Virtual function: This can be executed but it does not have any memory.

Page No.
Date

# OOPS WITH C++

↓

Object oriented programming system

BJARNE STOUSTRUP at AT & T Bell Laboratories New Jersey USA — 1980.

100%

| | C | C++ |
|---|---|---|
| 1972 | | |
| → | C is invented by Denis ~~Rechie~~ Ritchie | It is invented by Bjarne Stoustrup |
| → | It is based on Procedure oriented approach.. | It is based on object oriented approach. |
| → | It uses calloc, malloc and free operation for DMM. | • Object<br>• Class<br>• Encapsulation<br>• Abstraction<br>• Inheritance<br>• Polymorphism. |
| → | It does not have | It has friend function |
| → | Top-down approach | Bottom to up approach. |
| → | It does not have. | exception handling |
| → | subset of C++ | superset of C. |
| → | It does not have. | It has virtual fun.. |
| → | printf(), scanf() are the function in 'C' | cin(), cout() are the fn object |

(6) It has constructor and destructor. (✓)

Page No.
Date

Headerfile in C++ → <iostream.h>

input stream — output stream

(5) In C we use reference variable — It is not used in C++

Eg: scanf ("%d", &n)
↓
reference variable.

**OBJECT :** An entity that has state and behaviour is known as object

Eg: chair, table, keyboard.

**CLASS :** collection of object is called class.

* It is a logical entity.

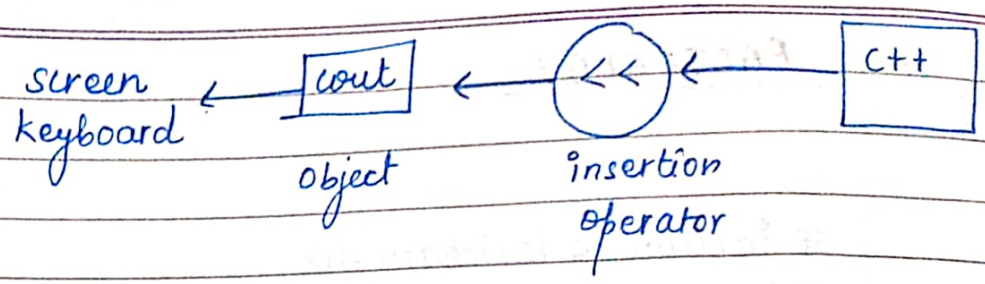
**Output operator :**

cout << " C++ is better than C " ;

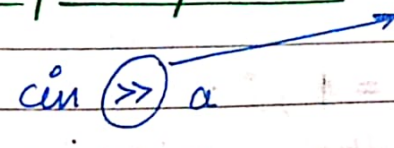
↓

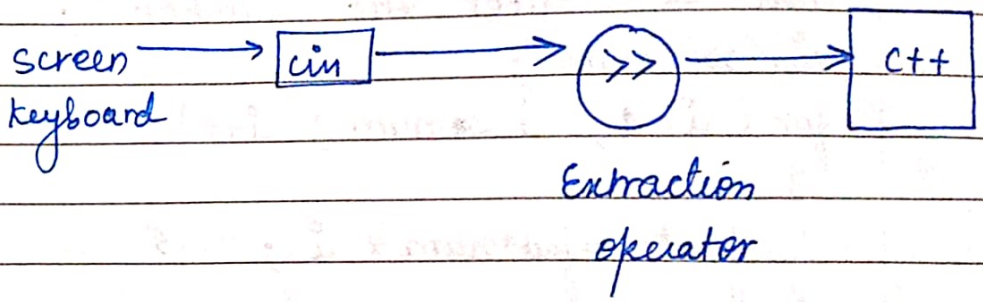
The identifier Cout is a predefined object that represent the standard output stream in C++.

The operator << insertion or put to operator.




