stream = sequence / chain of bytes.

console / character output. ⇐ Cout is an object which is used for the purpose of output
<< = put-to operator / Extraction operator

ch. input ⇐ Cin =
>> = get from

C ++

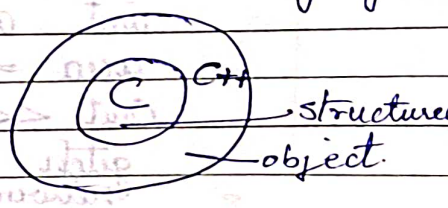
Q. WAP to find sum of two nos.

```
#include <iostream.h>        { input-output stream }
#include <conio.h>

using namespace std;
int main()
{
    int a, b;
    Cout << "Enter any 2 nos";
    Cin >>a;        } or  Cin >> a >> b;
    Cin >>b;
    int c;
    c = a+b;
    Cout << "\n Sum of 2 nos = " << c;
    getch();
    return 0;
}
```

Sum of " << a << "and " << b << " = " << c;

⇒ C++ was developed in 1980s (1982), by Mr. Bjarne Straus - ~~stupb~~ stroup.
ANSI = American National standard Institute.
(ANSI C++). (ATNT Bell Lab)

C = Structural / Procedural ~~Oriented~~ Language
C++ = Superset of C
Extension of C



HW Make any 10 programs of C in C++

1. PRODUCT OF TWO NOS :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

int main ()
{
    int a, b;
    Cout << " Enter any two nos ";
    cin >> a;
    Cin >> b;
    int c;
    c = a * b;
    Cout << " Product of two nos = " << c;
    getch ();
    return 0;
}
```

2. AREA OF CIRCLE :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;
int main ()
{
    int r;
    Cout << " Enter the radius of circle ";
    Cin >> r;
    int area;
    area = 3.14 * r * r;
    Cout << " Area of circle = " << area;
    getch ();
    return 0;
}
```

3. FIND SIMPLE INTEREST :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

int main ()
{
    int p, r, t, si;
    Cout << " Enter principal, rate & time ";
    Cin >> p >> r >> t;
    si = (p * r * t) / 100;
    Cout << " Simple interest = " << si;
    getch ();
    return 0;
}
```

4. PRODUCT OF THREE NOS :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

int main ()
{
    int a, b, c, p;
    Cout << " Enter any 3 nos ";
    Cin >> a >> b >> c;
    p = a * b * c;
    Cout << " Product of 3 nos " << p;
    getch ();
    return 0;
}
```

5. AREA OF RECTANGLE :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

int main()
{
    int l, b, ar;
    Cout << "Enter the dimensions ";
    Cin >> l >> b;
    ar = l * b;
    Cout << "Area of rectangle " << ar;
    getch();
    return 0;
}
```

6. AREA OF TRIANGLE :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

int main()
{
    int b, h, ar;
    Cout << "Enter the dimensions of triangle";
    Cin >> b >> h;
    ar = 0.5 * b * h;
    Cout << "Area of triangle " << ar;
    getch();
    return 0;
}
```

7. CONVERT °C → °F :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

int main()
{
    auto int t, f;
    Cout << "Enter the temp. in celcius ";
    Cin >> t;
    f = (t * 9/5) + 32;
    Cout << "Temp. in fahrenheit = " << f;
    getch();
    return 0;
}
```

8. SQUARE OF NO. :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

int main()
{
auto int a, ar sq;
    Cout << "Enter any no ";
    Cin >> a;
    sq = a * a;
    Cout << "Square of no=" << sq;
    getch();
    return 0;
}
```

9. AVERAGE OF 2 nos :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

int main ()
{
    int a, b, c;
    Cout << " Enter any 2 nos ";
    Cin >> a >> b;
    c = (a+b)/2;
    Cout << " Average of 2 nos=" << c;
    getch ();
    return 0;
}
```

10. PERCENTAGE OF A STUDENT :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

int main ()
{
    int a, b, c, d, e, p;
    Cout << " Enter marks in 5 subjects ";
    Cin >> a >> b >> c >> d >> e;
    p = (a+b+c+d+e)/5;
    Cout << " Percentage of student = " << p;
    getch ();
    return 0;
}
```

Class :- Extended form of structure. (Contains access specifier)
Structure - Does not have access specifier.

* CLASS :-

Class is a collection of member - variables and set of member - functions which operate over those variables.
Whenever we write any class, then that class itself become a new user - defined datatype and as WKT, to use any datatype we need to declare variable of that datatype. In same way, to use any class, we need to declare variable of that class. The variable of class is known as object.

∴ We can declare any no. of variables of any particular datatype. Similarly, we can declare any no. of objects of any particular class. That is why, we can say that a class is a collection of same types of objects.

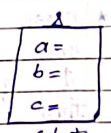
Memory for an object is allocated at run-time. Hence an object is also called run-time entity.
Memory for an object will remain only during execution of any program. That is why an object is also called instance of class.

Q. W. an object - oriented programme to find sum of 2 nos.

* Private members are accessible only inside class
* Public members are accessible everywhere.

```
#include <iostream.h>
#include <conio.h>
using namespace std;

class Sum
{  private :
      int a, b, c ;
   public :
   void input ()
   {
      Cout << " Enter any two nos ";
      Cin >> a >> b;
   }
   void add ()
   {
      c = a+b;
   }
   void output ()
   {
      cout << "\n Sum of 2 nos. = " << c;
   }
};
int main ()
{
   Sum s;
   s. input ();
   s. add ();
   s. output ();
   getch ();
   return 0;
}
```

access specifier / modifier / visibility mode



6 byte.

⇒ Two important concepts of object - oriented prog :-
(i) One (class) person should perform one task.
(ii) Don't interfere in other's work.

Q. Area of circle
Q. Simple Interest
Q. Greater b/w 2 nos.
Q. Table of any no
Q. Factorial of any no.

① AREA OF CIRCLE :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

class Area
{  private :
      int r ; float ar ;
   public :
   void input ()
   {
      Cout << " Enter the radius of circle";
      Cin >> r;
   }
   void mul ()
   {
      ar = 3.14 * r * r;
   }
   void output ()
   {  Cout << "\n Area of circle = " << ar;
   } };
```

```
int main ()
{
    Area a;
    a. input ();
    a. mul ();
    a. output ();
    getch ();
    return 0;
}
```

2 SIMPLE INTEREST :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

class Interest
{   private :
        int p, r, t;
        float si;
    public :

    void input ()
    {   Cout << " Enter principal, rate & time ";
        Cin >> p >> r >> t;
    }
    void mul ()
    {
        si = (p*r*t)/100;
    }
    void output ()
    {   Cout << "\n Simple Interest = " << si;
    } };
```

```
int main ()
{
    Interest i;
    i. input ();
    i. mul ();
    i. output ();
    getch ();
    return 0;
}
```

3 GREATER No. B/w 2 Nos :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

class Greater
{   private :
    ~~four~~ int a, b; e
    public :

    void input ()
    {   Cout << " Enter any 2 nos ";
        Cin >> a >> b;
    }
    void ~~Diff~~ output ()
    {   if (a > b)
            ~~return a;~~
        else
            ~~return b;~~
    }
};
```

Cout << " G N = " << a

Cout << " G N = " << b;

```
int main ()
{
    Greater g;
    g.input ();
    g.output ();
    getch ();
    return 0;
}
```

4. Table of any no :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

class Table
{   private: i, n,
    int i, n;
    public:

    void input ()
    {   Cout << "Enter any number";
        Cin >> n;
    }
    void output()
    {
        for (i = n; i <= n * 10; i = i + n)
        {
            Cout << "Table of no = \n" << i;
        }
    }
};
```

```
int main ()
{
    Table t;
    t.input ();
    t.output ();
    getch ();
    return 0;
}
```

