ARRAYS

INTRODUCTION :

If a program to store the roll numbers of students of a class is to be written., then we will have 2 options:-

Suppose , students = 60

Option 1:- declare 60 different variables and store each roll no in a variable.

Option 2:- declare a single variable in such a way that, we can store all the roll no's in it.

Definitely, the programmer's choice will be option-2. In order to declare a variable, to store many values in it, we have to use the concept of "ARRAY" .

DEFINITION:

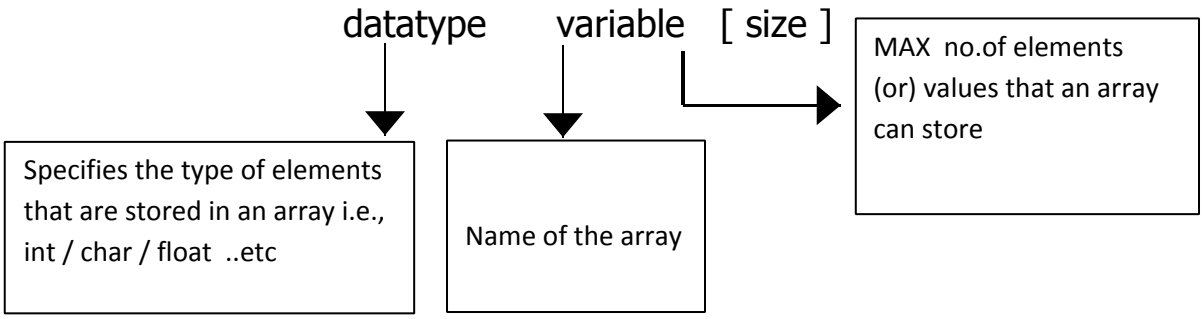
An array can be defined as, a group of related data items that share a common name. these data items may be int / char / float / double ..etc

- All these are stored in continuous memory locations (on RAM)
- Sometimes an array can be called as a SUBSRCIPTED VARIABLE

Difference between ARRAY and VARIABLE:

A variable can store only a single value at a time. Where as an array can store a multiple values at a time.

Syntax of Array:



Note : An array at a time can hold multiple values of similar data type only i.e., at a time array can hold group of integers, (or) group of floating point numbers etc..,

Declaration of variable:

```
int a;
```

this syntax indicates:-

- can hold 1 integer value
- only 2 bytes of memory is registered. Therefore it is integer data type.

Declaration of array:

```
int  a [ 5 ] ;
```

- can hold 5 integer values
- for each integer value 2 bytes of memory will be registerd i.e.,

5 * 2 = 10 bytes of memory is registerd

Total memory size of Array:

Total size = size * (size of data type)

Ex:-

(i) float k[10];

total size = 10 * 4 => 40 bytes

(ii) char c[5];

total size = 5 * 1 => 5 bytes

Declaring an array and Defining an array:

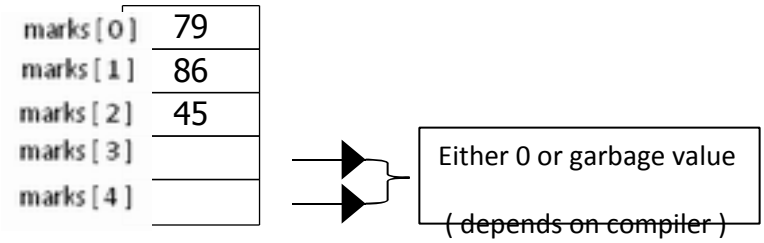
(i) int marks [5] = { 70, 86, 45, 67, 89 };

in the above declaration, the array 'marks [5]' will store the values as ahown below:

marks [0]	79
marks [1]	86
marks [2]	45
marks [3]	67
marks [4]	89

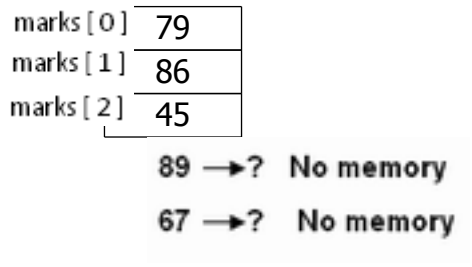
(ii) int marks [5] = { 79, 86, 45 };

in the above declaration the values are stored in memory as follows:-



(iii) int marks [3] = {79, 86, 45, 89, 67};

Memory allocation will not be done properly for the above declaration and it results in an error - (too many parameters in declaration)



This is because , size of array is given as only 3. Thefore for 4th and 5th elements memory will not allocated, which results in an error.

