OBJECT - ORIENTED PROGRAMMING

[OOPs]

JSG. Cbb

```
# include <iostream>
Using namespace std;
int main()
{
    int L, B, A;
    printf("Enter Length \n");
    Scanf("%d", &L);
    Cout << "Enter the Length : \n";
    Cin >> L;
    Cout << "Enter the breadth : \n";
    Cin >> B;
```

(or)

```
    Cout << "Enter Length and Breadth : \n";
    Cin >> L >> B;
    printf("\n Area = %d \n", A);
    A = L * B;
    Cout << "Area = " << A << "\n";
    Cout << A << " is Area \n";
Cout << "Length =" << L << "\n" << "Breadth =" << B;
```

<< → Insertion operator

>> → Exertion operator

output

```
Area = 50
50 is Area
Length = 5 Breath 10
```

class / ~~struct~~ BANK → A user defined data type

{ ← Private →

float bal ; // Publicly default in struct (Member Variable , Property, Attri

public :

Void openAccount ()

{

Cout << "Enter opening Balance : \n" ;

Cin >> bal ; // 500

}

Member functions, Methods, Behaviour

Void Show ()

{

Cout << "\n Balance = " << bal << "\n" ;

}

};

int main () // Non - Member function

{

BANK amn ; → an object

~~amn~~. bal ~~=~~ 500 ; // Error (Not Accessible)

✗ Cout ~~<<~~ amn. bal ~~<<~~ endl ; // 500 ✗

✓ amn. openAccount () ;

✓ amn. Show () ;

~~amn~~. bal ~~=~~ 100 ;

}

HOMEWORK

Q- Write a note on class in C++.

A- A class in C++ is the building block, that leads to object oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

Data members are the Data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behaviours of the objects in a class.

☆ An object is an instance of a class. When a class is defined, no memory is Allocated but when it is instantiated (i.e an object is created) memory is Allocated.

Q- Write differences b/w structure and classes.

A- In C++, a structure is same as a class Except for a few differences. The most important of them is security. A structure is not secure and cannot hide its implementation details from the end user while a class is secure and can hide its programming and designing details.

☆ Members of a class are private by default and members of a Struct are public by default.

* when Deriving a struct from class / struct, Default Access Specifier for a base class / struct is public. And when Deriving a class, default Access specifier is private.

Q- Write a program for Simple Interest using class.

A-
```
Class Simple Interest
{
    float P, R, T, S;
    public:
    Void Principal value ( )
        {
            Cout << " Enter principal value : \n";
        } Cin >> P;
    Void Rate of Interest ( )
        {
            Cout << " Enter Rate of Interest : \n";
        } Cin >> R;
    Void Time( )
        { Cout << " Enter Time : \n";
        } Cin >> T;
```

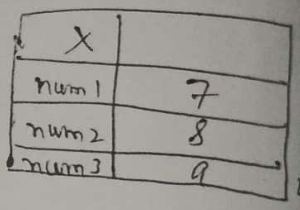
```
Void SI ()
{
    S = P * R * T / 100 ;
    Cout << " Simple Interest = " << S << "\n";
}
};
int main ()
{
    Simple Interest  x ;
    x. Principal value ();
    x. Rate of Interest ();
    x. Time ();
    x. SI ();
}
```

Q- Write a program for greatest for 3 no-s

A -

```
Class Greatest
{
    float A, B, C;
    public:
    Void num1 ()
    {
        Cout << " Enter the value of A: \n";
        Cin >> A ;
    }
    Void num2 ()
    {
        Cout << " Enter the value of B: \n";
        Cin >> B;
    }
    Void num3 ()
    {
        Cout << " Enter the value of C: \n";
        Cin >> C;
    }
```

```
Void Answer()
{
    if (A>B && A>C)
    Cout << " A is greatest";
    else if (B>A && B>C)
    Cout << " B is greatest";
    else
    Cout << " C is Greatest";
}
};
int main()
{
    Greatest X;
    X.num1();
    X.num2();
    X.num3();
    X.Answer();
}
```



(Service) Instance methods belong to instance (calling).

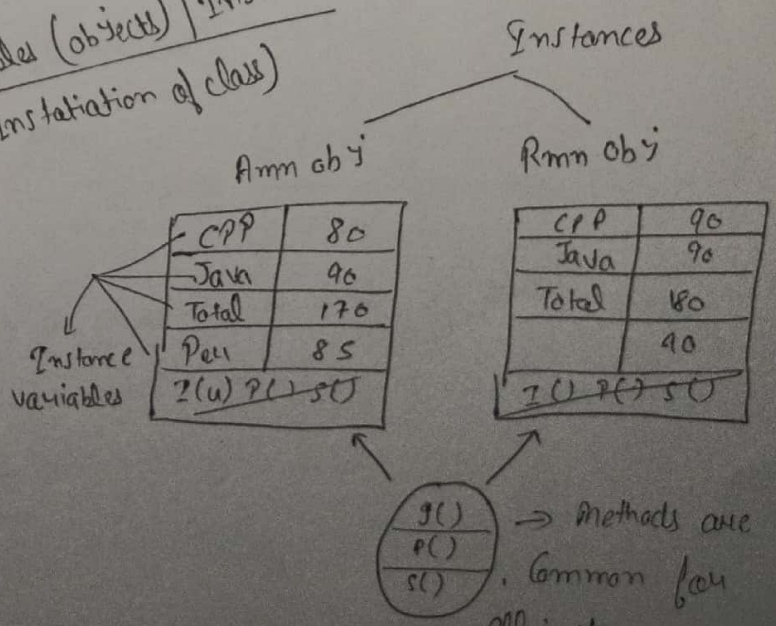
Imp Note :- Class doesn't Exist without main function, if there is a main function having instances, then only class Exist.

✱ Instance / Instant Variables :-

```
Class student
{
  int CPP, Java, total, Per ;  // Instance variables
                               (depends upon instances for existance)
  public :
    Void Input marks ()
    {
      Cout << " Enter CPP and Java marks : ";
      Cin >> CPP >> Java ;
      process ();  // Calling (Nesting the methods)
    }
    Void Process ()
    {
      total = CPP + Java ;
      per = total * 100 / 200 ;
    }
    Void show ()
    {
      Cout << " Total = " << total << " Per = " << Per << "\n" ;
    }
};
int main ()
{
  Student Amn obj, Rmn obj ;
  Amn obj . Input marks ();
  Rmn obj . Input marks ();
  Amn obj . show ();
  Rmn obj . show ();
}
```

(Service) Instance methods belong to instance for (Calling).

Variables (objects) / Instances
(Instatiation of class)



Home Work

Q- Write a program for volume of Cone and Cylinder using C

A-

```
Class Volume
 {
    float r, h
    public:
    Void radius ()
     {
       Cout << " Enter Radius : \n";
       Cin >> r;
     }
    Void height ()
      {
       Cout << " Enter height : \n";
       Cin >> h;
      }
    Void VCO ()
     {
       V1 = 3.14 * r * r * h/3;
       Cout << " Volume of Cone : " << V1 << "\n";
     }
    Void VCL ()
      {
       V2 = 3.14 * r * r * h;
       Cout << " Volume of cylinder : " << V2 << "\n";
      }
};
```

```
int main ()
{
    Volume   Cone, Cylinder;

        Cone . Radius ();
        Cone . height ();
        Cylinder . radius ();
        Cylinder . height ();
        Cone . VCo ();
        Cylinder . VCL();
}
```

Q - Write a program for Gross Salary.

A -

```
Class Salary
    {
        float da, hra, base Salary = 10000, Gross;
        public:
        Void DEAR ()
          {
            Cout << " Enter Dear Allowance : \n";
            Cin >> da;
          }
        Void Rent ()
          {
            Cout << " Enter House Rent Allowance : \n";
            Cin >> hra;
          }
        Void Total ()
          {
            Gross = base salary + hra + da ;
            Cout << " Gross Salary = " << Gross ;
          }
    };
```

```
int main()
{
    salary Prince, hardik;
    Prince.Dear();
    Prince.Rent();
    hardik.Dear();
    hardik.Rent();
    Prince.Total();
    hardik.Total();
}
```

Q- write a menu driven program for Bank machine.

A-

```
Class Bank
{
    float bal = 5000, amount, total, Cb;
    int ch;
    public:
    void deposit()
    {
        Cout << " Enter amount to be deposited : \n";
        Cin>> amount;
        total = amount + bal;
        Cout << " Total balance : " << total << "\n\n\n\n";
    }
    Void withdraw()
    {
        Cout << " Enter amount to be withdrawn : \n";
        Cin >> amount;
        Cb = bal - amount;
        Cout << " Current balance : " << Cb << "\n\n\n\n";
    }
    Void Enquiry()
    {
        Cout << " balance : " << bal << "\n\n\n\n";
    }
    Void Cases()
    {
```

Write a program for ATM Machine

```
class ATM
{  int P;
   void Insert()
   {
      cout << " Insert ATM Card";
   }
   void Pin()
   {
      {  cout << " Enter 4-Digit Pin" << "\n";
         cin >> P;
         Pin++;
      }
```

```
do
{
   cout << "**** MENU DRIVEN PROGRAM*** \n";
   cout << " Press 1 for deposit \n";
   cout << " Press 2 for withdraw \n";
   cout << " Press 3 for balance Enquiry \n";
   cout << " Enter your choice : \n";
   cin >> ch;
   switch(ch)
   {
      case 1: deposit(); break;
      case 2: withdraw(); break;
      case 3: enquiry(); break;
   default : cout << " Invalid choice\n";
} } while (ch != 0);
```

```
int main()
{
   Bank x;
   x. Cased();
}
```

Q- Write a program for ATM Machine

A- ~~class ATM~~

```
{
    int P, A, bal = 18000, code, prince;
    public:
    Void Insert()
    {
        Cout << " Insert ATM Card : \n";
            Code = 123;
        if (code != 123)
        Cout << " Sorry Wrong Card";
        else
            prince = 1;
    }

Void Pin()
{
    if (prince != 1)
    Cout << " Complete previous Task" << "\n";
    else if (prince == 1)
    {
      Cout << " Enter 4-Digit Pin";
      Cin >> P;
      if (P == 5678)
          prince = 1;
      else
    }  Cout << " Wrong Pin";
}
```

```
void Amount ()
    {
      if (bPince != 2)
        Cout << " check your pin";
      else if (pPince == 2)
        {
          Cout << " Enter Amount to be withdraw";
          Cin >> A;
          if ( A > 18000)
            Cout << " In Sufficient balance";
          else
            Cout << " Collect your Cash and Receipt";
        }
    }
};

int main ()
{
    ATM  x;
    x. Insert ();
    x. Pin ();
    x. Amount ();
}
```

// Methods with Arguments (Setter Method) With Return (Getter)

```
Class Student
{
    float CPP, Java, total;  // Instance variables
    public:
        Set marks
        void Input (float C, float J)  // Setter
        {
            CPP = C;
            Java = J;
        }
        void process ()
        {
            total = CPP + Java;
        }
        Get total
        float Show ()  // Getter
        {
            return (total);
        }
};
int main
{
    Student amn obj;
    amn obj. Set marks (80,90);
    amn obj. process ();
    float total = amn obj. Get total ();
    Cout << "In total =" << total;
    float per = total*100/200;
    Cout << "In per =" << per;
}
```

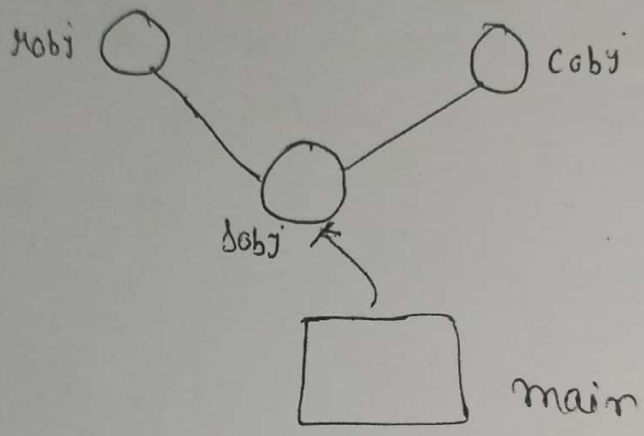
Local variables

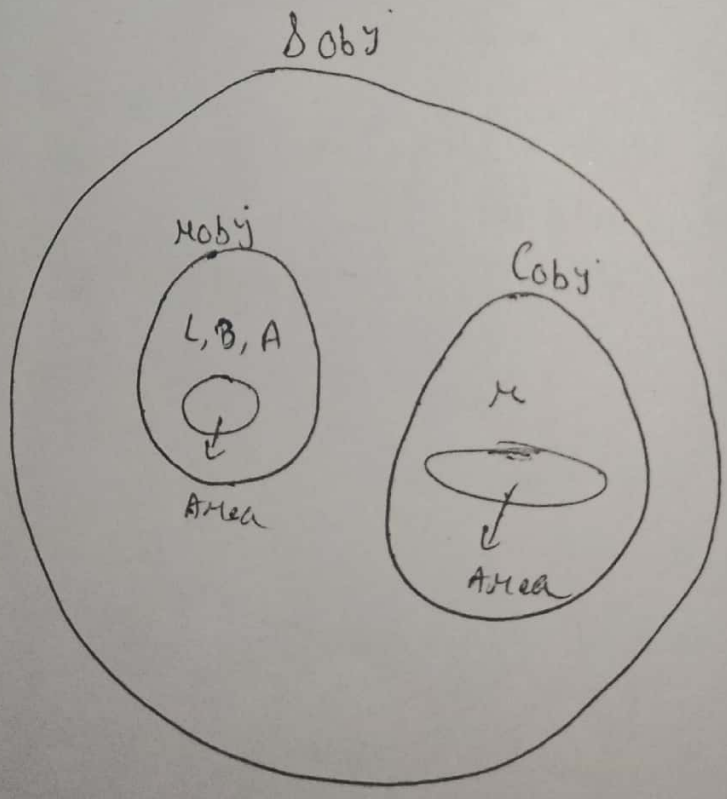
Container class | Composition

A class containing object of another class is known as Container class. The methods of Container class can use these objects for calling other member functions.

```
Class RECT
  {
    int L, B, A;
    public:
      Void R area ()
        {
          Cout << " Enter L, B: \n";
          Cin >> L >> B;
          A = L * B;
          Cout << A << "\n";
        }
  };
Class Circle
  {
    int r; float A;
    public:
      Void C area ()
        {
          Cout << " Enter radius : \n";
          Cin >> r;
          A = 3.14 * r * r
          Cout << " Area of Circle: \n";
        }
  };
Class shapes  // Container class
  { RECT robj;
    Circle cobj;
    public:
    Void Call All ( )
      {
        robj. R area ();
        cobj. C area();
      }
```

```
int main()
{
    Shapes    Sobj;
        Sobj.CallAll();
}
```





Composition

Home Work

Write a program for greatest of 3 No's using Get & Set

```
Class Greatest
{
  int P, Q, R;
  public:
    void Setnum (int A, int B, int C)
    { P = A;
      Q = B;
      R = C;
    }
  int process ( )
    {
      if (P > Q && P > R)
        return P;
      else if (Q > P && Q > R)
        return Q;
      else
        return R;
    }
};
int main ( )
{
  Greatest X;
  int A, B, C;
  Cout << " Enter Value of A, B, C. \n";
  Cin >> A >> B >> C;
  X. Setnum (A, B, C);
  int R = X. process ( );
  Cout << R << " is greatest";
}
```

Q.: Write a program for volume of cylinder and Cone using set

A-

```
Class Volume
{
    float r, h, VCO, VCL;
    public:
    Void Set dimensions (float A, float B)
        {
            r = A;
        }   h = B;
    Void process ()
      {
          VCO = 3.14 * r * r * h / 3;
          VCL = 3.14 * r * r * h;
      }
    float Cone ()
      {
        return (VCO);
      }
    Cylinder
    float [redacted] ()
      {
        return (VCL);
      }
};
int main ()
  {
      volume CO; CY;
      float A, B;
      Cout << " Enter radius & height: \n";
      Cin >> A >> B;
      CO. Set dimensions (A, B);
      CY. Set dimensions (A, B);
      CO. process ();
      CY. process ();
```

```
float P = Co.Cone();
    Cout << "Volume of Cone : " << P;
float R = Co.Cylinder();
    Cout << " Volume of Cylinder : " << R;
}
```

Q- Write a program for Reverse of a No. using Get & Set.

