Chapter:5

- **Constructors**
- **Destructors**

Constructor

- **It is a member function which initializes a class.**
- **Member function that is automatically called when an object is created.**
- **Purpose is to construct an object and do initialization if necessary.**
- **Constructor function name is class name.**
- **Has no return type *specified***

Default Constructors

- A default constructor is a constructor that takes no arguments.

- A simple instantiation of a class (with no arguments) calls the default constructor:

```
Rectangle r;
```

- Let's see the simple example of C++ default Constructor.

Default Constructors

```
#include <iostream.h>
class Employee
 {
   public:
     Employee()
     {
       cout<<"Default Constructor Invoked“<<endl;
     }
};
```

Default Constructors

```
int main(void)
{
    Employee e1; //creating an object
    Employee e2; //creating an object
    return 0;
}
```

Default Constructors

The Output:

Default Constructor Invoked

Default Constructor Invoked

Passing Arguments to Constructors

- To create a constructor that takes arguments:
  - Indicate parameters in prototype:
    ```
    Rectangle(double, double);
    ```
  - Use parameters in the definition:
    ```
    Rectangle::Rectangle(double w, double len)
    {
       width = w;
       length = len;
    }
    ```

Passing Arguments to Constructors

- You can pass arguments to the constructor when you create an object:

```
Rectangle r(10, 5);
```

Example:1

```
#include <iostream>
using namespace std;
class rectangle
{
 private:
   float height;
   float width;
```

Example:1

```
public:
  rectangle(float h=4, float w=5)
   {
    height = h;
    width = w;
   }
  void display()
  { cout<<height<<"  "<<width<<endl;}

};
```

Example:1

```
main()
{
        rectangle r(10,7);
        r.display();
        return 0;
}
```

The Output

```
10 7
```

Example:2

```
#include <iostream>
using namespace std;
class Employee
{
   public:
      int id;
      string name;
      float salary;
```

Example:2

```
Employee(int i, string n, float s)
 {
    id = i;
    name = n;
    salary = s;
  }
  void display()
  { cout<<id<<"  "<<name<<"  "<<salary<<endl;}
};
```

Example:2

```
int main(void)
{
  Employee e1(101, "Sonoo", 8900);
  Employee e2(102, "Nakul", 5900);
  e1.display();
  e2.display();
  return 0;
}
```

Example:2

The Output

101 Sonoo 8900

102 Nakul 5900

Passing Arguments to Constructors

- If all of a constructor's parameters have default arguments, then it is a default constructor. For example:

```
Rectangle(double = 0, double = 0);
```

- Creating an object and passing no arguments will cause this constructor to execute:

```
Rectangle r;
```

Example:

```
#include <iostream>
using namespace std;
class rectangle
{
 private:
   float height;
   float width;
```

Example:

```
public:
  rectangle(float h=4, float w=5)
   {
    height = h;
    width = w;
   }
  void display()
  { cout<<height<<"  "<<width<<endl;}

};
```

Example:

```
main()
{
        rectangle r;
        r.display();
        return 0;
}
```

The Output

```
4 5
```

Overloading Constructors

- **You can have more than one constructor in a class, as long as each has a different list of arguments.**

Overloading Constructors

```
class rectangle
 {
 private:
   float height;
   float width;
   int xpos;
   int ypos;
```

Overloading Constructors

```
public:
  rectangle(float, float); // constructor
  rectangle( );             // another constructor
  void draw();              // draw member function
  void posn(int, int);      // position member function
  void move(int, int);      // move member function
};
```

Example:

```
#include <iostream>
using namespace std;
class rectangle
{
 private:
   inr height, width;
```

Example:

```
public:
  rectangle(int h, int w)
   {
    height = h;
    width = w;
   }
```

Example:

```
rectangle()
   {
    height = 7;
    width = 5;
   }
  void display()
  { cout<<height<<"   "<<width<<endl;}
};
```

Example:

```
main()
{
        rectangle r1, r2(10,9);
        r1.display();
        r2.display();
        return 0;
}
```

The Output:

```
7 5
10 9
```

Destructor

- **It is a member function which deletes an object.**

- **A destructor function is called automatically when the object goes out of scope:**

1) the function ends

2) the program ends

3) a block containing temporary variables ends

Destructor

- **A destructor has:**

  (i) the same name as the class but is preceded by a tilde (~).

