

UNIT 1 (Part -2)

Topics Covered:

Arrays: Definition, Single and Multidimensional Arrays, Representation of Arrays: Row Major Order, and Column Major Order, Application of arrays, Sparse Matrices and their representations.

Arrays

Definition: An array is a sequential collection of elements of same data type and stores data elements in a continuous memory location. The elements of an array are accessed by using an index. The index of an array of size N can range from 0 to N-1.

For example, if your array size is 5, then your index will range from 0 to 4 (5-1). Each element of an array can be accessed by using arr[index].

Consider following array. The size of this array is 5. If you want to access 34, then you can access it by using arr[2] i.e. 12.

| | | | | | |
|-------|---|----|----|---|----|
| Arr | 5 | 23 | 34 | 9 | 28 |
| Index | 0 | 1 | 2 | 3 | 4 |

Array declaration in C/C++

Declaring an array you must specify, the following:

- **Size of the array:** This defines the number of elements that can be stored in the array.
- **Type of array:** This defines the type of each element i.e. number, character, or any other data type.

Example: intarr[5];

Note: This is a static array and the other kind is dynamic array, where type is just enough for declaration. In dynamic arrays, size increases as more elements are added to the array.

Array Initialization:

Array can be initialized either at the time of declaration or after that.

(i) The sample format if an array is initialized at the time of declaration is

Syntax: type arr[size]={elements}

Example: intarr[5]={4,12,7,15,9};

(ii) An array can be initialized after declaration by assigning values to each index of the array as follows:-

Syntax: type arr[size]
arr[index]=12

Example:

```
intarr[5];  
arr[0] = 4;  
arr[1] = 12;
```

Single dimensional Arrays

A one-dimensional array is also called a single dimensional array where the elements will be accessed in sequential order. This type of array will be accessed by the subscript of either a column or row index.

Syntax: *data-type Array-name[size]*

Example: *intx[10];*
char name[20];
float f[5];

Multidimensional Arrays

When the number of dimensions specified is more than one then it is called as a multi-dimensional array.

Two-Dimensional Arrays:

Syntax: *data-type Array-name[size][size]*

A two-dimensional array can be thought of a matrix with row and columns. For example if we declare a 2-D array as: *intarr[3][4];* then this declaration means a 2-D arr of 3 rows and 4 column which can be represented as follows:

| | Col 1 | Col 2 | Col 3 | Col 4 |
|------|-----------|-----------|-----------|-----------|
| Row1 | Arr[0][0] | Arr[0][1] | Arr[0][2] | Arr[0][3] |
| Row2 | Arr[1][0] | Arr[1][1] | Arr[1][2] | Arr[1][3] |
| Row3 | Arr[2][0] | Arr[2][1] | Arr[2][2] | Arr[2][3] |

The first row and column starts at 0. Inter section section of row and column becomes the position of elements of the 2-D array for e.g., first row(0) and second column(1) element will be *arr[0][1]*

Row major order

Row Major Order is a method of representing multi dimension array in sequential memory. In this method elements of an array are arranged sequentially row by row. Thus elements of first row occupies first set of memory locations reserved for the array, elements of second row occupies the next set of memory and so on. Consider a Two Dimensional Array consist of N rows and M columns. It can be stored sequentially in memory row by row as shown below:

| | | | | |
|---------|----------|----------|-------|------------|
| Row 0 | A[0,0] | A[0,1] | | A[0,M-1] |
| Row 1 | A[1,0] | A[1,1] | | A[1,M-1] |
| | | | | |
| Row N-1 | A[N-1,0] | A[N-1,1] | | A[N-1,M-1] |

Example:

Consider following example in which a two dimensional array consist of two rows and four columns is stored sequentially in row major order as:

| | | |
|------|---------|-------|
| 2000 | A[0][0] | Row 0 |
| 2002 | A[0][1] | |
| 2004 | A[0][2] | |
| 2006 | A[0][3] | |
| 2008 | A[1][0] | Row 1 |
| 2010 | A[1][1] | |
| 2012 | A[1][2] | |
| 2014 | A[1][3] | |

The Location of element A[i, j] can be obtained by evaluating expression:

$$\text{LOC (A [i, j])} = \text{Base_Address} + \text{W [M (i) + (j)]}$$

Here,

Base_Address is the address of first element in the array.

W is the word size. It means number of bytes occupied by each element.