(iv) int marks [3] = { 79, 86, 45, 89, 67 } ;

for this declaration , memory will be allocated based on the input values of an array. i.e.,

in above declaration , 5 values are being given as inputs, the compiler automatically, fits the size of array as 5 and gives 5*2 bytes = 10 bytes of memory.

NOTE: Initialization of an array must be done using curly braces (flower brackets) only.

Datatype name[size] = {element1,…};

WITHOUT ARRAY

```
void main()
{
int a-10, b=20, c=30;
printf(" a = %d ,", a );
printf("b = %d ,", b );
printf("c = %d .", c );
}
output:
a=10 , b=20 , c=30 .
```

WITH ARRAY

```
void main()
{
int a[5]= {10, 20, 30}, i;
printf("array elements are:");
for(i=0;i<5;i++)
{
Printf(" %d ",a[i]);
}          }
Output: array elements are: 10 20 30
```

NOTE: all the array elements are numbered, starting from zero (0).

int m[5] ={75, 85, 68, 97, 54};
In above declaration,
m[2] =68 i.e. 3rd element of the list, as it starts from 0, not 1.

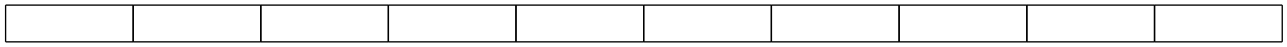
ENTERING DATA INTO AN ARRAY:

Generally, values at runtime can be assigned to an array using "for loop".

Ex: printf("enter marks");

```
for(i=0;i<10;i++)
{
scanf("%d",&marks[i]);
}
```

The for loop causes the process of assigning values at a runtime, till i=9 i.e. 10 values (0-9) are stored in an array. Values will be stored, in the array as:-



// WAP to accept values for five integers, at runtime and display them.

```
Void main()
{
Int a[20], i;
```

```
Printf("enter values into array");
For(i=0;i<5;i++)
{
Scanf("%d",&a[i]);
}
```

➔ To accept values into an array.

```
Printf("values in the array are:");
For(i=0;i<5;i++)
{
printf("%d",&a[i]);
}
```

→To display values of array.

NOTE: Even to print the values of array, "for loop" should be used

//Program to accept 2 arrays and add the corresponding elements of those arrays

```
void main()
{
        int a[10], b[10], sum[10];
        int I, j;
        printf("enter size of Arrays");
        scanf("%d", &j);
        printf("enter values for array – 1");
        for (i=0; i<j; i++)
        {
                Scanf("%d",&a[i]);
        }
        Printf("enter values for array – 2");
        for (i=0;i<j;i++)
        {
                Scanf("%d",&b[i]);
        }
        for(i=0;i<j;i++)
        {
                Sum[i]=a[i]+b[i];
        }
        printf("sum of arrays is : ");
        for(i=0;i<j;i++)
        {
                printf("%d",sum[i]);
        }
}
```

OUTPUT:-

Enter size of arrays 4
Enter values for array -1 10 20 30 40
Enter values for array -2 5 10 15 20
Sum of arrays is : 10 30 45 60

Elements are stored in array as shown below:-

a[0]	10		b[0]	5		sum[0]	15
a[1]	20	+	b[1]	10	=	sum[1]	30
a[2]	30		b[2]	15		sum[2]	45
a[3]	40		b[3]	20		sum[3]	60

//Program to print the largest element of the array

```
#include<stdio.h>
Main()
{
        int x[6], large;
        int i;
        printf("enter elements in array");
        for(i=0; i<6;i++)
        {
                scanf("%d",&x[i]);
        }
large=x[0];                     /*Assuming x[0] in large */
for(i=1, i<6;i++)
{
        if(x[i]>large)
        {
                large=x[i];
        }
}
printf("largest element is %d",large);
}
```

SEARCHING

Gathering any information (or) trying to find any data is said to be a Searching process.

Searching technique can be used more efficiently if the data is present in an ordered manner.

