Data Structure and Algorithms

Dr. Subhra Rani Patra
SCOPE, VIT Chennai

Array

Observations from above Picture:

- ✓ *It's a box of 'Tie'.*
- ✓ *All the compartments are contiguous.*
- ✓ *Each compartment can be identified uniquely.*
- ✓ *Size of the box is fixed and cannot be modified (as they come from standard manufacturers).*

10	20	30	40	50	60

Properties of Array:

- ✓ *Array can store data of specified data type'.*
- ✓ *It has contiguous memory location.*
- ✓ *Every 'cell' of an Array has an unique 'Index'.*
- ✓ *'Index' starts with 0.*
- ✓ *'Size of Array' needs to be specified mandatorily and can not be modified.*

Definition

Array is a data structure consisting of collection of elements, each identified by an array index

Need of an Array

Problem Statement: We want to store 1 million similar data types in memory

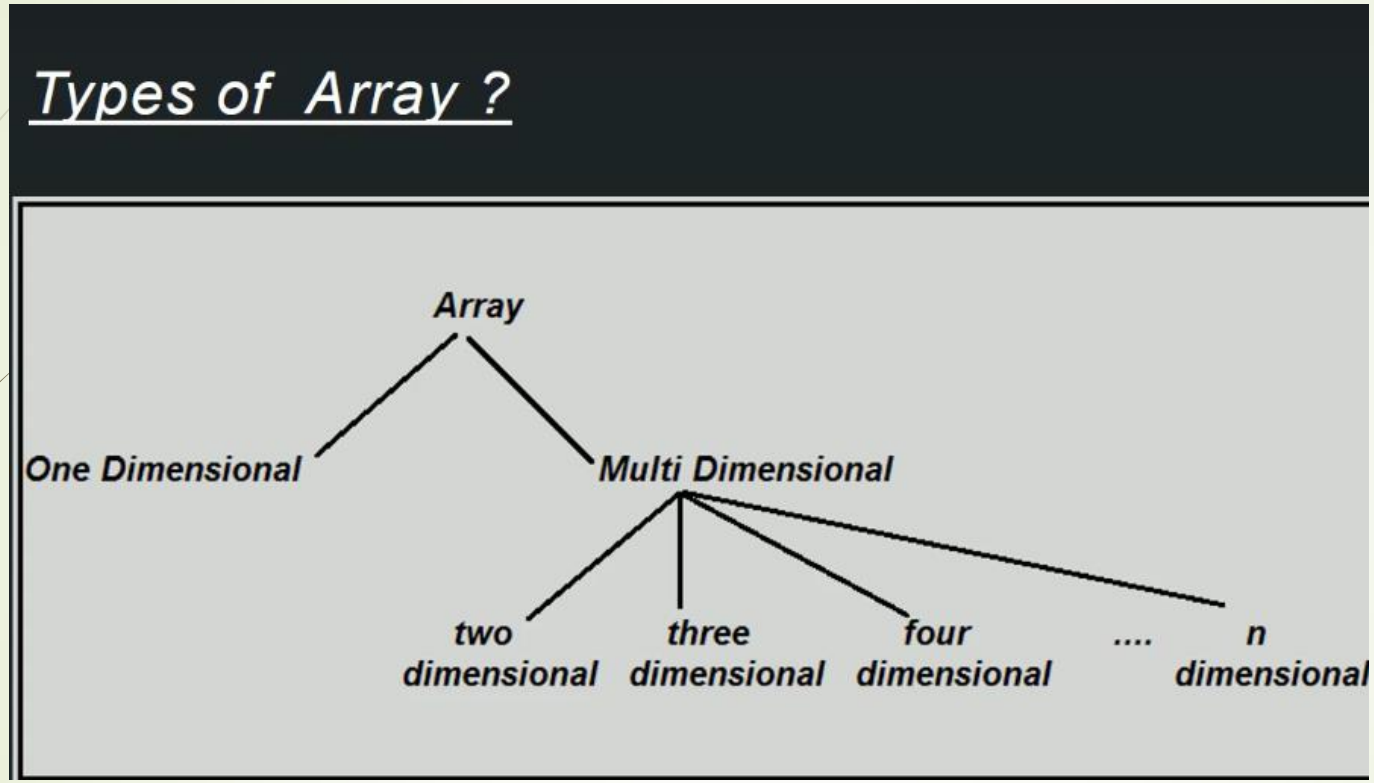
Solution1

- We declare 1 million primitive data structure :Integer, Float, Character, Boolean
- Problem is How will we maintain such a huge list of variables?