5. FACTORIAL OF No. :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

class Factorial
{   private:
    int f=1, n;
    public:

    void input ()
    {   Cout << "Enter any number";
        Cin >> n;
    }
    void fact ()
    {   int i; } f=1; }
    for (i = 1; i < n; i++)
    {
        f = f * i;
        Cout << "Factorial = " << f;
    }
};
```

To reduce complexity & ↑ efficiency :-
for (i=2; i<n; i++)
f = f * i;

```
    void output ()
    {
        cout << " Factorial = " << f ;
    }
};

int main ()
{
    Factorial ft ;
    ft. input ();
    ft. fact ();
    ft. output ();
    getch ();
    return 0;
}
```

Q. #
#
```
using namespace std ;

class greater
{   private :
    int a, b, c, d ;
    public :
    void input ()
    {   cout << "\n Enter any 2 nos " ;
        cin >> a >> b ;
    }
    void great ()
    {   if (a > b)
        cout << a << " is greater than " << b ;
```

```
int great ().
{ if (a>b)
    return a;
  else
    return b;
}
```

```
void output ()
{ cout << "\n Greater no = " << great ();
}
g. input ();
g. output ();
```

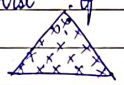
```
    else
    cout << b << " is greater than " << a ;
    }
};

int main ()
{
    greater g ;
    g. input ();
    g. great ();

    greater g2 ;
    g2. input ();
    g2. great ();
    getch ();
    return 0;
}
```

g
a =
b =

g2
a =
b =

* ~~When~~ e In C++, we cannot declare value of any variable directly inside private or public section of any class.

Q. Prime no.
Q. Reverse of no.
Q.



Q. Calculator using switch - case.

1. PRIME NUMBER :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

class Number
{
    private :
    int n, flag =1
    public :
    void input ()
    {
        Cout << " Enter any no ";
        cin >> n;
    }
    void nprime ()
    { int flag = 1; int i;
      for (i = 2; i < n ; i++)
      {
          if (n % i == 0)
          {
              flag = 1;
              break;
          }
      }
    void prime ()
    {
      if (flag == 0)
        Cout << "n << " is prime ";
      else
      Cout << "n << " is not prime ";
} };
```

```
int main ()
{
    Number p;
    p.input ();
    p.nprime ();
    p.prime ();
    getch ();
    return 0;
}
```

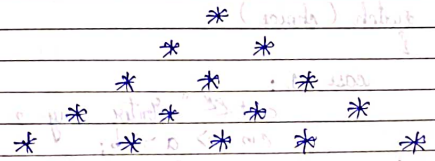
2. REVERSE OF NO :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;

class Reverse
{   private :
    int n, r, x=0;
    public :
    void input ()
    {
        cout << " Enter any no ";
        cin >> n;
    }
    void rev ()
    {   while (n != 0)
        {
            r = n % 10;
            n = n / 10;
            x = x * 10 + r;
        }
}
```

```
void output ()
{
    cout << " No. in reversed order = " << x;
}
};

int main ()
{
    Reverse r;
    r.input ();
    r.rev ();
    r.output ();
    getch ();
    return 0;
}
```

3. STAR PYRAMID :-

```
#
#
using namespace std;
class Pyramid
{
    private :
    int i, j, k, n;
    public :
    void input ()
    {
        cout << "\n Enter any no ";
        cin >> n;
    }
    void pyr ()
    {
        for (i = 1; i <= n; i++)
        {
            for (k = 1; k <= n-i; k++)
            {
                printf (" ");
            }
            for (j = 1; j <= i; j++)
            {
                printf ("* ");
            }
            printf ("\n");
        }
    }
};
int main ()
{
    Pyramid p;
    p.input ();
    p.pyr ();
    getch ();
    return 0;
}
```



```
#
#
#
using namespace std;
class Calculator
{
    private :
     int a, b, c;
    public :

    void input ()
    {
        Cout << " Enter any 2 no ";
        Cin >> a >> b;
    }
    void add ()
    {   c = a+b;
     }
    void sub()
    {   c = a-b;
    }
    void mul()
    {   c = a*b;
    }
    void div()
    {
        c = a/b;
     }
    void output ()
    {  Cout << " \n Answer = " << c;
    }
};
```

```
int main ()
{
    Calculator x;
     int ch;

    do
    {  Cout << " \n Enter 1 for + ";
       Cout << " \n Enter 2 for - ";
       Cout << " \n Enter 3 for * ";
       Cout << " \n Enter 4 for / ";
       Cout << " \n Enter 5 for exit ;
       Cin >> ch;

        switch (ch)
        {   case 1:
                    x.input(); x.add(); x.output();
                    break;
            case 2:
                    x.input(); x.sub(); x.output();
            case    break;
            case 3 :
                    x.input(); x.mul(); x.output();
                    break;
            case 4 :
                    x.input(); x.div(); x.output();
                     break
            case 5 :
                     exit (0);
            default:
                 Cout << " Enter right choice ";
        }
    }while (1);   }
```

⇒ SCOPE RESOLUTION OPERATOR :- (::)
This operator is used to resolve scope of any variable / function.

(i) It is used to access global members in any program.

⇒ INNER BLOCK :-
An inner block is used to separate scope of a function into smaller parts. The members written inside inner block cannot be accessed outside inner block but an inner block can access members of its outer block.

Ex eg :-

Q. WAP to understand first use of (::) operator & inner block.

```
#
#
using namespace std;
int a = 10;
int main()
{   int a = 20, b = 30;
    {
        cout << "\n Inside inner block - 1";
        int a = 50, b = 60;
(50)    cout << "\n Value of a=" << a;
(60)    cout << endl << "Value of b=" << b;
(10)    cout << endl << "Value of Global a = " << ::a;
    }
```

endl = manipulator (changes line)

```
    cout << "\n Back in main function";
    {
        cout << "\n Inside inner block - 2";
        int a = 100;
(100)   cout << "\n Value of a = " << a;
(30)    cout << "\n Value of b = " << b;
(10)    cout << "\n Outer Global a = " << ::a;
    }
(20) cout << endl << "Value of a = " << a;
(30) cout << endl << "Value of b = " << b;
(10) cout << endl << "Global a = " << ::a;
    getch();
    return 0;
}
```

Q. WAClass called FindArea, find area of circle when type of radius is integer / float / long

```
#include <iostream.h>
#include <conio.h>
using namespace std;

class FindArea
{   private:
        int r; int area; float r1; long r2;
    public:
    void input()
    {   cout << "Enter the radius of circle";
        cin >> r;
    }
    void areaf()
    {   area = 3.14 * r * r;
        area = 3.14 * r1 * r1;
        area = 3.14 * r2 * r2;
```

```
void public :
  void iarea (int r)
  {
     cout << "\n Area of circle = " << 3.14*r*r;
  }
  void farea (int float r)
  {
   cout << "\n Area of circle = "<<3.14*r*r;
  }
  void : larea (long r)
  {
   cout << "\n Area = "<< 3.14 * r * r;
  }
};
int   main()
{
  FindArea f;
  f. area (3);
  f. area ((float) 6.5);
  f. area ((long) 2619);
  getch ();
  return 0;
}
```

⇒ Function overloading :-
In this, names and return type of functions are same. But difference bhi functions is made by either their no. of parameters or datatype of parameters.
Overloading :- One name, many forms.

EXPLICIT TYPE :- Function overloading me first function me datatype nhi batana padta. Likin neeche ke function me datatype batana padta hai.

```
#
#
using namespace std;
class FindArea
{
  public :
    void area (int r)
    { cout << "\nArea = " << 3.14 * r * r;
    }
    void area (float r)
    { cout << "\n Area = " << 3.14 * r * r;
    }
    void area (long r)
    { cout << "\n Area = " << 3.14 * r * r;
    }
};
int main()
{  FindArea f;
   f. area (3);.    f. area ((float) 6.5);
   f. area ((long) 2619);
   getch (); return 0;
}
```

Q. WAC to find SI, when (i) P = int, R = int, T = int (ii) P = f, R = f, T = f using function overloading

```
#
#
using namespace std;

class Interest
{
```

Parameter :- Values sent in a function at the time of calling function.
Argument :- Values that are catched in a function.