A-

```
class Reverse
{
    int A, Rev=0;
    public:
    Void Set value (int P)
        { A = P;
        }
    Void process ()
        {
            int R;
            While (A != 0)
                {
                R = A % 10;
                Rev = Rev * 10 + R;
                A = A/10;
                }
        }
    int Get_num ();
        { Return Rev; .
        }
};
int main ()
{
    Reverse X;
    int P;
    Cout << " Enter Value of P : \n";
    Cin >> P;
    X.Set value (P);
    X.process ();
    int M = X.Get num ();
    Cout << " Reversed number : "
         << M;
}
```

Q- Write a program for Gross salary using Container class

A-

DOUBT

ON NEXT PAGE

```
Class DR
{
  float Da;
  public:
  Void Dear Allowance (float A)
      {
        Da = A;
      }
};

Class HRA
{
  float HA;
  public
  Void House Rent (float B)
      {
        HA = B;
      }
};

Class Salary
{
  DR    Hardik, Prince;
  HRA   Hardik, Prince;
  float b1 = 10000., b2 = 20000;
```

Q- Write a program for Gross Salary using Get & Set

A-

```
Class Salary
  {
    float Da, hra, base salary = 10000, Gross;
    public:
      Void Dear allowance (float A)
        {
          Da = A;
        }
      Void House rent (float B)
        {
          hra = B;
        }
      Void process ()
        {
          Gross = base salary + Da + hra;
        }
      float Get Salary ()
          {
            Return Gross;
          }
  };
int main ()
  {
      float A, B, A1, B1;
    Salary  Hardik, Prince;
      Cout << " Enter Dear Allowance & House Rent: \n";
      Cin >> A >> B;
      Hardik. Dear allowance (A);
      Hardik. House rent (B);
      Hardik. process();
      float m = Hardik. Get Salary ();
      Cout << " Gross Salary = " << m;
```

& Hardik

```
Cout << "\n";
Cout << " Enter Dear Allowance & House rent for prince
Cin >> A1 >> B1;

Prince . Dear allowance (A1);
Prince. House rent (B1);
Prince . process ( ),
float N = Prince . Get salary ( );

    Cout << " Gross Salary of prince : " << N << "\n";
    Cout << " Gross Salary of hardik : " << M;
}
```

// ARRAY OF OBJECTS :- It is used to store and process data about no. of entities of same type.

```
Class EMPLOYEE
{
  float bs, da, hra, gs;  // private instance variables
  public:
  Void Input (int i)
   {
     Cout << " Enter basic of " << i+1 << " Employee : \n";
     Cin >> bs;
   }
  Void process ()
   {
     da = bs * 50/100;
     hra = bs * 30/100;
     gs = bs + da + hra;
   }
  Void Show ()
    {
      Cout << " gross salary = " << gs << " \n";
    }
  float Get gs ()
      {
        return gs;
      }
};
```

```
int main()
{ int N, i;
  Cout << "Enter No. of employee: \n";
  Cin >> N;
  EMPLOYEE E[N]; // creation of array of objects
  for (i=0 ; i<N; i++)
    {
      E[i]. Input (i);
      E[i]. process ();
    }
  Cout << " salary slip: \n";
  float max = E[0]. Get gs ();
    float total =0 ;
    for (i=0; i<N; i++)
      {
        E[i]. show ();
          if (max < E[i] . Get gs ())
                max = E[i] . Get gs ();
          total = total + E[i]. Get gs();
      }
  Cout << " Maximum salary = " << max << "\n";
}
```

E[0]	bs	100000
	da	50 k
	hra	30k
	gs	180 k
E[1]	bs	200000
	da	100000
	hra	60000
	gs	360k
E[2]	bs	—
	da	—
	hra	—
	gs	—

Input ()
process ()
show()

Salary slip:
— — —
— —
— —
Max. Salary: -

Referencing And Call by Reference

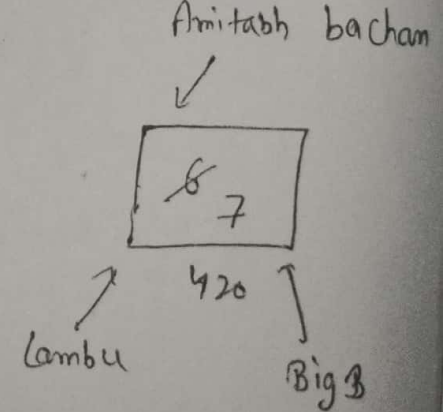
:- ~~The call by Reference of 'C' is~~ known as Call by Address in ++. Redefined the term referencing and call by reference.

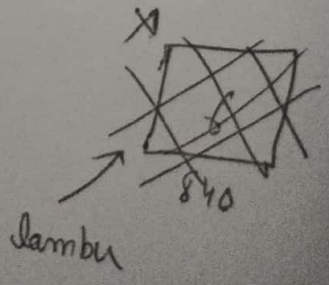
Definition :- The process of giving Alias name to Existing variable is known as referencing.

Eg :-

```
int main()
{
    int amitabh Bachan = 6;
    int & Lambu = amitabh bachan;
    Cout << Lambu << "\n" ; //6
    Lambu = 7 ; // modification
    Cout << amitabh bachan; //7
    int & big B = lambu;
    Cout << "\n" << big B ; //7
}
```

A reference Variable



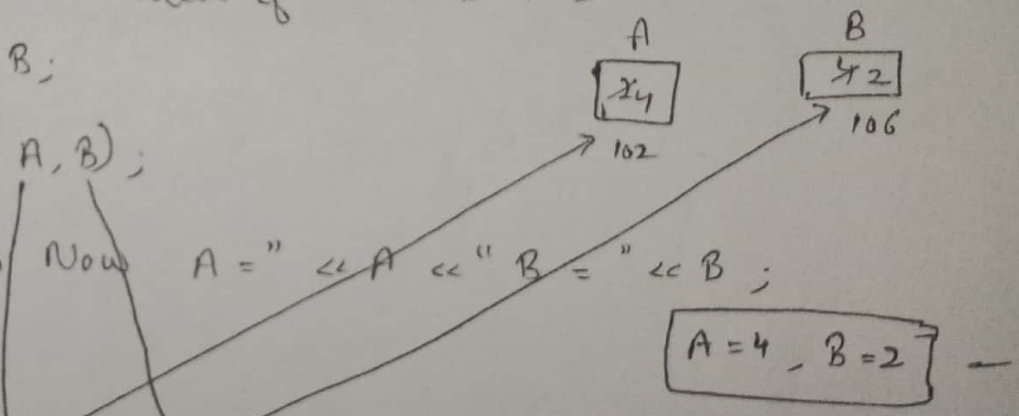


// Call by Reference

In call by Reference the formal Argument variables are taken as reference variables. If any change is made in the value using these reference variables, it will modify the original arguments

Eg :-

```
int main()
{
  Void SWAP (int &x, int &y);
    int A, B; // local variables of main()
  Cout << " Enter value of A and B : \n";
  Cin >> A >> B;
    Swap (A, B);
  Cout << " \n Now A = " << A << " B = " << B;
}
Void SWAP (int &x, int &y)
  {
    int temp = x;
        x = y;
        y = temp;
  }
```



Q- Write a program for N students using Array of ob... also find Maximum Percentage and Minimum Total mar...

A -

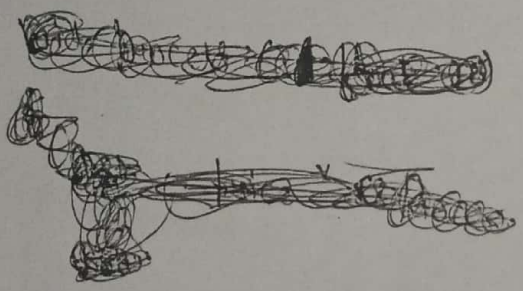
```
Class students
{
    float M, P, C, total, per
    public:
    void Input [redacted] (int i)
    {
        Cout << " Enter marks in Maths, Physics, chemistry of
                 i+1 << " student : \n" ;
        Cin >> M >> P >> C ;
    }
    void process1 ( )
    {
        total = M + P + C ;
    }
    void process 2()
    {
        per = M+P+C / 300 * 100 ;
    }
    float Get total()
    {
        return total ;
    }
    float Get Percentage ()
    {
        return per ;
    }
};
```

```
int main()
{ int N, i;
  Cout << " Enter No. of Students : \n";
  Cin >> N;
   Students s[N];
  for (i = 0; i<N; i++)
    {  s[i]. Input (i);
       s[i]. process1();
       s[i]. process 2();
     }
    float  Max = s[0]. Get Percentage ();
    float  Min = s[0]. Get total ();
  for (i = 0 ; i< N; i++)
    {   if ( max < s[i]. Get Percentage())
              Max = s[i]. Get Percentage ();
        if ( Min > s[i]. Get total())
              Min = s[i]. Get total ();
    }
    Cout << " Maximum Percentage = " << Max << "\n";
    Cout << " minimum Total = " << Min;
}
```

Q- Write a program for Array of objects of our choice most Expensive item

A-

```
Class SHOP
{
    float price, gst, amount
    public:
    void Input (int i)
    {
      Cout << " Enter price of " << i+1 << " item" <<
      Cin >> price;
    }
```



```
    void process ()
    {
      gst = price * 18/100;
    }
    void total ()
    {
      amount = price + gst;
    }
    float Get total ()
    {
      Return amount;
    }
};
```

```
int main()
{
    [redacted]
    int N, i;
    Cout << "Enter No of items : \n";
    Cin >> N;
    SHOP S[N];
    for (i=0; i<N; i++)
    {
        S[i]. Input (i);
        S[i]. process ();
        S[i]. total ();
    }
    float max = S[0]. Get total ();
    for (i=0; i< N; i++)
    { if (max < S[i]. Get total ())
        {
            max = S[i]. Get total ();
            p = i+1;
        }
    }
    Cout << " Most Expensive thing of store have price : " << max << "\n";
    Cout << " Item no. : " << P;
}
```

Scanned with CamScanner

Q- Write a program for sum of 2 No.'s Using Referencing

A -

```
class Number
{
    int A;
    public:
    Void Add (int &x, int &y)
    {
        x = 8;
        y = 10;
    }
};
int main ()
{
    Number x;
    int P, Q; int m;
    Cout << " Enter value of P and Q : \n";
    Cin >> P >> Q;
    x. Add (P, Q);
    m = P + Q;
    Cout << " Sum of P & Q = " << m;
}
```

★ PoRo :- Passing object Returning object

Class BANK → user defined Data type

```
{
    int Sb;
    int cb;
  public:
      Void SetbQ(int sb1 , int cb1)
          {
            Sb = Sb1;
            cb = Cb1;
          }
```

→ local variable

```
      Void show ( )
          {
            Cout << Sb << "    " << Cb << "\n";
          }

      Void barty ( BANK Moby)
          {
       (4)  Sb = Sb + Moby.sb;
       (6)  Cb = Cb + Moby.cb;
          }
```

→ local object

```
      BANK barty 2 (BANK aboby)
          {
       (7)    BANK bhai oby;
          bhaioby.Sb = Sb + aboby.sb;
          bhaioby.Cb = cb + aboby.cb;
       (10)
              return (bhai oby);
          }
};
```

Sb	4
cb	6

bhai oby

Sb	7
Cb	10

returnig object

```
int main ()
{
    BANK amn obj; // local object of main
    amnobj.setbal (1,2);
    BANK hmn obj;
    hmnobj.setbal (3,4);
    amnobj.party (hmn obj);   → passing object
    amnobj.show(); // 4-6
    hmnobj.show(); // 3-4
    BANK chmn obj;
    chmnobj = hmn obj.party2 (amn obj);
    chmnobj.show(); // 7-10
}
```

amnobj

Sb	1	4
Cb	2	6

hmn obj

Sb	3
Cb	4

chmn obj

Sb	7
Cb	10

Home Work

Q- Write a program for Difference b/w Height of 2 persons.

A-

```
Class Height
{
   int F ;     int H;
   int I ;

   public :
     Void Set Height ( int P, int Q)
          {
            F = P;
            I = Q ;
          }
     Void process ( )
     {
       H = F * 12 + I ;
     }
Height  Difference ( Height Prince)
     {
        Height Z ;
       Z. H =  H - Prince. H ;
     }      Return (z);
Void Show ( )
     {
        F = H/12;
        I = H%12;
        Cout << "Difference is : " << F << "feet " << I << "Inch";
     }
};
```

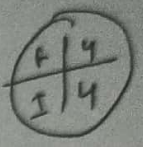
Local variables

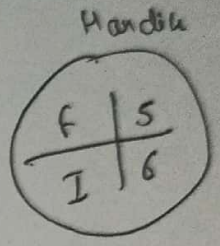
Local object

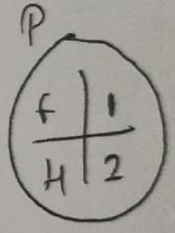
Returning object

```
int main ()
{
    Height Prince , Hardik ; // Local objects of main
    int P1, Q1 ;
    Cout << " Enter Height of Prince : \n" ;
    Cin >> P1 >> Q1 ;
    int P2, Q2 ;
    Cout << " Enter Height of Hardik : \n" ;
    Cin >> P2 >> Q2 ;
    Prince . Set Height (P1, Q1) ;
    Hardik . Set Height (P2, Q2) ;
    Prince . process () ;
    Hardik . process () ;
    Height P ;
    P = Hardik . Difference ( Prince) ;.
    P . show () ;
}
```

passing Instances







Q- Write a program for Difference b/w time of two clocks.

