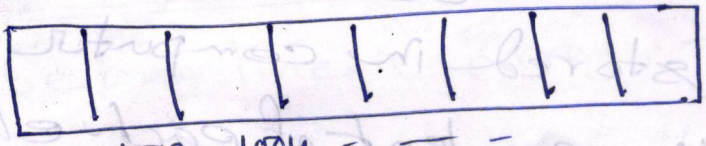
Array

Def:→ Array is a finite collection of same type of elements stored in adjacent memory locations. By finite we mean that there are specific number of elements in an array & similar implies that all the elements in an array are of the same type.

Ex- int a[50]

Representation of Linear array in memory:→ The memory of computer is simply a sequence of addressed locations as shown in the given figure.



Let arr be a linear array stored in the memory of computer.

∴ Addr(arr[k]) = address of kth element in the array arr.

All the elements of the array 'arr' are stored in the consecutive memory cells, the computer does not need to keep track of the address of every element of the array. It only need to keep track of the address of the 1st element which denoted by Base(arr).

(2)

It is called the base address of the array arr. Using the base address, the compute calculates the address of any element of the array by using the following formula.

Addr(arr[k]) = Base(arr) + w(k - lower boun

where w is the ~~size of~~ width of the datatyp of the array.

Ex→ int arr[20];

Base(arr) = 1000 & lb = 0

Addr(arr[5]) = 1000 + 2(5-0) = 1010.

Traversal:→ Let A be in collection of data elements stored in computer memory. To print the content of each element of A or count the number of elements of A with a given property, each element of A will have to be accessed or processed at least once This is called traversing.

Insertion & Deletion:→ If an element is to inserted at the end of the array, then the task is easily done, provided there is enough space to accommodate additional elements in the array.

But if we need to insert an element in the middle of an array, then half of the elements must be moved downwards to new locations to accommodate the new element & retain the order of other elements.

Similarly, deleting the elements from the end of an array is not a problem, but deleting from the middle requires movement to each element upwards in order to fill up the void in the array.

Algorithm of Insertion:→

Let A be a linear array, the function is Insert (A, N, K, ITEM), N is the number of items, K is the positive integer s.t. $K \leq N$. The following algorithm inserts an element ITEM into the Kth position of array A

Insert (A, K, N, ITEM):→

S1:→ Set J = N

S2:→ Repeat steps 3 & 4 while $J \geq K$

S3 Move Jth element downward
Set A[J+1] = A[J]

S4:→ Set J = J-1

S5:→ Set A[K] = ITEM

S6:→ N = N+1

S7:→ Exit.

(4) Delete(A, N, K, ITEM)

S1:→ Set ITEM=A[K]

S2:→ Repeat For J=K to N-1

set A[J]=A[J+1]

end of loop

S3:→ N=N-1

S4:→ Exit.

Linear Search:→

LSearch(arr, ITEM, LOC, N)

S1:→ Set flag=0

S2:→ For I=0 to N-1

S3:→ IF (ITEM= arr[I])

LOC=I+1, Flag=1

break

end of S2 loop

S4:→ If Flag=0 then

print "ITEM" not found.

S5:→ Else

Print "ITEM" found, location is LOC

S6:→ Exit.

Binary Search:→ Binary search technique is very fast & efficient. It requires the list of elements to be in sorted order.

In this method, to search an element we compare it with the element present at the center of the list. If it matches then the search is successful, otherwise, the list is divided into 2 halves — one from the 0th element to the (mid-1) element (first half) another from (mid+1) element to the last element (2nd half). As a result all the elements in the 1st half are smaller than middle element where as all the elements in 2nd half are grater than the middle element.

The searching will now proceed in either of the 2 halves depending upon whether the target element is grater or smaller than the middle element. If the element is smaller than the middle element then the searching is done in the 1st half, otherwise it is done in the 2nd half.