```
public :
void sinterest ( int p, int r, int t)
{
  cout << "\n Simple Interest = " << (p*r*t)/100;
}

void sinterest ( float p, float r, float t)
{
  cout << "\n Simple Interest = " << (p*r*t)/100;
}
};

int main ()
{
  Interest x;
  x. sinterest ( 1000, 2, 4);
  x. sinterest ((float) 2050.50, (float) 2.5, (float) 5.5);
  getch ();
  return 0;
}
```

⇒ DEFAULT ARGUMENT IN FUNCTION :-
In C++, we can specify default value of any particular argument. If at the time of calling, we do not specify value of that argument, then its default value will be used as the value of that argument.
For eg :-

For default argument in a function :-
void interest (int p, float r = 3.2, int t); ✗
void interest (int p, float r = 3.2, int t = 5); ✓

```
#
#
using namespace std;
class Simple_Interest
{
  public :
  void interest ( int p, int t, float r = 3.2)
  {
    cout << "\n Simple Interest = " <<(p*r*t)/
  }
};
int main ()
{
  int p, t, age;
  Simple_Interest si;
  cout << "\n Enter principal & time ";
  cin >> p >> t;

  cout << "\n Enter your age ";
  cin >> age;
  if (age >= 60)
    si.interest (p, t, 5.2);
  else
    si.interest (p, t);
  getch ();
  return 0;
}
```

Q. WAC called student with member variables name, address, classname, age. Then create a function called input & output() to enter and print details of student. Then store & print info of 5 students

```
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
using namespace std;

class student
{
    private:
    char name[30];
    char address[50];
    char classname[10];
    int age;

    public:
    void input()
    {
        cout << "\nEnter your name = "; fflush(stdin);
        cin >> name;   cin.getline(name, 30)
        cout << "\nEnter your address = ";
        cin >> address;
        cout << "\nEnter your classname = ";
        cin >> classname;
        cout << "\nEnter your age = ";
        cin >> age;
    }
    void output()
    {
        cout << "\nStudent Details are : ";
        cout << "\n Your name is " << name;
        cout << "\n Your address is " << address;
        cout << "\n Your classname is " << classnam
        cout << "\n Your age is " << age;
    }
};

int main()
{
    student s;   int n, i;
    cout << "\nEnter no. of students ";
    cin >> n;

    for(i=0; i<n; i++)
    {
        cout << "\nEnter details of students"
        s.input();
    }
    for(i=0; i<n; i++)
    {
        cout << "\nDetails of students:"
        s.output();
    }
    getch();
    return 0;
}
```

Default access specifier of C++ is private.
getline() ⇒ similar gets().
(Cin >> name;) ⇒ only one word

```
#
#
#
using namespace std;
class student
{
    char name[30], add[50], cname[30];
    int age;

  public:
    void input()
    {
      Cout << "Enter name of student";
      fflush(stdin);
      cin.⇒ getline(name, 30);
      Cout << "Enter address";
      fflush(stdin);
      Cin.getline(add, 50);
      Cout << "Enter class name";
      fflush(stdin);
      Cin.getline(stdin);
      Cout << "Enter age of student";
      ~~fin~~
      cin >> age;
    }
    void output()
    {
      Cout << endl << "Name of student =" << name;
      Cout << endl << "Address = " << add;
      Cout << endl << "Class Name =" << cname;
      Cout << endl << "Age = " << age;
    }
};
```

```
int main()
{
    student s[100];
    int n, i;

    Cout << "Enter total no. of students";
    Cin >> ~~n~~ n;
    Cout << "Enter information of "<<n<<" students";

    for (i=0; i<=n-1; i++)
    {
      s[i].input();
    }
    Cout << "Information of all students";

    for (i=0; i<=n-1; i++)
        s[i].output();

    getch();
    return 0;
}
```

Q. WAC called employee & store & print name designation & salary of all employee.

⇒ ARRAY OF OBJECTS :-
An array in which all elements are of type objects.

```
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
using namespace std;

class Employee
{
    char name[30], desig[30];
    int salary;

public:
    void input()
    {
        Cout << "Enter name of employee";
        fflush(stdin);
        Cin.getline(name, 30);
        Cout << "Enter designation of employee";
        fflush(stdin);
        Cin.getline(desig, 30);
        Cout << "Enter salary of employee";
        Cin >> salary;
    }
    void output()
    {
        Cout << endl << "Name of employee =" << name;
        Cout << endl << "Designation =" << desig;
        Cout << endl << "Salary = " << salary;
    }
};
```

```
int main()
{
    Employee e[100];
    int n, i;

    Cout << "Enter total no. of employees";
    Cin >> n;

    Cout << "Enter info of " << n << "employees";
    for(i=0; i<=n-1; i++)
        e[i].input();

    Cout << "Information of employees:";
    for(i=0; i<=n-1; i++)
        e[i].output();

    getch();
    return 0;
}
```

Q. Data of Faculty :

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

class Faculty
{
    char name[30], subject[30];
    int salary;
public:
```

```
void input ()
{
    Cout << endl << "Enter faculty name: ";
    fflush (stdin);
    Cin.getline (name, 30);
    Cout << endl << "Enter subject taught, ";
    fflush (stdin);
    Cin.getline (subject, 30);
    Cout << endl << "Enter salary: ";
    ## Cin >> salary;
}
## void output ()
{
    Cout << endl << "Faculty name: " << name;
    Cout << endl << "Subject taught: " << subject;
    Cout << endl << "Salary: " << salary;
}
};
int main ()
{
    Faculty f[100];
    int n, i;
    Cout << "Enter total no. of faculties: ";
    Cin >> n;
    Cout << "Enter info of " << n << " faculties: ";
    for (i=0; i<=n-1; i++)
    ## f[i].input ();
    Cout << "Info. of all the faculties: ";
    f[i].output ();
    getch ();
    return 0;
}
```

Q. Data of Books :-

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

class Book
{
    char name[30], author[50];
    int ## E, cost;
    public:
    void input ()
    {
        Cout << "Enter name of book: ";
        fflush (stdin);
        ## Cin >> Cin.getline (name, 30);
        Cout << "Enter author's name: ";
        fflush (stdin);
        Cin.getline (author, 50);
        Cout << "Enter cost of book: ";
        fflush (stdin);
        Cin >> cost;
    }
    void output ()
    {
        Cout << endl << "Book name: " << name;
        Cout << endl << "Author name: " << author;
        Cout << endl << "Cost of book: " << cost;
    }
};
```

```
int main ()
{
    Book b[100];
    int i, n;
    Cout << "Enter total no. of books : ";
    Cin >> n;

    Cout << "Enter info of " << n << "books";
    for (i=0; i<=n-1; i++)
        b[i].input ();

    Cout << "Info of books ";
    for (i=0; i<= n-1; i++)
        b[i].output ();
    getch ();
    return 0;
}
```

⇒ CONSTRUCTOR :-

A constructor is a special member () of class which has the same name as that of class & it is used to initialize object of class.

A constructor has following properties :-

(i) The name of constructor must match with the name of class.
(ii) A constructor cannot have any return-type.
(iii) A constructor should be declared inside public section of class.

Types of constructor :-

∴ There are 3 types of constructor :-

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor.

There are 2 ways by which we can call any constructor :-

1. Explicit calling of constructor
2. Implicit calling of constructor.

Q. WAP to demonstrate 'default constructor'.

```
#
#
using namespace std;
class Test
{
    int a;
    float b;
public:
    Test ()
    {  a = 10;
       b = 4.5;
    }
    void input ()
    {  Cout << "Enter any integer & float value";
       Cin >> a >> b;
    }
    void output ()
```

input() is used to upgrade the values of ~~obj~~ variable
Test() is used to initialize the object.

Date / / Page no.