  (ii) no arguments and return no values.

- **If you do not specify a destructor, the compiler generates a default destructor for you.**

Example:1

```
#include <iostream>
using namespace std;
class rectangle
{
 private:
   inr height, width;

 public:
   rectangle(int h, int w);     // constructor
   ~rectangle( );               // destructor
};
```

Example:1

```
rectangle::rectangle(int a, int b)
{
 height=a;
 widht=b;
}

rectangle::~rectangle()
{
    cout<<"destructure called"<<endl;
}
```

Example:1

```
int main()
{
   Rectangle r1(10,5);
   return 0;
}
```

The output

```
destructure called
```

Example:2

```
class person
{
  private:
    int Birth;
  public:
   person( )          // constructor
   {
      Birth=1970;
      cout<<"constructure called"<<endl;
   }
```

Example:2

```
    ~person( )      // destructor
    {
      cout<<"destructure called"<<endl;
    }
};
```

Example:2

```
int main()
{
   cout<<"main start"<<endl;
   creature obj;
   cout<<"main end"<<endl;
   getch();
   return 0;
}
```

Chapter:6

Inheritance

Introduction

- **In C++, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviors which are defined in other class.**

Introduction

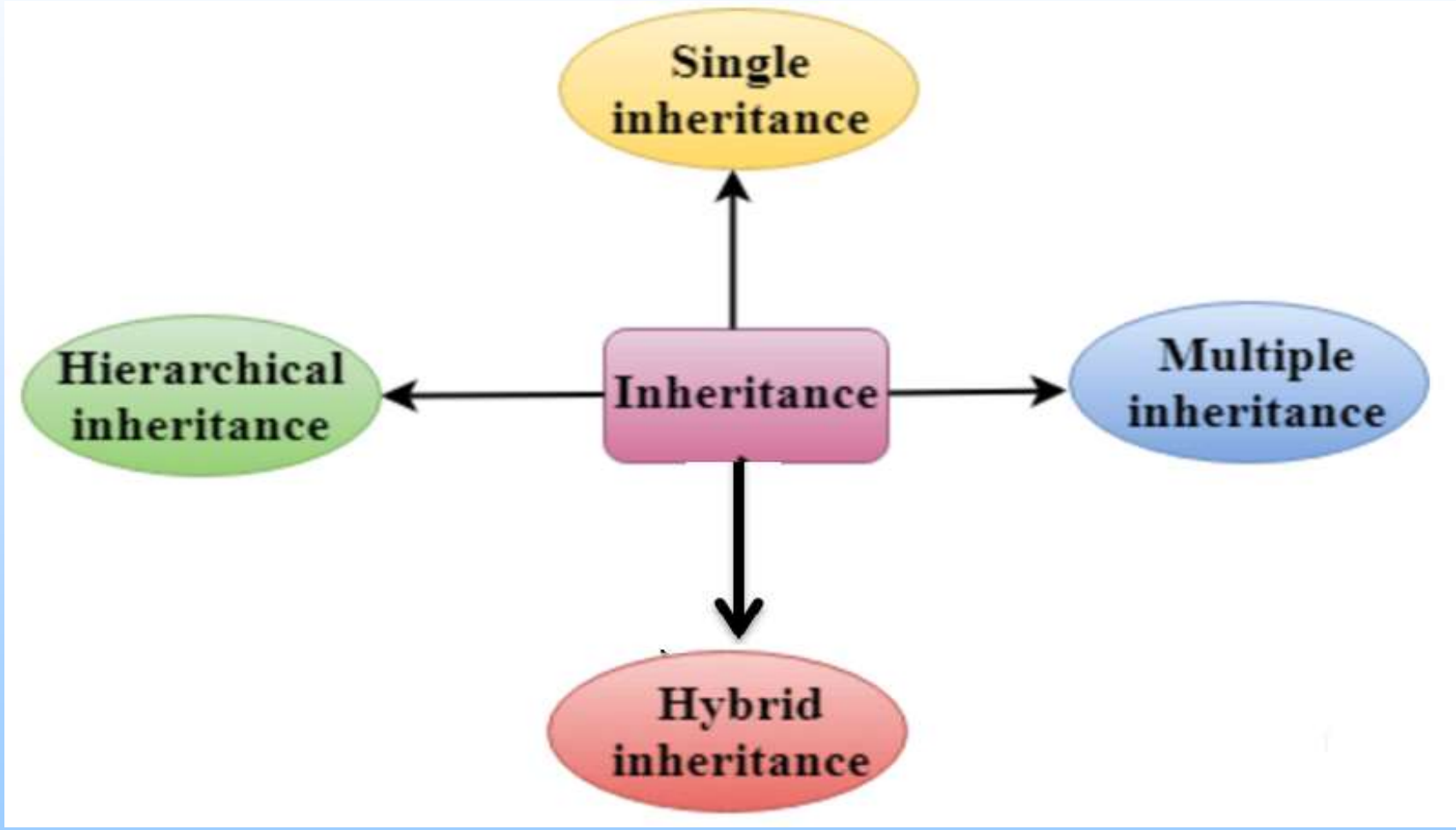
- **In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.**