## Input operator




1.
```
#include <iostream.h>
#include <conio.h>
void main ()
{
  cout << " Hello" ;
}
```

2. AVERAGE

```
#include <iostream.h>
#include <conio.h>
void main ()
{
  int a, b, avg ;
  cin >> a >> b;
  avg = (a+b)/2 ;
  cout << avg ;
}
```

\n is used for next line
or endl

Page No.
Date

3. **FACTORIAL**

```
# include < iostream.h>
#include < conio.h>
void main ()
{
  int num, fact = 0 , i ;
cout << " Enter the number " ;
cin >> num ;
for ( i = 0 ; i <= num ; i++)
  {
     fact = fact*num * i ;
  }
cout << " fact = " << fact << "\n" ;
}
```

i 2

```
i = n
while (i != 0)
{
 fact = fact * i)
 i--;
}
```

Page No.
Date

## Features of C++

(1) **Encapsulation** : wrap up data into a single unit.

class — Keyword
classname — (user-defined)

```
{
  /variables/*;
  /functions/*;
};
```

(2.) **Abstraction** : hiding the complexity and showing the useful parts.

(3.) **Inheritance** : Public, Private, Protected } Access specifier

(4.) **Polymorphism** : many - forms.

OOPs – Object Oriented Programming system.

* uses of objects and classes.

Features of OOPs ——— Encapsulation
Abstraction
Inheritance
Base (Parent) / Child

Polymorphism.
many forms
Eg: operators, functions.

CONCEPTS OF Object Oriented programming System

* It uses objects in programming and aims to implement real world entities like inheritance, polymorphism etc.

* **Heading object**

These are basic run time entities in OOPS. Where objects are instances of a class. and provides its reference. It is user defined.

* **Class**

It is also user defined and has a collection of data and function

**Encapsulation**

It is the concept of wrapping the data and functions into a single unit. The data is not accessible to the outside world and only those functions which are wrapped in the class can access it...

**Data Abstraction**

It refers to providing only needed information

to the outside world and hiding implementation details.
The advantage of abstraction is we can change implementation at any point of time but the user wont get affected because the interface remains same.

## Inheritance

It is the process by which objects of one class acquires the property of objects in another class.
It supports the concept of hierarchical classification.
It provides reusability to the data and functions.

## Polymorphism

It means ability to take more than one form.
It supports operator overloading and (C++) function overloading which means only operator except a few in C++ can be overloading and functions can also be overloaded.

WAP to enter student details and display. (Outside defi).

```
#include <iostream.h>
#include <conio.h>
class BCA
    { public :-
      int age ;
      char name [20];
    void getdata (void);
    void display (void);
    };

void BCA :: getdata (void)
    {
      cout << "enter :" << endl ;
      cin >> name;
      cout << " enter age :" << endl;
      cin >> age;
    }
void BCA :: display (void)
    {
      cout << "name :" << name;
      cout << "age :" << age;
    }
    void main ()
    {
      clrscr ();
      BCA a1 , a2,
      a1. getdata ();
```

```
a2.getdata();
a1.display();
a2.display();

getch();
}
```

WAP to enter student detail and display (inside def).

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>

class BCA
{ public:
    int age;
    char name[20];

    void getdata(void)
    {
     cout << "Name: ";
     cin >> name;
     cout << "Age: ";
     cin >> age;
    }
    void display(void)
    {
     cout << "Name: " << name;
     cout << "Age " << age;
    } a1;
```

```
void main()
{
 clrscr();
 a1.getdata();
 a1.display();
 getch();
}
```

WAP : { emp name, sal, add sal, display. (outside class defination)

5 employes.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>

class employee
   { char name[20];
     float salary;

     void enter(void);   // There will not be garbage value;
     void display(void);
     float expense(int)
   } e[5];

void employee::enter(void)
   {
     cout << "Name: ";
     gets(name);
     cout << "salary: ";
     cin >> salary;
   }
```

```
void Employee :: display(void)
{
    cout << "Name : " << name;
    cout << "\n salary : " << salary;
}
float Employee :: expense (int *e)
{
    int i; float total = 0.0;
    for (i=0 ; i<5 ; i++)
    {
        total += e[i].salary;
    }
    return total;
}
void main()
{ clrscr();
  for (int i = 0; i< 5 ; ++i )
  {
      e[i]. enter();
  }
for (int j = 0; j<5 ; ++j)
{
    e[j] . display();
}
float sum = e[1]. expense(e);
cout << "Total expense home : \n";
cout << sum;
getch();
}
```

loop should always be used only in the main.