```
    { cout << "\n Value of integer = " << a;
      cout << "\n Value of float variable = << b;
    }
};
int main()
{
    Test t1 = Just();   // -> Explicit calling
    t1.output();
    Just t2;            // -> Implicit calling.
    t2.output();
    getch();
    return 0;
}
```

t_1	t_2
a = 10	a = 10
b = 4.5	b = 4.5

* Default constructor of c++ :-

```
Just()
{   a = 0;
    b = 0.0;
}
```

When we don't declare the constructor, c++ makes a default constructor of itself ie. with the initial values of 0, 0.0, NULL, etc.

⇒ Parameterised Constructor :-
A constructor in which we can pass parameters & set values as per our requirement is called parameterised constructor

Date / / Page no.

Q. WAP to demonstrate parameterised constructor :-

```
#
#
using namespace std;
class Just
{
    int a; float b;
    public:
    Test()
    {
        a = 10; b = 4.5;
    }
    Just(int x, float y)
    {
        a = x; b = y;
    }
    void input()
    {
        cout << "Enter any integer & float value";
        cin >> a >> b;
    }
    void output()
    {
        cout << "\n Integer value = " << a;
        cout << "\n Float value = " << b;
    }
};
int main()
{
    Just t1;
    t1.output();
```

```
    Just J2 (3, 9.2);
     j2.output ();
    Just j3 (25, 3.3);
     j3.output ();
     getch ();
     return 0;
}
```

⇒ CONSTRUCTOR OVERLOADING :-
If in a class, we write more than one constructors of different types, then it is called constructor overloading.

⇒ Reference Variable in C++ :-
Reference variables are used to create second name of existing variable. They are also known as 'alias'.

```
int main ()
{
    int a = 3;
    int &x = a;
    Cout << a;
    Cout << x;
    getch ();
    return 0;
}
```

a → 3 ← x

Reference variable (a ka dusra naam)

⇒ Copy Constructor :-
It is used to copy one object into another object.
Eg :-

```
#
#
using namespace std;
class Just
{
    int a; float b;
    public:
    Just ()
    {
        a = 10; b = 4.5;
    }
    Just (int x, float y)
    {
        a = x; b = y;
    }
    void input ()
    {
        Cout << "Enter any integer & float value"
        Cin >> a >> b;
    }
    void output ()
    {
        Cout << "In Integer value " << a;
        Cout << "In Float value " << b;
    }
    Just (Just &ob)
    {
        a = ob.a;
        b = ob.b;
    }
};
```

t1: a = 10, b = 4.5
t2: a = 3, b = 9.2
t3: a = 10, b = 4.5

Constructor Overloading ka test example → Copy Cons. because for copy const. we need another const. also in the same class. It cannot be written above.

```
int main()
{
    Test t1;
    t1.output();
    Test t2(3, 9.2);
    t2.output();
    Test t3(t1);
    t3.output();
    getch();
    return 0;
}
```

~~→ DESTRUCTOR :-~~

Q. WAC called Box with 3 private integer member variables, L, B, H. Write 5 diff constructors of Box class & a function called show() which print dimension of box. Then create 5 objects of Box class using all constructors & print their dimensions.

```
#include <iostream.h>
#include <conio.h>
using namespace std;
class Box
{
    int l, b, h;
    public:
    Box()
    {
        l = 10; b = 12; h = 13;
    }
    Box(int x, int y, int z)
    {
        l = x; b = y; h = z;
    }
    Box(int x)
    {
        l = x; b = x; h = x;
    }
    Box(Box &ob)
    {
        l = ob.l;
        b = ob.b;
        h = ob.h;
    }
    void input()
    {
        cout<<"Enter the dimensions of box";
        cin >> l >> b >> h;
    }
    void show()
    {
        cout << "\n Length of Box " << l;
        cout << "\n Breadth of Box " << b;
        cout << "\n Height of Box " << h;
    }
};
```

```
int main()
{
    Box b1;
    b1.output();        b1.show();
    Box b2(5, 6, 7);
    b2.output();        b2.show();
    Box b3(15, 18);
    b3.output();        b3.show();
    Box b4(5);
    b4.output();        b4.show();
    Box b5(b1);
    b5.show();
    getch();
    return 0;
}
```

⇒ DESTRUCTOR:-

A destructure destructor is a special member () of class which has the same name as class but it begins with tilde (~) sign. A destructor is used to destroy any object.

A destructor is used to perform any task at the time of destruction (memory deallocation of any object).

A destructor has following properties :-

1. It cannot have any return-type.
2. It cannot have parameter. i.e. we cannot overload a destructor.

Eg:-

```
class Destruct_Demo
{
    private:
    int x;
    public:
    Destruct_Demo(int a)
    {
        x = a;
    }
    void show()
    {
        cout << "\n Value of class variable" << x;
    }
    ~Destruct_Demo()
    {
        x=0;
        cout << "\n Value of class variable" << x;
    }
};
int main()
{
    Destruct_Demo d1(10);
    d1.show();
    getch();
    return 0;
}
```

⇒ STATIC VARIABLE :-

If we want to share value of any class variable b/w different objects of class then we should declare that variable as 'static variable'.

(iii) A static variable is called by class name not by class object.

It has following properties :-

(i) Its default initial value is 0.

(ii) The scope of static variable is only inside block in which it is defined but its lifetime is entire program.

Eg :-

```
#
#
void increment()
{
  int i=0;
  cout << "\n Value of i=" << i;
  i++;
}
void main()
{
  increment();
  increment();
  increment();
  getch();
}
```

Output :-
Value of i = 0
Value of i = 0
Value of i = 0

```
#
#
void increment()
{
static int i=0;
  cout << "\n Value of i" << i;
  i++;
}
void main()
{
  increment();
  increment();
  increment();
  getch();
}
```

Output :-
Value of i = 0
Value of i = 1
Value of i = 2

Because for static variable memory is allocated only once & until the program ends.

* Memory of static variable is allocated before the declaration of object.

⇒ STATIC FUNCTION :-

If we want to call any class () without creating object of class, then we should declare that function as 'static function'.

It has following properties :-

(i) It is called by classname not by class object.

(ii) It can access only static members of class whereas a non-static function can access both.

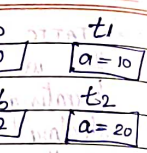
Eg :-

```
#
#
using namespace std;
class Static_Test
{
  private:
    int a;
    static int b;
  public:
    Static_Test(int x)
    {
      a = x;
      b++;
    }
    void output()
    {
      cout << "\n Value of non-static variable" << a;
    }
    static void show()
    { cout << "\n Value of static variable" << b;
    }};
```

if error comes ; then write :-

```
int static_Test :: b = 0;
void static_Test :: show ();
```

```
int main ()
{
    static_Test :: show ();
    static_Test t1 (10);
        t1.output ();
    static_Test t2 (20);
        t2.output ();
    static_Test show ();
    getch ();
    return 0;
}
```



** SCOPE RESOLUTION OPERATOR

(ii) It is used to access static members of class.

Q. WAP to demonstrate constructor, distructor, static variable & inner block.

