1 Basic IO in C++

1.1 Output / (<<) / Insertion / "Put to" operator:-

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

- The string in the quotation marks is to be displayed on the screen.
- The identifier cout is a predefined object that represents the standard output stream in c++.
- It is same as used in c as "printf" statement.
- The operator << is called the insertion or put to operator.
- It inserts or sends the contents of the variable on its right to object on its left.

1.2 Input / (>>) / extraction / "get from" operator : -

The statement

```
cin>>number1;
```

is an input statement and causes the program to wait for the user to type.

- The identifier 'cin' is a predefined object in C++ that corresponds to the standard input stream.
- It same as used in c as "scanf" statement.
- The operator '>>' is known as extraction or get from operator.
- It extracts or takes the value from the keyboard and assigns it to the variable on its right (number1).

2 Array in C++: introduction, declaration, initialization of one, two and multi dimensional arrays, operation on arrays

2.1 Introduction

- An array is a group of logically related data items of the same data type addressed by a common name, and all the items are stored in contiguous memory locations.
- For the instance, the statement:
  ```
  Int marks[10];
  ```
  defines an array by the name marks that can hold a maximum of ten elements.
- The individual elements of an array are accessed and manipulated using the array name followed by their index.
- The marks stored in the first subject is accessed by marks[0] and the marks stored in the 10th subject as marks[9].
- In this case, a sequence of ten integers representing the marks are stored one after another in memory.

2.2 Declaration

- Like other normal variables, the array variable must be defined or declared before its use. The syntax for defining an array……

```
DataType  ArrayName[ArraySize];
```

- In the definition, the array name must be a valid c++ variable, followed by an integer value enclosed in square braces.
- The integer value indicates the maximum number of elements the array can hold. The following are some valid array definition statements:

```
Int marks[100];             //integer array of size of 100
Float salary[25];           //floating-point array of size 25
Char name[50];              //character array of size 50
Int a[10], b[12], c[25];    //defines three array
Double d1, num[10];         // defines a variable and double arrray
```

- the representation of an array defined using the statement

```
int age[5];
```

is shown in following figure by assuming that each elements of array occupies two bytes.

2.3 Initialization of one dimension array

- A one-dimensional-arrays can be initialized at the point of their definition as follows:

```
Data-type array-name[size]={element-1, element-2,…..element-n};
```

- For instance, the statement

```
Int a[3]={ 6, 9, 3 };
```

defines one dimensional array of order 3 and initializes all its elements.

2.4 Initialization of double dimension array

- A two-dimensional-arrays can be initialized at the point of their definition as follows:

```
Data-type array-name[row size][col size]={
                                  {elements of first row},
                                  {elements of second row},
                                  ………..
                                  {elements of n-1 row},
                              };
```

- For instance, the statement

```
Int a[3][3]={
          {4, 7, 8},
          {2, 1, 9},
          {6, 9, 3},
       };
```

defines two dimensional array of order 3x3 and initializes all its elements.

- The first subscript (size of the row) may be omitted. Hence, he above definition can be replaced by:

```
        Int a[][3]={    {4, 7, 8},
                        {2, 1, 9},
                        {6, 9, 3}
                    };
```

- The inner braces can be omitted, permitting the numbers to be written in one continues sequence as follows:

```
        Int a[][3]={4, 7, 8, 2, 1, 9, 6, 9, 3};
```

- It has the same effect as the earlier definitions, but it suffers from readability.

2.5 Operations of array

- Example:

```
#include<iostream.h>
Void main()
{
        Int age[5];
        Float sum=0;
        For(int i=0; i<5; i++)
        {
                Cout <<“enter person”<<i+1<<” age: ”;
                Cin>>age[i];
        }
        For(i=0;i<5;i++)
        {
                Sum+=age[i];
        }
        Cout <<”average age is “<<sum/5;
}
```

3 *Working with string: introduction, declaration, manipulation and array of string*

3.1 Introduction of strings :

- Strings are used in programming language for storing and manipulating text, such as words, names, and sentences.
- It is represented as an array of characters and the end of the string is marked by the NULL (‘\0’) character. String constants are enclosed in double quotes.
- For instance,

  “Hello World”

  is a string. A string is stored in memory by using the ASCII codes of the characters that form the string. The representation of the string hello world in memory is shown in below.