Most widely used Searching methods are:-

i) Linear Search (Sequential Search)
ii) Binary Search

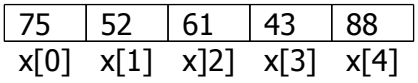
LINEAR SEARCH:-

Suppose an array is given, which contains 'n' elements. If no other information is given and we are asked to search for an element in array, than we should compare that element, with all the elements present in the array. This method which is used to Search the element in the array is known as Liner Search. Since the key element/ the element which is to be searched in array, if found out by comparing with every element of array one-by-one, this method is also known as Sequential Search.

Example:-
An array 'x' is given, which contains 5 elements in it i.e.,

int x[5]= { 75, 52, 61, 43,88};

These are stored in the memory in continuous memory location.
Fig.(a)



We are asked to search for an element '43' in the array.

Then we have to compare '43' with each and every element of the array. This is represented in the below figures:-

Fig.(b)

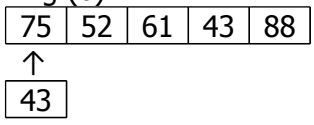
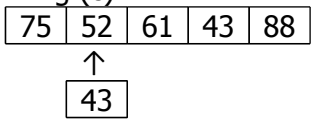


fig.(c)



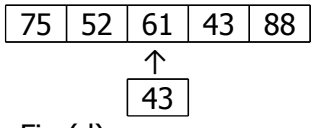


Fig.(d)

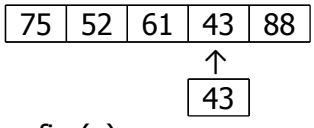


fig.(e)

// WAP to demonstrate LINEAR SEARCH

```
#include<stdio.h>
#include<conio.h>
Void main()
{
int  linear[20],n,i,k,temp=0;
clrscr();
printf(" enter range of elements");
scanf(" %d",&n);
printf("enter elements into array");
for(i=0; i<n; i++)
{
        scanf("%d",&linear[i]);
}
printf("enter search key:");
scanf("%d",&k);

for(i=0, i<n; i++)
{
        if(k==linear[i])
        {
        temp=1;
        printf("%d is found at location %d", k, i+1);
        break;
        }
}
if(temp!=1)
{
        printf("element not found");
}
getch();
}
```

OUTPUT:-

```
Enter range ofelements
5
Enter elements into array
45  68  75  83  99
Enter search key
83
83 is found at location 4
```

Reason for using 'temp" variable:

Without "temp", if we write else statement (or) else-if, they should be written after 'if' inside for loop. If we write outside for loop and error misplaced else will occur.

If we write inside for loop, every time those statements will be executed. Therefore we use 'temp' variable and assign its value to 1, and use the 'temp' variable in 'if' condition. Whenever search key is not found in the list.

EXPLANATION:-

Number of elements for which Linear search technique is to be performed, should be taken i.e. array size. (n).

Let n=5.

5 elements are entered into an array using for loop and stored in continuous memory locations.

	1		2	3	4	5
	72	45	68	59	83	
Linear	[0]		[1]	[2]	[3]	[4]

The element which is to be searched is taken i.e., Search key element (k)

k=59

According to the linear search method, now, '59' should be compared with every element in the list and when '59' matches with the element in the list, then its position should be displayed. To implement this logic, the c-Program code is:-

```
for(i=0; i<n; i++)
{
        if(k==liner[i]);
        {
        temp=1;
        printf("%d found at %d",k, i+1);
        }
}
if (temp!=1)
printf("element not found");
```

BINARY SEARCH:-

Efficient search method for large arrays. Liner search, will required to compare the key element, with every element in array. If the array size is large, liner search requires more time for execution.

In such cases, binary search technique can be used.

To perform binary search:-

i) Elements should be entered into array in Ascending Order

ii) Middle element of the array must be found. This is done as follows

Find the lowest position & highest position of the array i.e., if an array contains 'n' elements then:-

Low=0
High =n-1
Mid =(low+high)/2

Note: We are calculating mid position of the array not the middle element. The element present in the mid position is considered as middle element of array.

Now the search key element is compared with middle element of array. Three cases arises

Case 1 : If middle element is equal to key, then search is end.
Case 2 : If middle element is greater than key, then search is done, before the middle element of array.
Case 3 : If middle element is less than key, then search is done after the middle element of array.

This process is repeated till we get the key element (or) till the search comes to an end, since key element is not in the list.

