Arrays (1D)

Arrays are defined as the collection of similar types of data items stored at contiguous memory locations. It is one of the simplest data structures where each data element can be randomly accessed by using its index number.

In C programming, they are the derived data types that can store the primitive type of data such as int, char, double, float, etc. For example, if we want to store the marks of a student in 6 subjects, then we don't need to define a different variable for the marks in different subjects. Instead, we can define an array that can store the marks in each subject at the contiguous memory locations.

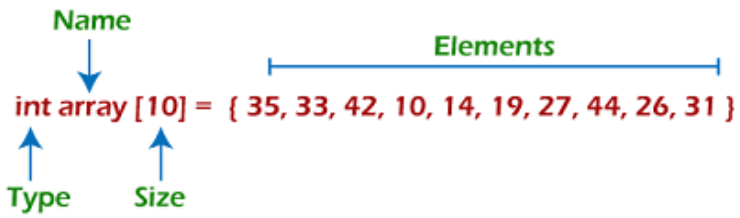
Properties of array :

There are some of the properties of an array that are listed as follows -

- Each element in an array is of the same data type and carries the same size that is 4 bytes.
- Elements in the array are stored at contiguous memory locations from which the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Representation of an array :

We can represent an array in various ways in different programming languages. As an illustration, let's see the declaration of array in C language -

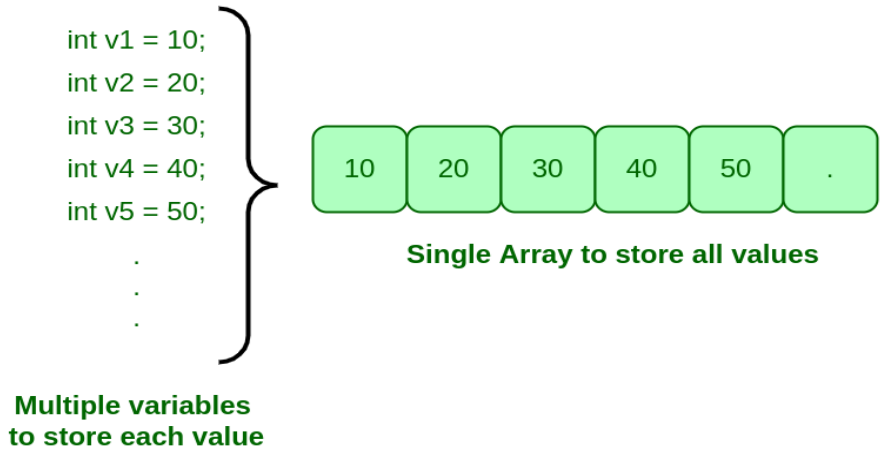


As per the above illustration, there are some of the following important points -

- Index starts with 0.
- The array's length is 10, which means we can store 10 elements.
- Each element in the array can be accessed via its index.

Why do we need arrays?

We can use normal variables (v1, v2, v3, ..) when we have a small number of objects, but if we want to store a large number of instances, it becomes difficult to manage them with normal variables. The idea of an array is to represent many instances in one variable.



Advantages :

- **Code Optimization:** we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages :

- **Size Limit:** We can store only fixed size of elements in an array. It doesn't grow its size at runtime.
- **Homogeneous Data:** No dissimilar type of data are allowed to store in an array

Applications of Arrays :