A-

```
class clock
{
    int H;      int U;
    int m;
    int S;

    public:
    void SetTime ( int P, int Q, int R)
        {
          H = P;
          m = Q;
          S = R;
        }
    void process ( )
    {
        U = H * 60 * 60 + m * 60 + S;
    }
Clock Difference ( Clock Y)
{
    Clock Z;
    Z.U = U - Y.U;
    return Z;
}
void show ( )
{
    H = U/(60*60);
    m = (U % (60*60)) /60;
    S = (U % (60*60)) % 60;
Cout << " Difference is : " << H << "Hours" << m << "mins" << S << "sec";
}
```

Local variable

Local instance

returning instance

```
int main ( )
{
    Clock X, Y; // Local instances of main
    int H1, m1, S1, H2, m2, S2;
    Cout << " Enter Time in clock X : \n";
    Cin >> H1 >> m1 >> S1;
    Cout << " Enter Time in clock Y : \n";
    Cin >> H2 >> m2 >> S2;
    X. Set time (H1, m1, S1);
    Y. Set time (H2, m2, S2);
    X. process ( );
    Y. process ( );
    Clock Z;
    Z = X. Difference (Y);
    Z. show ( );
}
```

X	
H1	2
m1	16
S1	20

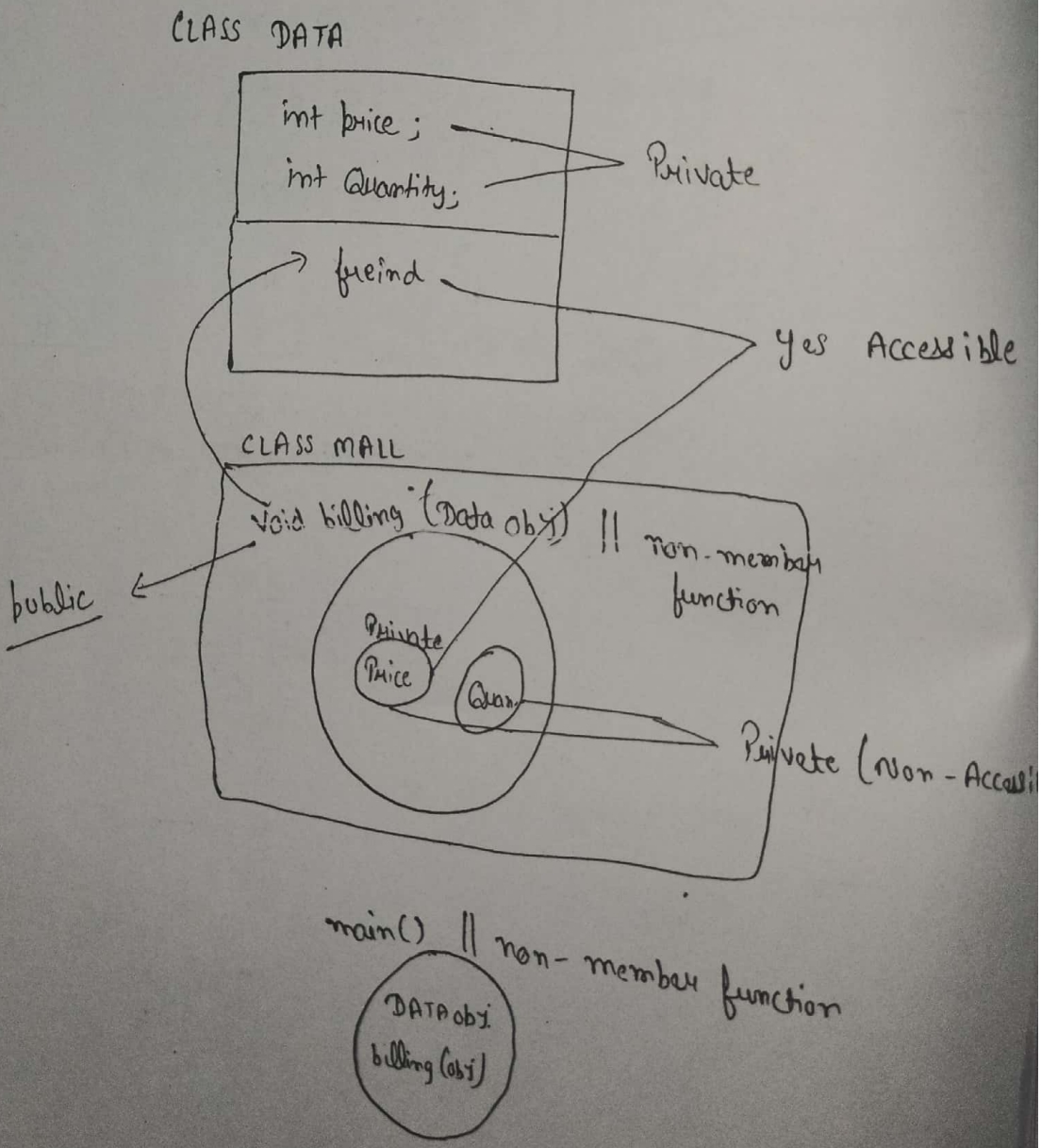
Y	
H2	1
m2	29
S2	40

Z	
H	1
m	19
S	40

passing instance

// Accessing private member of a class in non-member fun

FRIEND FUNCTION :- As we know that private variables of class are not accessible by a non-member function, but to make it possible the function must become freind of the classes of which it want to Access.



```
Class DATA
{
    int price, Quantity;
    public:
    Void Input()
      {
        Cout << " Enter price & Quantity : \n";
        Cin >> price >> Quantity;
      }
    freind Void billing (DATA obj);
};
Void billing (DATA obj)   // freind function
  {
    int bill = obj.price * obj Quantity;
    Cout << " Billing = " << bill;
  }
int main()
  {
    DATA obj;
    obj.Input();
    billing (obj);
  }
```

Receiving object

yes Accessible

~~ERROR~~

Passing object

* **Freind Class** :- A class cannot access private variables of other class but it is possible using freind as shown in program below.

```
Class PRODUCT
  {
    int price, Quantity;
    public:
      void Input ()
      {
        Cout << " Enter price and Quantity : \n";
        Cin >> price >> Quantity;
      }
      freind class MALL;
  };
Class MALL
  {
    float bill;
    public:
      void billing (Product obj)
      {
        bill = obj.price * obj.Quantity;
        Cout << "billing = " << bill;
      }
  };
int main ()
  {
    PRODUCT lab;
    MALL ml;
      lab.Input();
  }   ml.billing(lab);
```

passing object as argument

Home Work

Q- ✓ Class Salary
✓ Class Tax
✓ Class Net Salary → Using freind class

A-

```
CLASS SALARY
  {
    float Salary;
     public:
      void Input()
       {
         Cout << " Enter Salary : \n";
         Cin >> Salary;
       }
  };    freind class NETSALARY;

Class TAX
  {
    float Tax;
     public:
      void Input()
       {
         Cout << " Enter Tax : \n"
         Cin >> Tax;
       }
  };   freind class NET SALARY;
```

```
class NET SALARY
 {
   float Nt;
   public :
   void Net [redacted] ( SALARY X, TAX Y)
    {
      Nt = X.Salary -((Y.[redacted] Tax /100) * [redacted] X.Sala
      Cout << " Net Salary = " << Nt;
    }
};
int main ()
 {
   SALARY X;
   TAX    Y;
   NET SALARY Nt;
   X. Input ();
   Y. Input ();
   Nt. Net (X, Y);
}
```

passing object as Argument

Q-

- Class Salary
- Class Tax
- Net Salary

→ Using freind function

```
Class TAX ; // Forward Declaration, So that it can be used by Class SALARY.
Class SALARY
{
    float salary;
    public
    Void Input ( )
    {
      Cout << " Enter Salary : \n";
      Cin >> salary;
    }
    freind Void Net Salary (SALARY x, TAX y);
};

Class TAX
{
    float tax;
    public
    Void Input ( ).
    {
      Cout << " Enter Tax : \n";
      Cin >> tax;
    }
};  freind Void Net Salary (SALARY x , TAX y );
```

Compilation (Top to Bottom)

Execution (Bottom to top) (Starts from main())

```
Void NetSalary ( SALARY x, TAX y)
{
    float Nt;
    Nt = x.Salary - y.tax;
    Nt = x.salary - ((y.tax/100)* x.salary));
    Cout << " Net salary = " << Nt ;
}

int main ()
{
    SALARY x;
    TAX y;
    x.Input();
    y.Input();
    NetSalary (x, y);
}
```

passing object as Argument

```
a = Avg/12;
b = (Sum/12) -/.12;
Cout << "Average heigh :" << a << " feet" <<
    " Inches";
}
```

Q- Write a program using freind function of own choice.
Class HEIGHT 1;

A-

```
Class HEIGHT 1
{
    int f;
    public:
    Void Input ( int i)
    {
      Cout << " Enter feet of " << i+1 << " student " << "\n";
    } cin >> f;
    friend Growth (HEIGHT1 X [] , HEIGHT 2 Y [], int N);
};

Class HEIGHT 2
{
    int I;
    public:
    Void Input (int i)
    {
      Cout << " Enter Inches of " << i+1 << " Student << "\n";
    } cin >> I;
    friend Growth (HEIGHT 1 X[] , HEIGHT 2 Y [], int N);
};
Void Growth ( HEIGHT 1 X [ ], HEIGHT 2 Y [ ], int N)
{ int i, n, Sum = 0; float Avg ; int a, b;
  [illegible]
for (i = 0 ; i < N ; i ++)
  {
    Sum = Sum + [illegible] X [i]. F * 12 + Y[i]. I;
  }
  Avg = Sum/N;
```

```
int main ()
{
    [redacted] int N, i;
    Cout << " Enter No. of Students : \n";
    Cin >> N;
    HEIGHT 1  X[N];
    HEIGHT 2  Y[N];
    for (i=0; i<N; i++)
    {
        X[i]. Input (i);
        Y[i]. Input (i);
    }
    [redacted]
    Growth ( X, Y, N);
}
```

Q- write a program having friend class & friend function as well.

A-

```
class Mathematics
{
    int m;
    public:
    void Input ( )
    {
        Cout << " Enter marks in Mathematics :\n";
        Cin >> m;
    }
    friend class Total ;
};
class Science
{
    int S ;
    public :
    void input ( )
    {
        Cout << " Enter marks in Science :\n";
        Cin >> S;
    }
}; friend class Total;
class Total
{
    int total ;
    public ;
    void Tot ( Mathematics MA , Science SA)
    {
        total = MA.M + SA.S
    }
}; friend void Percentage ( Total TA);
```

```
void Percentage ( Total TA)
    {
        float P = TA.total /2 ;

        Cout << " Percentage = " << P ;
    }


int main ( )
  {
      Mathematics MA;
      Science SA;

      MA. Input ();
      SA. Input ();
      Total TA;
      TA. total (MA, SA);
          Percentage (TA);
  }
```

Q- A program in which Container class is a friend class

A- This program will not show any error but it will give wrong output.

```
Class SALARY
{ float salary;
  public:
    void Input ()
    {
      Cout << " Enter Salary : \n";
      Cin >> salary;
    }
    friend class Net salary;
};
Class TAX
{ float tax;
  public:
  void Input ()
  {
    Cout << " Enter Tax : \n";
    Cin >> tax;
  }
  friend class Net salary;
};
```

```
class Net Salary
{
    float P;
    public:
    SALARY X;
    TAX  Y;
    void InCome ()
    {
      P = X. salary -  ( (y. tax /100) * X. Salary);
    }
    void show ()
      {
        Cout << " Net Salary =" << P;
      }
    friend int main ();
};

int main ()
{
    SALARY x;
    TAX y;
     x. Input ();
     Y. Input ();
     Net Salary nt;
     nt. InCome ();
     nt. show ();
}
```

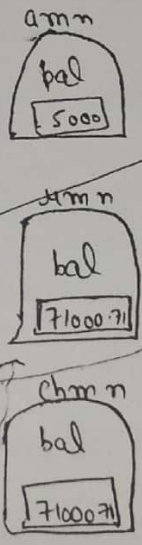
☆ Constructors and Destructors :-

- Constructors are special functions / methods which are called automatically when the object is created in memory.

```
Class BANK
{
  float bal;
public:
  BANK ()  // Default Constructor function
  {
    bal = 5000;
  }
  Void show ()
  {
    Cout << "In Balance =" << bal << "\n";
  }
  BANK (float b) // Parameterized Constructor
  {
    bal = b;
  }
  BANK (BANK &obj) // Copy Constructor
  {
    bal = obj.bal;
    obj.bal = 0;
  }
 ~BANK ()  // Destructor function
  {
    Cout << " Alvida ... \n";
  }
};
```

```
int main ()
{
  BANK amn;
  amn.show(); // 5000
  BANK hmn (71000.71);
  hmn.show (); // 71000.71
  BANK chmn (hmn);
  chmn.show(); // 71000.71
  hmn.show (); // 0
}
```

Constructor function called by default



~~by value~~ passing object by reference.

Note :- A Copy Constructor Cannot have object passed by value.

Features of Constructors :-

- Its Name should be same as class Name.
- It should not have Return type.
- As many Constructors are allowed as we want.

* **Destructors** :- They are special functions which are called automatically when the object is destroyed from memory.

Features of Destructors :-

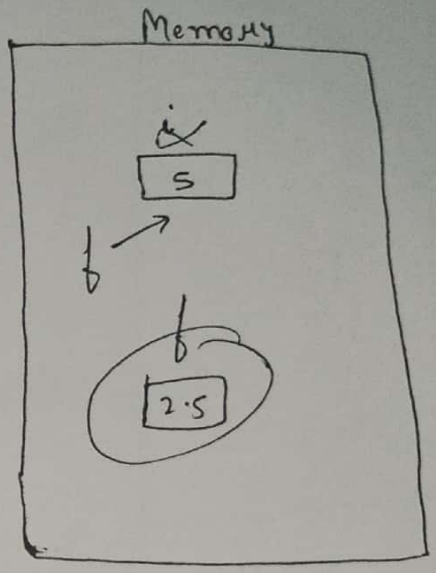
- Its name should be same as class name.
- It should not have Return type.
- preceded with tilde sign.
- only one distructor is allowed.
- It is used to release the resources occupied by an object, i.e. memory / file etc.