The process of comparing the required element with the middle element & if not found then dividing the elements into 2 halves is repeated till the element is found or the division of half parts gives 1 element.

(6) Algorithm:

BINARY (A, LB, UB, ITEM, LOC)

S1:→ Set BEG=LB, END=UB & MID=INT((BEG+END)/2)

S2:→ Repeat S3 & S4 while BEG≤END &
A[MID] ≠ ITEM

S3:→ IF ITEM < A[MID] then
Set END=MID-1
ELSE
Set BEG=MID+1
end of If

S4:→ Set MID=INT((BEG+END)/2)
end of S2 Loop

S5:→ IF A[MID] = ITEM then
Set LOC=MID
ELSE
Set LOC=NULL
end of If

S6:→ Exit

Ex:→ Let us take an array arr that consists of 10 sorted numbers & 46 is the element that is to be searched.

	0	1	2	3	4	5	6	7	8	9
arr →	0	1	2	9	10	11	15	20	46	72
	Beg=0									End=9

Complexity of Binary Search → The complexity is measured by the number $f(n)$ of comparisons to locate ITEM in A where A contains n elements. Observe that each comparison reduces the sample to locate ITEM where

$$2^{f(n)} < n$$

$$\text{i.e, } f(n) < \frac{\log n}{\log 2}$$

$$\Rightarrow f(n) < \log_2 n$$

That is, the running time for the worst case approximately equal to $\log_2 n$. One can also show that the running time for average case is approximately equal to the running time for the worst case.

Two Dimensional Array → The 2-dimensional array is a collection of elements placed in m rows & n columns. There are 2 subscripts in the syntax of 2D array in which one specifies the no. of rows & the other specifies no. of columns. In 2-D array each element is itself an array.

$$\begin{bmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \end{bmatrix}$$

(8)

Representation of 2-D array in Memory :- Let A be a 2-D m x n array. Although A is pictured as a rectangular array of elements with m rows & n columns, the array will be represented in memory by a block of m.n sequential memory locations. The 2 ways in which elements can be represented in computer's memory are — Row-major representation & the column-major representation.

Row-major Representation :- In row major representation the 1st row of the array occupies the 1st set of memory location, 2nd occupies the next set & so on. The diagramatic representation of 2-D array arr[2][3] in row-major order is given below.

arr	[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	[1][2]

Memory locations are 1st occupied by the 1st row of the array.

Now Base(arr) has the address of the 1st element of the array arr[0][0].

The formula to calculate the (i, j)th element of 2-D array of mxn dimension is

~~Addr(a[i][j]) = Base Address + w[n(i-1)+(j-1)]~~

Addr(a[i][j]) = Base addr + w[n(i-1) + (j-1)]

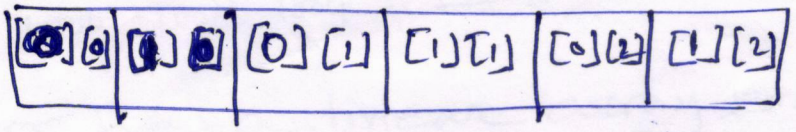
Addr[a[i][j]] = BaseAddr + w[n(i-lr) + (j-lc)]

where lr → lower limit of row
lc → lower limit of column
w → width of datatype.

Ex: Let Baseaddr = 100, Here w = 2
Find the address of a[2][3] using row-major formula where m = 3 & n = 4.

Addr[a[2][3]] = 100 + 2[4(2-0) + (3-0)]
= 100 + 2[8+3]
= 122

Column-Major Representation :→ Similarly, for the column-major order representation, consider the same matrix



To find the address of (i,j) the element in an array mxn dimension of w width of each element, the following formula can be used.

Addr(a[i][j]) = BaseAddr + w[m(j-lc) + (i-lr)]

Ex:→
Addr[a[2][3]) = 100 + 2[3[3-0] + (2-0)]
= 100 + 2[9+2]
= 122