WAP to enter marks of 5 students along with their names and also calculate the avg marks. (using inside class definition)

```
#include <iostream.h>
#include < conio.h>
#include < stdio.h>

class student
{ public :
    float marks ;
    char name[20];
    float avg ; sum = 0.0 ;
    void input(void)
    {
        cout << "Enter the details of ~~object~~ student" <<
        cout << " Enter the marks and name" ;
        for(i=0; i<5 ; i++)
        {
            cin >> marks[i] ;
            gets (name[i]);
        } sum = sum + marks[i];
    }
    void display (void)
    {
        avg = sum/5 ;
        for (i=0 ; i<5 ; i++)
        {
    puts (name[i]) ;
        } 51;
```

```
void main()
{
  clrscr();
  s1.input();
  s1.display();
  getch();
}
```

## STATIC DATA MEMBER

There are certain specific characteristics of a static member variable.

(i) It is initialized to '0' when the first object of its class is created ...

(ii) Only one copy of that member is created for the entire class ..... and is shared by all the objects of that class.

(iii) It is visible only within the class but its lifetime is the entire program.

* Static variable are normally used to ~~hold~~ maintain values to the entire class.
Eg: a static data member can be used as a counter that records the occurances of all the objects.

### Static member function

* A static member function can be declared only if it fulfills certain properties

(i) A static function can have access to only other static members (variables and function) declared in the same class.

(ii) A static member function can be called/invoked using a class name instead of object.

class name :: func. name;

↓
Scope resolution Operator

Page No.

by default of.

The static function is used to keep a count of numbers of object created and maintained by static variable.

Points to keep in mind

1. Static variable
2. Static function
3. Calling of static variable for linking
4. Invoking of static function

Page No.
Date

Program of static variable and function

```
#
#
class bca
{       public:
        int x;
        static int count;

        void getdata(void);
        static void display(void);
};

void bca :: getdata(void)
{
    cout << " Enter x:";
    cin >> x;
    ++x;
}

void bca :: display(void)
{
```

Using static variable & member function print 1 to 10 counting.

```
#
#
class member count.
{   public:
    int x;
    static int counting;
    static void print();
};
int count :: counting;
void count :: print();
{   counting ++;
    cout << counting << endl;
}
void main()
{ clrscr()
    for(i=0; i<10; i++)
    { cout :: print(); }
  getch();
}
```

## THIS POINTER

```
#include <iostream.h>
#include <conio.h>
class bca
  { public:
      void getdata(void)
      {
        cout << "Add :" << this;
      };
    void main()
    {
      clrscr();
      bca a1, a2, a3;
      a1.getdata();
      a2.getdata();
      a3.getdata();
      getch();
    }
```

This is a unique keyword in c++ which is used to store the address of an object associated with a class.

It represents an object that invokes a member function. 'This' is a pointer that points to object for which 'this' function was called.

above program.

Page No.
Date

# CONSTRUCTOR

* It is a special member function whose task is to initilize the object of its class.
* It is special because It has same name as of class name. and it is invoked automatically when the object of its class is created.
* It has some special characteristics :

(a) They should be declared in public section.
(b) They are invoked automatically when the objects are created.
(c) They do not have return types, not even void because they can-not return values.
(d) It can't be inherited but it can be used through a derived class
(e) constructor can-not be virtual.
(f) An object with a constructor or destructor can not be used with a . operator.

## Default constructor

* It does not have parameters or if it has parameters, all the parameters have default value.

A constructor is used to state the behaviour of an object and must not have a return type.