This pt will be understandable at last

* Dynamic Memory Allocation (DMA) :-

Reusability of memory

```
int main ()
{
    int i = 5;      → Dynamic Static Memory Allocation
    i
    free i;
    float f = 2.5;
    ix   f
}
```



Here Using DMA, the available Memory can be re-used. Here we use keywords new and delete. New is used for memory Allocation and delete is used for memory de-allocation.

Memory Allocation to Primitive Variable.

Syntax :-

```
Data Type * Pointer = new Data Type ();
delete Pointer;
```

new → Allocation

delete → Deallocation.

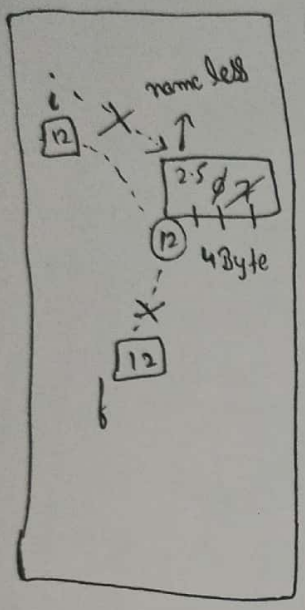
Eg :- int main ()

{

int *i = new int ( ) ; // Memory Allocation

Cout << *i << "\n" ; // 0

*i = 7 ;



Cout << "\n *i = " << *i << "\n" ; // 7

delete i ; // Memory De Allocation

float *f ;

f = new float ( ) ; // Memory re-allocation

*f = 2.5 ;

Cout << "\n *f = " << *f ; // 2.5

delete f ;

}

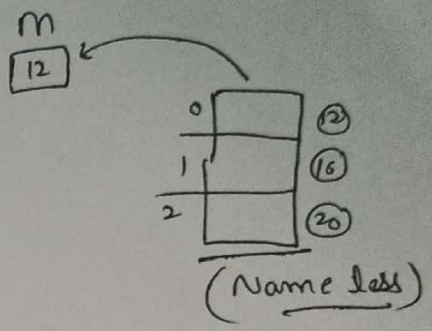
③ // Memory Allocation to Array

Allocation :- Data Type * Pointer = new int [size] ;

De Allocation :- delete [ ] Pointer ;

```
int main ()
{
    int N, i ;
    Cout << " Enter N : \n" ;
    Cin >> N ;
    int *m = new int [N] ; // Memory Allocation to Array
```

```
for (i=0 ; i<N ; i++)
{ Cout << " Enter value : \n";
  Cin >> m[i];
}
Cout << "\n output\n";
  for (i=0 ; i<N ; i++)
  {
    Cout << m[i] << "\n";
  }
  delete [] m ; // Memory De-Allocation
}
```



(3) // Memory Allocation to object

: Syntax :-

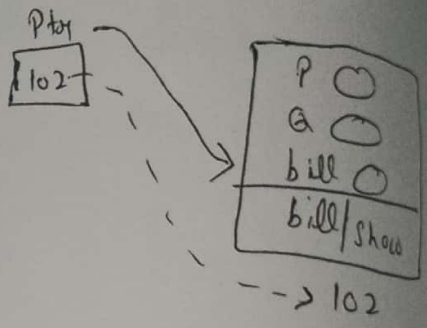
```
Class Name * Ptr = new class Name ();
delete Ptr;
```

```
Class PRODUCT
  {
    int price, Qty, bill;
    public:
      void billing ()  // Inline Methods
        {
          Cout << " Enter price & Quantity : \n";
          Cin >> price >> Qty ;
          bill = price * Qty;
        }
      void Show ();
  };
```

?

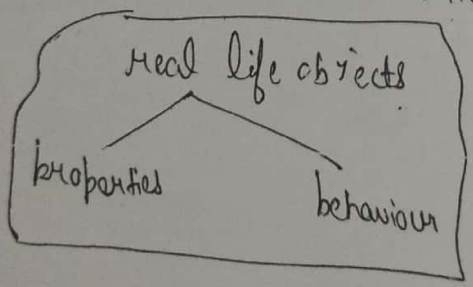
inline void PRODUCT :: show() // Non-Inline Methods

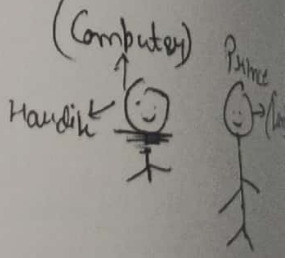
```
{
    cout << " Bill = " << bill;
}

int main()
{
    PRODUCT *Ptr;
    Ptr = new PRODUCT();
          creation of object

    (*Ptr).billing();
    or
    Ptr -> billing();
    Ptr -> show();
    delete Ptr;
}
```



Q- Why C++ is object-oriented Programming?

A-





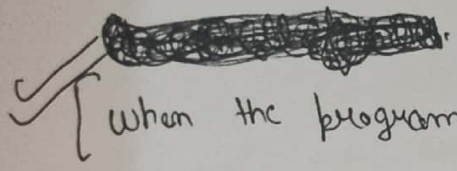

```
class BANK
{
    int P, R;
public:
    void billing();
};

int main()
{
    BANK. amn;
}
```

amn → [redacted] variable

Amn is an [redacted] variable but it has properties of class BANK, so it belongs to this class and hence called as object and we will work this object in C++, thats why it is known as object-oriented programming

Q- what are Inline Methods?

A- ⇒ C++ provides an inline functions to reduce the function call overhead. Inline function is a function that is Expanded inline when it is called. when the inline function is called whole code of the inline function gets substituted at the point of inline function call. This substitution is performed by C++ Compiler at compile time.

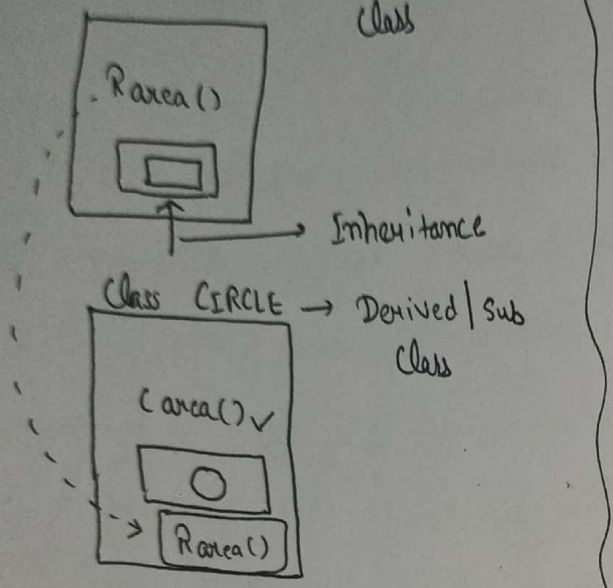


[when the program Executes the function call instruction the CPU stores the memory address of the instruction following the function call then executes the function code, & return control to Calling function This become overhead if the Execution time of function is less than the switching time from the caller function to called function]. and this overhead occur for small function.

Imp :- Inlining is a request to Compiler not a Command and Compiler can ignore it also or may not perform in such circumstances:

1) If a function Contains loop.
2) If a function is recursive.
3) If a function Contains static variable.
4) If a function Contains switch or goto statement.

★ **Inheritance** :- It is the process of creating new classes from existing classes. The existing class is known as base class and newly created class is known as derived/sub class. The actual benefit of Inheritance is reusability.



main()

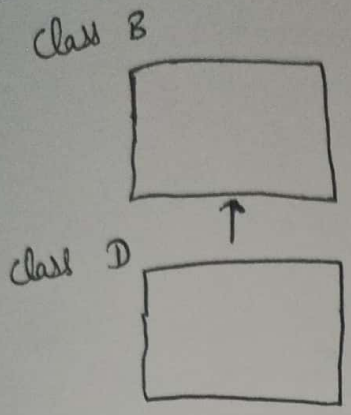
```
CIRCLE obj;
obj.CArea();
obj.RArea();
```

```
Class Lib
{
  public:
  float getSquare(float);
  {
    return (r*r);
  }
};
Class CIRCLE : public Lib
{
  float ar, r;
  public:
  void Carea()
  {
    Cout << " Enter radius:
    Cin >> r;
    ar = 3.14* getSquare(
    Cout << "\n Area = " << ar;
  }
};
int main()
{
  circle obj;
  Cobj.Carea();
```

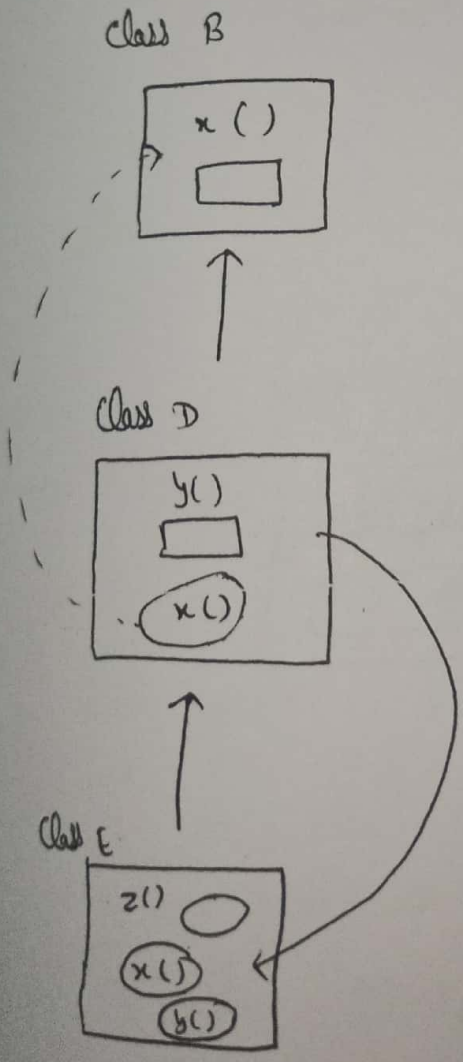
Reusability

* Types of Inheritance :-

① Single Level Inheritance :-

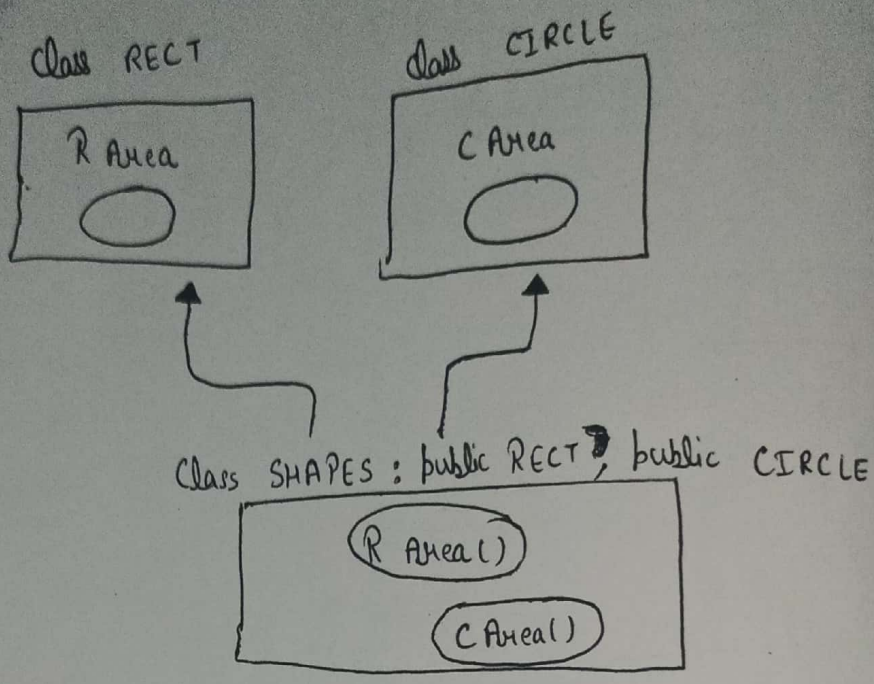


② multilevel Inheritance :-

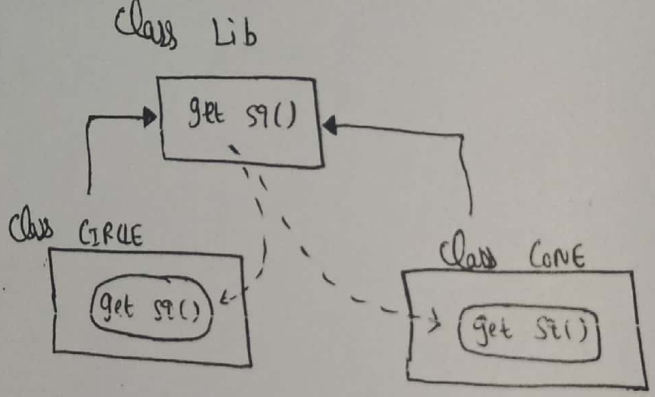


Here (B) is Direct base for (D) and is indirect base for (E).