```
class ConsDes
{
    private :
        static int count ;
    public :
        ConsDes ()
        {
            count ++;
            Cout << "\n Object no " << count << "created";
        }
        ~ConsDes ()
        {
            Cout << "\n Object no " << count << " Destroyed";
            count -- ;
        }
};
int ConsDes :: count = 0;
int main ()
{
    ConsDes C1, C2, C3 ;
    {
        Cout << "\n Inside inner block - 1";
        ConsDes C4, C5 ;
    }
    Cout << "\n Back in main function ";
    {
        Cout << "\n Inside inner block - 2.";
        ConsDes C6 C7 ;
    }
    getch ();
    return 0;
}
```

Output :-

```
Object no 1 created.
Object no 2 created
Object no 3 created
Inside inner block -1
Object no 4 created
Object no 5 created
Back in main function
Inside inner block -2
Object no 6 created
Object no 5 distroyed
Object no 4 distroyed
Back in main ()
Inside inner block - 2
```

Google Scholar → Research Paper ① IEEE ② ACM
(Abstract)
(Short overview in starting)

Object no 4 created.
Object no 5 created
Object no 5 destroyed
Object no 4 destroyed

will not be seen → Object no 3 destroyed
Object no 2 destroyed
Object no 1 destroyed.

⇒ PROPERTIES OF OBJECT - ORIENTED PROGRAMMING :-

1. Class
2. Object
3. Data encapsulation :-
Binding of data & functions together into a single unit is called data encapsulation
For eg :- class
4. Data abstraction :-
Act of representating essential features by hiding background details is called data abstraction.
Eg :- All predefined library functions, ATM, humans, electronic devices.
5. Data hiding :-
Hiding of data from external world is called data hiding.
Eg :- private members.
6. Inheritance
7. Polymorphism
8. Dynamic binding :- & What is working we come to know at runtime. Eg :- Virtual function.
9. Message Passing :-

Early binding - compile-time
Late binding - run-time (Virtual ())

⇒ INHERITANCE :-
It is the process by which one class will acquire properties of another class.
For eg :- A child inherits properties of parents, smartphones, vehicles, etc.
Syntax for inheritance in C++ is as follows :-

```
class Derived_Class_Name : inheritance_mode
                                Base_Class_Name
{
    body
};
```

The class whose properties gets inherited into another class is called base class & the class which inherits properties of another class is called derived class.

There are 3 modes in which we can inherit a base class into derived class :-

1. private mode :-
When we inherit a base class into derived class using private access specifier, then public and protected members of base class will become private members in derived class. Hence they are not available for further inheritance.

2. protected mode :-
When we inherit a base class into derived class using protected access specifier, then public and protected members of base class will become

protected members in derived class. Hence they are available for further inheritance.

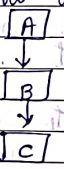
3 public mode :-
When we inherit a base class into derived class using public access specifier, then public and protected members of base class will become public & protected members in derived class. Hence they are available for further inheritance.

NOTE :- Private members cannot be inherited.

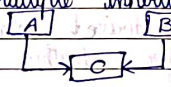
⇒ Types of inheritance :-
C++ categorize inheritance into 5 types :-

1. Single inheritance

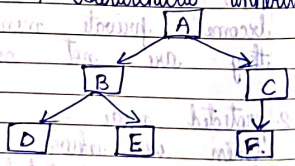
2. Multilevel inheritance



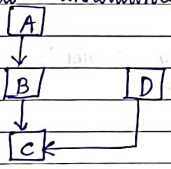
3 Multiple inheritance



4. Hierarchical inheritance



5. Hybrid inheritance :-



⇒ Access specifier in C++ :-

1. private :-
Private members are accessible only inside class. They cannot be accessed from outside class and cannot be inherited.

2. public :-
Public members are accessible everywhere.

3 Protected :-
Protected members are accessible from inside class and its derived class but cannot be accessed from outside class.

Q. WAP to demonstrate single inheritance.

```
class A
{
    private :
        int x;
    protected :
        int y;
    public :
        int
```

```
void input()
{
    cout << "Enter any 2 nos";
    cin >> x >> y;
}
void output()
{
    cout << "In Private number=" << x;
    cout << "In Protected number=" << y;
}
};
class B: public A
{
    private:
        int z;
    public:
        void inp()
        {
            cout << "In Enter any 2 nos";
            cin >> z;
        }
        void show()
        {
            cout << "In Value of derived class variable" << z;
        }
};
int main()
{
    B obj;
    obj.input();
    obj.output();
```

Protected: int y;
public: void input();
void output();

```
    obj.inp();
    obj.show();
    getch();
    return 0;
}
```

Q. WAP to demonstrate multi-level inheritance.

```
class A
{
    same ---
};
class B: public A
{
    same --
};
class C: public B
{
    private:
        char ch;
    public:
        void in()
        {
            cout << "In Enter any character value";
            cin >> ch;
        }
        void display()
        {
            cout << "In Value of class - c variable" << ch;
        }
};
```

protected int y;
public: input(), output(), inp(), show()

```
int main()
{
    C obj;
    obj.input();
    obj.output();
    obj.inp();
    obj.show();
    obj.in();
    obj.display();
    getch();
    return 0;
}
```

Q. WAP to demonstrate multiple inheritance.

```
#
class A
{
public:
    void showA()
    {
        cout << "In show function of class-A";
    }
};
class B
{
public:
    void showB()
    {
        cout << "In show function of class-B";
    }
};
```

showA() showB()

```
class C : public A, public B
{
public:
    void showC()
    {
        cout << "show function of class-C";
    }
};
int main()
{
    C x;
    x.showA();
    x.showB();
    x.showC();
    getch();
    return 0;
}
```

⇒ Ambiguity Problem in inheritance :-

If a derived class inherit properties of more than one base classes & those base classes have a property with same name then in derived class, we get multiple properties of same name & the object of derived class cannot determine which property it should call. This problem is known as ambiguity problem in inheritance.

The sol^n of above problem is scope resolution operator.

```
int main()
{
    C x;
    x.A::show();
    x.B::show();
    x.showC();
    getch();
    return 0;
}
```

⇒ FUNCTION OVERRIDING :-
If the names, returntype & parameters of functions in base class & derived class are same but their implementations are different then it is called function overriding

```
#
#
using namespace std;
class A
{
public:
    void show()
    {
        cout << "In show function of class-A";
    }
};
class B : public A
{
public:
    void show()
    {
        cout << "In show function of class-B";
    }
};
int main()
{
    B x;
    x.show();      x.A::show();
    getch();
    return 0;
}
```

Agar object jis class ka hai, pehle uska function call hoga. Agar was class me vo nhi hai, toh base class me jayega. otherwise no.

Hierarchical inheritance is well-organised single inheritance.

Q. WAC called student with member-variables name & enroll. no. WAF called setStudent() & a function called showStudent() to enter & print name & enroll no of student.

Then create a derived class of student class called Marks with 5 protected integer number variables m_1, m_2, m_3, m_4, m_5 which represents marks of student in all 5 of subjects. WAF called setMarks() & a function called showMarks() to enter & print marks of all subjects.
Then create a derived class of Marks class called Result with one member variable called total. WAF called showResult() which will print complete detail of student along with marks & total marks.

Then store & print info of all students of your class.

```
#include <iostream.h>
#include <conio.h>
using namespace std;

class student
{
    char name[30];
    char eno[15];
    public:
    void setStudent()
    {
        Cout << "\n Enter name of student";
        fflush(stdin);
        Cin.getline(name, 30);
        Cout << "\n Enter enrollment no";
        fflush(stdin);
        Cin.getline(eno, 15);
    }
    void showStudent()
    {
        Cout << endl << "Name of student=" << name;
        Cout << endl << "Enrollment no=" << eno;
    }
};
class Marks : public student
{
protected:
    int m1, m2, m3, m4, m5;
public:
    void setMarks()
    {
        Cout << "Enter your marks in 5 subjects=";
        Cin >> m1 >> m2 >> m3 >> m4 >> m5;
    }
    void showMarks()
    {
        Cout << "Marks of student are;";
        Cout << "Marks scored in subject-1=" << m1;
        Cout << "Marks in subject-2=" << m2;
        Cout << "Marks in subject-3=" << m3;
        Cout << "Marks in subject-4=" << m4;
        Cout << "Marks in subject-5=" << m5;
    }
};
class Result : public Marks
{
    private:
    float total=0;
    total=(m1+m2+m3+m4+m5);
    void showResult()
    {
        total = m1+m2+m3+m4+m5;
        Cout << "Total marks of student" << total;
        Cout << "Marks in individual subjects:";
        Cout <<
Cout << "subject-1=" << m1 << "\n subject-2=" << m2 <<
"\n subject-3=" << m3 << "\n subject-4=" << m4
<< "subject-5=" << m5;
    }
};
```

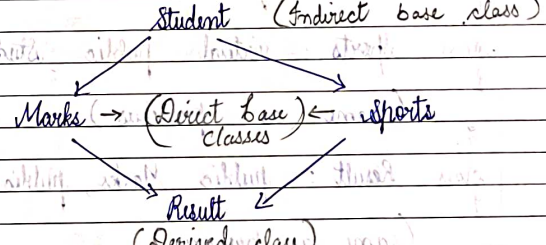
```
int main ()
{
    Result r; Result r[100];
    r. setStudent
    int i, n;
    Cout << " Enter no. of Students ";
    Cin >> n;
    for (i=0; i<n; i++)
    {
        r[i].setStudent ();
        r. showStudent ();
        r[i].setMarks ();
        r. showMarks ();
        r. showResult ();
    }
    for (i=0; i<n; i++)
    {
        r[i] showStudent ();
        r[i] showMarks ();
        r[i].showResult ();
    }
    getch ();
    return 0;
}
```

Q. WAP to demonstrate hybrid inheritance.