Solution2

- We declare an array of Size 1 million
- Advantage: We just need to reference the cell number of the array and we can access that cell

Types of Array ?



✓ ***One Dimensional Array:*** *In it each element is represented by a single subscript. The elements are stored in consecutive memory locations. Ex: Arr [7], Arr[col]*

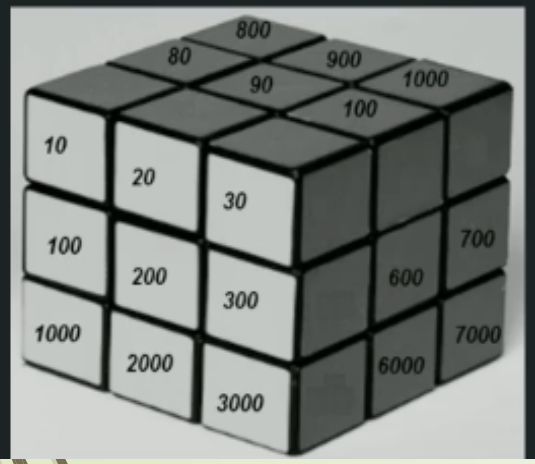
10	20	30	40	50	60	70

✓ ***Multi Dimensional Array:***

✓ ***Two dimensional array:*** *In it each element is represented by two subscripts. Thus a two dimensional m x n array A has m rows and n columns and contains m*n elements. Ex: Arr [3] [7] has 3 rows and 7 columns and 2*3 = 6 elements. Arr[row][col]*

10	20	30	40	50	60	70
100	200	300	400	500	600	700
1000	2000	3000	4000	5000	6000	7000

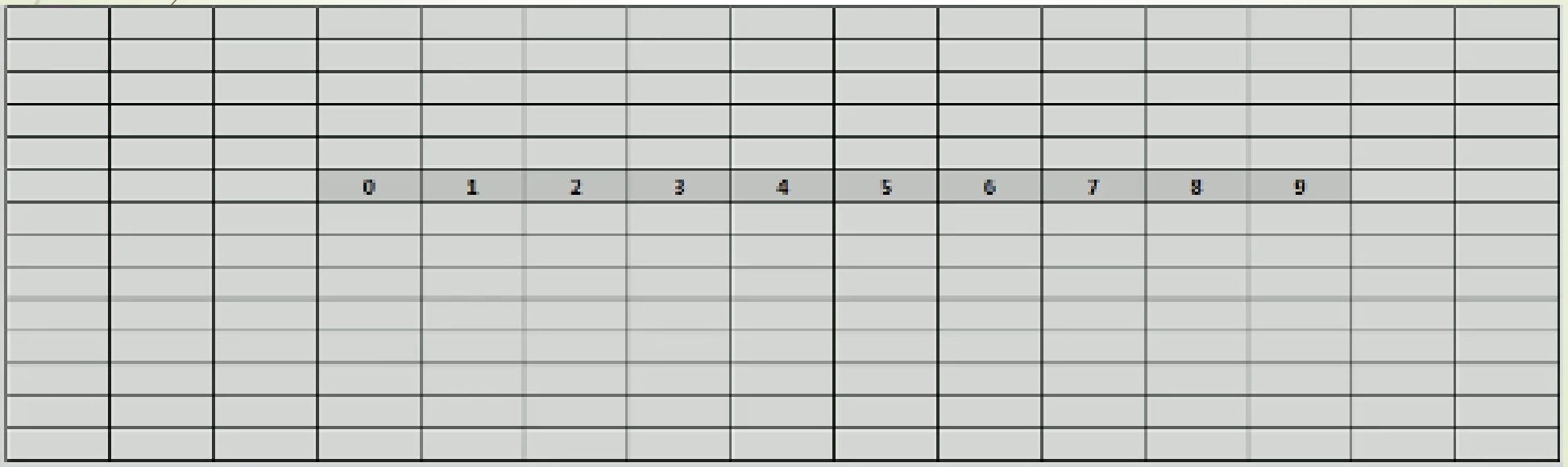
✓ ***Three dimensional array:*** *In it each element is represented by three subscripts. Thus a three dimensional m x n x l array A contains m*n*l elements. E.g. A [3] [3] [3] has 3*3*3 = 27 elements. Arr[depth][row][col]*



How is an Array represented in Memory ?

