Inheritance

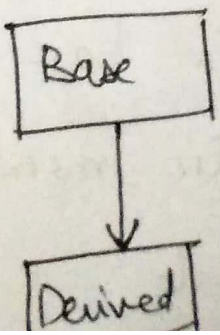
The mechanism of deriving a new class from an old one is called inheritance (or derivation). The old class is called base class & the new one is called derived class (or subclass). The derived class inherits some or all of the traits from the base class. A class can also inherits properties from more than one base class and more than one class can inherit properties from a single base class.

Benefits of inheritance :-

1. Code reusability :- The derived class inherits the properties of base class. So, it reuses the base class code.
2. Saves Time :- Since code is reused, it is not needed to write the code again in derived class. Hence, saves time.
3. Cost Effective :- Reusing the code not only saves time, but money also. and reduces frustration.
4. Reliability increases :- Since the code that is reused has been used and tested already, so it is more reliable to to use that code than to create new one.

Syntax of Inheritance

```
class derived-class-name : visibility-mode base-class-name
{
  // Body of derived class
};
```



Scanned by CamScanner

The colon (:) indicates that the derived-class-name is derived from base-class-name. The visibility mode is optional and if present, it may be either private or public. By default it is private. It specifies whether the features of the base class are privately derived or publicly derived.

Base class	Derived class: Public	Private	Protected
Public	Public	Private	Protected
Private	×	×	×
Protected	Protected	Private	Protected

* Visibility of Inherited Members:-

↳ When a base class is privately inherited:-

① Public members of the base class become private members of derived class

② Private " " " " " are not inherited.

③ Protected " " " " " become private in derived class.

↳ When a base class is publicly inherited:-

① Public members of the base class become public members of derived class.

② Private members are not inherited.

③ Protected members of the base class become protected in derived class.

When base class is protectedly inherited:-

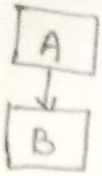
Public members of base class become protected in derived class

Private " " " " are not inherited.

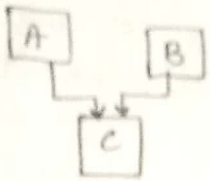
Scanned by CamScanner

Types of Inheritance

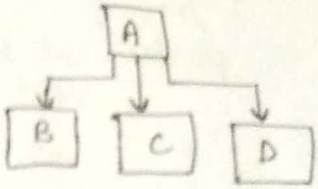
(a) Single inheritance



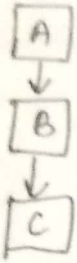
(b) Multiple Inheritance



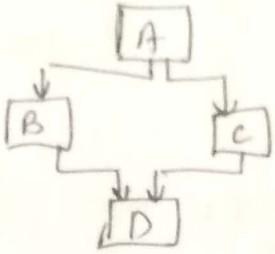
(c) Hierarchical Inheritance



(d) Multilevel Inheritance

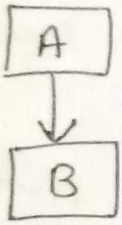


(e) Hybrid Inheritance



(1) Single Inheritance :-

A derived class with only one base class is called single inheritance



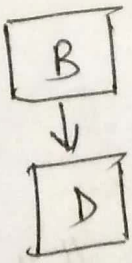
Syntax

```
class derived-class : visibility mode base-class
{
    // Body of derived class
};
```

eg (1)

Inherited Publicly

```
class B
{
    int p;      // not inheritable
public: int a,b;  // inheritable
    void setdata()
    {
    a=5; b=10;
    }
    void show()
    {
    cout << a << b;
    }
};
```




```
class D : public B
{ int c;
public: void mul()
    {
    c = a * b;
    }
    void display()
    {
    cout << "Multiplication";
    cout << c;
    }
};
void main()
{ D d;
  d.setdata(); d.show();
  d.mul(); d.display(); }
```