## Parameterized Constructor

* It is used for initialising the argument or parameters
* It is necessary to initize various data elements of different objects with different values when they are created. This is achieved by using and passing arguments to constructor function when the objects are created, so the constructor that can take arguments are called parameterized constructor.

## COPY Constructor

It is a constructor which creates an object by initialising it with an object of the same class which has been created previously.

The copy constructor is used to

(a) Initialising one object from another of the same type
(b) Copy an object to pass it as an argument to a function
(c) Copy an object to return it from a function

A copy constructor is a member function which initializes the object using another object of the same class.

## Syntax

### (1) Default constructor

```
Class xyz
{
  public :
  xyz (void)          // constructor declared
{
};                    // constructor defined
void main()
{
  xyz x1;             // constructor invoked
  getch();            automatically when
}                     object created.
```

### (2) Parameterized constructor

```
class xyz
{
  public :
  xyz (int, int)      // constructor with
{                     Parameter declared.

};                    // defined.
void main()
{
  clrscr();
  xyz a1 (1,2);       // invoked and value passed
  getch();
}
```

3. Multiple constructor

```
class xyz
{
 public:

 xyz(void)
{
}
 xyz(int,char)
{ }
};
void main()
{
  clrscr();
 xyz x1;
 xyz x1(1,9);
 getch();
}
```

Program

Default Constructor

```
class employee
{
 public:

employee(void)
{ float salary;
 cin >> salary;
 cout << salary;
}
};
void main()
{
  employee e1;
 getch();
}
```

Parameterized Constructor

```
class employee
{
 public:
employee(int a, int b)
{
 int area;
 area = a*b;
 cout << area;
}
};
```

```
void main()
{
  clrscr();
  int c,d;
  cin >> c;
  cin >> d;
  employee e(c,d);
  getch();
}
```

## DESTRUCTOR

It is a special member function which destructs or deletes an object. It works just opposite to constructor unlike constructors that are used for initialising an object, destructors destroy the object; similar to constructor the destructor name should exactly match with the class name but it is followed by a tilde sign. (~)

Explicit Call to destructor is only necessary when object is placed at a particular location in memory by using new operator.

When is destructor called ??

1. A destructor function is called automatically when the object goes out of scope.
2. The function ends.
3. The program ends.
4. A block containing local variable ends.
5. A delete operator is called.

* Destructors don't take any argument and don't return anything.

Syntax :

```
class xyz
{ public:
  xyz(void);
  ~xyz();
};
```

Program : Destructor Implicit program.

```
class xyz
{
  public:
  xyz (void)
  { int a;
  cout << "Enter the number";
  cin >> a;
  a++;
  cout << a;
  }
  ~xyz (void)
  {
  cout << "memory freed";
  }};
void main ()
{
  clrscr();
  xyz x1;
  //~xyz();
}
```

Output :

IMPLICIT CALLING

```
#
#
class dest
{ public:
int d, b, area;
dest ();   { cout << " constructor called";
~dest ();
void area (void)
{
  cout << " Enter length";
  cin >> d;
  cout << " Enter breadth";
  cin >> b;
  area = d * b;
  cout << " area is " << area;
}
void main()
{
  clrscr();
  dest x1;          // Constructor invoked.
  x1. area();       // area function invoked.
  getch();
}
```

# FRIEND FUNCTION

Friend function declaration is preceded by the keyword friend. A friend function possesses certain characteristics :

1. It is not in the scope of the class only to which it has been declared as friend.
2. It is not in the scope of class so it can-not be called using object of that class.
3. It can be invoked like a normal function without the help of any object.

```
Eg: friend mean(char);
    :
    void main
    {
      mean(x);
```

4. It can-not access the number names variable. directly and has to use an object name with (.) operator with each member
5. It can be declared either in the public or the private part of class.
6. Friend functions are used in operator overloading
7. Member function of one class can be friend function of another class and in such case they are defined using scope resolution ~~numbe~~ operator.

* All the member function of one class can also be declared as friend function of another class. and that concept is known as friend class.