3.2 Declaration of strings :

- An array of characters representing a string is defined as follow:
  ```
  Char array-name[size];
  ```
- As usual, the size of the array must be an integer value. For instance, the statement
  ```
  Char name[50];
  ```
  defines an array and reserves 50 bytes of memory for storing a set of characters.
- The length of this string cannot exceed 49 since, one storage location must be reserved for storing the end of the string marker.
- The program name.cpp defines an array and uses it to store characters.
  ```
  #include<iostream.h>
  void main()
  {
      char name[50];                //string definition
      cout << "enter your name <49-max>: ";
      cin>>name;
      cout<<"your name is "<<name;
  }
  ```
  RUN
  ```
  Enter your name <49-max>: abc
  Your name is : abc
  ```
- In main(), the statement
  ```
  Cin>>name;
  ```
  Reads characters and stores them into the variable name. The statement
  ```
  Cout<<"your name is "<< name;
  ```
  Outputs the contents of the string variable name.

3.3 Strings Manipulation:

- C++ has several built-I functions such as strlen(), strlwr(), etc., for string manipulation. To use these functions, the header file string.h must be included in the programming using the statement
  ```
  # include<string.h>
  ```

3.3.1 String Length

- The string function strlen() returns the length of a given string.
- A string constant or an array of characters can be passed as an argument.
- The length of the string excludes the end-of-string character (NULL).
- Syntax : strlen(s1);
- Example :

```
#include<iostream.h>
#include<string.h>
void main()
{
    char s1[25];
    cout<<"Enter your name";
    cin>>s1;
    cout<<"strlen(s1): "<<strlen(s1) <<endl;
}
```

3.3.2 String Copy

- The string function strcpy() copies the contents of one string to another.
- It takes two arguments, the first argument is the destination string array and the second argument is the source string array.
- The source string is copied into the destination string.
- Syntax : strcpy(s1, s2);
- Example :

```
#include<iostream.h>
#include<string.h>
void main()
{
    char s1[25],s2[25];
    cout<<"Enter string";
    cin>>s1;
    strcpy(s2,s1);
    cout<<s2;
}
```

3.3.3 String Concatenation

- The string function strcat() concatenates two strings resulting in a single string.
- It takes two arguments, which are the destination and source strings.
- The destination and source strings are concatenated and the resultant string is stored in the destination(first) string.
- Syntax : strcat(s1, s2);
- Example :

```
#include<iostream.h>
#include<string.h>
void main()
{
char s1[40], s2[25];
cout<<"Enter string s1";
cin>>s1;
cout<<"Enter string s2";
cin>>s2;
strcat(s1, s2 );
cout<<s1;
}
```

3.3.4 String Comparison

- The string function strcmp() compares two strings, character by character.
- It accepts two strings as parameters and returns an integer, whose value is

<0	if the first string is less than the second
==0	if both are identical
>0	if the first string is greater than the second.

- Whenever two corresponding characters in the string differ, the string which has the character with the higher ASCII value is greater.
- Syntax : strcat(s1, s2);
- Example :

```
#include<iostream.h>
#include<string.h>
void main()
{
char s1[25], s2[25];
cout<<"Enter string s1";          cin>>s1;
cout<<"Enter string s2";          cin>>s2;
int status = strcmp(s1, s2 );
cout<< " strcmp( s1, s2) : ";
if (status == 0) {  cout<< s1 << " is greater than " << 2;      }
else if(status > 0) {       cout<< s1 << " is greater than " << s2;    }
else        {        cout<< s1 << " is less than " <<s2;         }
}
```

3.3.5 String Upper/Lower case

- The string function strlwr() and strupr() convert a string to lower-case and upper-case respectively and returns the address of the converted string.
- Syntax : strlwr(s1);
- Syntax : strupr(s1);
- Example :

```
#include<iostream.h>
#include<string.h>
void main()
{

            char s1[25], temp[25];
              cout<<"Enter a string";
              cin>>s1;
              strcpy(temp, s1);
              cout<<"strupr(temp): "<<strupr(temp) <<endl;
              cout<<"strlwr(temp): "<<strlwr(temp) <<endl;
}
```

