ARRAYS

Arrays: An array is a finite collection of similar elements stored in adjacent memory locations

→ An array with n number of elements is referenced using an index that ranges from 0 to n-1.

→ The lowest index of an array is called the lower bound (LB)

→ The highest index is called the upper bound (UB).

→ The no. of elements in an array is called its range.

∴ LB = 0

UB = n-1

size of an array = UB - LB + 1

= n-1-0+1

= n

Array declaration in C

```
int age[20];
```

data type — Name of the array — Size of the array

Array initialization

Eg

```
int age[5] = {2, 10, 5, 15, 20};
char name[5] = "Teena";
```

or

```
char name[] = {'T', 'e', 'e', 'n', 'a'};
```

(optional)

Reading the values of an array from keyboard

Eg:-

```
float f[5];
```

Both are referencing to the address of first element of an array

```
scanf("%d", &f[0]); // is equivalent to
scanf("%f", f);
```

Representation of Linear arrays in Memory

The memory of the computer is simply a sequence of addressed locations as shown in fig.

As the elements of the array arr are stored in consecutive memory cells, the computer does not need to keep track of the address of the first element

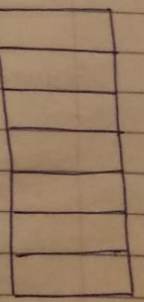


Fig: Computer Memory

2018/8/31 15:37

of the array. It only needs to keep track of the address of the first element of the array which is denoted by

Base(arr) ← Base add. of arr.

Formula for calculating the address of the k^{th} element in arr

Add(arr[k]) = Base(arr) + w(k - LB)

where w → size of the data type of the array arr.

For eg:

```
int arr[5]; // int takes 4 bytes of memory
```

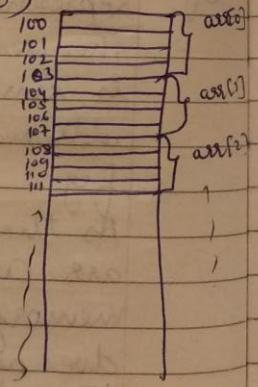
Assume Base(arr) = 100

Then calculate the address of the third element of the array

Add(arr[2]) = 100 + 4(2 - 0)

= 100 + 8

= 108



Array Operations

① Traversal

Let A be the collection of data elements stored in the computer's memory. To display or count the no. of elements of A with a given property, each element of A will have to be accessed at least once. This is called traversing.

Algorithm for Array Traversal

A : Linear array

LB, UB : Lower bound and Upper Bound of A

PROCESS : Operation to be applied on each element of A while traversing.

STEP 1 : [Initialization]
Set counter := LB

STEP 2 : Repeat STEP 3 and 4
while counter ≤ UB

STEP 3 : [Visit element]
Apply PROCESS to A[counter]

STEP 4 : [Increase counter]
Set counter = counter + 1

STEP 5 : [Finished]
Exit

A_TRAV.C

2018/8/31 15:37

NOTE: If an element is to be inserted at the end of the array, then there is enough space to accommodate additional elements in the array. But if an element is to be inserted in the middle of an array then half of the elements must be moved downwards to accommodate the new element & retain the order of the other elements.

INSERTION

Insertion refers to the operation of adding another element to the array.

Algorithm: INSERT (A, N, K, ITEM)
A: Linear Array
N: No of items
K: +ve integer, such that $K \leq N$
ITEM: element i.e. to be inserted at K^{th} position of array A.

STEP 1: [Initialize Counter]
Set J := N
STEP 2: Repeat Steps 3 and 4
while J >= K
STEP 3: [Move J^{th} element downward]
Set A[J+1] := A[J]
STEP 4: [Decrease Counter]
Set J := J - 1
End of Step 2 loop
STEP 5: [Insert element]
Set A[K] := ITEM
STEP 6: [Reset N]
Set N := N + 1
STEP 7: [FINISHED]
Exit

A_INSERT.C

DELETION

Deletion refers to the operation of removing an element from the array.
Let A be a linear Array
Function used to delete from the array is DELETE (A, N, K, ITEM)

Algorithm: DELETE (A, N, K, ITEM)
A: Linear array
N: No. of items
K: +ve integer, such that $K \leq N$
ITEM: element i.e. to be deleted from K^{th} position from the array A

STEP 1: SET ITEM := A[K]
STEP 2: Repeat for J = K to N-1
[Move J+1 element upward]
Set A[J] := A[J+1]
End of loop
STEP 3: [Reset the no of elements in A]
Set N := N - 1
STEP 4: [FINISHED]
Exit

A_DELETE.C

2018/8/31 15:37

TWO DIMENSIONAL ARRAYS

It is collection of elements placed in m rows and n columns.
A 2-D array is also called a matrix.
A matrix is said to be of order m×n
where m → rows & n → columns

	col 0	col 1	col 2
ROW 0		a[0][1]	
ROW 1			

Row Major and Column Major Order

All elements of a matrix get stored in the memory in a linear fashion.
There are two ways in which elements can be represented in Computer's memory -
- Row major
- Column major

ROW MAJOR : In this, the first row occupies the first set of memory locations, second occupies the next set and so on.

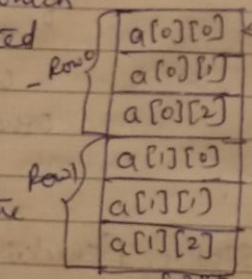


Fig: Representation of 2-D Array in Row Major Order.

Calculate the add. of an element in 2-D array

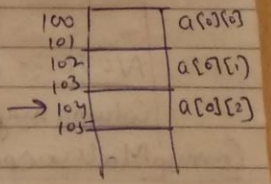
Address of k^{th} element in j^{th} row, with m×n dimensions.

$$A(j,k) = base(A) + W[N(j-1) + (k-1)]$$

where W → word size (i.e. the size of data type)
N → no. of columns.

int a[2][3]; & base(A) = 100, W = 2

Then Address (1,3)
= 100 + 2[3(1-1) + (3-1)]
= 100 + 2(0+2)
= 100 + 4 = 104



Column Major : In this the first column occupies the first set of memory location, second occupies the next set and so on.

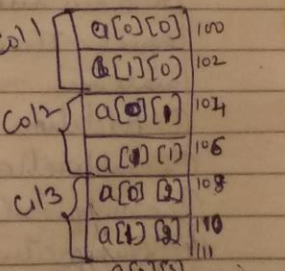


Fig:- Representation of 2-D Array in Column Major order

2018/8/31 15:38

Calculate the add of k^{th} element in j^{th} row in an array of m x n in column major

$$A(j,k) = Base(A) + W[M(k-1) + (j-1)]$$

Where W → word size

M → No. of rows

int a[2][3]; Base(A) = 100, W = 2

Address(1,3) = 100 + 2[3(3-1) + (1-1)]

= 100 + 2[4 + 0]

= 100 + 8

= 108

NOTE :- Row major order is used in C/C++

Column major order is used in Fortran, MATLAB etc.

Common Matrix operations

* When the no. of rows and no. of columns are equal then the matrix is called square matrix

1. Addition of two matrices
2. Subtraction
3. Multiplication
4. Transposition

2018/8/31 15:39