N is number of rows in array.

M is number of columns in array.

Suppose we want to calculate the address of element A [1, 2].

It can be calculated as follow:

Here,

Base_Address = 2000, W= 2, M=4, N=2, i=1, j=2

$$\text{LOC (A [i, j])} = \text{Base_Address} + \text{W [M (i) + (j)]}$$

$$\text{LOC (A[1, 2])} = 2000 + 2 * [4*(1) + 2]$$

$$= 2000 + 2 * [4 + 2]$$

$$= 2000 + 2 * 6$$

$$= 2000 + 12$$

$$= 2012$$

Column major order

Column Major Order is a method of representing multi dimension array in sequential memory. In this method elements of an array are arranged sequentially column by column. Thus elements of first column occupies first set of memory locations reserved for the array, elements of second column occupies the next set of memory and so on.

Consider a Two Dimensional Array consist of N rows and M columns. It can be stored sequentially in memory column by column as shown below:

| | | | | |
|------------|----------|----------|--|------------|
| Column 0 | A[0,0] | A[1,0] | | A[N-1,0] |
| Column 1 | A[0,1] | A[1,1] | | A[N-1,1] |
| | | | | |
| Column N-1 | A[0,M-1] | A[1,M-1] | | A[N-1,M-1] |

Example:

Consider following example in which a two dimensional array consist of two rows and four columns is stored sequentially in Column Major Order as:

| | | |
|------|---------|----------|
| 2000 | A[0][0] | Column 0 |
| 2002 | A[1][0] | |
| 2004 | A[0][1] | Column 1 |
| 2006 | A[1][1] | |
| 2008 | A[0][2] | Column 2 |
| 2010 | A[1][2] | |
| 2012 | A[0][3] | Column 3 |
| 2014 | A[1][3] | |

The Location of element A[i, j] can be obtained by evaluating expression:

$$\text{LOC (A [i, j])} = \text{Base_Address} + \text{W [N (j) + (i)]}$$

Here,

Base_Address is the address of first element in the array.

W is the word size. It means number of bytes occupied by each element.

N is number of rows in array.

M is number of columns in array.

Suppose we want to calculate the address of element A [1, 2].

It can be calculated as follow:

Here,

Base_Address = 2000, W= 2, M=4, N=2, i=1, j=2

LOC (A [i, j]) = Base_Address + W [N (j) + (i)]

LOC (A[1, 2]) = 2000 + 2 *[2*(2) + 1]

= 2000 + 2 * [4 + 1]

= 2000 + 2 * 5

= 2000 + 10

= 2010

Derivation of Index Formulae for 1-D, 2-D, 3-D and n-D Array

Address calculation of 1D:

| | | | | | |
|------|------|-----------|--|--|-------|
| A[1] | a[2] | a[3]..... | | | A[11] |
| | | | | | |

Suppose address of a[1] in BA and word size W is 1.then

Address of a[1]=BA

Address of a[2]=BA+1

Address of a[3]=BA+2

So Address of a[i]=BA+(i-1)

Address of a[i]=BA+W(i-1)

Address of a[i]=BA+W*E₁

Where E₁ is effective index of i.(E₁=i - lower bound of array)

Address calculation of 2 D (Row major order):

| | | | | | |
|----|---|---|---|-------|----|
| | 1 | 2 | 3 | | L2 |
| 1 | | | | | |
| 2 | | | | | |
| | | | | | |
| | | | | | |
| L1 | | | | | |

Suppose address of a[1] in BA and word size W is 1.then

Address of a[1][1]=BA

Address of a[1][2]=BA+1

Address of a[1][3]=BA+2

So Address of a[1][L2]=BA+L2-1

Address of a[2][1]= BA+L2-1+1=BA+L2

Address of a[2][2]=BA+L2+1

Address of a[2][3]=BA+L2+2

So Address of $a[2][L2] = BA + L2 + L2 - 1$

So Address of $a[3][1] = BA + L2 + L2 - 1 + 1 = BA + 2 * L2$

So Address of $a[i][1] = BA + (i - 1) * L2$

Address of $a[i][2] = BA + (i - 1) * L2 + 1$

Address of $a[i][3] = BA + (i - 1) * L2 + 2$

Address of $a[i][j] = BA + (i - 1) * L2 + (j - 1)$

Address of $a[i][j] = BA + W(E_1 * L2 + E_2)$

Where E_1 and E_2 are effective index of i and j and $L1$ and $L2$ are length of row and column dimensions. ($E_i = i - \text{lower bound}$)