```
#
#
using namespace std;
class Student
{
    char name[30];
    char enroll[10];
public:
    void setStudent ()
    { HW.
    }
    void showStudent()
    { HW.
    }
};
class Marks : public Student
{ protected :
    int m1, m2, m3, m4, m5;
public :
    void set_marks ()
    { HW.
    }
    void show_marks ()
    { HW.
    }
};
class sports
{ protected :
    int score;
public:
    void setScore ()
    {
        Cout << "\n Enter score in sports ";
        Cin >> score;
    }
    void show_score ()
```

```
    {
        cout << "\n score in sports = " << score;
    }
};
class Result : public Marks, public Sports
{
    int total;
public:
    void showResult()
    {
        showStudent();
        showMarks();
        showScore();
        total = m1 + m2 + m3 + m4 + m5 + score;
        cout << "\n Final marks of student-" << total;
    }
};
int main()
{
    Result r[100];
    int i, n;
    cout << "\n Enter no of students";
    cin >> n;
    cout << "\n Enter info of " << n << "students";
    for(i=0; i<n; i++)
    {
        r[i].setStudent();
        r[i].setMarks();
        r[i].setScore();
    }
    cout << "\n Info. of " << n << "students";
    for(i=0; i<n; i++)
    {
        r[i].showResult();
    }
    getch();
    return 0;
}
```

⇒ Diamond Problem in Inheritance :-



If a derived class inherit properties of more than one direct base classes & those direct base classes have a common indirect base class, then in derived class, properties of indirect base class will be inherited more than once. Hence at the time of calling, object of derived class cannot decide which property it should call. This problem is known as "Diamond problem in inheritance.

The sol" is above problem is 'virtual' keyword that means we should inherit indirect base class into direct base classes using 'virtual' keyword.

For eg :-

#
#

class student
{ (same as previous)
};
class Marks : public virtual student
{
(same as previous)
};
class sports : virtual public student
{
(same as previous)
};
class Result : public Marks, public sports
{
(same as previous)
};
int main ()
{
(same as previous)
};

Q. WAC called First with 2 private int. member variables x & y. Write a parameterised constructor of class & a function called show () which will print values of variables x & y. Then create a derived class called Second with one member variable z of type float. Write a func" called input () & output () to enter & print value of variable z. Then store & print values of all variables.

#
#
using namespace std ;
class First
{
private :
int x, y ;
public :
First (int a, int b)
{
x = a; y = b;
}
void show ()
{
Cout << " Value of x=" << x ;
Cout << " Value of y=" << y ;
}
};
class Second : public First
{
float z ;
public :
void input ()
{
Cout << " Enter a value of float type" ;
Cin >> z ;
}
void output ()

```
Second (int m, int n, float p): First (m,n)
{ z=p;
}
{
Cout << "Value of ~~float type~~ z=" << z;
}
};
int main ()
{
Second s(10, 20, 3.8);
s.show();
s.input();
s.output();
getch();
return 0;
}
```

⇒ CONSTRUCTOR IN INHERITANCE :-

If we write a constructor in base class, then we have to write a constructor in derived class also and in derived class constructor, we must call base class construc

* POLYMORPHISM :-

One name, many forms OR
One interface, multiple methods ; is called polymorphism.
Eg:- Function overloading.

C++ categorize polymorphism into two types :-

POLYMORPHISM

Compile-time polymorphism
Eg:-
→ Function overloading
→ Constructor overloading
→ Operator overloading

Run-time polymorphism
Eg:-
→ Virtual Func"

** Function overiding is an example of compile-time as well as run-time polymorphism.

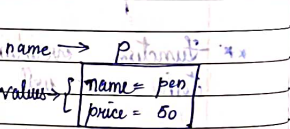
⇒ POINTER-TO-OBJECT :-

A pointer which stores address of an object is called pointer-to-object.

Eg:-

```
#
#
using namespace std;
class Product
{
    char name[30];
    int price;
```

```
    public :
        void input ();
        void output ();
};
void Product :: input ()
{
    Cout << " Enter name of product ";
    Cin. getline (name, 30);
    Cout << " \n Enter price of product ";
    Cin >> price ;
}
void Product :: output ().
{
    Cout << "\n Name of product = " << name;
    Cout << " \n Price of product = " << price;
}
int main ()
{
    Product P;
    Product *ptr ;
    ptr = & P
    ptr -> input ();
    ptr -> output ();

    getch ();
    return 0;
}
```



⇒ POINTER TO ARRAY OF OBJECTS :-
A pointer which stores address of an array of objects is called pointer to array to objects.

Name of array represents the address of its first elements.

```
#
#
using namespace std ;
class Product
{   same as previous -
};
void Product :: input ()
{ - - - - }
void Product :: output ()
{ - - - - }
int main ()
{ Product P[10]
    Product * ptr ;
    ptr = & P[0];
    int i;
    Cout << " \n Enter values of 10 Products ";
    for ( i = 0 ; i < 10 ; i ++)
    {
        ptr -> input ();
        ptr ++ ;
    }
    Cout << " \n Information of all products = ";
    {
        ptr = P;
        for ( i = 0 ; i < 10 ; i ++).
        {
            ptr -> output ();
            ptr ++ ;
        }
    getch ();
    return 0;
}
```

⇒ DYNAMIC MEMORY ALLOCATION in C++ :-
To dynamically allocate memory at runtime in C++, new keyword is used & to deallocate memory, delete keyword is used.
Eg :-

```
#
#
using namespace std;
class Product
{
    same as previous
};
void Product :: input ()
{
}
void Product :: output ()
{
}
int main ()
{
    Product *ptr;
    ptr = new Product;
    ptr -> input ();
    ptr -> output ();
    delete ptr;
    getch ();
    return 0;
}
```

⇒ POINTER IN INHERITANCE :-
A base class pointer may refer to derived class object but using pointer of base class, we can access only those members of derived class which are available in base class as well as in derived class, that means only over-ridden members.
For eg :-

```
#
#
using namespace std;
class Base
{
    public :
        int b;
        void show ()
        { cout << " Value of base class variable " << b;
        }
};
class Derived : public Base
{
    public :
        int d;
        void show ()
        { cout << " In Value of derived class variable " << d;
        }
};
int main ()
{
    Base ob;
    Base *bptr;
    bptr = & ob;
    bptr -> b = 10;
```

```
bptr -> show ();
Derived od.
Derived *dptr;
dptr = &od;
dptr -> b = 20;
dptr -> d = 30;
dptr -> show ();
bptr = &od;
bptr -> b = 50;
~~bptr -> d = 60;~~ XX   { Will not work }
bptr -> show ();          // Base class function

((Derived *) bptr) -> show ();   // Derived class version
(Explicit Type casting)

~~((Derived *) bptr) -> d = 100;~~ XX   // Will not work.
getch ();
return 0;
}
```

⇒ VIRTUAL FUNCTION / RUN-TIME POLYMORPHISM :-

To implement run-time polymorphism in C++ virtual function is used. If we declare a function as virtual () in base class & we declare a function with same prototype in derived class, then which func" will be called at the time of calling will depend on base class pointer. If base class pointer points to base class object then it will call base class member & if base class pointer points to derived class object then it will call derived class member. In this way, run-time polymorphism is implemented in C++

For eg :-