- ✓ **1D Array:**
  - ✓ Arr[col]
  - ✓ Arr[10] = {0,1,2,3,4,5,6,7,8,9};
  - ✓ Storage in RAM: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

One-D

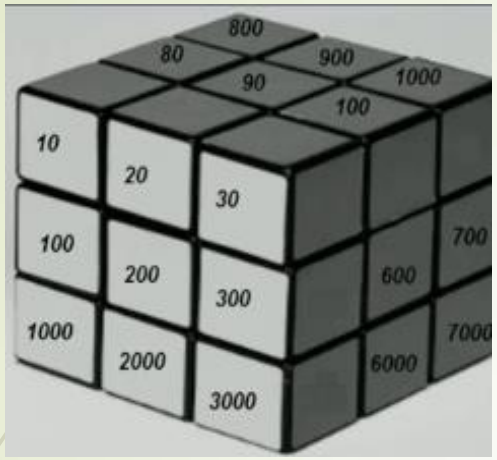


Two-D

				0	10							
				20	30							
				40	50							
				60	70							
				80	90							

Two-D

			00	10	20	30	40	50	60	70	80	90		



Three-D

arr[2,3,3]=

```
{
    { {11, 12, 13}, {15, 16, 17}, {19, 20, 21} },
    { {23, 24, 25}, {27, 28, 29}, {31, 32, 33} }
};
```

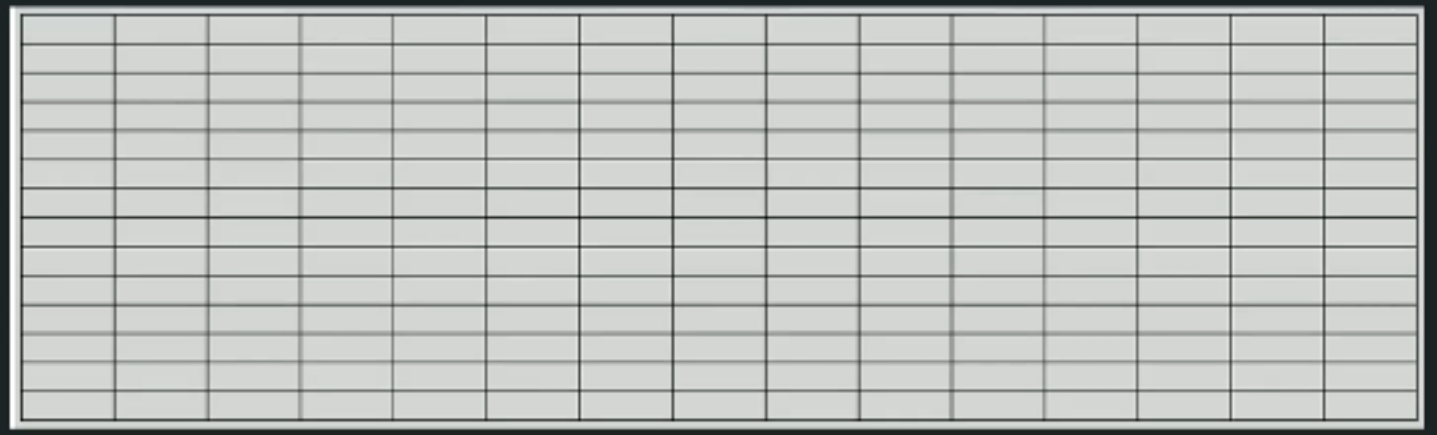
	11	12	13	15	16	17	19	20	21	23	24	25	27	28	29	31	32	33	

Common operations of an array

- Declaring, Instantiating and initializing an array
- Traverse operation
- Inserting a value
- Deleting a value
- Searching a given value
- Updating a value

Declaring, Instantiating, Initializing a 1D Array:

- ✓**Declare:** Creates a reference to Array
- ✓**Instantiation of an Array:** Creates a Array
- ✓**Initialization:** Assigns values to cells in Array

