇒ A private member is accessible only to member of the class in which the private member is declared. They can not be inherited.

→ A private member of the base class can be accessed in the derived class through the member functions of the base class.

→ A private member is accessible to member of its own class and to any of member in derived class

→ A public member is accessible to member of its own class, member of derived class, and outside users of the class.

ex

```
class base {
    public:
        int x;
    protected:
        int y;
    private:
        int z
};
class publicDerived : publicbase
{
    // x is public
    // y is protected
    // z is private so not accessible
    //   from publicDerived class
};
```

```
class protectedDerived : protected base
{   // x is protected
    // y  "    "
    // z is not accessible from
    //   protectedDerived
};
class privateDerived : Private base
{
    // x is private
    // y is private
    // z is not accessible
    //   from privateDerived
};
```

Inheriting constructor & Destructor - The derived class need not have a constructor as long as the base class has a no-argument constructor.

→ if the base class to have a constructor with argument, pass the arguments to the base class constructor is mandatory for the derived class constructor.

ex.- WAP in C++ to demonstrate inherited constructor

```
# include <iostream.h>
class B1                    // base class
 {
  public:
   B1() {
     cout<< "constructor in the base class" <<endl;
          }
   };
 class B2
  { public:
   B2() {
     cout<<"constructor of class B2"<<endl;
   } };
 class C: public B2, public B1          // Derived class
 { public:
    C() {
     cout<<" No Argument constructor in C" <<endl;
     }
  };
 void main() {
      C objc;
  }
```

Constructor for virtual base class ⇒ instead of order in which the base class is appear in the declaration of the derived class, the virtual base constructor are invoked first.

Ex WAP in C++ to demonstrate constructor for virtual base class.

```
# include <iostream.h>
class B1                    // base class
{
  public :
     B1(){
     cout << " No argument constructor of B1 class";
  };
class B2
     {
       public :
       B2()
       {cout << " constructor of B2 class";
     };
class C: public B1, virtual B2
     {  public:
          C() : B1(), B2()
        {cout << " constructor in class C";
        };
void main() {
          C objC;}
```

O/P constructor of B2 class
No argument constructor of B1 class
constructor in class C