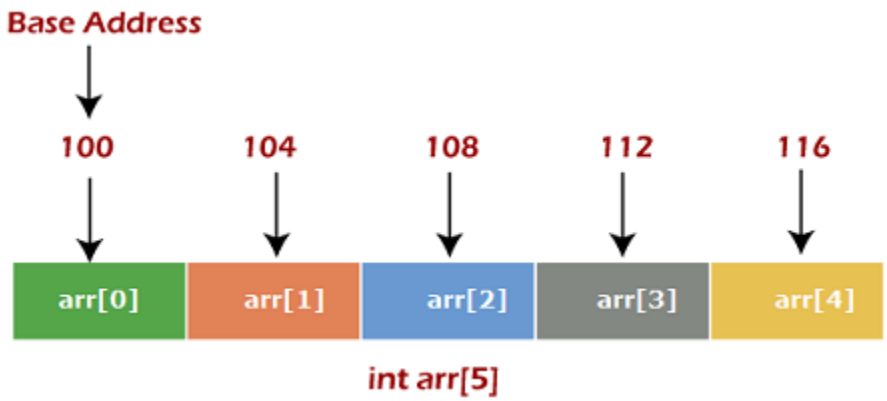
1. Array stores data elements of the same data type.
2. Maintains multiple variable names using a single name. Arrays help to maintain large data under a single variable name. This avoid the confusion of using multiple variables.
3. Arrays can be used for sorting data elements. Different sorting techniques like Bubble sort, Insertion sort, Selection sort etc use arrays to store and sort elements easily.
4. Arrays can be used for performing matrix operations. Many databases, small and large, consist of one-dimensional and two-dimensional arrays whose elements are records.
5. Arrays can be used for CPU scheduling.
6. Lastly, arrays are also used to implement other data structures like Stacks, Queues, Heaps, Hash tables etc.

Memory allocation of an array

As stated above, all the data elements of an array are stored at contiguous locations in the main memory. The name of the array represents the base address or the address of the first element in the main memory. Each element of the array is represented by proper indexing.

We can define the indexing of an array in the below ways -

- ➔ 0 (zero-based indexing): The first element of the array will be arr[0].
- ➔ 1 (one-based indexing): The first element of the array will be arr[1].
- ➔ n (n - based indexing): The first element of the array can reside at any random index number.



In the above diagram, we have shown the memory allocation of an array arr of size 5. The array follows a 0-based indexing approach. The base address of the array is 100 bytes. It is the address of arr[0]. Here, the size of the data type used is 4 bytes; therefore, each element will take 4 bytes in the memory.

Basic operations on Array

Now, let's discuss the basic operations supported in the array -

1. **Traversal -** This operation is used to print the elements of the array.
2. **Insertion -** It is used to add an element at a particular index.
3. **Deletion -** It is used to delete an element from a particular index.
4. **Update -** It updates an element at a particular index.
5. **Search -** It is used to search an element using the given index or by the value.
6. **Sort -** It is used to rearrange a given array or list elements according to a comparison operator on the elements.

   We are discussing above mentioned top 4 operations of Array, searching and sorting operations are discussed in the upcoming units.

Traverse Operation

This operation is to traverse through the elements of an array.

- **Algorithm :**

  Let LA is a Linear Array unordered with N elements.

```
1. Start
2. Set i = 0
3. Repeat Step 4 - 5 while i<N
4. Access LA[i]
5. Set i = i + 1
6. Stop
```

- **C Program:**

```
#include <stdio.h>
void 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]);
   }
}
```

- **Output :**

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

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.

- **Algorithm:**

  Let LA is a Linear Array unordered with N elements and K is a positive integer such that K<=N. Below is the algorithm where ITEM is inserted into the K th position of LA –

```
1. Start
2. Set J=N
3. Set N = N+1
```

```
4. Repeat steps 5 and 6 while J >= K
5. Set LA[J+1] = LA[J]
6. Set J = J-1
7. Set LA[K] = ITEM
8. Stop
```

- C Program :

```
#include <stdio.h>
void 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]);
     }
}
```

- Output :

```
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.

- ### Algorithm:

  Consider LA is a linear array with N elements and K is a positive integer such that K<=N. Below is the algorithm to delete an element available at the K th position of LA.

```
1. Start
2. Set J=K
3. Repeat steps 4 and 5 while J < N
4. Set LA[J-1] = LA[J]
5. Set J = J+1
6. Set N = N-1
7. Stop
```

- ### C Program:

```
#include <stdio.h>
void 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]);
   }
}
```

- ### Output:

```
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
```

Update Operation :

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

- **Algorithm:**

  Consider LA is a linear array with N elements and K is a positive integer such that K<=N. Below is the algorithm to update an element available at the Kth position of LA.

  ```
  1. Start
  2. Set LA[K-1] = ITEM
  3. Stop
  ```

- **C Program:**

  ```
  #include <stdio.h>
  void 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]);
     }
  }
  ```

- **Output:**

  ```
  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
  ```