```
#
#
using namespace std;
class Base
{
    public:
    virtual void show ()
    {
        cout << "\n show function of base class";
    }
    void display ()
    {
        cout << "\n Display function of base class";
    }
};
class Derived : public Base
{
    public:
    void show ()
    {
        cout << "\n show func" of derived class";
    }
    void display ()
    {
        cout << "\n Display func" of derived class";
    }
};
```

```
int main()
{
    Base ob;
    Base * bptr;
    bptr = &ob;
    bptr -> show();          // base class()
    bptr -> display();       // base class()

    Derived od;
    Derived * dptr;
    dptr = &od;
    dptr -> show();          // derived class()
    dptr -> display();       // derived class()

    bptr = &od;
    bptr -> show();          // derived class()
    bptr -> display();       // base class()
    getch();
    return 0;
}
```

⇒ PURE VIRTUAL FUNCTION IN C++ :- (Abstract class (to inherit it))

A pure virtual func" does not have any implementation of its own. The implementation of p.v.f. is provided by derived class. If is used to provide function prototype to derived class. If derived class does not provide implementation of p.v.f. then derived class will not compile.

For eg :-

Jis class me p.v.f. declare kiya hai, uska object nhi bana sakte. Kyuki hume inheritance krna hai

```
#
#
using namespace std;
class Circle
{
    public:
        virtual void area(float r) = 0;
};
class CircleArea : public Circle
{
    public:
        void area(float r)
        {
            cout << "\n Area of circle = " << 3.14*r*r;
        }
};
int main()
{
    CircleArea c;
    c.area(5.2);
    getch();
    return 0;
}
```

⇒ OPERATOR OVERLOADING :- (Feature in C++)

If one operator performs more than ~~forms~~ one task in different situations, then it is known as operator overloading.

Its syntax is as follows :-

⇒ returntype operator operator_sign (parameters).
{
    statements;
}

-10
5-3] (-) is unary & binary

'operator' keyword is used for operator overloading.
There are some operators which cannot be overloaded.
(i) number access operator (.)
(ii) scope resolution operator (::)
(iii) conditional operator (?)
(iv) sizeof() operator

NOTE:- In operator overloading, the usual meaning of operator should not be changed

Q. WAP to overload unary operator (-).

```
#
#
using namespace std;
class Over
{
    private :
        int x, y;
    public :
        Over ()
        {
            x = 0; y = 0;
        }
        Over (int a, int b)
        {
            x = a; y = b;
        }
        void show ()
        {
            cout << "\n Value of x = " << x;
            cout << "\n Value of y = " << y;
        }
        void operator -()
        { x = -x;
        } y = -y;
int main ()
{
    Over u (10, -20);
    u.show ();
    -u ;                    // u.operator -();
    u.show ();
    getch ();
    return 0;
}
```

u: x = 10, y = -20

u: x = -10, y = 20

TWO Q. WAP to overload unary operator '++'

Q. WAP to overload binary operator '+'.

```
#
#
using namespace std;
class Boxer
{
    private :
        int x, y;
    public :
        Boxer ()
        {
            x = 0; y = 0;
        }
        Boxer (int a, int b)
        {
            x = a; y = b;
        }
        void show ()
        { cout << "\n Value of x = " << x;
          cout << "\n Value of y = " << y;
        }
```


```
Bover operator + (Bover m)
{
    Bover ans;
    ans.x = x + m.x;
    ans.y = y + m.y;
    return ans;
}
};
int main()
{
    Bover b1(10, 20);
    b1.show();
    Bover b2(3, 4);
    b2.show();
    Bover b3;
    b3 = b1 + b2;   // b1.operator+(b2);
    Cout << "In Object after addition";
    b3.show();
    getch();
    return 0;
}
```

ans
x=13
y=24

Q. WAC called Complex ~~st~~ with 2 private int mem variables real & image, WA default & parameterised constructor of class & a function called show-Complex() which will print complex no. in proper form. Then overload binary operator + to find sum of 2 complex nos.

```
#
#
using namespace std;
class Complex
{
    private:
        int real, image;
    public:
        Complex()
        {
            real = 0; image = 0;
        }
        Complex(int a, int b)
        {
            real = a; image = b;
        }
        void showComplex()
        {
            Cout << "In ∀ Complex No = " << real <<
                    " + i " << image;
        }
        Complex operator + (Complex c)
        {
            Complex ans;
            ans.real = real + c.real;
            ans.image = image + c.image;
            return ans;
        }
};
```

```
int main ()
{
    Complex c1 (2, 3);
      c1.showComplex ();
    Complex c2 (5, 7);
      c2.showComplex ();
    Complex c3;
      c3 = c1 + c2;
    Cout << "\n Sum of complex nos = ";
      c3.showComplex ();
      getch ();
      return 0;
}
```

HW Q
```
#include <iostream.h>
#include <conio.h>
using namespace std;
class Upover
{
    private :
      int x, y;
    public :
      Upover ()
      {
          x=0; y=0;
      }
      Upover (int a, int b)
      {
          x=a; y=b;
      }
      void show ()
      {
          Cout << "\n Value of x = " << x;
          Cout << "\n Value of y = " << y;
      }
      void operator ++ ()
      {
          x = ++x;
          y = ++y;
      }
};
int main ()
{
    Upover u (10, 20);
    u.show ();
    ++u;
    u.show ();
    getch ();
    return 0;
}
```

⇒ FRIEND FUNCTION :-

Friend () is a special outside function which is not a member of class but it can access public members of class directly without object & private members of class using class object.

'friend' keyword is used to declare a func" as friend () of class.

(VIOLATION OF DATA HIDING)

Q. WAC called Time with 2 private int member variables hour & minute. Overload binary op. + using friend () to find sum of 2 different time.

```
#
#
using namespace std;
class Time
{
    private:
        int hour, minute;
    public:
        Time ()
        {
            hour = 0; minute = 0;
        }
        Time ( int x, int y)
        {
            hour = x; & minute = y;
        }
        void showTime ()
        {
            Cout << " \n Time = " << hour << "Hours &"
                 << minute << " minutes ";
        }
        friend Time operator + (Time x, Time y);
};
Time operator + (Time x, Time y)
{
    Time ans;
    ans. hour = X. hour + Y. hour ;
```

t1: hour = 2, min = 40
t2: hour = 3, min = 50
t3: hour = 6, minute = 30

```
    ans. minute = X. minute + Y. minute ;
    if ( ans. minute >= 60 )
    {
        ans. hour ++ ;
        ans. minute - = 60 ;
    }
    return ans ;
}
int main ()
{
    Time t1 (2, 40);
      t1. showTime ();
    Time t2 (3, 50);
      t2. showTime ();
    Time t3 = t1 + t2 ;  // operator + (t1, t2);
    Cout << " \n Final time after addition = ";
    t3. showTime ();
    getch ();
    return 0;
}
```

ans: hour = 5→6, minute = 90→30

Q. WA friend () to find sum of two different time

```
~~Time~~ friend Time ()

Time add ()
{
    int z;
    z = x + y ;
    if (
```

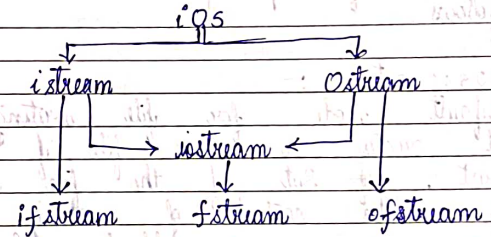
```
Time add ( int x, int y)
{
    int ans;
    ans = 0;
}