o/p

```
5 10
Multiplication
50
```

(2) Inherited Privately

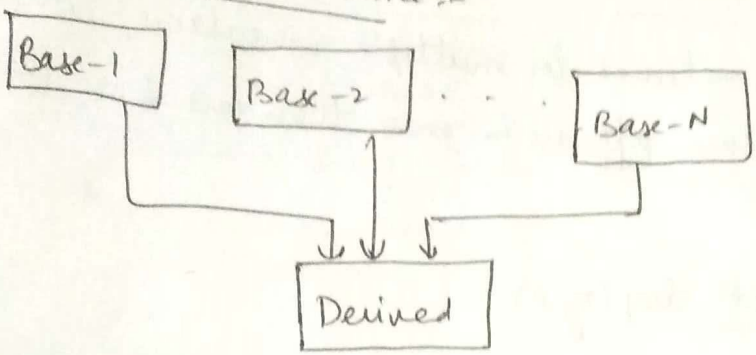
```
class B
{ int a;          // private, not inheritable
  public: int b;  // public, inheritable
          void getdata();
          int get_a();
          void show_a();
};
class D : private B          // private derivation
{ int c;
  public: void mul();
          void display();
};
void B :: getdata()
{ cout << "Enter a, b:";
  cin >> a >> b;
}
int B :: get_a()
{ return a;
}
void B :: show_a()
{ cout << "a=" << a;
}
void D :: mul()
{ getdata();
  c = b * get_a();
}                    // a cannot be accessed directly
void D :: display()
{ show_a();
  cout << "b=" << b << "c=" << c;
}
int main()
{ D d;
  d.getdata // won't work
  d.mul();
  d.show_a(); // won't work
  d.display();
  // d.b = 20; // won't work
  d.mul();
  d.display();
  return 0;
}
```

off

```
Enter a, b: 5 10
a=5 b=10
c=50
Enter a b
a=2 b=20
c=40
```

Scanned by CamScanner

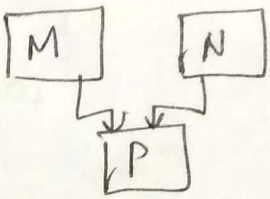
Multiple Inheritance:-



A class can inherit the attributes of two or more classes. This is known as Multiple Inheritance.

Syntax

```
class Derived : visibility Base-1, visibility Base-2, ---, visibility Base-N
{
  // Body of derived class
};
```



Example

```
class M
{
 public: int m;
         void getM();
         {
          cout << "Enter m";
          cin >> m;
         }
};
class N
{ public: int n;
          void getN()
          {
           cout << "Enter n";
           cin >> n;
          }
};
class P : public M, public N
{
 public: int P;
         void mul()
         {
         // getM();
         // getN();
           P = m * n;
           cout << "mul" << P;
         }
};
void main()
{
  P p;
  p.getM();
  p.getN();
  p.mul();
}
```

O/P

```
Enter m
10
Enter n
20
Mul 200
```

Scanned by CamScanner

Ambiguity in Inheritance

This problem arises sometimes in multiple inheritance, when function with same name appears in more than one base class.

Eg.

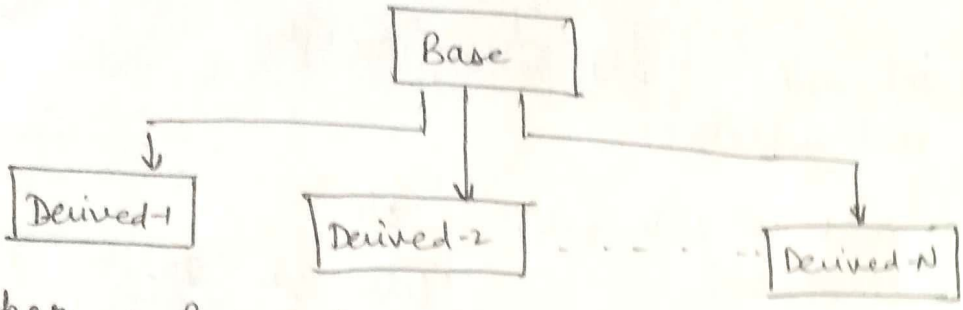
```
class M
{ public : void display()
           {
             cout << "class M";
           }
};
class N
{ public : void display()
           {
             cout << "class N";
           }
};
class P : public M, public N
{ public : void display() // over
    {
      cout << "class P";
      M :: display();
    }
};
void main()
{
  P p;
  p.display();
}
```

Problem Which display() function is used by derived class when we inherit class M, N?

Solution We can solve this problem by defining a named instance within the derived class using scope resolution operator with the function.

Scanned by CamScanner

Hierarchical Inheritance

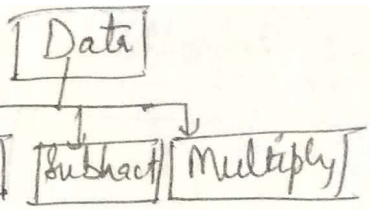


When a Base class is inherited by various derived classes, this is known as Hierarchical Inheritance.

