Arrays

What is an Array?

- Array is the collection of similar data types or collection of similar entity stored in contiguous memory location.
- Each data item of an array is called an element.
- Each element is unique and located in separated memory location.
- Each of elements of an array share a variable but each element having different index no. known as subscript.

Declaration of an Array

Syntax: datatype arrayname[size];

Example:

```
int arr[100];
int marks[100];
int a[5]={10,20,30,100,5}
```

The declaration of an array tells the compiler that, the data type, name of the array, size of the array and for each element it occupies memory space.

Initialization of an Array

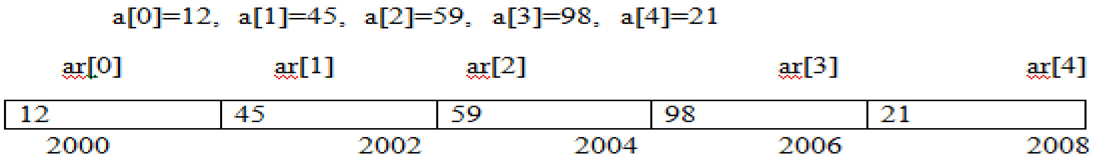
Syntax: data type arrayname [size] = {value1, value2, value3…};

Example: int ar[5]={20,60,90, 100,120};

- Array subscript always start from zero which is known as lower bound and upper value is known as upper bound and the last subscript value is one less than the size of array.
- Total size in byte for 1D array is:
  - Total bytes=sizeof (data type) * sizeof array.
  - Example : if an array declared is: int [20];
    - Total byte= 2 * 20 =40 byte.

Accessing and Modifying Array Elements

The array elements are 12 45 59 98 21



Suppose if we modify 1st array element as 125,
a[0]=125;

Array elements after modification:

a[0]	a[1]	a[2]	a[3]	a[4]
125	45	59	98	21

Example program to display Array Elements

```
#include<stdio.h>
int main()  {
int arr[5],i;
for(i=0;i<5;i++)  {
printf(“enter a value for arr[%d] \n”,i);
scanf(“%d”,&arr[i]);
}
 printf(“the array elements are: \n”);
for (i=0;i<5;i++)  {
printf(“%d\t”,arr[i]);
}
return 0;
}
```

Output

Enter a value for arr[0] = 12

Enter a value for arr[1] =45

Enter a value for arr[2] =59

Enter a value for arr[3] =98

Enter a value for arr[4] =21

The array elements are 12 45 59 98 21

2-D Array

- Two dimensional array is known as matrix.
- The array declaration in two dimensional array use two subscripts.
- Its syntax is data-type array name[row][column];
- Total no. of elements in 2-D array is calculated as **row*column**

2-D Array Initialization

- int mat[4][3]={11,12,13,14,15,16,17,18,19,20,21,22};

- int mat[4][3]={{11,12,13},{14,15,16},{17,18,19},{20,21,22}};

2-D Array Example

int a[2][3];

Total no of elements=row*column is 2*3 =6

It means the matrix consist of 2 rows and 3 columns For example:-

20	2	7
8	3	15

Positions of 2-D array elements in an array are as below

00	01	02
10	11	12
a [0][0]	a [0][1]	a [0][2]
a [1][0]	a [1][1]	a[1][2]

20	2	7	8	3	15
2000	2002	2004	2006	2008	2010

•

Accessing 2-D Array

For processing 2-d array, we use two nested for loops.

The outer for loop corresponds to the row and the

Inner for loop corresponds to the column.

For example int a[4][5];

For reading value:-

```
for(i=0;i<4;i++) {
   for(j=0;j<5;j++) {
     scanf("%d",&a[i][j]);
    }
  }
```

For displaying value:-

```
for(i=0;i<4;i++) {
   for(j=0;j<5;j++) {
    printf("%d",a[i][j]);
    }
  }
```

Array Advantages and Disadvantages

- Efficient and fast access
- Memory Efficiency
- Versatility
- Compatibility with hardware

- Fixed size
- Memory allocation issues
- Insertion and deletion
- Lack of flexibility

Array Applications

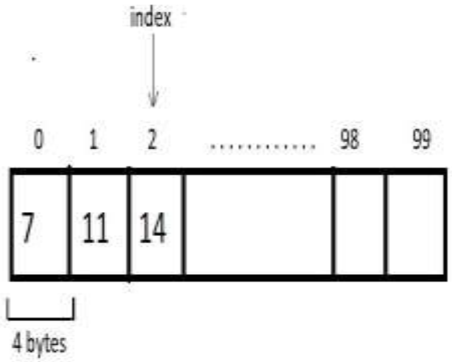
- Storing and accessing data
- Searching
- Matrices
- Dynamic Programming
- Implementing other data structures
- Data Buffers