Introduction

C++ supports five types of inheritance:

- **Single inheritance**
- **Multiple inheritance**
- **Hierarchical inheritance**
- **Multilevel inheritance**
- **Hybrid inheritance**

Introduction



Derived Classes

- A Derived class is defined as the class derived from the base class.

- The Syntax of Derived class:

```
class derived_class_name :: visibility-mode base_class_name
{
   // body of the derived class.
}
```

Derived Classes

Where,

derived_class_name: It is the name of the derived class.

visibility mode: The visibility mode specifies whether the features of the base class are publicly inherited or privately inherited. It can be public or private.

base_class_name: It is the name of the base class.

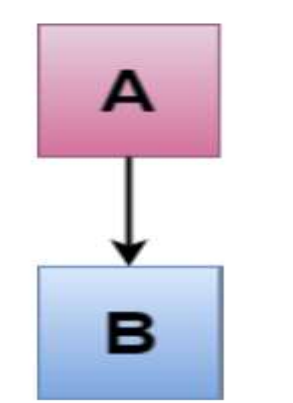
Derived Classes

Note:

- **In C++, the default mode of visibility is private.**
- **The private members of the base class are never inherited.**

Single Inheritance

- **Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.**



Where 'A' is the base class, and 'B' is the derived class.

Single Inheritance

- When one class inherits another class, it is known as single level inheritance. Let's see the example of single level inheritance which inherits the fields only.

Example:1

```
#include <iostream>
 class Account {
   public:
   float salary = 60000;
 };
 class Programmer: public Account
 {
   public:
   float bonus = 5000;
 };
```

Example:1

```
int main(void) {
    Programmer p1;
    cout<<"Salary: "<<p1.salary<<endl;
    cout<<"Bonus: "<<p1.bonus<<endl;
   return 0;
}
```

The Output:

```
Salary: 60000
Bonus: 5000
```

Example:2

```
#include <iostream.h>
 class Animal
 {
   public:
   void eat()    { cout<<"Eating..."<<endl; }
 };
 class Dog: public Animal
 {
   public:
   void bark()  {  cout<<"Barking..."; }
};
```

Example:2

```
int main(void)
{
    Dog d1;
    d1.eat();
    d1.bark();
    return 0;
}
```

The Output:

```
Eating...
Barking…
```

Example:3

```
#include <iostream.h>
class A
{
    int a = 4;
    int b = 5;
    public:
    int mul()
    { return a*b; }
};
```

Example:3

```
class B : private A
{
    public:
    void display()
    {  cout <<"a * b is: "<<mul()<<endl; }
};
```

Example:3

```
int main()
{
   B b;
   b.display();
   return 0;
}
```

The Output:

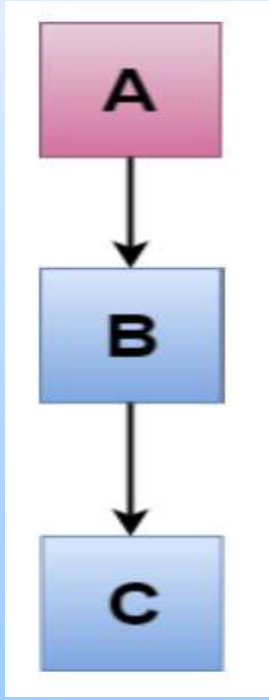
```
a * b is : 20
```

Example:3

- In the above example, class A is privately inherited. Therefore, the mul() function of class 'A' cannot be accessed by the object of class B. It can only be accessed by the member function of class B.