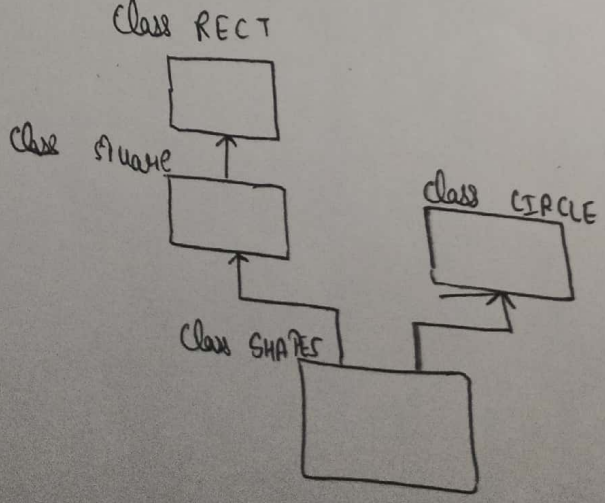
③ Multiple Inheritance :-



④ Heirarchial Inheritance :-



⑤ Hybrid Inheritance :-

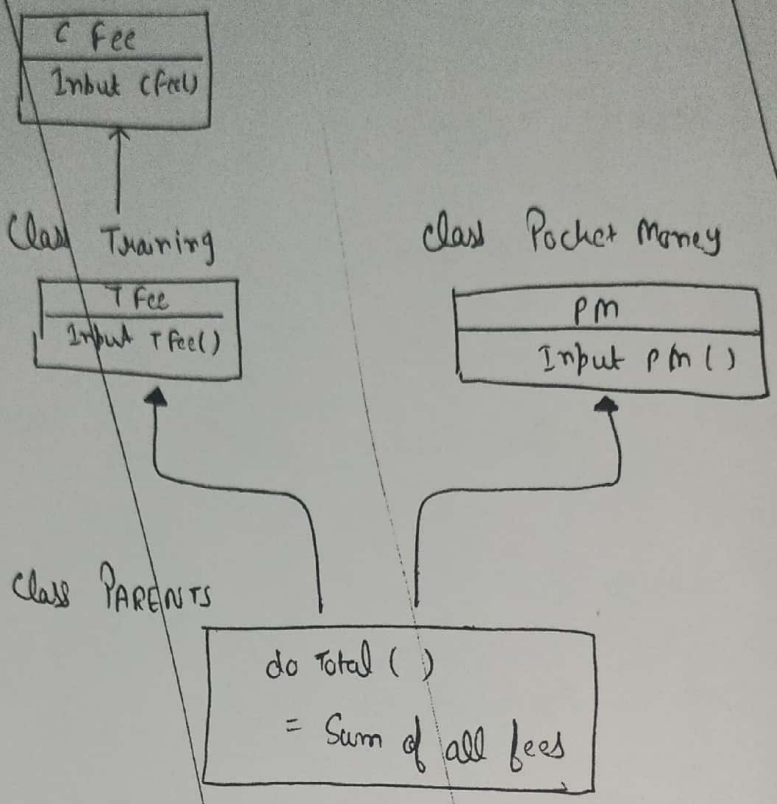


Inheritance

```
Class RECT
{ public :
  Void R area()
    {
      Cout << " Area of Rect In";
    }
};

Class SQuare : public RECT
{
  public :
    Void S area ( )
     {
       Cout << " Area of SquareIn";
     }
};

Class CIRCLE
   { public :
       Void C Area ( )
       {
         Cout << " Area of circle In";
       }
    };

Class SHAPES : public square , public CIRCLE
     { public :
       Void Call All ( )
          {
             R area();
             S area();
             C Area();
          }
       };
int main ()
     { SHAPES obj;
       obj. Call All();
     }
```

Home Work

Q- Class COLLEGE



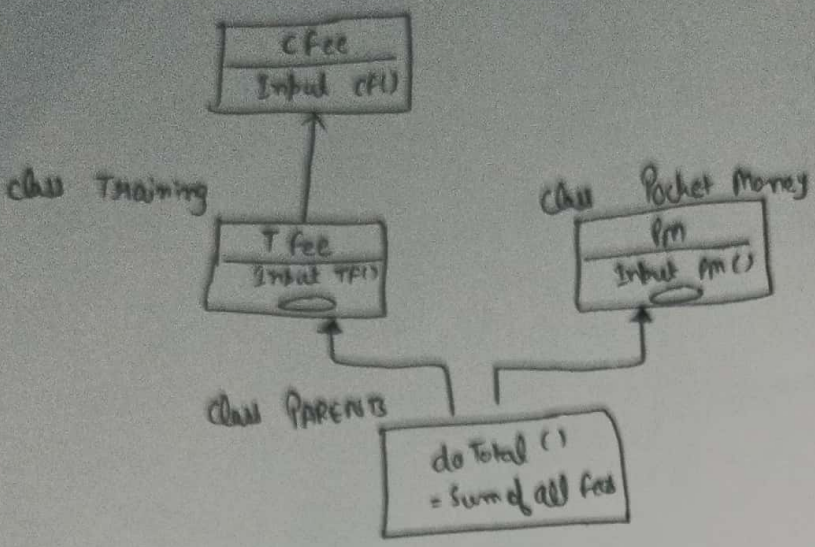
A -

```
Class COLLEGE
    {
      float CFee;
      public:
       void input ( )
          {
            Cout << " Enter fee : \n";
            Cin >> CFee;
          }
       friend class Training ;
    };
```

```
Class Training : public COLLEGE
    {
        float T fee;
        public:
        void Input1 ()
          {
            Cout << " Enter Training fee : \n";
            Cin >> T fee;
            Void Input ();
          }
    };
class PocketMoney
    {
        float PM;
        public
        void Input2 ()
          {
            Cout << " Enter pocket money : \n";
            Cin >> PM;
          }
    };
Class PARENTS : public Training, public PocketMony
    {
        float S;
        public:
        void doTotal ()
          {
```

Home Work

Q- class COLLEGE



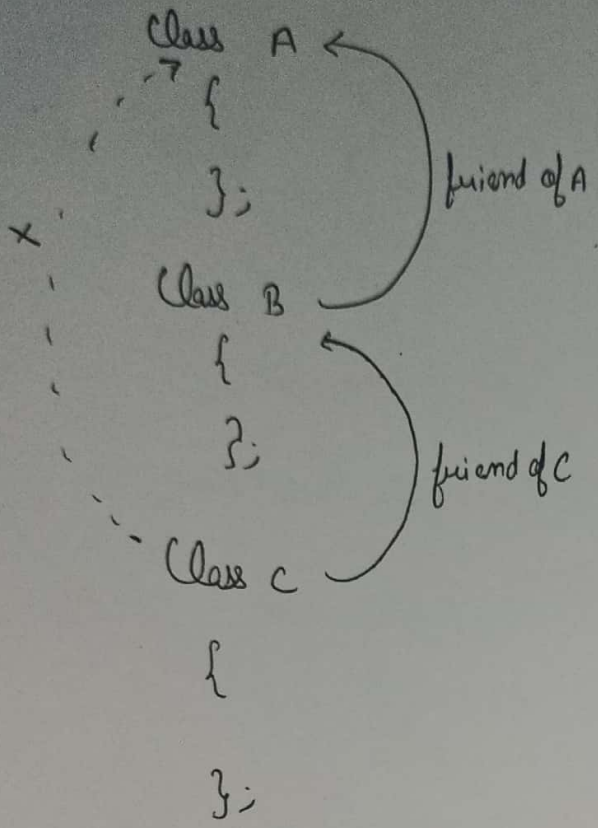
A - Class COLLEGE

```
{
    float CFee;
    public:
    float Input (float x)
        {
            CFee = x;
            return (CFee);
        }
    float ;
};

Class TRAINING : public COLLEGE
{
    float TFee, x;
    public;
float Input 1 ( )
        {
            Cout << " Enter Training fee : \n";
            Cin >> TFee;
```

```
Cout << "Enter College fee : \n";
Cin >> x;
float m = TFee + Input (x);
};      return m;
```

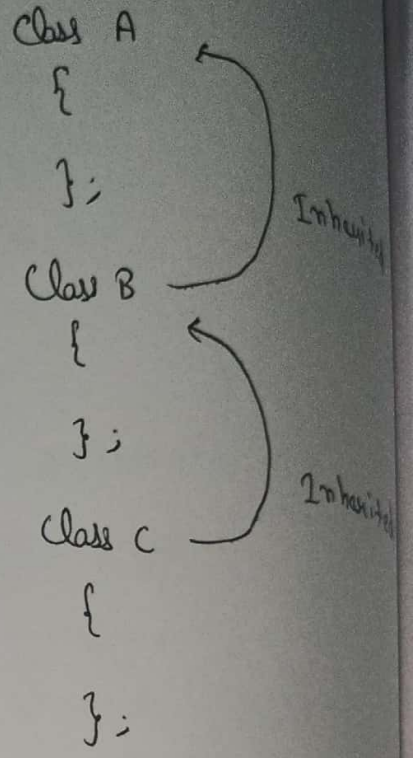
```
class Pocket Money
{
    float PM;
    public:
    float Input3 ( float y)
        { PM = y;
          return PM;
        }
};

Class PARENTS : public TRAINING, public Pocket Money
{
    float S, Y;
    public:
    void TF()
    {
        float P = Input2();
        Cout << " Enter pocket Money : \n";
        Cin >> y;
        float Q = Input3(y);
        float R = P + Q;
        Cout << "Sum of Total fees : " << R;
    }
int main()
{
    PARENTS X;
    X.TF();
}
```

FRIEND CLASS



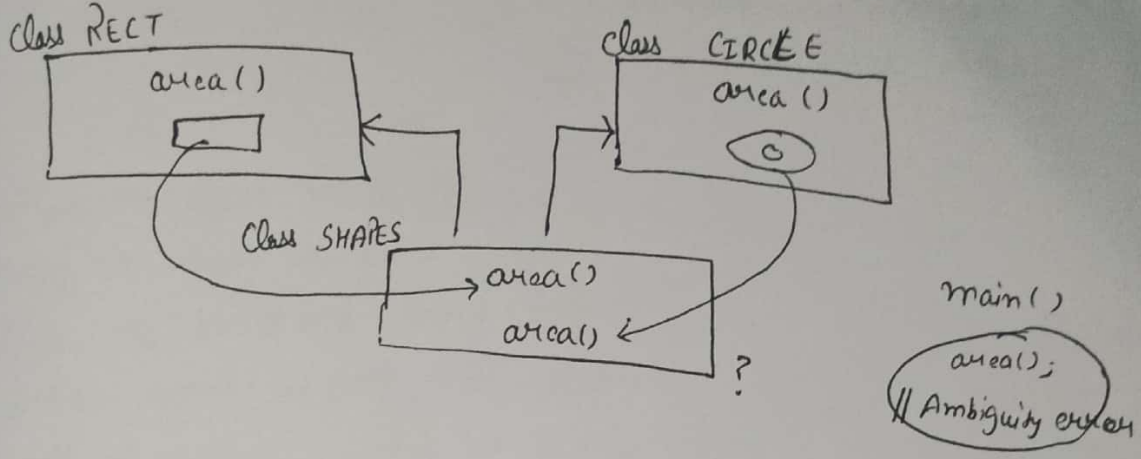
If Class B is friend of Class A than it will call methods of class A, if class C is friend of class B than it will call methods of class B, but During this, Class C is not friend of class A, so it will not call methods of class A, but if we want, than we have to make it as a friend, so, making friend of every and each class is lengthy and time Consuming, thats why we use inheritance.

INHERITANCE



If class B inherited class A than it will call methods of Class A and if class C inherited Class B, than it will call methods of class B, as well as class 'A', this is a difference w.r.t friend class

Ambiguity :- In case of multiple Inheritance, when multiple base classes contain methods having same name and prototype, than the derived class will also have multiple methods of same prototype, it leads to error of Ambiguity. as shown in program below.

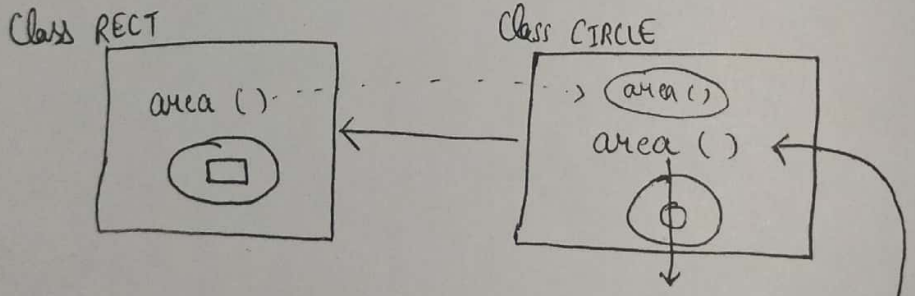


Eg :-
```
Class RECT
  {
    public :
      Void Area()
        {
          Cout << " Area of Rectangle \n";
        }
  };
Class CIRCLE
    {
      public :
      Void Area()
      {
        Cout << " Area of circle \n";
      }
  };
Class SHAPES : public RECT, public CIRCLE
    {

    };
```

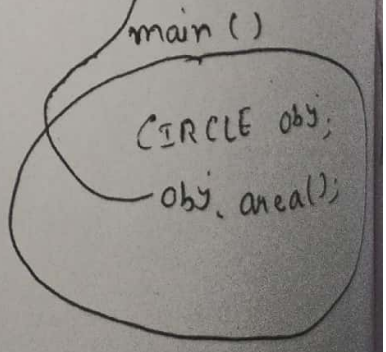
```
int main()
{
    SHAPES sobj;
    sobj.area();  // Error of Ambiguity
    // To solve the problem we use (::) Scope resolution operator.
    sobj.RECT :: area();
    sobj.CIRCLE :: area();
}
```

We use Scope resolution operator bcz when we work in future, we can face some problems on declaration of methods, bcz may be these method names already exist in Library class, so Here we will use Resolution operator.

★ **Method over-riding** :- when base class and derived class duo Contains methods having same name and Prototype then method over-riding takes place as shown below:



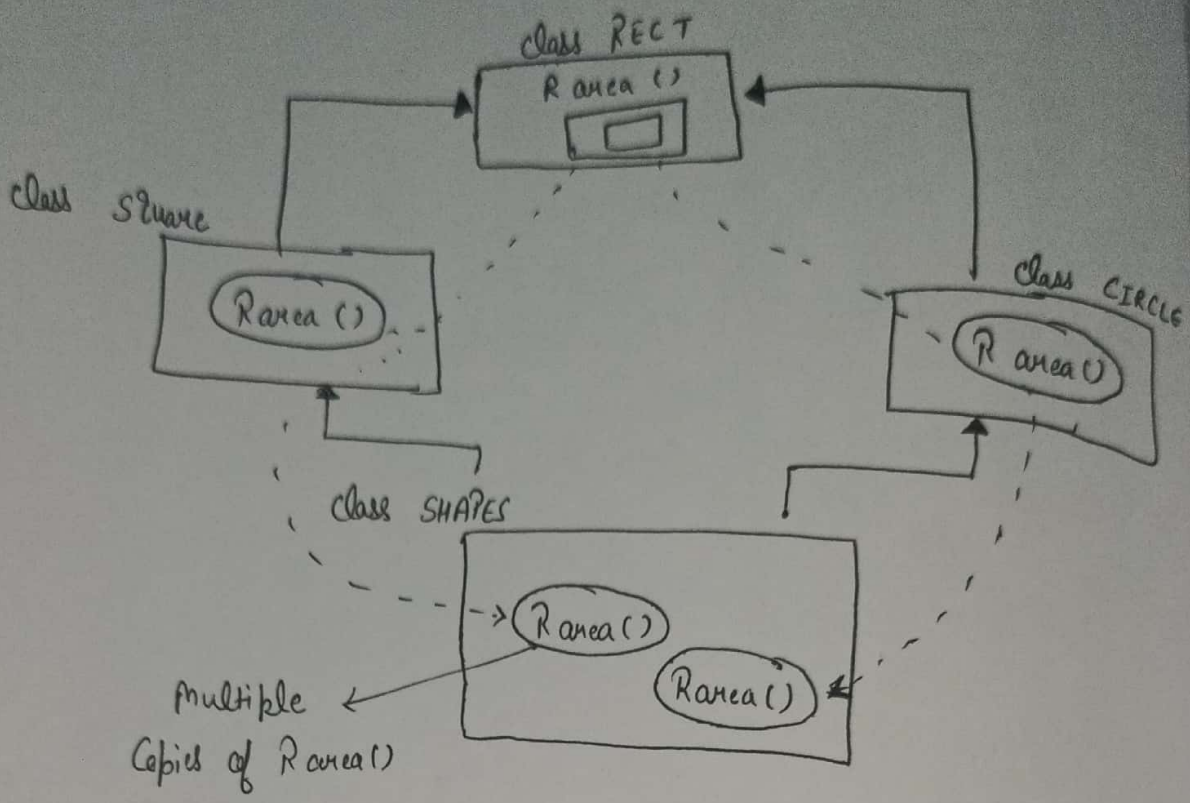
In this, Jis class ka object, usi ka method call hoga.