3.4 Array of string

- An array of strings is a two dimensional array of charcters & is defined as follows:
  Char array-name[rwo_size][column_size];
  For instance the statement
  Char person[10][15];
- Defines an array of string which can store names of 10 persons & each name can't exceed 14 characters; 1 character is used to represent the end of string. The name of the first person accessed by the expression person[0], & the second person by person[1] & so on.
- For example;
  Char city[5][10]={"anand","vvn","ahmedabad","baroda","surat"};

4 *Classes and objects in c++*

4.1 Classes :

- A class is a way to bind the data and its associated functions together. It allows the data (and functions) to be hidden, if necessary from external use.
- Defining a class means creating a new abstract data type that can be treated like any other built – in data type.
- A class specification has two parts:
  1. Class declaration.
  2. Class function definitions.

4.1.1 Class declaration:

- It describes the type and scope of its members.

4.1.2 Class function definitions:

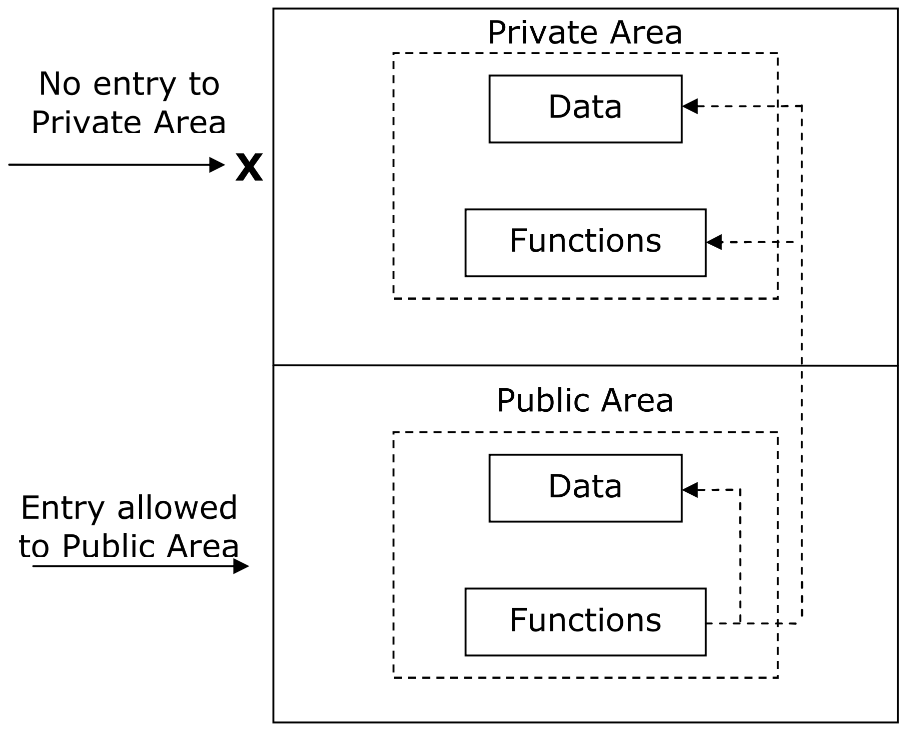
- It describes how the class functions are implemented.
- General form of class declaration:

```
class class_name
{
        private:
                variable declarations;
                function declarations;

        public:
                variable declarations;
                function declarations;
};
```

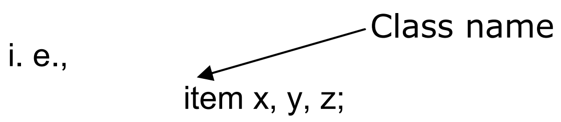
- The keyword class specifies that what follows is an abstract data of type class name.
- The body of a class is enclosed within braces and terminated by a semicolon.
- The class body contains the declaration of variables and functions. These functions and variables are collectively called class members.
- They are usually grouped under two sections, namely, private and public. The keywords private and public are known as visibility labels. These keywords are followed by a colon.
- The class members that have been declared as private can be accessed only from within the class.
- The variable declared inside the class are known as data members and the functions are known as member functions.
- The public members (both functions and data) can be accessed from outside the class. The binding of data and functions together into a single class type variable is referred to as encapsulation.