//PROGRAM TO DEMONSTRATE BINARY SEARCH

```
#include<stdio.h>
#include<conio.h>
Void main()
{
int binary[20],n, i, k, low, mid, high;
clrscr();
printf("enter range of elements");
scanf("%d",&n);
printf("enter elements into array");
for(i=0, i<n; i++)
{
        scanf("%d",&binary[i]);
}
printf("enter search key");
```

```
scanf("%d",&k);
low=0;
high=n-1;
while(low<=high)
{
        mid=(low+high)/2;
        if(binary[mid]<k)
        {
                low=mid+1;
        }
        else if(binary[mid]>k)
        {
                high=mid-1;
        }
        else
        break;
}
if(binary[mid]==k)
{
        printf("element is found at location %d",mid+1);
}
else
printf("element not found");
getch();
}
```

OUTPUT:-

```
Enter the range of elements
5
Enter elements into array
22    28   34    45    58
Enter search key
34
Element found at location 3
```

SORTING

Sorting means arranging the given data in a particular order.
Some of the Sorting techniques are:

i) Bubble sort
ii) Selection sort

BUBBLE SORT

In bubble sort each element is compared with its adjacent element.

If first element is larger than second one, then both elements are swapped. Other wise, element are not swapped.

Consider the following list of numbers.
Example:

74	39	35	97	84
a[0]	a[1]	a[2]	a[3]	a[4]

74 & 39 are compared
Bcoz 74 > 39, both are swapped. Now list is

39	74	35	97	84
a[0]	a[1]	a[2]	a[3]	a[4]

74 &35 are compared
Bcoz 74 > 35, both are swapped, Now list is

39	35	74	97	84
a[0]	a[1]	a[2]	a[3]	a[4]

74 & 97 are compared
Bcoz 74 < 97 list remain unchanged

97 & 84 are compared
Bcoz 97> 84, both are swapped, the list becomes

39	35	74	84	97
a[0]	a[1]	a[2]	a[3]	a[4]

Note: After first pass, largest element in given list occupies the last position.
After second pass, second largest element is placed at second last position and so on..

//PROGRAM TO DEMONSTRATE BUBBLE SORT

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int  i, j, n, bubble[20], temp;
        clrscr();
        printf("enter range of elements");
        scanf("%d",&n);
        printf("enter elements");
        for(i=0, i<n; i++)
        {
                scanf("%d",&bubble[i]);
        }
for(i=0; i<n; i++)
{
        for(j=0;j<n-1;j++)
        {
        if(bubble[j] > bubble[j+1])
        {
        temp=bubble[j];
        bubble[j]=bubble[j+1];
        bubble[j+1]=temp;
        }
        }
}
printf("after sorting");
for(i=0; i<n; i++)
{
        printf("%d",bubble[i]);
}
}
```

OUTPUT

```
enter range of elements
5
Enter elements
3 2 5 4 6
After sorting
2 3 4 5 6
```

SELECTION SORT:

In selection sort, element at first location (0^{th} location) is considered as least element, and it is compared with the other elements of the array. If any element is found to be minimum than the element in first location, then that location is taken as minimum and element in that location will be the minimum element.

After completing a set of comparisions, the minimum element is swapped with the element in first location (0^{th} location).

Then again element second location is considered as minimum and it is compared with the other elements of array and the process continues till the array is sorted in ascending order.

Note: After first pass, smallest element in given list occupies the first position.
After second pass, second largest element is placed at second position and so on..

//PROGRAM TO DEMONSTRATE SELECTION SORT

```
#include<stdio.h>
#include<conio.h>
void main()
{       int i, j, n, a[100], t, min;
        clrscr();
        printf("enter range of elements");
        scanf("%d",&n);
        printf("enter elements:");
        for(i=0, i<n; i++)
        {
                scanf("%d",&a[i]);
        }
        printf("elements before sorting");
        for(i=0; i<n; i++)
        {
                printf("%d",a[i]);
        }
        for(i=0; i<n-1; i++)
        {
                min=i;
                for(j=i+1; j<n; j++)
                {
                        if(a[j]<a[min])
                        min=j;
                }
                if(min!=i)
                {
                        t=a[i];
                        a[i]=a[min];
                        a[min]=t;
                }
```

```
        }
                printf("elements after sorting are");
                for(i=0; i<n; i++)
                {
                        printf("%d", a[i]);
                }
        getch();
}
```

OUTPUT

```
enter range of elements
5
Enter elements
3 2 5 4 6
After sorting   2 3 4 5 6
```