1-D:

Address of an element of an array say " $A[I]$ " is calculated using the following formula:

$$\text{Address of } A[I] = B + W * (I - LB)$$

Where,

B = Base address

W = Storage Size of one element stored in the array (in byte)

I = Subscript of element whose address is to be found

LB = Lower limit / Lower Bound of subscript, if not specified assume 0 (zero)

Example:

Given the base address of an array **B[1300.....1900]** as 1020 and size of each element is 2 bytes in the memory. Find the address of **B[1700]**.

Solution:

The given values are: $B = 1020$, $LB = 1300$, $W = 2$, $I = 1700$

$$\text{Address of } A[I] = B + W * (I - LB)$$

$$= 1020 + 2 * (1700 - 1300)$$

$$= 1020 + 2 * 400$$

$$= 1020 + 800$$

$$= 1820$$

2-D:

A two dimensional Array A is the collection of ' $m \times n$ ' elements. Programming language stores the two dimensional array in one dimensional memory in either of two ways -

1) Row Major Order:

First row of the array occupies the first set of memory locations reserved for the array; Second row occupies the next set, and so forth.

To determine element address $A[i,j]$:

$$\text{Location } (A[i,j]) = \text{Base Address} + (N * (I - 1)) + (j - 1)$$

For example :

Given an array $[1 \dots 5, 1 \dots 7]$ of integers. Calculate address of element $T[4,6]$, where $BA=900$.

Solution:- $I = 4$, $J = 6$, $M = 5$, $N = 7$

$$\text{Location } (T[4,6]) = BA + (7 * (4 - 1)) + (6 - 1)$$

$$= 900 + (7 * 3) + 5$$

$$= 900 + 21 + 5$$

$$= 926$$

2) Column Major Order:

Order elements of first column stored linearly and then comes elements of next column.

To determine element address $A[i,j]$:

$$\text{Location} (A[i, j]) = \text{Base Address} + (M \times (j - 1)) + (i - 1)$$

For example :

Given an array $[1 \dots 6, 1 \dots 8]$ of integers. Calculate address element $T[5,7]$, where $BA=300$.

Solution:- $I = 5$, $J = 7$, $M = 6$, $N = 8$

$$\begin{aligned}\text{Location} (T [4,6]) &= BA + (6 \times (7-1)) + (5-1) \\ &= 300 + (6 \times 6) + 4 \\ &= 300 + 36 + 4 \\ &= 340\end{aligned}$$

3-D:

In three - dimensional array also address is calculated through two methods i.e; row-major order and column-major method.

To calculate address of element $X[i,j,k]$ using row-major order :

$$\text{Location} (X[i,j,k]) = BA + MN(k-1) + N(i-1) + (j-1)$$

To calculate address of element $X[i,j,k]$ using column-major order

$$\text{Location} (X[i,j,k]) = BA + MN(k-1) + M(j-1) + (i-1)$$

For example :

Given an array $[1..8, 1..5, 1..7]$ of integers. Calculate address of element $A[5,3,6]$, by using rows and columns methods, if $BA=900$?

Solution:- The dimensions of A are :

$$M=8, N=5, R=7, i=5, j=3, k=6$$

Rows - wise :

$$\text{Location} (A[i,j,k]) = BA + MN(k-1) + N(i-1) + (j-1)$$

$$\begin{aligned}\text{Location} (A[5,3,6]) &= 900 + 8 \times 5(6-1) + 5(5-1) + (3-1) \\ &= 900 + 40 \times 5 + 5 \times 4 + 2 \\ &= 900 + 200 + 20 + 2 \\ &= 1122\end{aligned}$$

Columns - wise :

$$\text{Location} (A[i,j,k]) = BA + MN(k-1) + M(j-1) + (i-1)$$

$$\begin{aligned}\text{Location} (A[5,3,6]) &= 900 + 8 \times 5(6-1) + 8(3-1) + (5-1) \\ &= 900 + 40 \times 5 + 8 \times 2 + 4 \\ &= 900 + 200 + 16 + 4 \\ &= 1120\end{aligned}$$

n-D:

Let C be an n -dimensional array.