Class



4.2 Creating Objects:

- Once a class has been declared, we can create variables of that type by using the class name. in C++ the class variables are known as objects. We may declare more than one object in one statement.



- The necessary memory space is allocated to an object at this stage. Objects can also be created when a class is defined by placing their names immediately after the closing brace, as we do in the case of structures.
- i.e.,

```
class item
{       …………
        …………
        …………
}x, y, z;
```

4.3 Accessing Class Members:

- The private data of a class can be accessed only through the member functions of that class. The main() cannot contain statements that access data members.
- Format of calling a member function:

  object-name . function-name (actual-arguments);

- A member function can be invoked only by using an object of the same class.

4.4 Defining Member Functions:

- Member functions can be defined in two places.

  - Outside the class definition.
  - Inside the class definition.

4.4.1 Outside the class definition:

- Member functions that are declared inside a class have to be defined separately outside the class.
- An important difference between a member function and a normal function is that a member function incorporates a membership “Identity Label” in the header. This “Label” tells the compiler which class the function belongs to.

- **General form of member function definition is:**

```
return-type class-name :: function-name(argument declaration)
{

        Function body

}
```

- **The member functions have some special characteristics as bellow:**

  - Several different classes can use the same function name. The “membership label” (class-name) will resolve their scope.
  - Member functions can access the private data of the class. A non-member function cannot do so, but the friend function can access private data.
  - A member function can call another member function directly, without using the dot operator.

<u>4.4.2 Inside the class Definition:</u>

- Another method of defining a member function is to replace the function declaration by the actual function definition inside the class.
- Example:

```
class item
{
        private:
                int number;
                float cost;
        public:
                void getdata();                         //Declaration.
                //Inline function
                void putdata()
                {
                        cout<<“Number :”<<number<<endl;
                        cout<<“Cost :”<<cost<<endl;
                }
};

void item :: getdata()
{
        cout<<“\nEnter Number :”;
        cin>>number;
        cout<<“Enter Cost : ”;
        cin>>cost;
}

void main()
{
        item i1, i2;

        i1.getdata();
        i1.putdata();
```

- When a function is defined inside a class, it is treated as an inline function. Therefore all the restrictions and limitations that apply to an inline function are also applicable here.
- Normally, only small functions are defined inside the class definition.

5 *Constructors: default, parameterized, copy, overloading and destructor*

Constructor:-

- A constructor is a special member function whose task is to initialize the objects of its class. It is special because its name is the same as the class name.
- The constructor is invoked whenever an object of its associated class is created.
- It is called constructor because it constructs the values of data members of the class.
- The syntax for defining a constructor is shown below:

- Syntax:

```
class  ClassName
{
        public :
                ClassName( )   // This is a constructor.
                {
                    :
                    :
                 }
};
```

- Example:

```
class  test
{
        public :
                test( )   // This is a constructor.
                {
                    cout<<"I am constructor" ;
                 }
};
void   main()
{
        test    t ;   // This is call to constructor.
}
```

- It is possible to define a class without constructor. In such case, compiler calls a dummy constructor (i.e. which performs no action) when its object is created.

- Characteristics of a constructor:
- A constructor has the following characteristics:
  - It has the same name as its class.
  - It does not have any return type.
  - It is used to initialize data members of a class.
  - It is used to allocate memory to data members of a class.
  - It is the first member function to be executed automatically when the object of the class is created.