Syntax

```
class Base
{
   //Body of Base
};

class Derived-1 : visibility-mode Base
{
  // Body of Derived-1
};
class Derived-2 : visibility-mode Base
{
   //Body of Derived-2
};
  .
  .
  .
class Derived-N : visibility-mode Base
{
  // Body of Derived-N
};
```

Scanned by CamScanner




```
class Data
{
  public : int a, b;
           void getdata()
           {
            cout << " Enter a,b";
            cin >> a >> b;
           }
};

class Add : public Data
{
  public : int c;
           void sum()
           {
            c = a+b;
            cout << " Sum is: " << c;
           }
};

class Subtract : public Data
{
  public : int d;
           void minus()
           {
            d = a-b;
            cout << " Subtraction is: " << d;
           }
};

class Multiply : public Data
{
  public : int m;
           void mul()
           {
            c = a*b;
            cout << " Multiplication is: " << m
           }
};

void main()
{
  Add x;
  x.getdata();
  x.sum();

  Subtract y;
  y.getdata();
  y.minus();

  Multiply z;
  z.getdata();
  z.mul();
}
```

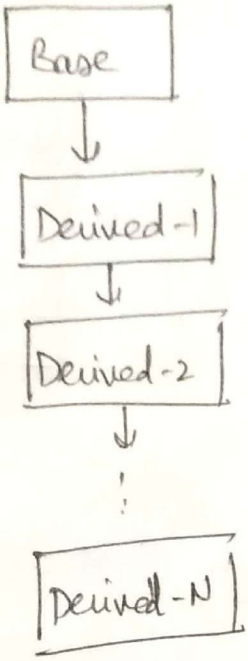
O/P

```
Enter a,b  20, 10
Sum is 30

Enter a,b 20 10
Subtraction is 10

Enter a,b  5 2
Multiplication is 10
```

Scanned by CamScanner

Multilevel Inheritance

when a class is derived from base class and it is further inherited by another derived class, it is called multilevel inheritance.



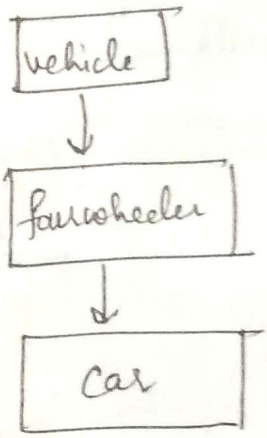
Syntax

```
class base-class
{
   // Body of Base
};
class Derived-1 : Visibility mode Base
{
  // Body of derived-1
};
class Derived-2 : visibility-mode derived-1
{
  // Body of derived-2
};
:
class Derived-N : visibility-mode derived-N-1
 { // Body of derived-N
};
```

Scanned by CamScanner

Example

```
#include<iostream.h>
class vehicle
{
public : vehicle()
         {
          cout<<" This is vehicle";
         }
};

class fourwheeler : public vehicle
{
public:
        fourwheeler()
        {
         cout<<" Objects with 4 wheels";
        }
};

class car : public fourwheeler
{
  public :
          car()
          {
          cout<<" This is car";
          }
};
int main()
{
 car c; // creating object of subclass will
 return 0;      invoke constructor of base
}                   class.
```



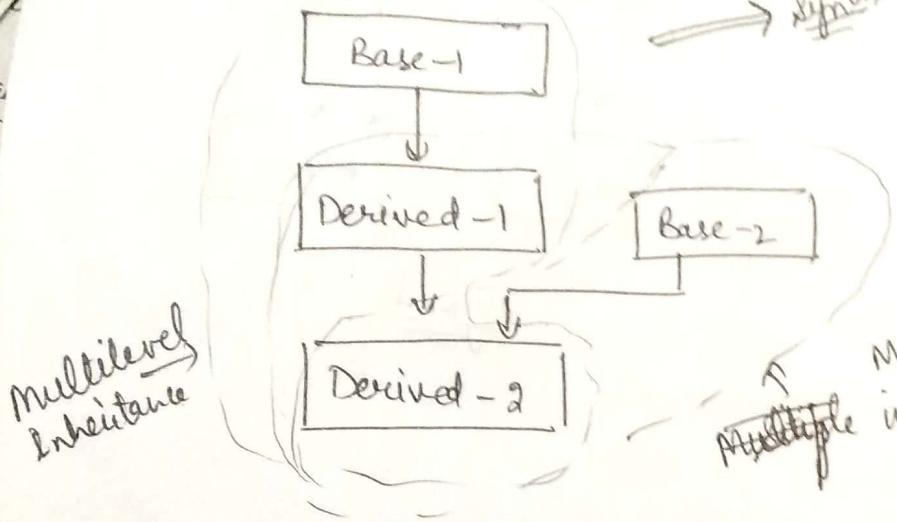
o/p

```
This is vehicle
Objects with 4 wheels
This is car
```

