# 4. Array

Array is a collective name given to a group of similar elements. These similar elements could be int, float or char etc. data type.

→ All elements of an arrays are of same data type.

→ Array can be represented as -

i) One dimensional array - i.e. linear list of variables

ii) Two dimensional array - i.e table of variables

iii) Multidimensional array.

## Declaration of one dimensional Array

→ Like other variables an array needs to be declared before it is used, so that compiler can allocate space for it in memory.

→ General form of array declaration is

```
data_type variable_name [ size ];
```

→ data type specifies the type of element that can be stored inside the array.

eg. - int group [10] ;

↓ ~~subscript~~ / dimension

↑ ~~subscript~~ ~~variable~~

declares group as an array that contain maximum 10 elements of int type.

[] tells compiler that we are dealing with array.

## Initialization of one dimensional Array

→ After an array is declared, its elements must be initialized, otherwise it will contain garbage

→ An array can be initialized at following stages -

i) At compile time

ii) At run time

i) Compile time Initialization :-

→ In this, array is initialized when it is declared.

General form is -

data_type array_name [size] = { list of ~~variable~~ values };

the values in list are separated by commas.

int count [4] = { 6, 8, 44, 50 };

char name [6] = { 'C', 'P', 'B', 'N', '\0' };

→ If size of array is greater than number of values, then remaining locations are filled with zero.

→ if size of array is less than numb values then unexpected problem may occur program.

→ We can also write as

```
int count = { 4, 8, 40, 60 };
```

this will creat an array of size 4.

ii) Run time initialization :-

→ For initializing large arrays, this method is used. This can be done in following way:

```
- - - - - -
- - - - - -
for (i=0; i<100; i++)
{
   if i<50
      sum[i] = 0.0;
   else
      sum[i] = 1.0;
}
- - - - - - -
- - - - - - -
```

→ so, first 1st 50 elements are initialized to zero while remaining 50 elements are initialized to 1, at run time

# Array Operation

## Insertion

→ Insertion operation is to insert one or more data elements into an array. Based on requirement, a new element can be added at beginning, end or an given index of array.

→ Steps to perform insertion :-

i) Get the element to be inserted. Say value.

ii) Get the position at which element is to be inserted. Say pos index

iii) Then shift the array elements (all) from this position to one position ahead

iv) Insert element x (value) in position pos. (index)

```
# define max 5          // length of array
main()
{
    int array[max] = {1, 2, 4, 5};
    int N = 4;          // No. of elements present in array
    int i = 0;          // loop variable
    int index = 2;      // index location to insert new value
    int value = 3;      // new data to be inserted
// Print array before insertion
    printf("Printing array before insertion - \n");
```

```
for (i=0 ; i<N ; i++)
{
    Printf (" array [%d] = %d \n", i, array[i
}
// shift rest elements forward
for (i=N , i >= index ; i--)
{
    array [i+1] = array [i] ;
}
// add new data at desired index position
array [index] = value ;
// increase N to show present no of elements
N++ ;
// print array after insertion
printf (" Printing array after insertion - \n");
for (i=0 ; i <N ; i++)
{
    printf (" array [%d] = %d \n", i, array[i]);
}
}
```

After compilation this program will give following result

Output-

printing array before insertion -

array [0] = 1

array [1] = 2

array [2] = 4

array [3] = 5

printing array after insertion -

array [0] = 1

array [1] = 2

array [2] = 3

array [3] = 4

array [4] = 5

## Searching

→ For an array, search operation for an array element can be performed based on its value or its index.

→ Steps to perform searching using index value :-

I) Get the element to be searched say value

II) Set j = 0, i.e. set index to zero.

III) If array [j] is equal to value, then jump to step

IV) Set j = j + 1

V) Repeat step (III) & (IV) while j < N

vii) Print j, value

```
main ( )
{   int   n = 5;
    int   array [ n ]  =  { 1, 3, 5, 7, 8 };
    int   value = 5 ;
    int   j = 0 ;
    printf (" The original array elements are : \n");
    for (i=0 ; i <n ; i++)
    {
        printf (" array [%d] = %d  \n", i , array[i]);
    }
     while  ( j < n )
    {
        if ( array [j] = = value )
        {
            break ;
        }
        j ++ ;
     }
        printf (" found element %d at pindex
                value %d \n", value , j );
}
```

after compilation, result will be

The original array elements are :

array [0] = 1

array [1] = 3

array [2] = 5

array [3] = 7

array [4] = 8

found element 5 at position 3.

## Deletion Operation

→ Deletion refers to removing an existing element from array and re-organising all elements of an array.

→ steps to perform deletion operation :-

i) Get the index value from where element to be deleted. say index.

ii) set j = index

iii) set array [j] = array [j+1]

iv) Set j = j+1

v) Repeat step (iii) & (iv) until j is equal to less than total no. of elements in array say N

vi) set N = N-1

```
main ()
{   int   N = 5;
    int   array [N] = {1, 3, 7, 8, 5};
    int   index = 3;
    int   i , j ;
Printf (" The original array elements are : \n");
for (i=0 ; i<N ; i++
{
   Printf  (" array [%d] = %d  \n" , i , array[i] );
}
   j = index ;
while  (j < N)
{
arrBy[ j-1] = array [j];
  j = j+1 ;
}
   N = (N-1) ;
Printf (" The array elements after deletion : \n");
for (i=0 ; i<n ; i++)
{
  Printf  (" array [%d] = %d  \n" , i , array[i]);
  }
}
```

After compilation following result is produced

Output

The original array elements are :

array [0] = 1

array [1] = 3

array [2] = 7

array [3] = 8

array [4] = 5

The array elements after deletion are :

array [0] = 1

array [1] = 3

array [2] = 8

array [3] = 5

# String in C

→ Sequance of character terminated at null character.

→ ASCII code of null character is zero.

→ String is not a data type, so string is declared as an array of character.

→ General form of declaration of string variable is-

~~char name [10];~~

char string_name [size];

→ size of string should be equal to no. of character in array plus one. Extra one space is for null character.

→ Eg. String can be initialize at compile time in following way -

char name[10] = {'B', 'H', 'O', 'P', 'A', 'L', '\0'};

or char name [10] = "BHOPAL";

→ Operation on string we have already studied in library functions.

```
    clrscr();
    printf("%s", s);
    getch();
}
```

→ here %s represents string data type and it can directly print a string.

→ for certain calculation we can use char with loop and for directly printing the string, we can use %s data type.

→ There is one more method instead of printf, we can use

```
puts(s);
```

→ This will print the string, this command is specially used to print string only and also it changes line itself after printing a string i.e no need of \n.

→ Now, we have one more method to assign string in character array. i.e

```
char s[10] = "SUSHMA";
```

→ This is much easier

→ If we want to take string input from user, we will use this program.

2. Devel

```
{
    char s[20];
    clrscr();
    printf("Enter your name - ");
    scanf("%s", s);
    puts(s);
    getch();
}
```

→ In this program we increased string size as we don't know how long word user will enter.

→ also for string

s = &s[0]

so we need not to write a & before s. it will automatically store the 1st character in s[0] and cosequently other characters in others.

→ But we cannot write two words using this program because scanf can not take multiword string because scanf considers space as data separator. [Enter, tab]

→ so we can use other input instruction

gets(s);

→ This can input many words of 20 characters as we have limited it.