## MULTIPLE CLASS

```
# include <iostream.h>
# include <conio.h>
class A
{
 public:
  int x,y;
  void enter (void)
  {
  cout << "Enter value of x & y :" << endl;
  cin >> x >> y;
  cout << "sum of x & y =" << x+y << endl;
  }
};
 class B
 {
  public:
  int p,q;
  void enter (int,int)
  {
   cout << "Enter value of p & q" << end;
   cin >> p >> q;
   cout << "sum of p & q = " << p+q << endl;
  }
 };
  void main ()
 { clrscr();          B b1;
  int o,m;            b1. enter(o,m)
  A a1;               getch();
  a1.enter()          }
```

## FRIEND FUNCTION

```
#include <iostream.h>
#include <conio.h>
class Program
{
  private:
  int a;
  int b;
  public:
 void value (void)
 {
   a=25;
   b=30;
 }
  friend float average (program p)
 {
     p.value();
     return (p.a+p.b)/2.0;
 }
};
  int main()
  {
    clrscr();
    program x
    x.value();          // Invoke.
    average(x);         // invoke friend function
 cout << "The value of average is" << average(x);
  getch();
  return 0;
 }
```

```
#include < iostream.h>
#include < conio.h>
class program
{
   private :
    int l ;
    int b ;
   public :
   void value (void)
   {
      l = 10;
      b = 5;
   }
   friend float area ( program a)
   {
          a.value()
      return ( a.l * a.b) ;
   }
};
   int main ()
   {
      program p;

      p. value ();
   // area (p);
   cout << " The area of rectangle is "<< area (p);
      getch ();
      return 0;
   }.
```

# POLYMORPHISM

Poly + Morphism

- Run time
  - ↓
  - virtual function
- Compile time
  - operator overloading
  - function overloading.

It is made up of two words Poly and morphism where poly means many and morphism means forms Polymorphism is a crucial technique of ~~oops~~ (OOPS)

Polymorphism occurs when there is a hierarchy of classes and they are related There are two types of Polymorphism

1. Compile time
2. Run time

* Compile time polymorphism is executed and implemented using overloading (Function and operator) where as run time polymorphism is implemented using virtual function.

## COMPILE TIME POLYMORPHISM

(•) **Function overloading**

It is a technique in which a program can have functions with same name but different arguments (in terms of number or types).

(•) **Operator overloading**

It is a technique where an operator can be represented and reused as many times as a program demands

To define the task of an operator we must specify what it means in relation to the class to which the operator is applied.
This is done with the help of a special function called operator function which describe the task to it.

## RUN TIME POLYMORPHISM

(•) **Virtual function**

It is a technique of polymorphism used to implement run time polymorphism where a member function of a class is represented as a function which does not occupies any memory but exist only at the run time.

Virtual function can be invoked using arrow operator. C++ supports a mechanism in which, at run time the class objects are under consideration and the appropriate version of function is invoked. Since the function is linked with a particular class much later after the compilation. This process is termed as **LATE BINDING** and it is also known as **Dynamic binding** because the selection of the appropriate function is done dynamically at run time.

19/8/19.

# FUNCTION OVERLOADING

```
#include <iostream.h>
#include <conio.h>
class abc
{
   int a;
  void enter (void)
  {  int rollno;
     cout << " Enter the details ";
     cin >> rollno;
  }

  void enter (int b, int c)
  {
     cout << " Enter the id and marks ";
     cin >> b >> c;
  }

}
void main ()
{
   abc a1;
  a1. enter ();
  a1 . enter (5, 60);
  getch ();
}
```

```
#include < iostream.h>
#include < conio.h>
class xyz
{
   public :
  int func (int x, int y)
  {
     return (x-y);
  }
  int func (float a, float b)
                    float c
  {
     return ( a+b);
  }
  void main
  {
     xyz s;
  cout << " The result of x and y is " << s.func(4,3
  cout << " The result of a and b is " << s. func(4.0,
                                                  7.
   getch();
}
```