Scanned by CamScanner

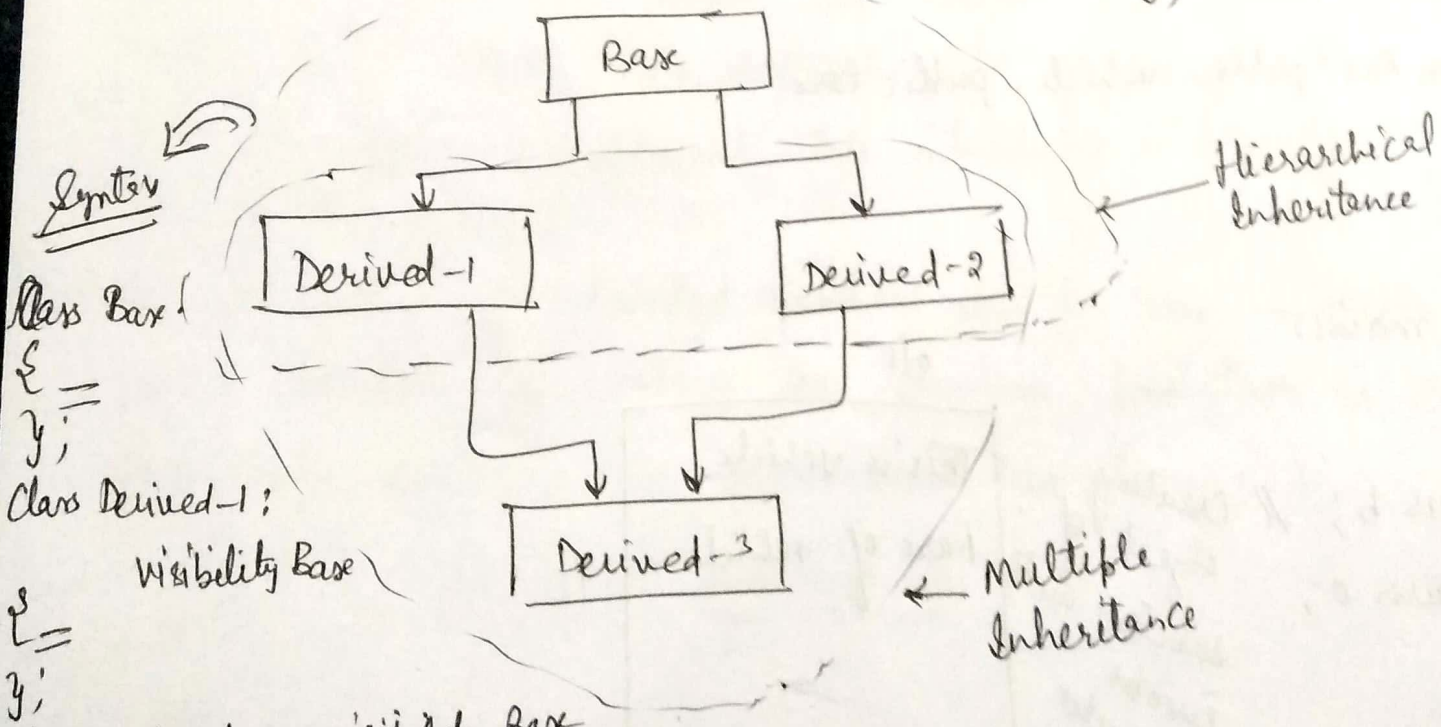
Hybrid Inheritance

There could be situations where we need to apply two or more types of inheritance to design a program. This combination of two or more types of inheritance is known as hybrid inheritance.