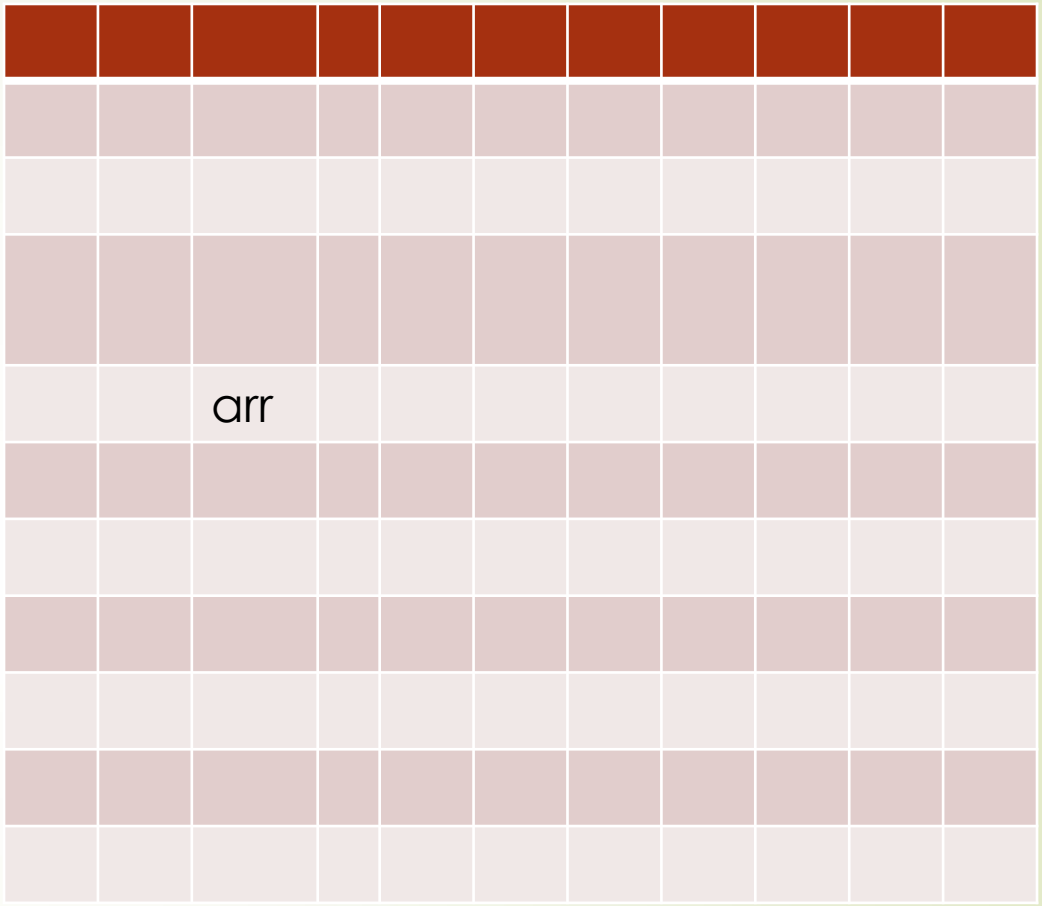


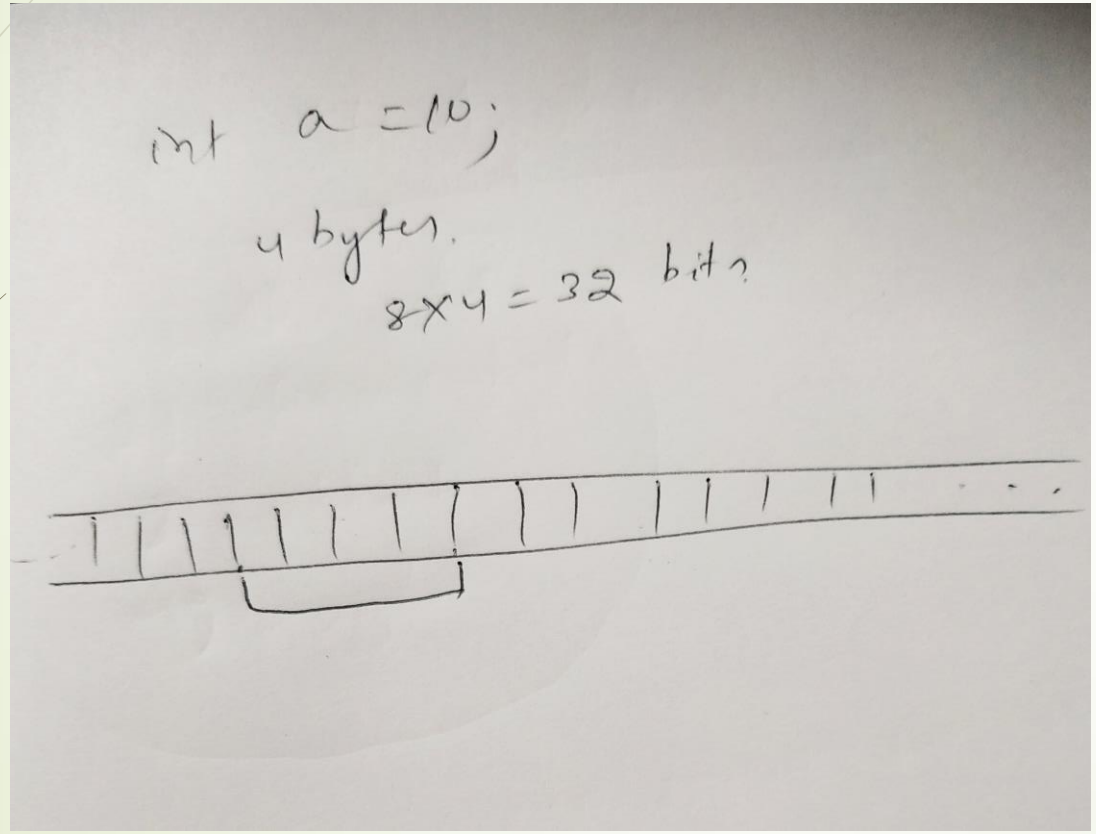