## OPERATOR OVERLOADING

```
#
#
class abc
{
  public :
    int rollno;
    char grade;
    float marks;

abc (int m, char n, float g)
{
    rollno = m;
    grade = n;
    marks = g;
}
  void operator++(int)
{
    rollno = rollno + 1;
    grade = grade + 1;
    marks = marks + 1;
}
};
  int main ()
{
  abc s (22, 'a', 20.1)
  cout << " Before overloading " << s.rollno
     << s.grade << s.marks;
  s++;
cout << " After overloading " << s.rollno << s.grade <<
  s.marks;
  getch();
}
```

## VIRTUAL FUNCTION

* **Rules for virtual function :**

1) Its a member fn which is redefined in the derived class and the fn can be overridden.

2) Virtual function can-not be static & also cannot be friend of another class.

3) Virtual function defined in base class should be accessed using pointer or reference to achieve run-time polymorphism.

4) If we have virtual function & it is overridden in derived class then we do not need virtual keyword in the derived class.

5) A base class pointer can point to the object of base class as well as to the object of derived class.

6) For invoking a virtual function, we do not need dot operator instead we will use arrow operator (→)

```
#
# class base
{
  public :
  virtual void show ()
  {
    cout << " We are in derived class " ;
  }
};
  int main ()
  {
    base * b ;
    derived d ;
    b = &d ;
    b → show ();
     getch();
    return 0;
  }
```

class derived : public base
{ public :
  void show ()
  { cout << "We are in derived class" }

# INHERITANCE

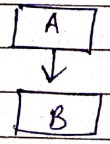
Inheritance is a feature of OOPs (Object oriented programming systems) which allows the creation of a new class (base) with derivation of its own features and allows child class to have its own unique features

This is basically done by creating new classes and reusing the properties of the existing one.

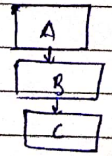
The mechanism of deriving a new class from the old one is known as Inheritance.

## Types of Inheritance

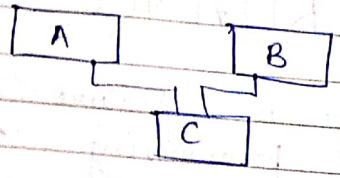
1) Single level Inheritance



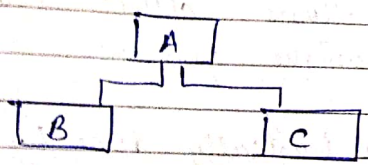

2) Multi-level Inheritance



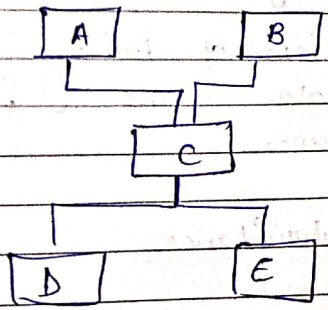
3) Multiple Inheritance




(4) Hierarchical Inheritance



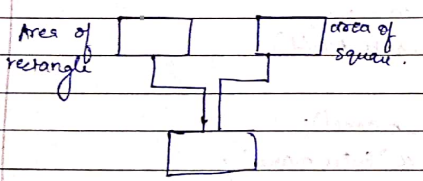

(5) Hybrid Inheritance




```
class A
{
  public:
  int roll;
  char name;
  void get_name(void)
  {
    cout << "Enter name";
    cin >> name;
    cout << "Enter rollno";
    cin >> rollno;
  }
};
class B : public A
{
  public:
  int m1,m2,m3;
  void marks(void)
  {
    get_name();
    cout << "Enter marks";
    cin >> m1 >> m2 >> m3;
  }
};
class C : public B
{
  public:
  int sum;
  void display(void)
  {
```