Array Operations

- Traversal
- Insertion
- Deletion
- Search
- Sorting

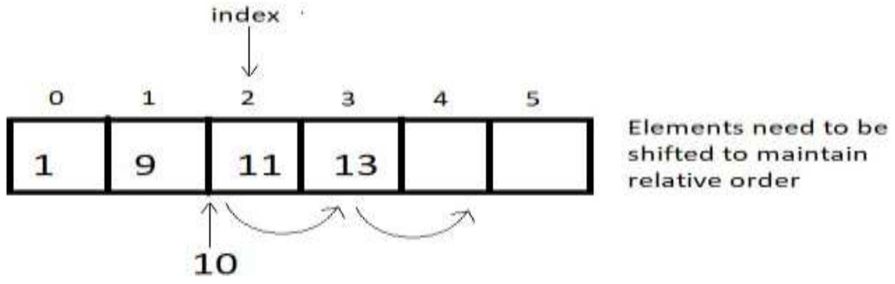
Array Operations - Traversal

- Visiting every element of an array once is known as **traversing** the array.
- **Why Traversal?**
  - For use cases like:
    - Storing all elements – Using scanf()
    - Printing all elements – Using printf()
    - Updating elements.
- Example



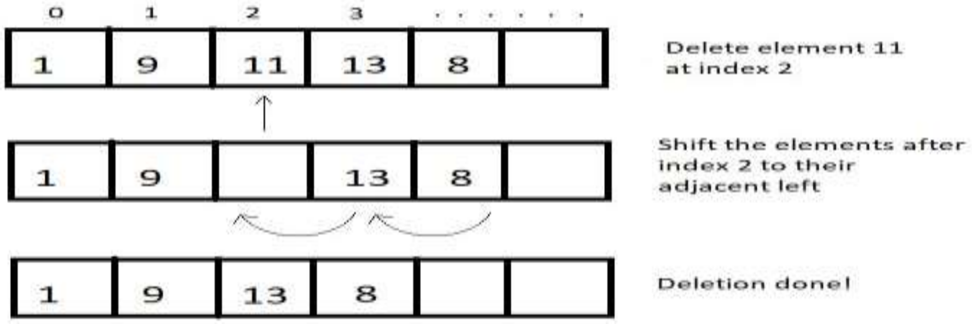
Array Operations - Insertion

- An element can be inserted in an array at a specific position.
- Suppose we want to add an element 10 at index 2 in the below-illustrated array, then the elements after index 1 must get shifted to their adjacent right to make way for a new element.
- Example



Array Operations - Deletion

- An element at a specified position can be deleted, creating a void that needs to be fixed by shifting all the elements to their adjacent left.
- Example



Address of any element in 1-D Array

INSTITUTIONS
ANDHRA PRADESH, INDIA

- Address of A[i]= B.A + (i-LB)*W
- Where B.A = Base Address

  i = index

  LB = Starting Index

  W = Size of type

Ex : A[1300—1900]

BA=1020, W=2

Find A[1700] address

Sol: Address of A[1700]

= 1020+(1700-1300)*2

= 1020+400*2

= 1020+800

= 1820

Address of any element in 2-D Array

- When it comes to organizing and accessing elements in a multi-dimensional array, two prevalent methods are **Row Major Order** and **Column Major Order**.
- These approaches define how elements are stored in memory and impact the efficiency of data access in computing.

Aspect	Row Major Order	Column Major Order
Memory Organization	Elements are stored row by row in contiguous locations.	Elements are stored column by column in contiguous locations.
Memory Layout Example	For a 2D array A[m][n]: [A[0][0], A[0][1], ..., A[m-1][n-1]]	For the same array: [A[0][0], A[1][0], ..., A[m-1][n-1]]
Traversal Direction	Moves through the entire row before progressing to the next row.	Moves through the entire column before progressing to the next column.
Access Efficiency	Efficient for row-wise access, less efficient for column-wise access.	Efficient for column-wise access, less efficient for row-wise access.
Common Use Cases	Commonly used in languages like C and C++.	Commonly used in languages like Fortran.
Applications	Suitable for row-wise operations, e.g., image processing.	Suitable for column-wise operations, e.g., matrix multiplication.

Row Major Order

- Address of A[i][j]= B.A + ((i-LR)*C + (j-LC))*W
- Where B.A = Base Address

i = row index
j = column index
C = no.of columns
LR = Starting Row Index
LC = Starting Column Index
W = Size of type

Ex: arr[1- - 10][1- - 15], BA=100, W=1
then find address of arr[8][6]?

Sol : arr[8][6] = 100+((8-1)*15 + (6-1))*1
= 100 + (7 *15 + 5) *1
= 100+(110)*1
= 210

Column Major Order

- Address of A[i][j]= B.A + ((j-LC)*R + (i-LR))*W
- Where B.A = Base Address

i = row index

j = column index

R = no.of rows

LR = Starting Row Index

LC = Starting Column Index

W = Size of type

Ex: arr[1- - 10][1- - 15], BA=100, W=1

then find address of arr[8][6]?

Sparse Matrix

- Matrix that has greater no.of zero elements than non-zero elements.

0	0	3	0	4
0	0	5	7	0
0	0	0	0	0
0	2	6	0	0

- Why do we need to use a sparse matrix instead of a simple matrix?
  - Storage
  - Computing time

Sparse Matrix Representation

- The non-zero elements can be stored with triples, ie. Rows, columns and value.
- Sparse matrix can be represented in the following ways:
  - Array representation
  - Linked list representation

Array Representation

- 2D array is used to represent a sparse matrix in which there are three rows named as
  - **Row**
  - **Column**
  - **Value**

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix} \Rightarrow$$

Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

Program

Exercise

1. Write a menu-driven program to perform the following operations on an array
   i. Insertion
   ii. Deletion
   iii. Display
2. Write a C program to check whether given matrix is sparse marix or not