Syntax

```
class Base-1
{ =
};
class Derived-1: Visibility Base-1
{ =
};
class Base-2
{ =
};
class Derived-2 : visibility
    Derived-1, visibility
    Base-2
{
  =
};
```

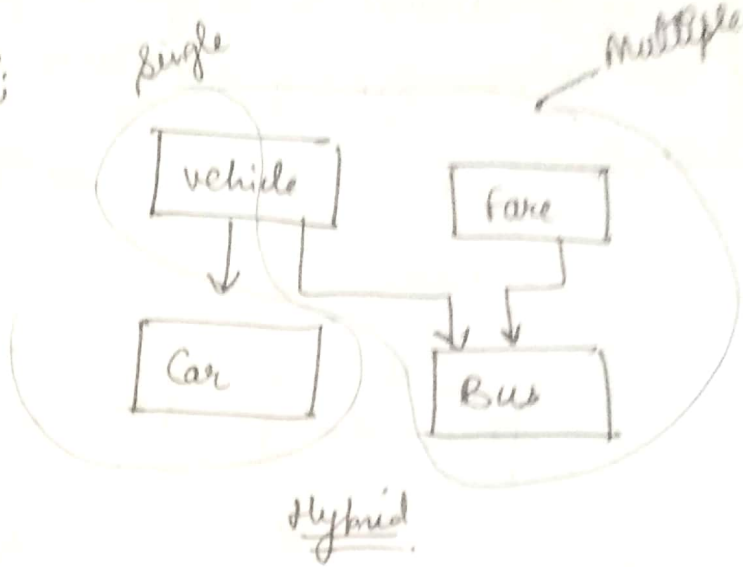


Syntax

```
class Base
{ =
};
Class Derived-1 :
    Visibility Base
{ =
};
class Derived-2 : visibility Base
{ =
};
Class Derived-3 : visibility Derived-1, visibility Derived-2
{ =
};
```

Scanned by CamScanner

```
#include <iostream.h>
class vehicle
{
public: vehicle()
        {
        cout << "This is vehicle";
        }
};
class fare
{
public : fare()
        {
        cout << "Fare of vehicle";
        }
};
class Car: public vehicle
{
};
class Bus: public vehicle, public fare
{
};
int main()
{
Bus b; // creating object of subclass will invoke constructor of baseclass
return 0;
}
```



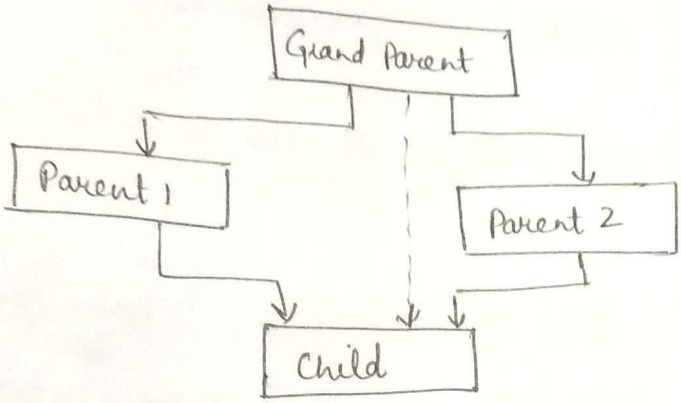
o/p

```
This is vehicle
Fare of vehicle
```

Scanned by CamScanner

Virtual Base Classes

Consider a situation where all the three kinds of inheritance, ie- multilevel, multiple & hierarchical inheritance are involved.



The 'child' has two direct base classes 'Parent 1' and 'Parent 2' which themselves have a common base class 'Grand Parent'. The 'child' inherits the traits of 'GrandParent' via two separate paths. All the public & protected members of GrandParent are inherited into 'child' twice via 'Parent 1' and 'Parent 2'. This means 'child' would have duplicate sets of the members inherited from 'GrandParent'. This introduces ambiguity and should be avoided.

The duplication of inherited members due to these multiple paths can be avoided by making the common base class as virtual class while declaring the intermediate base classes.

```
class A        // grand parent
{ --
};
class B1 : virtual public A    // parent-1
{
  =
};
```

They can be used in any order

```
class B2 : public virtual A    // parent 2
{
=
};
class C : public B1, public B2 // child
{
==
};
```

They can be used in any order

// only one copy of A will be inherited.

When a class is made virtual base class, only one copy of that class is inherited, regardless of how many inheritance paths exist between virtual base class & a derived class.