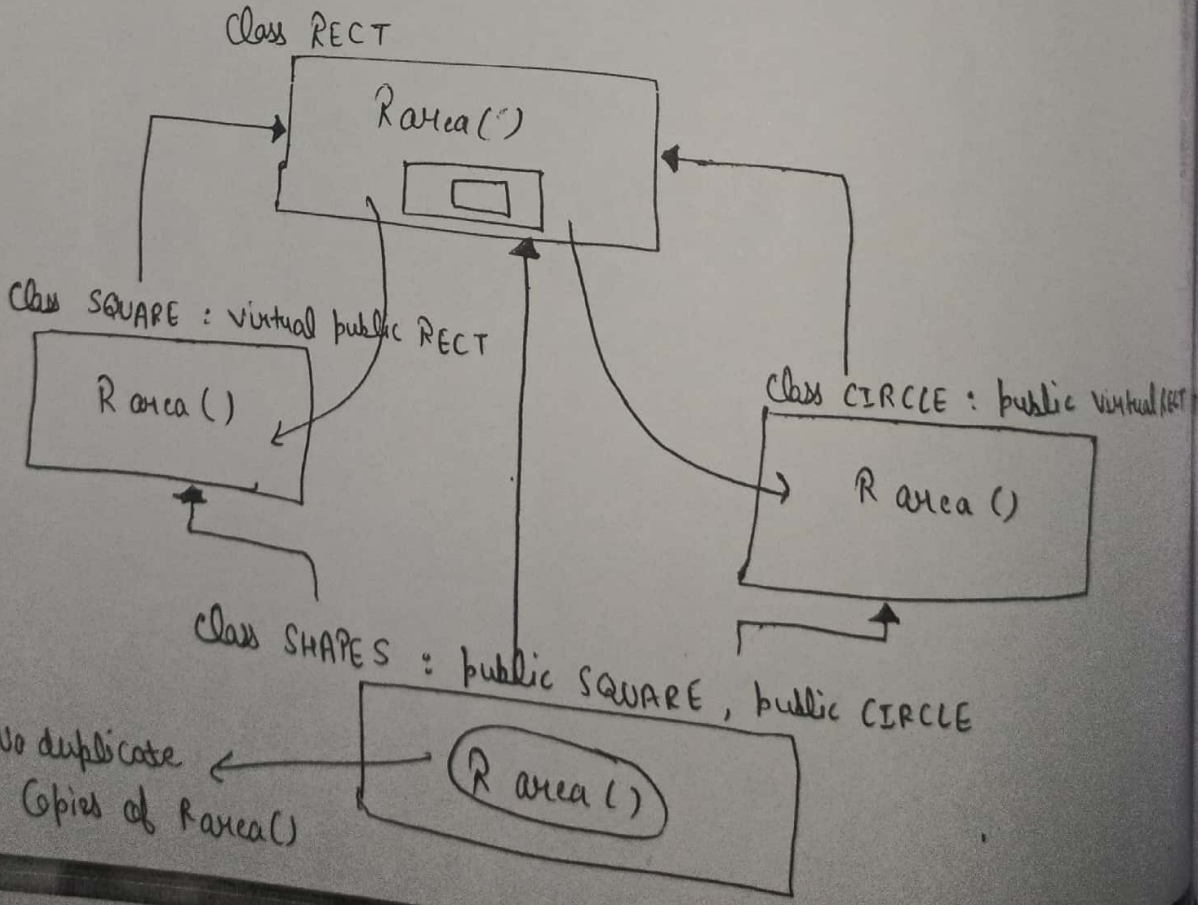
```
Class RECT
  {
    public:
      void Area()  // over Hidden method
        {
          Cout << " Area of Rectangle \n";
        }
  };
Class CIRCLE : public RECT
  {
    public:
    void Area ()  // over Hiding method
      {
        Cout << " Area of circle \n";
      }
  };
int main ()
  {
    CIRCLE Cobj ;
     Cobj. Area(); // area of circle
    Cobj RECT :: area(); // area of Rectangle
  }
```

Used to Call over Hidden method

* **Virtual Base Class** :- We can understand this concept with the following diagram:-



To avoid multiple copies of method R area (), we use the concept of virtual base class, as shown below:

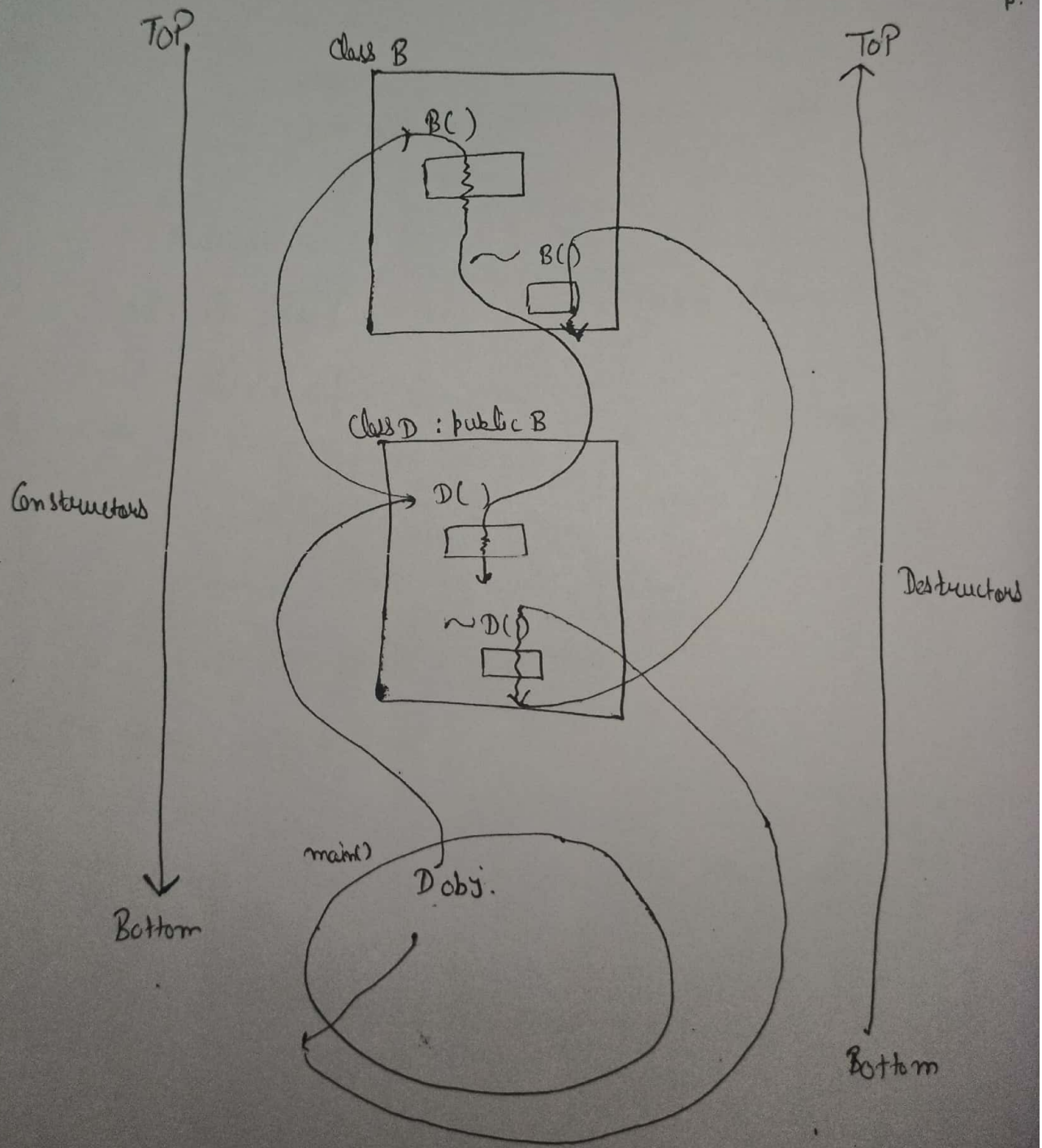


Eg :-

```
Class RECT
  {
    public :
      Void Rarea()
        {
          Cout << " Area of Rectangle \n";
        }
  };
Class SQUARE : public virtual RECT
      {
      };
Class CIRCLE : virtual public RECT
      {
      };
Class SHAPES : public SQUARE, public CIRCLE
          {
          };
int main()
  {
    SHAPES obj;
    obj. R area(); // No Ambiguity
  }
```

★ Constructors and Destructors With Inheritance :-

When the base class and Derived class duo Containing Constructors then the Constructor of base class executes before the Constructor of derived class, in other words Constructor Executes from top to bottom. On the other hand destructors execute from bottom to top.



① Treatement of Default Constructors and Destructors :-

```
Class B
{
  public:
   B() || ①
   {
    Cout << "In Default Constructor of Base";
   }
  ~B() || ④
   {
    Cout << " In Destructor of Base";
   }
};

Class D : public B
 {
  public:
   D() || ②
   {
    Cout << In Constructor of D :";
   }
  ~D() || ③
   {
    Cout << " In Destructor of D :";
   }
};
int main()
  {
    D obj;
  }
```

② Treatment of Parameterised Constructors :-

```
Class B
{
  public :
    B()                                  ——— (1)
    {
      Cout << " B default \n";
    }
    B(int i)                             ——— (2)   // Parameterized Constructor
    {
      Cout << I = " << i << "\n";
    }
};

Class D : public B
  {
    public :
      D()                                ——— (3)
      {
        Cout << " D Constructor \n";
      }
      D( float y, int s) : B (s)         ——— (4)   Initialisation List
      {
        Cout << "\n y = " << y;                    Parameterized Constant
      }
};
int main()
  {
    D obj ;  // 1, 3
    D obj2 (7.8, 6) ;  // 2-4
  }
```

★ Member Access Specifiers :-

① Public ② Private ③ Protected

① **Public** :- Public members of a class are Accessible Everywhere (verble members). i.e. Member functions, friend function, non-member function, Derived class, Non-Derived class (Container class).

② **Private** :- These are Accessible in member functions, friend functions and friend class only. not evan accessible in derived class.

Eg :- friend function / class.

③ **Protected** :- protected members (variables / functions) are Accessible in friend function, friend class, member functions and also accessible in derived class.

Eg :-
```
class RECT
{
  protected :
    int L, B;
};
class PROCESS : public RECT
{ int ar;
  public :
  void Area ()
  {
    Cout << " Enter Length & Breadth : \n";
    Cin >> L >> B;
    ar = L * B;
    Cout << ar;
  }
main ()
{
process obj;
obj. area ();
}
```

Home Work

Q- Program of fees on protected Access specifier.

A-

```
Class COLLEGE
  {
    protected :
     float Cfee;
    public :
     Void Input1( )
     {
       Cout << " Enter College fees : \n"
       Cin >> Cfee;
     }
  };
Class TRAINING Fee : public COLLEGE
   {
     protected :
     float TFee, Total;
      public :
      void Input 2 ( )
       {   input 1( );
          Cout << " Enter Training fees : \n";
          Cin >> TFee;
        }
   };
Class Pocket Money
   {
     protected
      float PM;
      public :
        void Input 3 ( )
          {
            Cout << " Enter pocket money : \n";
            Cin >> PM;
          }
     };
```

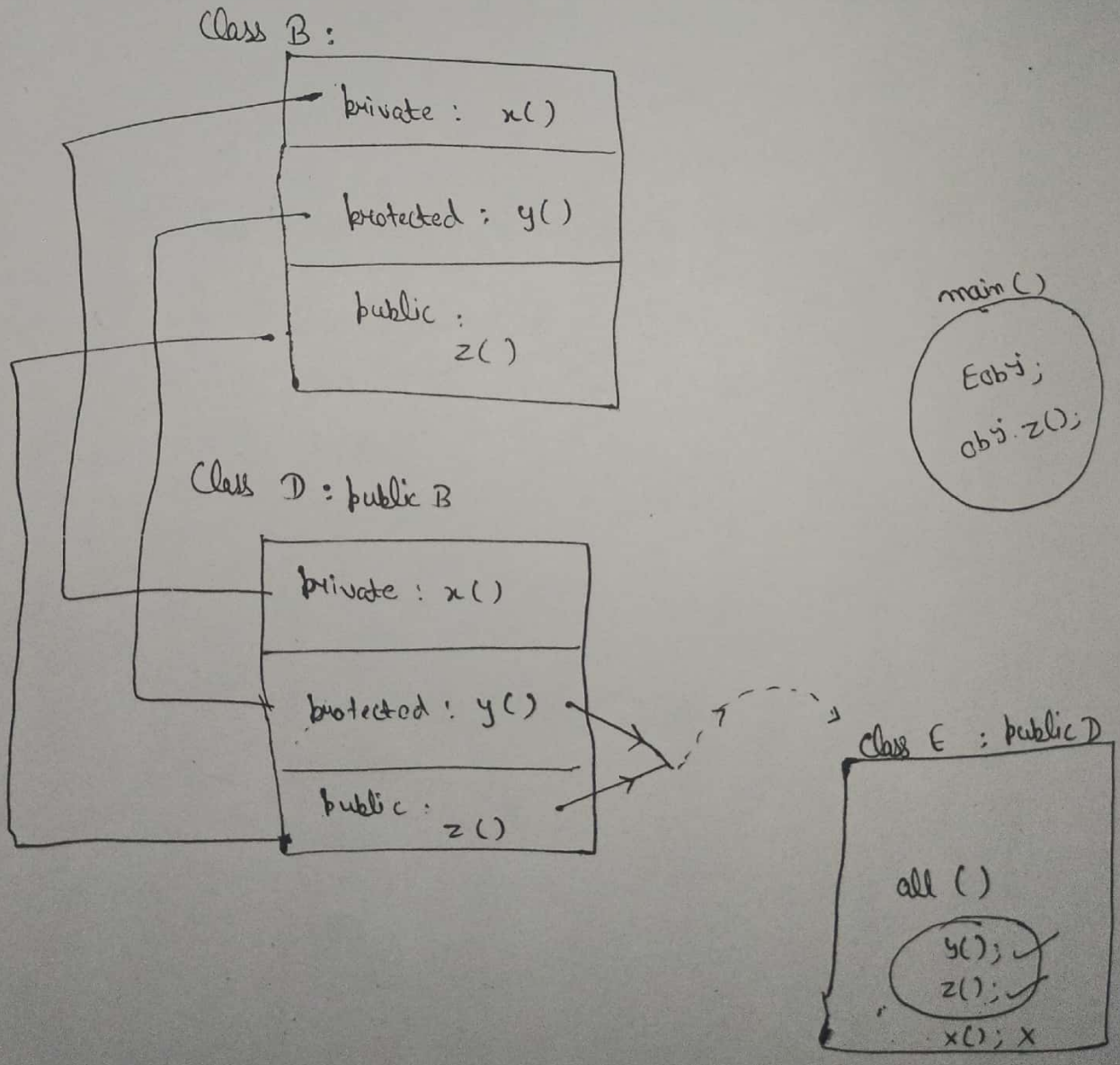
```
void process ( )
   { Total = TFee + Cfee;
   }
```

```
class PARENTS : public TRAINING fee, public Pocket money
{
    protected :
      float U;
    public :
      void doTotal ()
      { input 2();
        input 3();
        [redacted] U = cFee + TFee + Pm;
        Cout << " Sum of Total fees = " << U;
      }
};
int main()
{
    PARENTS X;
      X. doTotal ();
}
```