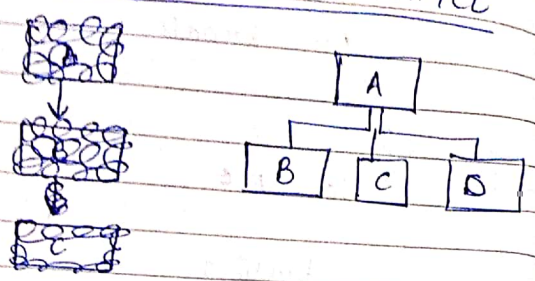
marks();
sum = (m1 + m2 + m3);
cout << "The name is" << name;
cout << "The rollno is" << rollno;
cout << "The marks are" << m1 << m2 << m3;
cout << "The sum is" << sum;
}
};
int main()
{
C obj;
obj.display();
getch();
return 0;
}

## MULTIPLE INHERITANCE



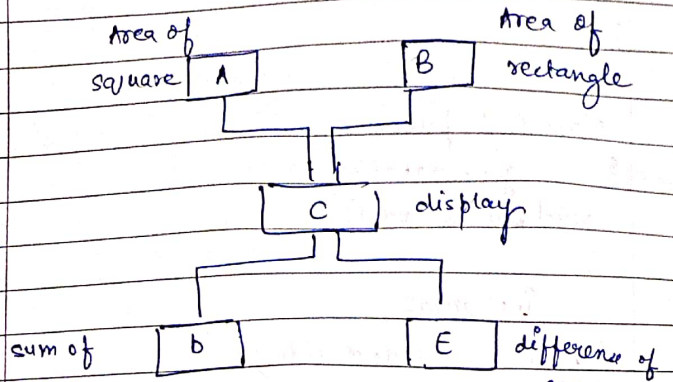

class A
{
public:
int length;
int breadth
void get_data(void)
{
cout << "Enter length";
cin >> length;
cout << "Enter breadth"
cin >> breadth;
}
};
class B
{
public:
int side;
void area(void)
{
cout << "Enter the side of square";
cin >> side;
}
};
class C : public b, public A
{
public:
int result;
void display (void)
{
area();
get_data();
cout << "The area of rectangle is" << length * breadth;
cout << "The area of square is" << side * side;
}
};
int main()
{
C obj;
obj.display();
getch();
return 0;

# HIERARCHICAL INHERITANCE




```
class A
{
    public :
    int l, b, side, base, height;

void get_data (void)
{
    cout << "Enter the values";
    cin >> l;
    cin >> b;
    cin >> side;
    cin >> base;
    cin >> height;
}
};

Class B : public A
{
 void calculate (void)
 {    get_data ();
      int area1;

 area1 = l * b;
 cout << " The area of rectangle is :" << area1;
 }
};

class C : public A
{    get_data();
 void area (void)
 {
     int area2;
  area2 = side * side;
  cout << " The area of square is :" << area2;
 }
};
class D : public A.
{

  void triarea (void)
  {
     int area3;
  area3 = 1/2 * base * height;
  cout << " The area of triangle is:" << area3;
  }
};
 int main ()
{
    D obj1;
    C obj2;
    B obj3;
  obj1. triarea ();
  obj2. area ();
  obj3. calculate ();
 return 0;
```

5. # HYBRID INHERITANCE




```
class A
{   public:
    int length;
    int breadth;

void calculate (void)
{
    int area1;
cout << " Enter the value of length and breath.";
cin >> length >> breadth;
area1 = length * breadth;
}
};

class B
{   public:
    int side;
void area (void)
{   int area2;
cout << " Enter the side of square ";
cin >> side;
area2 = side * side;
}
};

class C : public A, public B
{   public:
void display (void)
{ calculate ();
    area ();
cout << " The area of rectangle" << area1;
cout << " The area of square " << area2;
}
};

class D ; public C
{   public:
    void sum (void)
    {   int sum;
        display ();
sum = area1 + area2;
cout << " The sum of area is " << sum;
}
};
class E ; public C
{   public:
    void diff (void)
    {   int diff;
        display ();
diff = area1 - area2;
```

Page No.
Date

```
cout << " The difference is " << diff ;
 }
};
int main ()
{
    E obj;
    D obj;
    C obj;
  E. diff();
  D. sum();
  C. display();
  return 0;
}
```