Example:-

```
#include <iostream.h>
class student
{
protected : int rollno;
public : void getno (int a)
         {
           rollno = a;
         }
         void putno()
         {
         cout << "roll no:" << rollno;
};
class test : virtual public student
{
protected : float part1, part2;
public :    void getmarks (float x, float y)
            { part1 = x; part2 = y;
            }
            void putmarks()
            {
};          cout << "Marks obtained" << "Part1 = " << part1 << "Part2 = " << part2;
```

Scanned by CamScanner

```
class sports : public virtual student
{
protected : float score;
public : void get_score (float s)
         {
           score = s;
         }
         void put_score ()
         {
           cout << " Sports wt : " << score;
         }
};

class result : public test, public sports
{
   float total;
public : void display ();
};
void result :: display ()
{ total = part1 + part2 + score;
  put_numba ();
  put_marks ();
  put_score ();
  cout << "Total score : " << total;
}
int main()
{
  result r;
  r.getno(678);
  r.getmarks (30.5, 25.5);
  r.getscore (7.0);
  r.display ();
  return 0;
}
```

o/p

```
Rollno : 678
Marks Obtained :
Part1 = 30.5
Part 2 = 25.5
Sports wt : 7
Total Score : 63
```

In this example, only one copy of rollno is inherited by result class from student class.
∴ any changes made to rollno by any of the derived classes are made to this single instance. This ensures data integrity.

Scanned by CamScanner

An abstract class is one which is not used to create objects. It is designed only to act as base class (to be inherited by other classes). An abstract class is the one which has at least one pure virtual function.

eg

```
class vehicle // abstract class
{
private : int d1, d2;
public :
        virtual void spec() = 0;    // pure virtual function
};
class LMV : public vehicle
{
public :
        void spec()
        {
        ==
        }
};
class HMV : public vehicle
{
public : void spec()
        {
        ==
        }
};
class TW : public vehicle
{
public :
        void spec()
        {==
        }
};
```

Scanned by CamScanner

Constructors in Derived Classes

As long as no base class constructor takes any arguments, the derived class need not have a constructor function. However, if base class contains a constructor with one or more arguments, then it is mandatory for the derived classes have a constructor and pass the arguments to base class constructors.

When both the derived and base classes contain constructors, the base constructor is executed first and then the derived class constructor is executed.

In Case of Multiple Inheritance,
the base classes are constructed in the order in which they appear in the declaration of the derived class.

In Multilevel Inheritance,
the constructors will be executed in the order of inheritance.

The constructor of derived class receives the entire list of values as its arguments and passes them onto the base constructors in the order in which they are declared in the derived class. The base constructors are called and executed before executing the statements in the body of their derived constructor.

The constructors for virtual classes are invoked before any non-virtual base classes in the order in which they are declared.

Scanned by CamScanner

Method of Inheritance

```
class B : public A
{
};      // single
```

Order of Execution

A(); Base Constructor
B(); Derived "

```
class A : public B, public C
{
  //Multiple
};
```

B(); // Base (first)
C(); // Base (2nd)
A(); // Derived.

```
class A : public B , virtual public C
{
  // Multiple Inheritance
};    with one virtual base class
```

C(); // virtual base
B(); // ordinary base
A(); // derived.

Example

```
# include <iostream.h>
class A
{
  int x;
public: A(int i)
        {
          x=i;
          cout<<"A class";
        }
        void showdata()
        {
          cout<<"x="<<x;
        }
};
```

Scanned by CamScanner

```
class B
{
    float y;
public:  B(float y)
         {
          y=y;
          cout << " B class";
         }
        void showdata2()
         {
          cout << "y=" << y;
         }
};
class C: public B, public A
{
    int m,n;
public: C(int a, float b, int c, int d): A(a), B(b)
        {
          m=c;
          n=d;
          cout << " C class";
        }
        void showdata3()
        {
         cout << "m=" << m << "n=" << n;
        }
};
int main()
{
  C obj(5,10.75, 30,40);
  obj.showdata1();
  obj.showdata2();
  obj.showdata3();
  return 0;
}
```

Header line of the derived class constructor contains 2 parts:
Part 1 – Declaration of arguments passed to derived constructor
Part 2 – function calls to base constructors separated by a colon (:)

o/p

```
B class
A class
C class
x=5
y=10.75
m=30
n=40
```

Scanned by CamScanner