Declare and instantiate an array
type arrayName [arraySize];

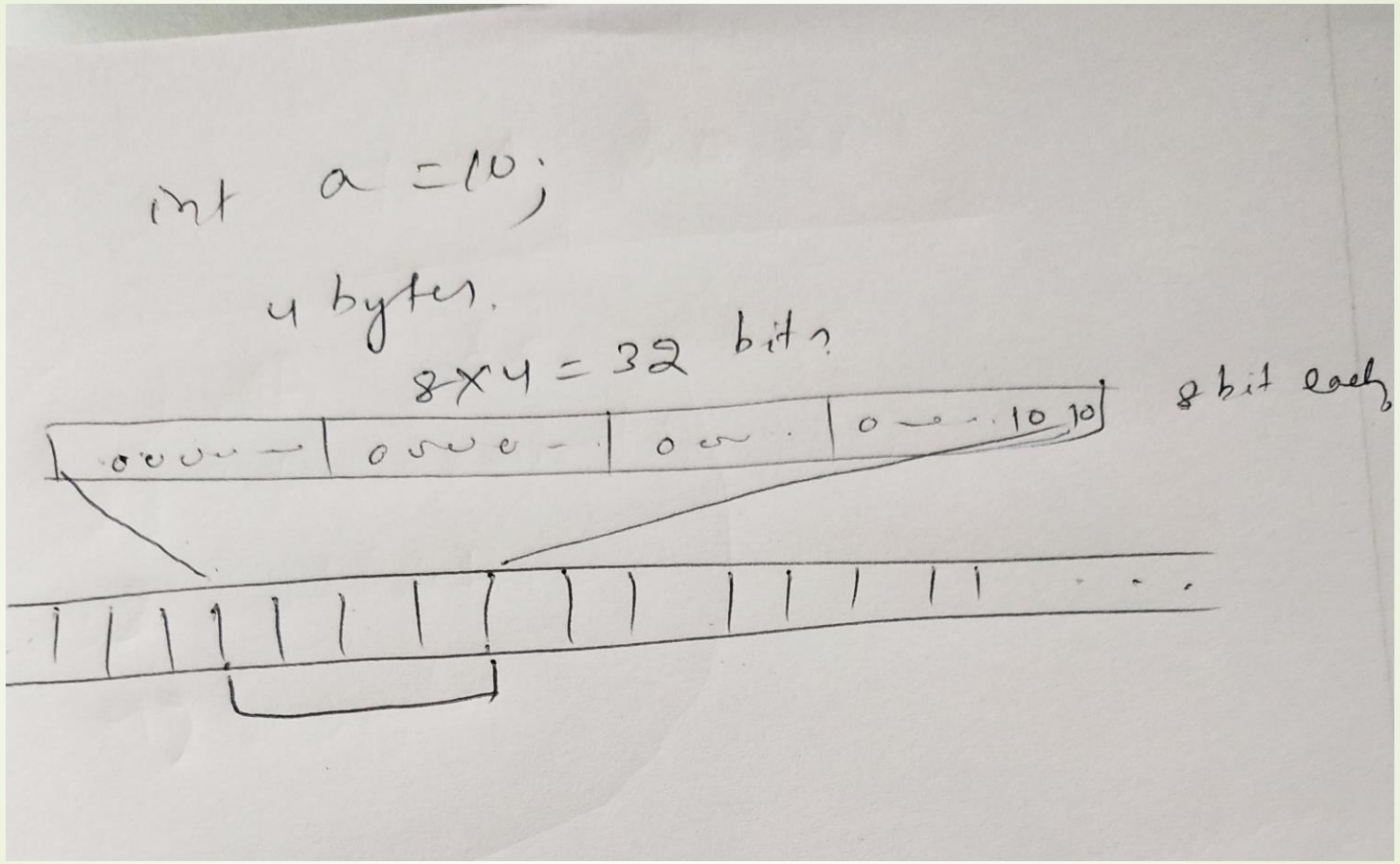
int a[5]