Types of Derivations | Visibility Control :-

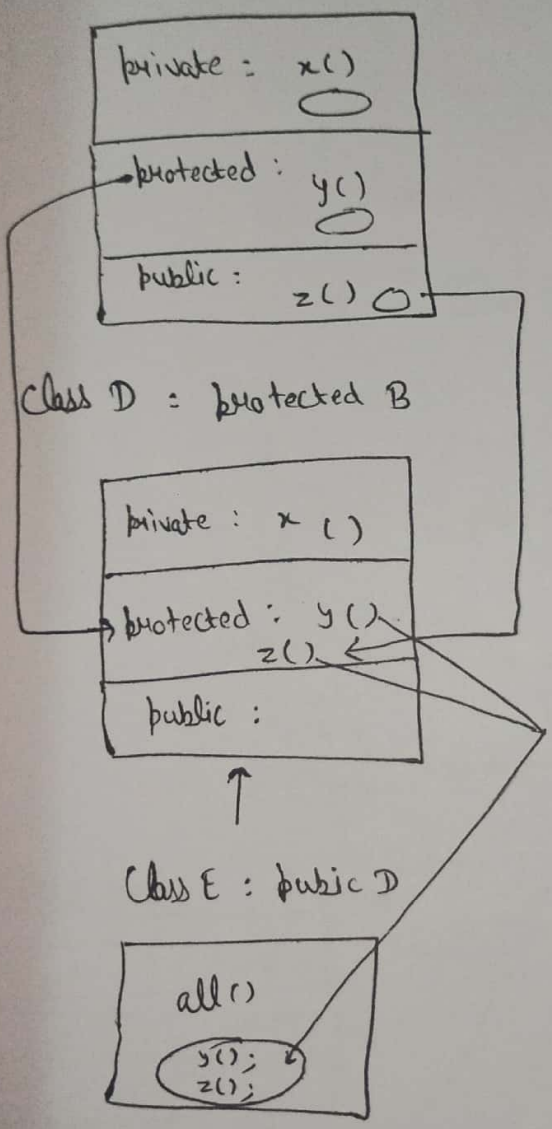
① : Public ② : Protected ③ : Private

① Public Derivation :- In case of public Derivation, public, protected and private members of base class remains public, protected and private respectively in derived class. But private members can be inherited but not be accessed in further derived class.



② Protected Derivation :-

Here public members of base class becomes protected in derived class and protected remains protected and private remains private



③ Private Derivation :

In case of private Derivation public and protected members of base class becomes private in derived class, and the y can't be Inherited ahead

Eg :-

```
class B
{ public :
  void x()
   { cout << "x";
   }
  protected :
   void y()
   { cout << "y";
   }
};
class D : private B
  {
  };
class E : public D
{ public :
   void all()
   { x();
     y();
   }
};
int main()
   { E obj;
     obj.all();
   }
```

ERROR

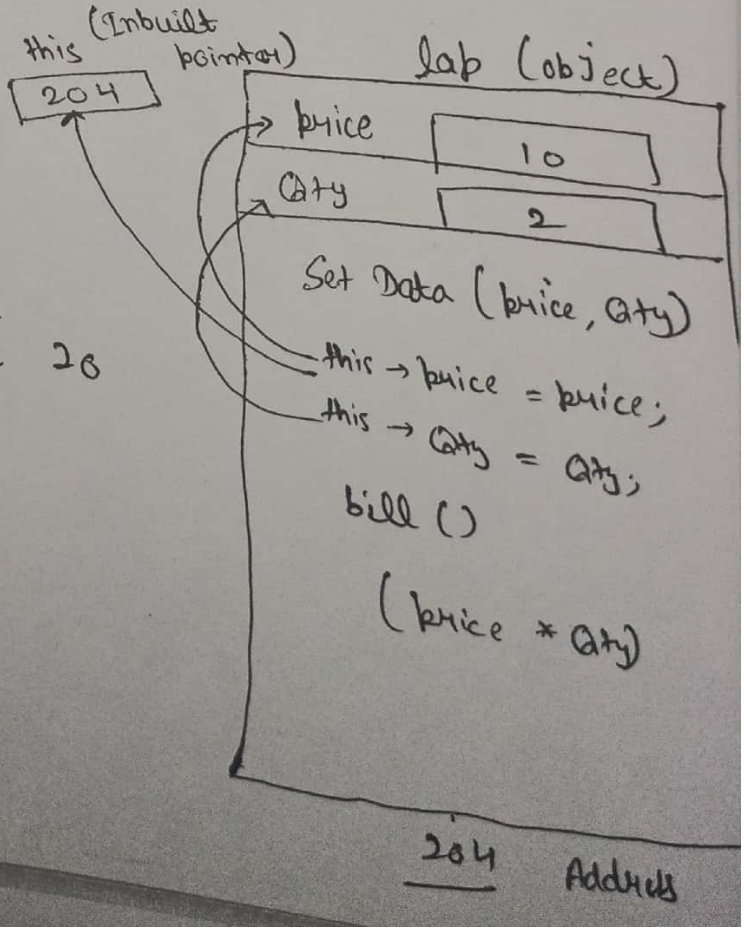
"This" Pointer :-

This is an inbuilt pointer which stores the address of current invoking object in memory.

Generally it is used to differentiate Instance variables, when they are having same names.

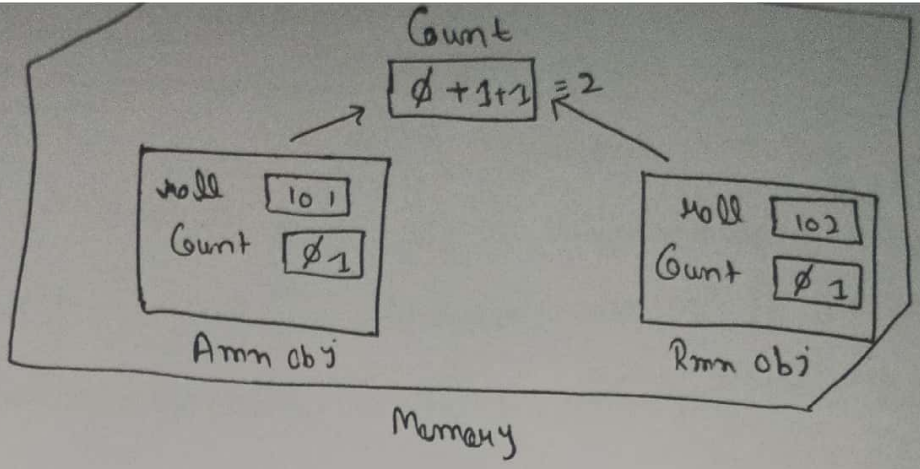
"This" Pointer can be used within member functions only.

```
Class PRODUCT
  {
    int price, Qty;
    public:
      Void SetData (int price, int Qty)
        {
IV <-- this -> price = price;  --> LV
IV <-- this -> Qty   = Qty;    --> LV
        }
      Void bill ()
        {
          Cout << " In Bill = " << price * Qty;
        }
  };

int main()
  {
    PRODUCT lab;
An invoking object --> lab. SetData (10,2);
    lab. bill (); // garbage value 20
  }
```



Static Members :- Static Variables and Methods

```
class STUDENTS
{
    int roll ; // Instance variable
    static int Count ; =0 ; // IV // class variable
    public :
    STUDENTS (int rol) // Parameterized Constructor
        {
          roll = rol ;
            Count ++ ;
        }
    Void show ()    // Instance Method
      {
        Cout << roll << " Count = " << Count << endl ;
      }
    Static Void showCount ()  // Static Method
        {
          Cout << "In Total Students = " << Count ;
        }
};
int STUDENTS :: Count = 0 ; // Creation of Static variable
                            // it will take value '0'
                               by default.
int main ()
{
  STUDENTS Amn (101) ;
      Amn . show () ; // 101 - 1
  STUDENTS Rmn (102) ;
        Rmn . show() ; // 102 - (1) (2)
  STUDENTS :: showCount () ; // Count = 2
}
```



⇒ Static variables :-

- Static variables are known as class variables.
- By Default it takes 0 value.
- It belongs to class instead of instances object.
- Its creation takes place outside the class using scope resolution operator (::)
- It can be used by duo instance methods and static methods.

⇒ Static Methods :-

- Static method is called using class Name with (::) operator.
- It belongs to class.
- But It can use only static variables.

★ **Operator overloading** :- We know that operators available in C++ cannot work with objects, to make it possible we need to overload the operator by re-defining its behaviour in class.

```
Class BANK
{
    int SB, CB;
    public:
    Void SetBal (int S, int C)
    {
        SB = S;
        CB = C;
    }
    Void show ()
    {
        Cout << SB << " " << CB << endl;
    }
Void operator ++ (int) // overloading postfix unary operator
    {
        SB++;
        CB++;
    }
Void operator -- () // overloading prefix Decrement operator
    {
        SB--;
        CB--;
    }
Void operator += (BANK Hobj) // overloading Short hand Assignment operator
    {
        SB = SB + Hobj.SB;
        CB = CB + Hobj.CB;
    }
```

```
BANK operator + (BANK aoby)   // overloading binary operator
  {
     BANK Bhai;
     Bhai.SB = SB + aoby.SB;
     Bhai.CB = CB + aoby.CB;
     return (Bhai);
  }
};

int main()
  {
     BANK Amnoby, Rmnoby;
     Amnoby.SetBal(1,2);
     Amnoby++;
     Amnoby.show(); // 2-3
     --Amnoby;
     Amnoby.show(); // 1-2
     Rmnoby.SetBal(2,3);
     Amnoby += Rmnoby;
     Amnoby.show(); // 3-5
     BANK chmnoby = Rmnoby + Amnoby;
     chmnoby.show(); // 5-8
  }
```

Operators which cannot be overloaded

1. Size of
2. :: → Scope of Resolution
3. . → Member Access specifier
4. .* → Pointer to Member.
5. ?: → Conditional

Home work

Q- write a Program for difference b/w Heights of 2 Persons using operator overloading.

A-

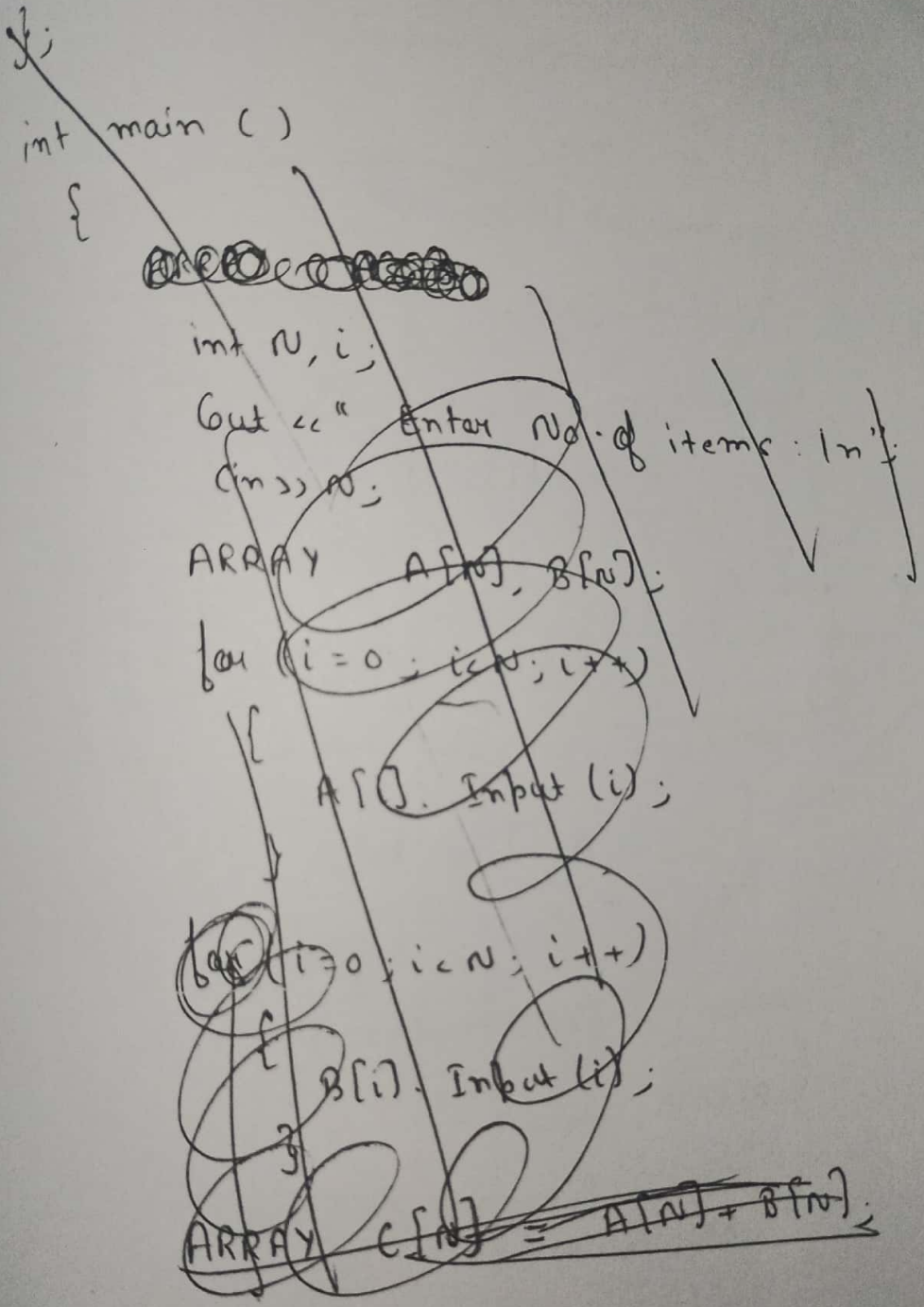
```
Class HEIGHTS
{
    int F;    int H;
    int I;

    public :

    Void Set Height (int P, int Q)
            {
                F = P;
                I = Q;
            }

    Void Process ( )
            {
                H = F * 12 + I;
            }

    HEIGHTS operator - ( HEIGHTS A)
     {
        HEIGHTS Z;
        Z.H = abs(H - A.H);
            Return (Z);
     }

    Void show ( )
     {
        F = H / 12;
        I = H % 12;
        Cout << " Difference is : " << F << "feet" << I << "Inch";
     }

};
```

```
int main()
{
    HEIGHTS Hardik, Prince;
    Prince.SetHeight(5,4);
    Hardik.SetHeight(4,1);
    Prince.process();
    Hardik.process();
    HEIGHTS SAKSHAM = Hardik - Prince;
    SAKSHAM.show();
}
```

Q- Write a program for operator overloading using Two 1-D Arrays.