5.1 Default Constructor:-

- **"**Constructors without any arguments(or parameters) are known as default constructor."
- The syntax for defining a default constructor is shown below:
-

```
class  ClassName
{                                There is no argument
        public :
                ClassName( )    // This is a  default constructor.
                {
                     :
                     :
                 }
};
```

- Example:

```
class  test
{
        public :
                test( )    // This is a  default constructor.
                {
                    cout<<"I am default constructor" ;
                 }
};
void   main()
{
        test    t ;   // This is call to default constructor.
}
```

5.2 Parameterized constructor:-

- "Constructor with arguments are known as parameterized constructor."
- The syntax for defining a parameterized constructor is shown below:

```
class  ClassName
```

```
{
        public :
                ClassName(int  a)    // This is a  parameterized constructor.
                {
                    :
                    :
                 }
};
```

Here, one or more parameters are allowed

- Example:

```
class   test
{
        public :
                test( int   a )    // This is a  parameterized constructor.
                {
                   cout<<"The value of a="<<a ;
                 }
};
void    main()
{
        test     t(5) ;   // This is call to parameterized constructor.
        test       t = 4 ; // Another way of calling parameterized constructor.
}
```

Output:

```
The value of a=5
The value of a=4
```

5.3 Constructor Overloading :-

- A class have multiple constructor, and they differ only in total number of arguments or their data types or both. This is called constructor overloading.
- Example: Write a C++ program to build a class addition, which have two integer data members (x and y).

```
#include<iostream.h>
class   addition
{
      public :
         int   x, y, sum;
        addition() ;          // Default constructor
        addition ( int  a ) ;
        addition ( int   a , int  b) ;
} ;
```

```
class   addition : : addition()
{       x=0;
        y=0;
        sum = x + y;
        cout<<"Sum ="<<sum;        }
class   addition : : addition( int  a )
{       x=a;
        y=0;
        sum = x + y;
        cout<<"Sum ="<<sum;        }
class   addition : : addition(int   a,  int   b)
{       x=a;
        y=b;
        sum = x + y;
        cout<<"Sum ="<<sum;        }
void  main()
{    addition    a1 ;        //This call default constructor, display Sum = 0
    addition    a2(5) ;   //This call constructor with one argument, display Sum = 5
    addition a3(5,7);   //This call constructor with two argument, display Sum = 12     }
```

5.4 Copy constructor:-

- **"**A constructor having a reference to an object of its own class as an argument is known as copy constructor."
- The argument of a constructor can be of any type except an object of its own class as a **value** parameter.

```
class     test
{                   This is invalid. It is an Error
        :
        test (   test   t1 ) ;
        :
}
```

But, an object of a class can be passed as a reference parameter as shown below.

```
class     test
{                   Reference to an obiect of the class test
        :
        test (   test   &t1 ) ;   // Copy constructor.
        :
}
```

- Example:

```
#include<iostream.h>

class test
{
        int i;
        public:
          test();               //default constructor
          test(test &x);        //copy constructor
          void disp();
};
test::test()
{
        i=1;
}
test::test(test &x)
{
        i=x.i+1;
}
void test:: disp()
{
        cout<<i<<endl;
}
void main()
{

        clrscr();
        test t1;        //call default constructor
        t1.disp();
        test t2(t1);    // call copy constructor
        t2.disp();
        test t3(t2);    // call copy constructor again
        t3.disp();
        getch();
}
```

<u>5.5 Destructor</u>:-

- "Destructor is a special member function, which is called automatically when object is destroyed. It has the same name as its class prefix by ~ (Tilde) character."

- The syntax for defining a destructor is shown below:

```
class  ClassName
{
        public :
                ClassName( )    // This is a constructor.
                {
                     :
                     :
                 }
                ~ ClassName( )    // This is a destructor.
                {
                     :
                     :
                 }
};
```

- A class cannot have more than one destructor. Like constructor, destructors have no return type. Unlike the constructor, the destructor does not take any arguments.

- <u>Example</u>:

```
class  test
{
        public :
                test( )    // This is a constructor.
                {
                    cout<<"I am constructor" ;
                 }
                ~test( )    // This is a destructor.
                {
                    cout<<"I am destructor" ;
                 }

};
void    main()
{
        test     t ;   // This is call to constructor.
        Cout<<"\n In main function";
}
```