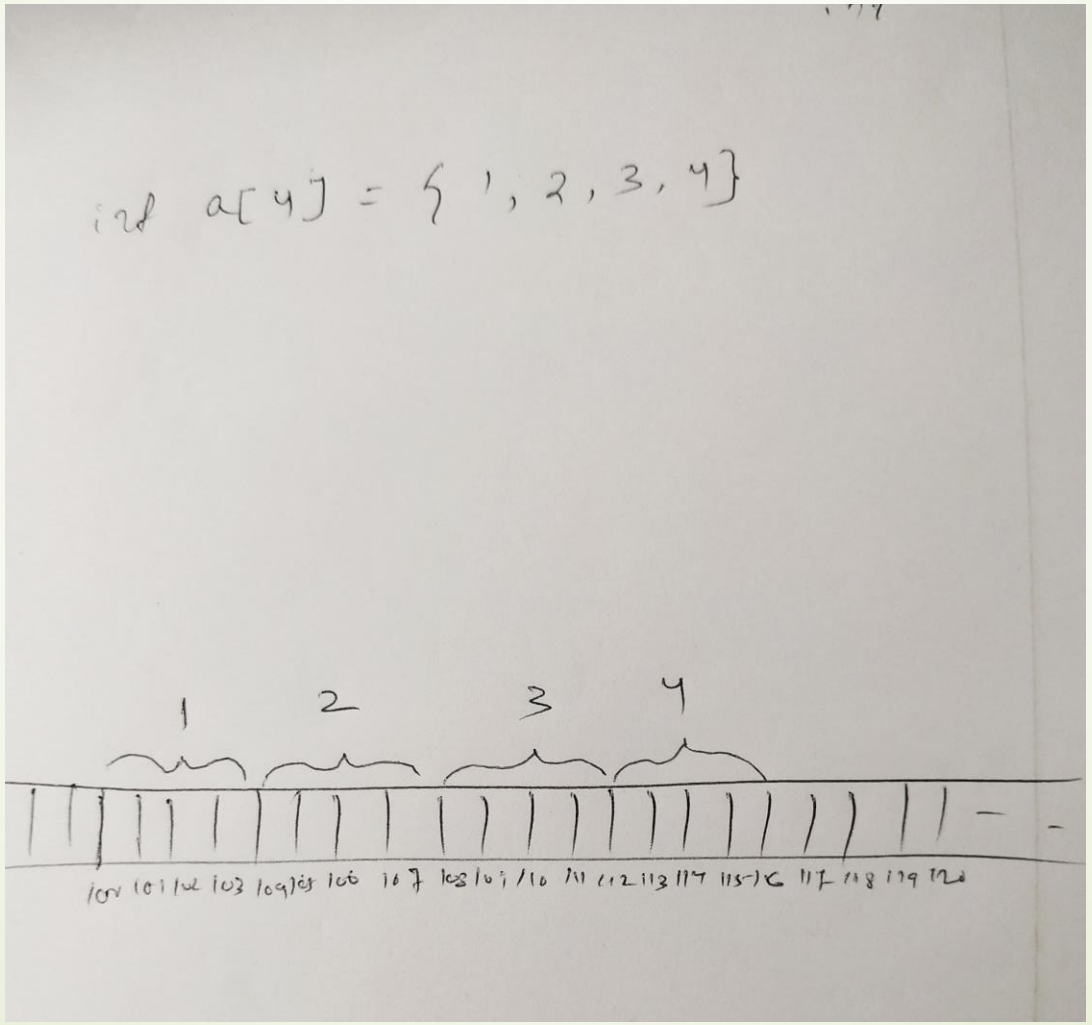
Initialize an array
a[0]=1000
a[1]=2
a[2]=3
a[3]=7
a[4]=50