Time add ( Time X, Time Y)
{
    int ans;
    ans.hours = X.hour + Y.hour;
    ans.minute = X.minute + Y.minute;
    if (ans.minute >= 60)
    {
        ans.hour ++;
        ans.minute -= 60;
    }
    return ans;
}
```

→ Call :- `Time t3 = add (t1, t2);`

If we want to include a function in class, make one object less than required.
Same as binary operator Bover

⇒ FILE HANDLING IN C++ :-

C++ provides a class hierarchy for file handling.



→ ios :- input - output stream
Base class of all classes in C++.

→ (i) istream (derived class of ios)
Cin belongs to this class.

(ii) ostream (derived class of ios)
Cout belongs to this

→ iostream :-
$ Present inside <iostream.h>

→ ifstream :- (derived class of istream)
To read data from file, we create object of ifstream class.

→ ofstream :-
To write data into file, we create object of ofstream

→ fstream :-
To write & read data from class ; we create object of fstream class.

⇒ Various file-opening modes in c++ are as follows :-

1. ios :: out :-
Default mode for file writing.
It creates a new file, if file does not exist. But if the file already exists, it deletes contents of existing file & then open a new blank file.

2. ios :: in :-
Default mode for file reading.
Opens a file in read only mode.

3. ios :: binary :-
Required mode for binary files.

4. ios :: app :-
It opens a file in append mode but it can add data only in forward direction.

5. ios :: ate :-
Brings cursor to the end of file while opening but can add data in both directions.

6. ios :: nocreate :-
Does not create a new file if file already exits.

7. ios :: trunc :-
It deletes all the contents of existing file.

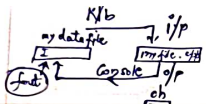
fail() :- Returns true when file operation fails otherwise false.

good() :- Returns true when file operation is successful otherwise false.

eof() :- This () returns true when file ends / end of file cond? occurs.
Returns false when file does not ends.

write() :- Writes all types of data.

read() :- Reads all types of data.



Q. WAP to write some text (character) data in a file.

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
#include <fstream>
using namespace std;

int main()
{   // code to open a file & stream.
ofstream fout("mydatafile.text", ios::out);

    // check whether file operation is successful or not
    if (fout.fail())
    {
        cout << "\n Can't create file, writing impossible";
        getch();
        exit(0);
    }
    // code to write data in a file.
    cout << "\n Enter some data in a file & press
            $ at end";
    char ch;
    do
    {
        cin >> ch;
        fout << ch;
    } while (ch != '$');
    cout << "\n Data saved successfully";
    // code to close a file
    fout.close();
} getch(); return 0;
```

Q. WAP to read text data from a file.

```
#
#
#
#
using namespace std;
int main()
{   // To read data from file, we create object of
    ifstream class.

    ifstream fin;
        // Code to open a file in reading mode
    fin.open("mydatafile.txt");
    if (fin.fail());
    {
        cout << "\n Can't open file, reading impossible";
        getch();
        exit(0);
    }
        // Code to read data from file.
    cout << "\n Contents of file are :";
    char ch;
    while (!fin.eof())
    {
        fin >> ch;
        cout << ch;
    }
        // Code to close file.
    fin.close();
    getch();
    return 0;
}
```

same file opening mode

Q. WAP to copy one file into another.

```
#
#
#
#
using namespace std;
int main()
{
    // Code to open a source file
    ifstream fin("myfile", ios::in);
    if (fin.fail())
    {
        cout << "\n Can't open source file, copy impossible";
        getch();
        exit(0);
    }
    // Code to open a target file.
    ofstream fout("yourfile", ios::out);
    if (fout.fail())
    {
        cout << "\n Can't open target file, copy impossible";
        getch();
        exit(0);
    }
    // Code to copy a file
    char ch;
    while (!fin.eof())
    {
        fin >> ch;
        fout << ch;
    }
    cout << "\n File Copied Successfully";
    // Code to close files
    fin.close();
    fout.close();
    getch();
    return 0;
}
```

Q. WAP to store students record in a file.

```
#include <iostream>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <fstream>
using namespace std;

class student
{
    private:
        char name[30], address[50];
        int roll;
    public:
        void input()
        {
            cout << "\n Enter the name of student";
            fflush(stdin);
            cin.getline(name, 30);
            cout << "\n Enter address of student";
            fflush(stdin);
            cin.getline(address, 50);
            cout << "\n Enter roll no of student";
            fflush(stdin);
            cin >> roll;
        }
```

```
    void output ()
    {
        Cout << "Name of student " << name;
        Cout << "\n Address of student " << address;
        Cout << "\n Roll no of student " << roll;
    }
};
int main ()
{
    fstream fout;
        // Code to open a file in binary mode.
    fout.open ("st_file", ios::out | ios::binary);
        // Code to check file opened / not.
    if (fout.fail())
    {
        Cout << "\n Can't create file, writing impossible";
        getch ();
        exit (0);
    }
        // Code to write student data in file.
    student s[100];
    int n, i;
    Cout << "\n Enter total no. of students";
    Cin >> n;
    Cout << "\n Enter info of " << n << "students";
    for (i=0; i<=n-1; i++)
    {
        s[i].input ();
        fout.write ((char *) & s[i], sizeof (student));
    }
Cout << "Data saved succesfully";
```

Typecasting →

```
    fout.close ();
    getch ();
    return 0;
}
```

Q. WAP to read students record from file.

```
#
#
#
#
#
using namespace std;
class student
{
    private:
        char name [30], address [50];
        int roll;
    public:
        void input ()
        {
            Cout << "\n Enter name of student ";
            fflush (stdin);
            Cin.getline (name, 30);
            Cout << "\n Enter address of student ";
            fflush (stdin);
            Cin.getline (address, 50);
            Cout << "\n Enter roll no of student";
            fflush (stdin);
            Cin >> roll;
        }
```

```
void output ()
{
    cout << "\n Name of student " << name;
    cout << "\n Address of student " << address;
    cout << "\n Roll No of student " << roll;
}
};
int main ()
{
    ifstream fin;
    // Code to open a file in binary mode.
    fin.open ("st_file", ios::in | ios::binary);
    // Code to check file opened or not.
    if (fin.fail())
    {
        cout << "\n Can't open file, reading impossible";
        getch();
        exit(0);
    }
    // Code to read students record from file
    student s[100];
    int i=0;
    cout << "\n Info of all students";
    while (fin.read ((char *) & s[i], sizeof (student)))
    {
        s[i].output ();
        i++;
    }
    fin.close ();
    getch ();
    return 0;
}
```

C++ = 48 keywords.
updated C++ = 63 keywords. | C = 32 keywords

Q. swapping of nos.

```
void swap (int, int);
{
void swap ()
{   int t;
    t = x;
    x = y;
    y = t;
}
```

⇒ TEMPLATES in C++ :- (Generic Programming)

Q. WAP to create a template func" with one generalised datatype.

```
#
#
using namespace std;
template <typename X>          → keyword for existing + new generalized datatype.
void swap (Xa, Xb)
{
    Xc;
    c = a;
    a = b;
    b = c;
    cout << "\n Values of variables after swapping
         = " << "\n a = " << a << "\n b = " << b;
}
```

```
int main()
{
    swap(10, 20);
    swap(90000, 870000);
    swap(4.5, 6.2);
    swap('A', 'B');
    getch();
    return 0;
}
```

Q. WAP to create a template func" with 2 generalize datatype.

```
#
#
using
template <typename X, typename Y>

void sum(X a, Y b)
{
    cout << "\n Sum of 2 nos = " << a+b;
}
int main()
{
    sum(10, 20);
    sum(90000, 8);
    sum(4.6, 2);
    getch();
    return 0;
}
```

char * = string (directly 'string' as datatype)

Q. WAP to create a template class.

```
#
#
using

template <typename X>

class student
{
    char *name;
    X roll;
    public:
        student(char *a, X r)
        {
            name = a;
            roll = r;
        }
        void output()
        {
            cout << "\n Name of Student " << name;
            cout << "\n Roll No. of student " << roll;
        }
};
int main()
{
    student <char *> s1("Ram", "CS123");
    s1.output();
    student < int > s2("Mohan", 10);
    s2.output();
    getch();
    return 0;
}
```

⇒ Templates are 'type safe' as it is checked 2 times : (i) When it is declared // created (ii) when it is instanticated.

Date: / / Page no:

TEMPLATES :-

They are used to create generalized functions or generalized classes. Template is also called 'parameterized type'.

Templates work only with parameters.

Library :- STL (Standard Template Library)