<u>Output</u>:

```
I am constructor
In main function
I am destructor
```

5.6 Differentiate between Constructor and Destructor

	Constructor		Destructor
1.	Constructor is a special member function, which is called automatically when object is created	1.	Constructor is a special member function, which is called automatically when object is destroyed
2.	Syntax: **Note** : Write down syntax here.	2.	Syntax:
3.	Example: **Note** : Write down example here.	3.	Example:
4.	Arguments can be passed to constructor.	4.	Arguments cannot be passed to destructor.
5.	Constructors cannot be virtual.	5.	Destructor can be virtual.
6.	It has the same name as its class	6.	It has the same name as its class but prefix by ~(Tilde) character.
7.	A class can have more than one constructor	7.	A class cannot have more than one destructor
8.	Constructor can be overloaded.	8.	Destructor cannot be overloaded.
9.	It is used to allocate the resources to the object of a class.	9.	It is used to release all the resources allocated to the object of a class.
10.	Types of constructor are: - Default constructor - Parameterized constructor - Copy constructor	10.	There are no any types of destructor.

6 Access specifiers, implementing and accessing class members

- Each user has different access privileges to the object. A class differentiate between access privileges by partitioning its contents and associating each one of them with any one of the following keword:
  - Private
  - Protected
  - public
- These keywords are called access control specifiers. All the members that follow a keword (upto another keyword) belong to that type.
- If no keyword is specified, then the members are assumed have private privilege. The following table specifies control of privileges.

Access Specifier	Accessible to			
	Own class member	Derived class member	Other class member	Object of a class
Private	Yes	No	No	No
Protected	Yes	Yes	No	No
Public	Yes	Yes	Yes	Yes

6.1 Private Member :

- The private members of a class have strict access control. Only the member functions of the same class can access these members.
- The private members of a class are inaccessible outside the class, thus, providing a mechanism for preventing accidental modifications of the data members.

```
Class person
{
        Private :
        // private members
        ………
        Int age ;
        Int getage();
        ………
};
Person p1;
A=p1.age;           // can not access private data
P1.getdata();       // can not access private data
```

- The following example illustrates the situation when all the members of a class are declared as private:

```
Class inaccessible
{
        Int x ;
        Void display();
        {
                Cout<<"\n data ="<<x;
        }
};
Void main()
{
        Inaccessible obj1;
        Obj1.x=5;           // error invalid access
        Obj1.display();     // error invalid access
}
```

- The class having all the members with private access control is of no use; there is no means available to communicate with the external world. Therefore , classes of the above type will not contribute anything to the program.

6.2 Protected Member :

- The access control of the protected members is similars to that of private members and has more significance in inheritance. The member function of the derived class can access the protected data member.

- Only class member and derived class member can access protected data, other class member can not access.

```
Class person
{
        Protected :    //access specifier
        // protected members
        .........
        Int age ;
Int getage();
.........
};
Person p1;
A=p1.age;              //same as private can not access protected member
P1.getdata();          //same as private can not access protected member
```

6.3 public members

- The members of class , which are to be visible (accessible) outside the class, should be declared in public section.
- All data members and functions declared in the public section of the class can be accessed without any restriction from anywhere in the program, either by function that belongs to the class or by those external to the class.
- Accessibility control of public members is shown in figure.

```
Class person
{
        Public  :    //access specifier
        // public  members
        .........
        Int age ;   public data
Int getage();  public function
.........
};
Person p1;
A=p1.age;              //class access public data
P1.getdata();          // can access public function
```

7 Working with objects : constant objects, nameless objects, live objects, arrays of objects

7.1 Constant Object:-

- C++ allows to define constant object of a class.
- **Syntax:**

Name of a class　　　　Name of an Object

```
        className    const    ObjectName ( Arguments ) ;
```

Keyword

- Constant objects are initialized only by a constructor during object creation. Once an object is created, no member function of its class can modify its data members. They can only read the data members. Such a data members are known read only data members and object is known as **read only object** or **constant object**.

```
//Constant object...
#include<iostream.h>
#include<conio.h>
#include<string.h>
class  student
{
   public :
      int  no;
      char name[50];

      student (int n, char nm[50]);
      void print();
};
student :: student(int n, char nm[50])
{
       no = n;
       strcpy(name,nm);
}
void student :: print()
{
       cout<<no<<name;
}
void main()
{
   student  const s1(1,"XYZ");
       clrscr();
       s1.no=5;  // Error :: Can not modify constant object.
       s1.print();
       getch();
}
```

7.2 Nameless Object:-

- C++ supports the creation of unnamed object. It means that in creation of an object, the name of an object need not mentioned.
- **Syntax:**

className (Arguments) **;**

Class name ↗ ↖ Arguments to constructor

- In above syntax, the name of object is not mentioned.
  - Passing arguments to an object is optional. If there is no argument, a default constructor is called. If there is argument, corresponding constructor is called.
  - The scope of a nameless object is limited only to the statement in which it is created because nameless objects are immediately destroyed after execution of a constructor.
  - The creation of nameless object is useful in function in function returning an object.