Multilevel Inheritance

Multilevel inheritance is a process of deriving a class from another derived class.



Multilevel Inheritance

- When one class inherits another class which is further inherited by another class, it is known as multi level inheritance in C++. Inheritance is transitive so the last derived class acquires all the members of all its base classes.

Example:1

```
#include <iostream.h>
class AAA
{
 public:
 void f1()
 { cout<<“AAAAAA..."<<endl;  }
};
class BBB: public AAA
{
   public:
   void f2()
   {  cout<<“BBBBBB..."<<endl; }
 };
```

Example:1

```
class CCC: public BBB
 {
    public:
    void f3()
    {  cout<<“CCCCCC...";  }
 };
int main(void)
 {
    CCC  d1;
    d1.f1();
    d1.f2();
    d1.f3();
     return 0;
}
```

Example:1

- **The Output:**

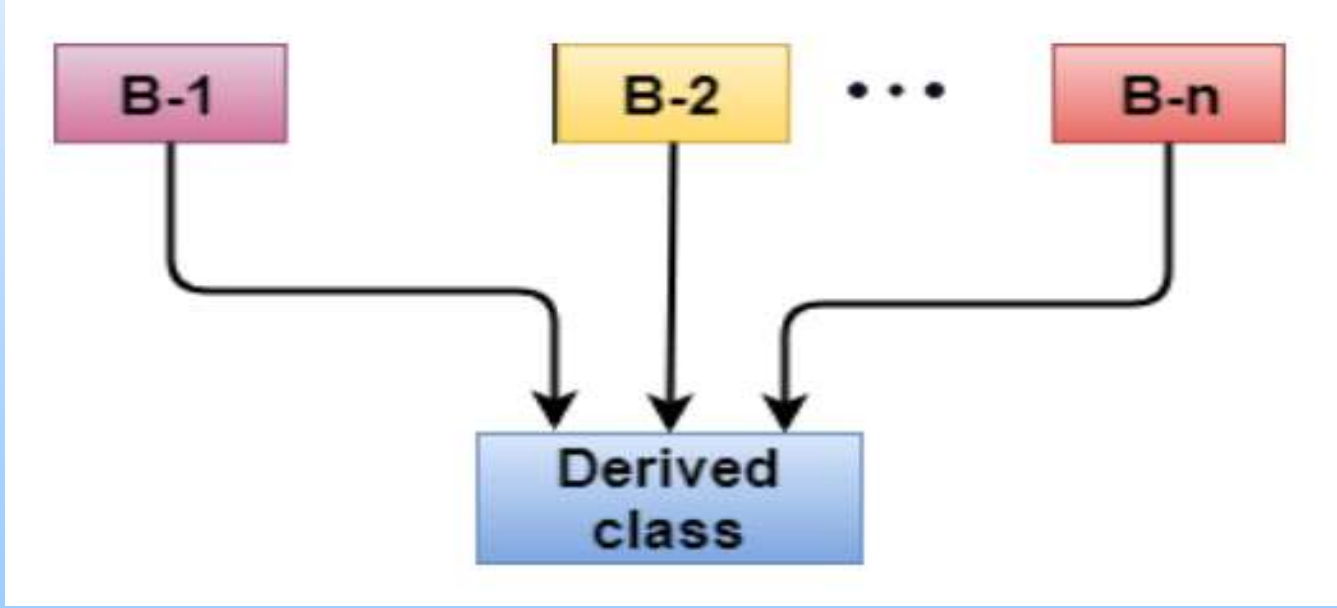
AAAAAA…

BBBBBB…

CCCCCC…

Multiple Inheritance

- **Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.**



Multiple Inheritance

- **Syntax of the Derived class:**

```
class D : visibility B-1, visibility B-2, ?
{
    // Body of the class;
}
```

Example:1

```
#include <iostream.h>
class A
{
    protected:
     int a;
    public:
    void get_a(int n)
    {  a = n; }
};
```

Example:1

```
class B
{
    protected:
    int b;
    public:
    void get_b(int n)
    {  b = n;  }
};
```

Example:1

```
class C : public A, public B
{
   public:
    void display()
    {
      cout << "The value of a is : " <<a<< endl;
      cout << "The value of b is : " <<b<< endl;
      cout<<"Addition of a and b is : "<<a+b;
    }
};
```