The length $L(i)$ of dimension i of C can be calculated by,

$$L(i) = \text{upper bound} - \text{lower bound} + 1$$

For a given subscript $K(i)$, the effective index $E(i)$ of $L(i)$ can be calculated from,

$$E(i) = K(i) - \text{lower bound}$$

Then address $C[K(1), K(2), \dots, K(n)]$ of element of C is,

$\text{Base}(C) + w[(((E(n)L(n-1) + E(n-1))L(n-2)) + \dots + E(3))L(2) + E(2))L(1) + E(1)]$, when C is stored in column major, and

$\text{Base}(C) + w[(\dots((E(1)L(2) + E(2))L(3) + E(3))L(4) + \dots + E(n-1))L(n) + E(n)]$, when C is stored in row major order.

$\text{Base}(C)$ denotes the address of 1st element of C and w denotes the number of words per memory location.

Sparse matrix:

What is Sparse Matrix?

In computer programming, a matrix can be defined with a 2-dimensional array. Any array with 'm' columns and 'n' rows represent a m X n matrix. There may be a situation in which a matrix contains more number of ZERO values than NON-ZERO values. Such matrix is known as sparse matrix.

Sparse matrix is a matrix which contains very few non-zero elements.

When a sparse matrix is represented with a 2-dimensional array, we waste a lot of space to represent that matrix. For example, consider a matrix of size 100 X 100 containing only 10 non-zero elements. In this matrix, only 10 spaces are filled with non-zero values and remaining spaces of the matrix are filled with zero. That means, totally we allocate $100 \times 100 \times 2 = 20000$ bytes of space to store this integer matrix. And to access these 10 non-zero elements we have to make scanning for 10000 times. To make it simple we use the following sparse matrix representation.

Sparse Matrix Representations

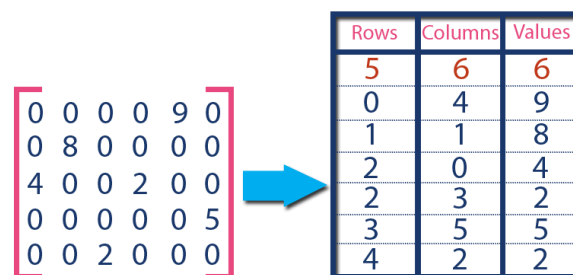
A sparse matrix can be represented by using TWO representations, those are as follows...

1. Triplet Representation (Array Representation)
2. Linked Representation

Method 1: Triplet Representation (Array Representation)

In this representation, we consider only non-zero values along with their row and column index values. In this representation, the 0th row stores the total number of rows, total number of columns and the total number of non-zero values in the sparse matrix.

For example, consider a matrix of size 5 X 6 containing 6 number of non-zero values. This matrix can be represented as shown in the image...



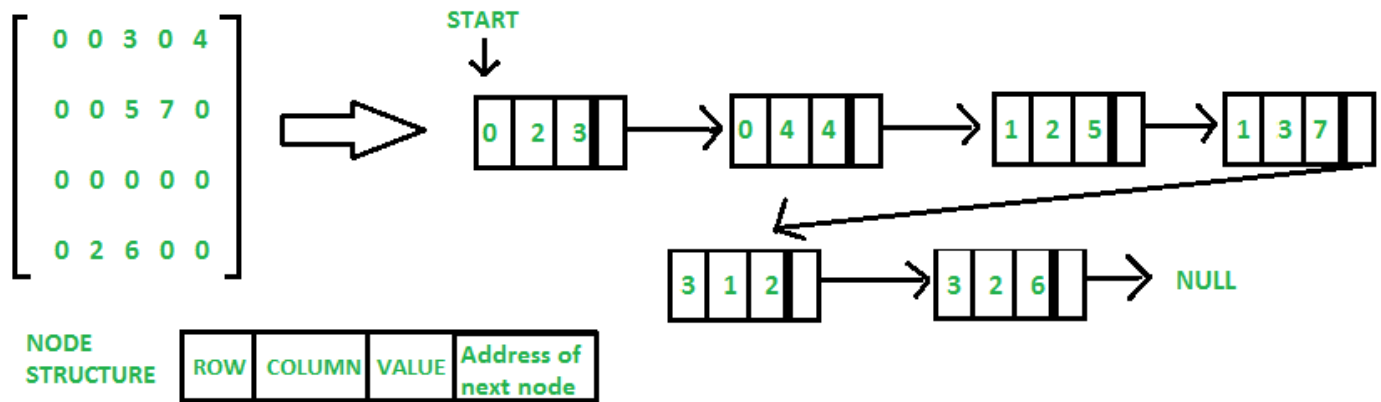
| Rows | Columns | Values |
|------|---------|--------|
| 5 | 6 | 6 |
| 0 | 4 | 9 |
| 1 | 1 | 8 |
| 2 | 0 | 4 |
| 2 | 3 | 2 |
| 3 | 5 | 5 |
| 4 | 2 | 2 |