- <u>Example</u>:

```
#include<iostream.h>
class   test
{
    public :
      int    a;
      test ()
};
test  ::  test ()
{
      cout<<"Constructor \n";
}
void main()
{
     test () ;   // This statement created as well as destroyed nameless object.
}
```

7.3 Live object:

- Objects created dynamically with their data members initialized during creation are known as live object. The new operator is used for creating a live object. The syntax for creating live object is as follows.

Ptr_to_object = **new** className (Parameters) ;

- A class whose live object is to be created must have at least one constructor.

```
#include<iosteam.h>
#include<conio.h>
class    student
{
     private :
               int    roll_no;
               char   *name;
     public :
        student(  int   no)
        {        roll_no = no;
                 name = NULL:        }
        student( int   no,  char  *nm)
        {        roll_no = no;
                 name  = new  char [ strlen(nm)+1 ] ;
                 strcpy(name,nm);    }
        ~student()
        {        delete   name;        }
};

void main()
{
        student    *s1, *s2;
                 s1   =   new    student( 1 );                          //Live object
                 s2   =   new     student( 1, "xyz") ;  //Live object
}
```

- The syntax for destroying live object is the same as that of normal dynamic objects.

```
delete     object_Name ;
```

- The delete operator is used to release memory.

<u>7.4 Array of object</u>:-

- It is possible to define object variable, which is an array. Such a variable is known as array of object.
- Consider the following class.

```
class       student
{
  public:
        int       no;
        char     name[25];
        int       marks;
       void   read();
      void   print();
};
```

```
void    student ::  read()
{
        :
}
void   student  :: print()
{
        :
}
```

- The syntax for defining an array of object is shown below.

```
        class      class_Name         object_Name [ size ] ;
                         or
      class_Name       object_Name [ size ];
```

- In above, class is keyword, class_Name is the name of the class, object_Name is any valid variable name and size is integer constant value. An array of the above class can be defined as follows.

```
      class        student       s[50];
                         or
      student       s[50];
```

- The syntax for accessing its members using an index.
  Class_Name [index]. MemberName   //for data member
  Class_Name [index]. MemberFunction(parameter) //for member function
- The following statement access members of the class.

  cin>> s[4].name ;   //It reads name of the 5th student.
  cin>>s[2].marks ; //It reads marks of the 3rd student.
  cout<< s[0].no ;   // It prints number of 1st student.
  S[2].read() ; //It calls read function of a class.

- Program: Following program reads the number, name and total marks of the n student and displays them.

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

class      student
{
  private :
      int       no;
      char     name[25];
      int       marks;
    public :
        void   read();
        void   print();
};
```

```
void   student ::  read()
{
                cout<<"Student number…";
                cin<<no;
                cout<<"Student name…";
                cin>>name;
                cout<<"Student total marks…";
                cin<<marks;
}
void   student  :: print()
{
        cout<<"\n"<< no<<"  "<<name<<"  "<<marks;
}
void main()
{
    student   s[50];
    int     i,n ;
        clrscr();
        cout<<Enter number of student…";
        cin>>n;
        for( i=0 ; i<n ; i++)        s[i].read();
        cout<<"\n Student Report ";
        cout<<"\n ------------------------";
        cout<<"\n No    Name   Marks";
        cout<<"\n -----------------------";
for ( i=0 ; i<n ; i++)        s[i].print();
        getch();
}
Input:
        Enter number of student… 2
        Student number…1
        Student name…Anjali
        Student marks…50
Student number…2
        Student name…Tushar
        Student marks…80
Output:
           Student Report
        -----------------------------
        No.     Name         Marks
        -----------------------------
        1       Anjali        50
        2       Tushar        80
```