Example:1

```
int main()
{
   C c;
   c.get_a(10);
   c.get_b(20);
   c.display();
   return 0;
}
```

Example:1

- **The Output:**

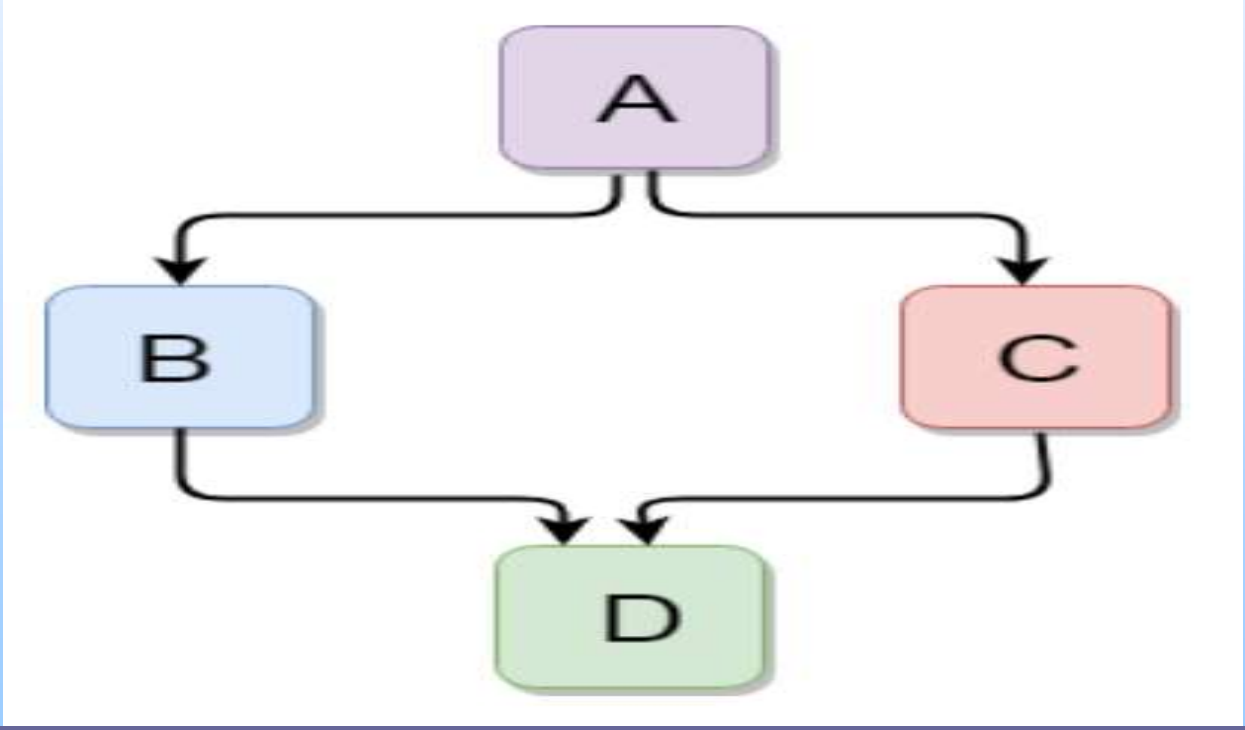
The value of a is : 10

The value of b is : 20

Addition of a and b is : 30

Hybrid Inheritance

- **Hybrid inheritance is a combination of more than one type of inheritance.**



Example:

```
#include <iostream>
class A
{
    protected:
    int a;
    public:
    void get_a()
    {  cout << "Enter the value of 'a' : ";
       cin>>a; }
};
```

Example:

```
class B : public A
{
    protected:
    int b;
    public:
    void get_b()
    {  cout << "Enter the value of 'b' : ";
       cin>>b;  }
};
```

Example:

```
class C
{
    protected:
    int c;
    public:
    void get_c()
    { cout << "Enter the value of c is : ";
        cin>>c;     }
};
```

Example:

```
class D : public B, public C
{
    public:
    void mul()
    {
      get_a();
      get_b();
      get_c();
      cout << "Multiplication of a,b,c is : " <<a*b*c<< endl;
    }
};
```

