Array

Array is a linear collection of similar type of data items. The array is called to as static structure because once an array is created its size cann't be changed. Although, we create a long array and manipulate only some elements assuming that rest of the array is vacant.

The array is static because of its memory mapping. The elements of an array are stored in contiguous memory locations. Therefore the adderes binding for an array is comple-ted at the compile time. Due to contiguous nature, the address of elements can be mathematically calculated if we know the address of any element. when an array is allocated in the memory the address of its first element is stored by the language to keep track. This address is called as base address of the array and by manipul-ting this base address the address of rest element are

traced. In 'C' language the base address of an array is stored by creating a constant pointer with the same name of the array. There are two types of array -

(i) Single dimensional array
(ii) Multi dimensional array

1) Single dimensional array -

A single dimensional array is a linear collection of data items, those are physically co-related. Each linear array has its own lower bound and upper bound. The value of lower bound and upper bound depends upon the implementation of language. Sometimes the lower bound is zero, sometimes it is one and sometimes it is specified by the programmer.

for example - In C and C++ it is zero.
In FORTRAN (Formula Translation) it is one
In PASCAL, it is specified

by the user.
The address of A[k] can be calculated by following the given formula.

(a) In general.
Add. of A[k] = base * w(k-lb)

(b) In C
Loc A[k] = base + w * k

Operation on single dimensional arra

⇒ **Creation of the array** - As any other variable array are also variables except that they occupy more than one memory location all of them sharing the same name. Therefore, we must have to create an array before we use it. The array can be created either by just declarin it or by defining it.

Declaration of array follow the given syntax-

Datatype arrayname [size];

or

Arrayname [size] as datatype

Example - In 'C'

```
int A[100];
float FA[100];
```

To prevent the array elements to be occupied with garbage value (in case of C), we can define them by providing explicit initial values.

Syntax -

Datatype Array[size] = list of values;

Ex - In 'C'

```
int A[5] = {1,2,3,4,5};
float FA[] = {1.1, 2.2, 3.3};
char C[20] = {'I', 'N', 'D', 'I', 'A'};
long LA[100] = {0};
```

⇒ Traversal of the array

Once the array is created we can traverse its elements either to read their value or to write values to them. Traversal means accessing more than one element of the array consecutively. The traversal can be done by using determinate loop and subscript.

-bing the counter to refer the elements one by one

⇒ **Inserting a data item in an array :-**

Insertion means adding a new element at a particular position. The insert process with-in an array is performed by pushing down the items from the required position. This work is done by starting from the last element and assigning the value to the next element until the given position is arrived. In the insert algorithm we should consider that array is a static structure. Therefore, no more ~~so~~ rooms can be made by insertion and if ~~we~~ the array is filled upto its full capacity. The last item will be dropped after insertion. The insert operation follows the given algorith -

Step 1: Start function Insert (A, I, P, N)

Step 2: Declare counter C

Step 3: for C = N-1 To P-1 ~~se~~ step-1

step 4: let A[c+1] = A[c]

step 5: End of for

step 6: let A[G] = I

step 7: End of function.

Deletion from an array :-

Since, array is a static structure no element can be physically deleted from an array. When we talk about deletion from an array, we actually say to delete a valid value from the array ~~and then delete a valid value from the array~~ and then to traverse the array one element less than the previous traversal. To delete an element at a position we start from the position given pull up the value untill the last value is move upward.

Deletion follows the given algorithm -

Step 1 : Start Delete (A[], pos, n)

Step 2 : Declare variable c

Step 3 : for (= pos-1 To n-2)

Step 4 : Let A[c] = A[c+1]

Step 5 : End for

Step 6 : End of function

⇒ Sorting of Array :—

Algorithm

Step 1 : start Bubble sort (A[], n)

Step 2 : Declare variable i, j, iv

Step 3 : for i = 1 to n-1

Step 4 : for j = 0 to n-2

Step 5 : if A[j] > A[j+1] then

(A) Let iv = A[j]

(B) Let A[j] = A[j+1]

(C) Let A[j+1] = iv

step 6 : End of if
step 7 : End of step (4) for
step 8 : End of step (3) for.
step 9 : End of function

* Bubble sorting :-

Bubble sorting is comparison between two values and interchanging them by one by one.

⇒ Flag sort :-

When the bubble sort is used with a large size of array. It is detected that after executing some iteration of the outer loop. The array becomes sorted and after then the repetition of the outer loop and with in which the repetition of inner loop is done unnecessarily. That is a misuse of programming processing time and reduces the time efficiency of

algorithm.

To eliminate the drawbacks of bubble sort we use a flag variable that is assigned with zero before starting of the inner loop and increased after every interchange of the elements. This flag is also checked to be greater than zero for the continution of the outer loop this called the flag sorting.

⇒ **Searching a value with in an array :-**

Sometimes we may need to search a value that either it is present in an array or not. If present in which index or on which position it is present. There are two algorithm to perform searching named as linear search and binary search.

① **Linear search :-** The linear search operation is based on the concept that starting from the zeroth element compare each element value with the given value and retu

with the index. If there is a match otherwise proceed with the next element until the last element value is checked. The main drawback of the linear search is that if the given value is not present in the array then we must have to check every element. It will be less efficiently particularly in the case when array is stored. The linear search follows the given algorithm :-

Step 1: Start Linear search (A[], n, v)

Step 2: Declare variable c

Step 3: for c = 0 to n - 1

Step 4: If A[c] = v Then
(A) Return with c

Step 5: End of if

Step 6: End of for

Step 7: Return with -1

② Binary search :- for a sorted array the linear search causes overhead in term of processing time. If In the case if the search value is not present in the array or present at the end of the array. Because where this value called be present can not be predected for a sorted array. To save this extra overhead we can use binary search technique.

The binary search depends upon the divides and conquer concept. In this searching we start the comparison from the middle pocket of the array which divides the array into two sub-array. If the value is found in the middle pocket we return its index otherwise by comparing the relationship between search value and middle value. We decide that either the searching will be continued in the upper sub array or into the lower subarray.

for that subarray the what steps are repeat again or we found the value array is exhausted.

Algorithm of Binary search

Step 1: Start Binsearch (A[], n, v)

Step 2: Declare varible lb, lb = 0, ub = n-

Step 3: let m = (lb + ub) / 2;

Step 4: while (lb <= ub)

Step 5: If A[m] = v Then

(A) return with m

Step 6: End of if

Step 7: If A[m] < v Then

(A) ub = m - 1

Step 8: Else

(A) lb = m + 1

Step 9: End if

Step 10: End of while

Step 11: Return with -1