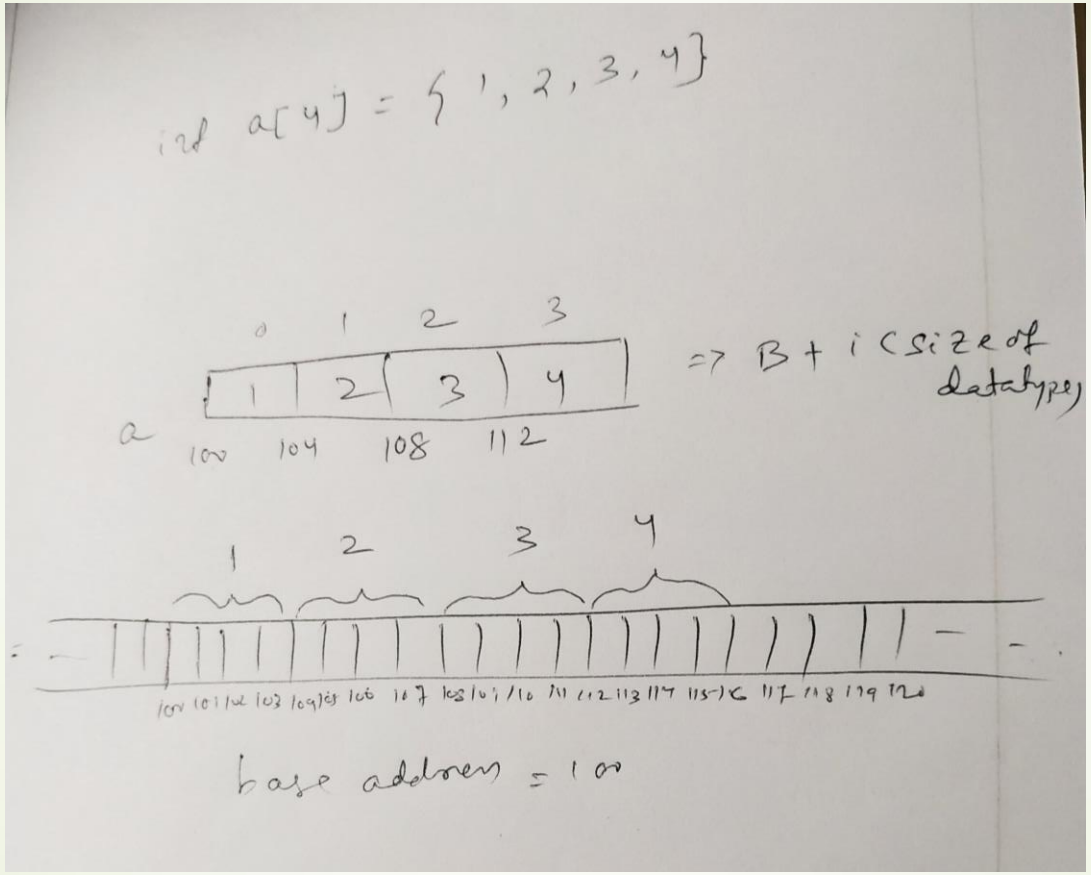
Declaration, Instantiation and Initiaization
arr=int a{1000, 2, 3, 7, 50};