In above example matrix, there are only 6 non-zero elements (those are 9, 8, 4, 2, 5 & 2) and matrix size is 5 X 6. We represent this matrix as shown in the above image. Here the first row in the right side table is filled with values 5, 6 & 6 which indicate that it is a sparse matrix with 5 rows, 6 columns & 6 non-zero values. The second row is filled with 0, 4, & 9 which indicate the non-zero value 9 is at the 0th-row 4th column in the sparse matrix. In the same way, the remaining non-zero values also follow a similar pattern

Method 2: Using Linked Lists

In linked list, each node has four fields. These four fields are defined as:

- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non zero element located at index – (row,column)
- **Next node:** Address of the next node



Why to use Sparse Matrix instead of simple matrix?

Storage: There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those non-zero elements.

Computing time: Computing time can be saved by logically designing a data structure traversing only non-zero elements..

Operations of Sparse Matrix

- Add
- Transpose and
- Multiply

Given two sparse matrices, perform the operations such as add, multiply or transpose of the matrices in their sparse form itself.

The result should consist of three sparse matrices, one obtained by adding the two input matrices, one by multiplying the two matrices and one obtained by transpose of the first matrix.

Example: Note that other entries of matrices will be zero as matrices are sparse.

Operations on Sparse Matrices

| Row | Column | Value | Row | Column | Value | Row | Column | Value | Result of Addition |
|--------------------------|--------|-------|-----------------|--------|-------|-----|--------|-------|---------------------|
| 1 | 2 | 10 | 1 | 3 | 8 | 1 | 2 | 10 | |
| 1 | 4 | 12 | 2 | 4 | 23 | 1 | 3 | 8 | |
| 3 | 3 | 5 | 3 | 3 | 9 | 1 | 4 | 12 | |
| 4 | 1 | 15 | 4 | 1 | 20 | 2 | 4 | 23 | |
| 4 | 2 | 12 | 4 | 2 | 25 | 3 | 3 | 14 | |
| | | | | | | 4 | 1 | 35 | |
| | | | | | | 4 | 2 | 37 | |
| Input : Matrix 1: (4x4) | | | Matrix 2: (4x4) | | | | | | |
| Result of Multiplication | | | Row | Column | Value | Row | Column | Value | Result of Transpose |
| | | | 1 | 1 | 240 | 1 | 4 | 15 | |
| | | | 1 | 2 | 300 | 2 | 1 | 10 | |
| | | | 1 | 4 | 230 | 2 | 4 | 12 | |
| | | | 3 | 3 | 45 | 3 | 3 | 5 | |
| | | | 4 | 3 | 120 | 4 | 1 | 12 | |
| | | | 4 | 4 | 276 | | | | |

Applications of Arrays in C

In c programming language, arrays are used in wide range of applications. Few of them are as follows...

1. Arrays are used to Store List of values

In c programming language, single dimensional arrays are used to store list of values of same datatype. In other words, single dimensional arrays are used to store a row of values. In single dimensional array data is stored in linear form.

2. Arrays are used to Perform Matrix Operations

We use two dimensional arrays to create matrix. We can perform various operations on matrices using two dimensional arrays.

3. Arrays are used to implement Search Algorithms

We use single dimensional arrays to implement search algorithmslike...

Linear Search

Binary Search

4. Arrays are used to implement Sorting Algorithms

We use single dimensional arrays to implement sorting algorithmslike...

Insertion Sort

Bubble Sort

Selection Sort

Quick Sort

Merge Sort, etc.,

5. Arrays are used to implement Data structures

We use single dimensional arrays to implement data structures like...

Stack Using Arrays

Queue Using Arrays

6. Arrays are also used to implement CPU Scheduling Algorithms