Example:

```
int main()
{
    D d;
    d.mul();
    return 0;
}
```

- **The Output:**

Enter the value of 'a' : 10

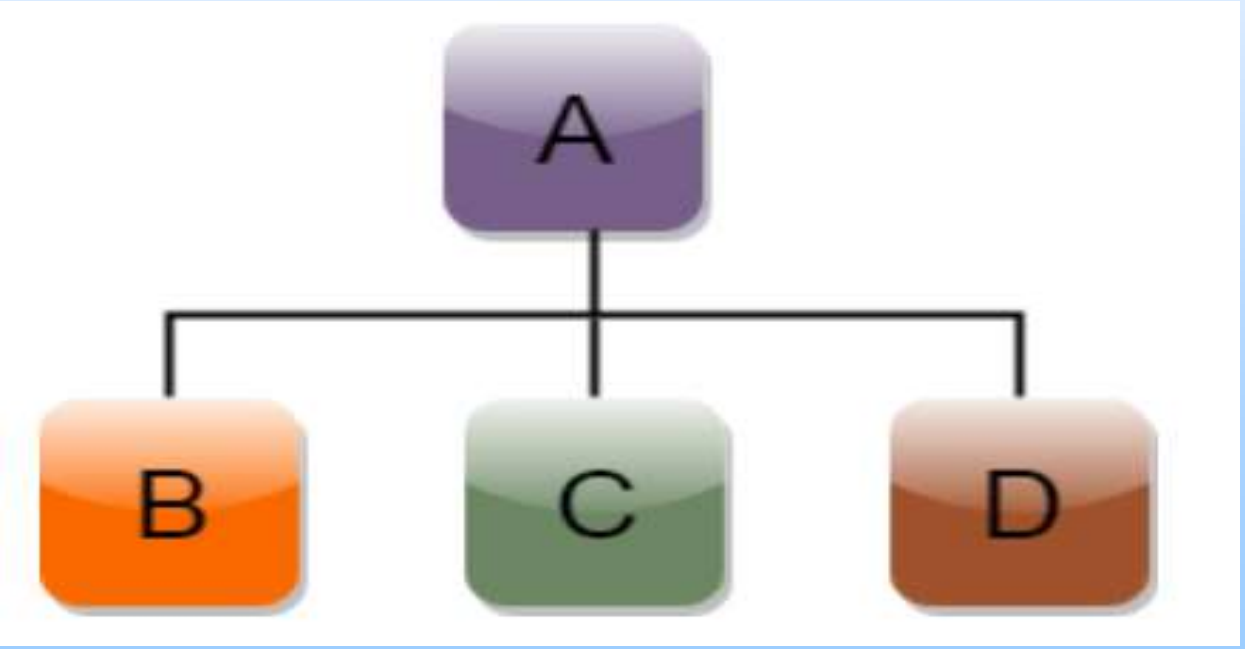
Enter the value of 'b' : 20

Enter the value of c is : 30

Multiplication of a,b,c is : 6000

Hierarchical Inheritance

- **Hierarchical inheritance is defined as the process of deriving more than one class from a base class**



Hierarchical Inheritance

- **Syntax of Hierarchical inheritance:**

```
class A
{ // body of the class A.  }
class B : public A
{ // body of class B.  }
class C : public A
{ // body of class C.  }
class D : public A
{ // body of class D.  }
```

Example:

```
#include <iostream>
class Shape
{
    public:
    int a;
    int b;
    void get_data(int n,int m)
    {  a= n;
        b = m;   }
};
```

Example:

```
class Rectangle : public Shape
 class
 {
    public:
    int rect_area()
    {
        int result = a*b;
        return result;
    }
 };
```

Example:

```
class Triangle : public Shape
{
    public:
    int triangle_area()
    {
        float result = 0.5*a*b;
        return result;
    }
};
```

Example:

```
int main()
{
    Rectangle r;
    Triangle t;
    int length,breadth,base,height;
    cout << "Enter the length and breadth: " ;
    cin>>length>>breadth;
    r.get_data(length,breadth);
    cout << "Area of rectangle is:"<< r.rect_area()
     <<endl;
```

Example:

```
 cout << "Enter the base and height:";
 cin>>base>>height;
 t.get_data(base,height);
 cout <<"Area of triangle is:"<< t.triangle_area();
 return 0;
}
```

Example

- **The Output:**

Enter the length and breadth: 23 20

Area of rectangle is : 460

Enter the base and height: 2 5

Area of triangle is : 5