A-

```
class ARRAY
{
    int U;
    public:
    void Input(int i)
    {
        cout << " Enter price of " << i+1 << " item ";
        cin >> U;
    }
ARRAY operator + ( [redacted] B[7])
{
```

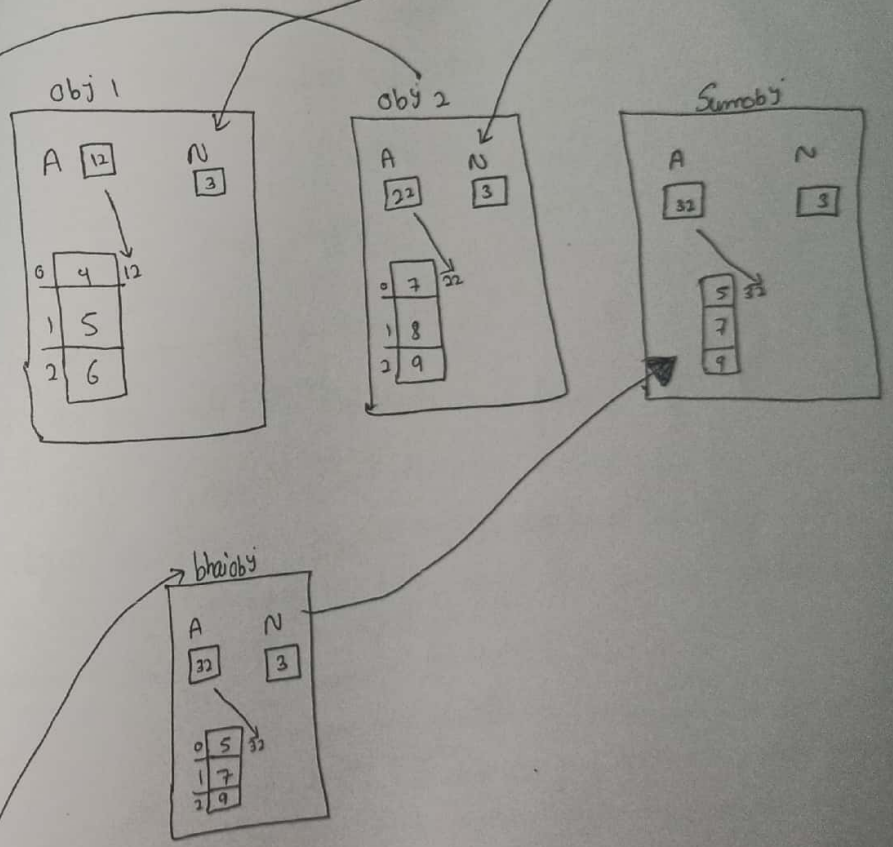


Q- Write a program for array within object using operator overloading

A-

```
class ARRAYS
{
    int *A, N, i;
    public:
    void InputArray (int N1)
    {
        N = N1;
        A = new int [N];
        for (i=0; i<N; i++)
        {
            Cout << " Enter Value : \n";
            Cin >> A[i];
        }
    }
void show()
{
    for (i=0; i<N; i++)
    {
        Cout << A[i] << "\n";
    }

ARRAYS operator + (ARRAYS obj2)
{
    ARRAYS bhaiobj;
    bhaiobj.A = new int [N];
    bhai.N = N;
    for (i=0; i<N; i++)
    {
        bhaiobj.A[i] = A[i] + obj2.A[i];
    }
    return (bhaiobj);
}
};
```

```
int main()
{
    ARRAYS obj1, obj2, Sumobj;
    obj1.InputArray(3);
    obj2.InputArray(3);
    Sumobj = obj1 + obj2;
    Sumobj.show();
}
```



☆ Overloading Relational Operators :-

int x = 12 < 3;

x0

```
Class HEIGHT
{
    int F, I;
    public :
    void Input()
    {
        Cout << " Enter feet and Inch : \n";
        Cin >> F >> I;
    }
    int Operator == (Height aobj)
        {
            if (F == aobj.F && I == aobj.I)
                Return (1);
            else
                Return (0);
        }
    int Operator < (Height aobj)
        {
            if (F*12 + I < aobj.F*12 + aBbj.I)
                Return (1);
            else
                Return (0);
        }
};
```

```
int main ()
{
    HEIGHT amnobj, Hmnobj;
    amnobj. Input();
    Hmnobj. Input();
    if (amn obj == Hmn obj)
        Cout << " Same Height";
    else
        Cout << " Not Same";
    if (Hmnobj < amnobj)
        Cout << " Hmn is shorter";
    else
        Cout << "In amn is shorter";
}
```

Q- Write a note on operator overloading.

A- In C++, we can make operators to work for user defined classes. This means C++ has the ability to provide the operators with a special meaning for data type, this ability is known as Operator overloading.

For Eg :- We can overload an operator '+' in a class like string so that we can concatenate two string by just using + . other Examples classes where arithmetic operators may be overloaded are Complex Number, fractional Number, Big integer, etc.

Polymorphism :- Calling overriding method of derived class using pointer of base class is called Polymorphism.

(many forms of functions)

Binding

(Linking) of one function with another function

Static Binding
or
Compile time Binding
or
Early Binding

It takes place in function overloading and operator overloading.

function overloading.

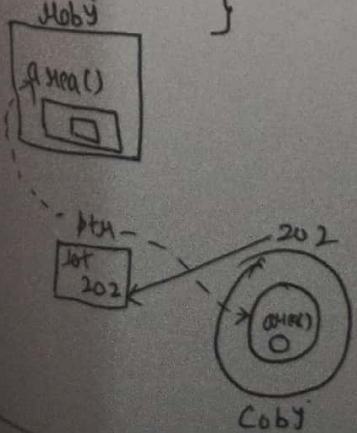
```
Void Area (int L, int B)
    {
      Cout << L*B;
    }
Void Area (float H)
    {
      Cout << 3.14*H*H;
    }
int main ()
  {
     Area (5,2);
     Area (7.0);
     Area (5,8);
  }
```

Dynamic Binding
or
Runtime Binding
or
Late Binding

Required / Needed things to do this

- # classes
- # inheritance
- # function over-riding
- # pointer to base class
- # virtual function.
  - → of base class

```
class RECT
{
  public :
virtual  void Area ()
  {
    Cout << "Area of Rect\n";
  }
};

class CIRCLE : public RECT
  {
    public :
    Void Area () // over-riding method
     {
       Cout <<"\n Area of circle";
     }
  };
int main ()
  {
    RECT Robj, *ptr;   → pointer to base class
       ptr = & Robj;
       ptr → Area(); // Area of rect
    CIRCLE Cobj;
       ptr = & Cobj;
       ptr → area(); // Area of ~~circle~~ ~~Rect~~ circle
  }
```



★ Pure virtual function | Abstract class | Abstract base class

A virtual function without body is known as pure virtual function.

Virtual Void Area () = 0 ;

A class Containing pure virtual function is known as Abstract class.

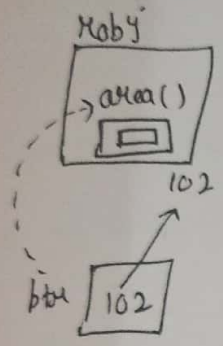
An abstract class cannot be instantiated (creation of objects).

It can be used for inheritance only.

Now the derived class is bound to override the pure virtual function of base class, otherwise it will also become abstract.

```
Class SHAPE // Abstract class
  {
   public:
   virtual void Area () = 0 ; // Pure virtual function
  };

Class RECT : public SHAPE
    {
      public:
      Void Area () // overriding
       {
         Cout << "In Area of Rect : In";
       }
    };
```

```
int main()
{
    SHAPE obj;
    SHAPE *ptr;
    RECT  Roby; || valid
          ptr = & Roby;
          ptr -> area(); || Area of Rect
}
```



Q- Definition of Polymorphism.

A- Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

As we know Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

Note on virtual function.

A- A virtual function is a member function which is declared within a base class and is re-defined (over-ridden) by a derived class. When you prefer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and Execute the derived class's version of the function.

- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function Call.
- They are mainly used to Acheive Runtime Polymorphism.
- Functions are declared with a virtual keyword in base class.

Rules for virtual functions

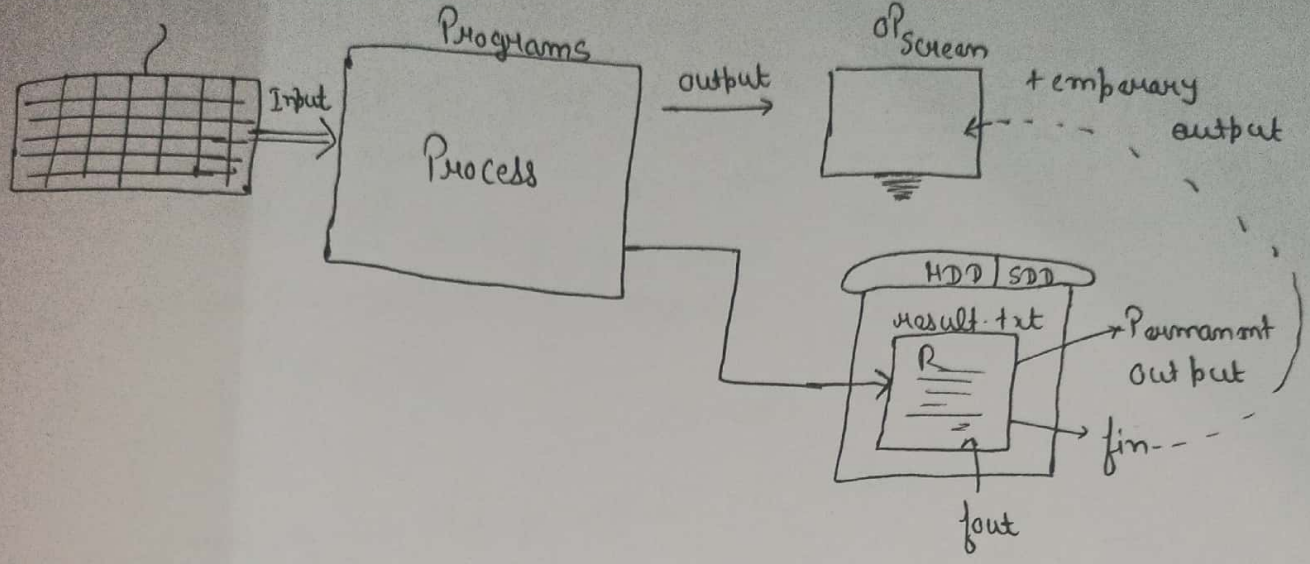
1.) Virtual functions cannot be static and also cannot be a friend function of another class.

2.) Virtual functions should be Accessed using pointer or reference of base class type to acheive run time Polymorphism.

3.) The Prototype of virtual functions should be same in base as well as Derived class.

4.) A class may have virtual Destructor but it cannot have a virtual Constructor.

★ File - Handling :-



Cout (<<) → operator overloading function

└→ an object of class OSTREAM

Cin >>

└→ An object of class ISTREAM

```
# include <fstream>
int main()
{
    int count = 0;
    ofstream fout;
    fout.open("result.txt", ios::out);   → Write mode
    fout << "Real In";
    fout << "Java\n";                    → Writting in file
    fout.close();                        → closing file
```

// — — — — — — —

```
ifstream fin;
 fin.open ("result.txt", ios :: in);   // read mode
  while (! fin.eof ()) // It checks end of file
  {
    char ch = fin.get (); // Reading [redacted]
                          // get() -> used to read a character from file
      Cout << ch;
      Count ++;          // Writting output on Screen
  }
  fin.close ();
 Cout << "\n length of file = " << Count;
}
```

// File - Copy Program

```
# include <fstream>
 Class File Copy
  {
    ifstream fin;
    ofstream fout;
    char src [20], des [20], ch;
    public :
     Void do Copy ()
     {
       Cout << " Enter file name to read :";
       Cin>> src;
        fin.open ( src, ios :: in); // opened for reading
         if (fin. fail ())
          { Cout << " file not found";
              return;
          }
```

```
Cout << " Enter file name to write :";
  Cin >> dat ;
    ↓
fout . open (dat , ios :: out);
  while ( ! fin. eof())
  {
     ch = fin. get(); // read a
     fout << ch;          character
     Cout << ch;
  }
}
```

→ write mode

ios:: app (It retains the existing contents and append new over retaining)

```
~ File Coby ()  // Destructor
  {
    fin. close ();
    fout . close ();
  };
```

→ realeasing the resources.

```
int main ()
{  File Coby obj ;
     obj. do Coby ();
}
```

Destruction Called