int a = 10;
4 bytes.
8 x 4 = 32 bits


int a = 10;
4 bytes.
8 × 4 = 32 bits
8 bit each
10
10


int a[4] = {1, 2, 3, 4}
1
2
3
4


int a[4] = {1, 2, 3, 4}
0 1 2 3
1 2 3 4
a
100 104 108 112
=> B + i (size of datatype)
1 2 3 4
base address = 100

Traverse Operation

In traverse operation of an array, each element of an array is accessed exactly for once for processing. This is also called visiting of an array.

```c
//Write a Program which perform traversing operation.
#include <stdio.h>
void main()
{
   int LA[] = {2,4,6,8,9};
   int i, n = 5;

   printf("The array elements are:\n");

   for(i = 0; i < n; i++)
   {
      printf("LA[%d] = %d \n", i, LA[i]);
   }
}
```

Insertion Operation

Insert operation is to insert one or more data elements into an array. Based on the requirement, new element can be added at the beginning, end or any given index of array.

Here, we see a practical implementation of insertion operation, where we add data at the end of the array −

Example

Below is the implementation of the above algorithm −

```
#include <stdio.h>
main() {
   int LA[] = {1,3,5,7,8};
   int item = 10, k = 3, n = 5;
   int i = 0, j = n;

   printf("The original array elements are :\n");

   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }

   n = n + 1;

   while( j >= k){
      LA[j+1] = LA[j];
      j = j - 1;
   }
```

```
   LA[k] = item;

   printf("The array elements after insertion :\n");

   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }
}
```

When compile and execute, above program produces the following result –

```
The original array elements are :
LA[0]=1
LA[1]=3
LA[2]=5
LA[3]=7
LA[4]=8
The array elements after insertion :
LA[0]=1
LA[1]=3
LA[2]=5
LA[3]=10
LA[4]=7
LA[5]=8
```

Deletion Operation

Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

Example

Below is the implementation of the above algorithm −

```
#include <stdio.h>
main() {
   int LA[] = {1,3,5,7,8};
   int k = 3, n = 5;
   int i, j;

   printf("The original array elements are :\n");

   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }

   j = k;

   while( j < n){
      LA[j-1] = LA[j];
```

```
      j = j + 1;
   }

   n = n -1;

   printf("The array elements after deletion :\n");

   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }
}
```

When compile and execute, above program produces the following result –

```
The original array elements are :
LA[0]=1
LA[1]=3
LA[2]=5
LA[3]=7
LA[4]=8
The array elements after deletion :
LA[0]=1
LA[1]=3
LA[2]=7
LA[3]=8
```

Search Operation

You can perform a search for array element based on its value or its index.

Example

Below is the implementation of the above algorithm −

```
#include <stdio.h>
main() {
   int LA[] = {1,3,5,7,8};
   int item = 5, n = 5;
   int i = 0, j = 0;

   printf("The original array elements are :\n");

   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }

   while( j < n){

      if( LA[j] == item ){
         break;
      }

      j = j + 1;
```

```
    }

    printf("Found element %d at position %d\n", item, j+1);
}
```

When compile and execute, above program produces the following result –

```
The original array elements are :
LA[0]=1
LA[1]=3
LA[2]=5
LA[3]=7
LA[4]=8
Found element 5 at position 3
```

Update Operation

Update operation refers to updating an existing element from the array at a given index.

Example

Below is the implementation of the above algorithm −

```c
#include <stdio.h>
main() {
   int LA[] = {1,3,5,7,8};
   int k = 3, n = 5, item = 10;
   int i, j;

   printf("The original array elements are :\n");

   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }

   LA[k-1] = item;

   printf("The array elements after updation :\n");

   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }
}
```

When compile and execute, above program produces the following result –

```
The original array elements are :
LA[0]=1
LA[1]=3
LA[2]=5
LA[3]=7
LA[4]=8
The array elements after updation :
LA[0]=1
LA[1]=3
LA[2]=10
```

LA[3]=7
LA[4]=8


Thank
you