Array in Data Structure

In this article, we will discuss the array in data structure. 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.

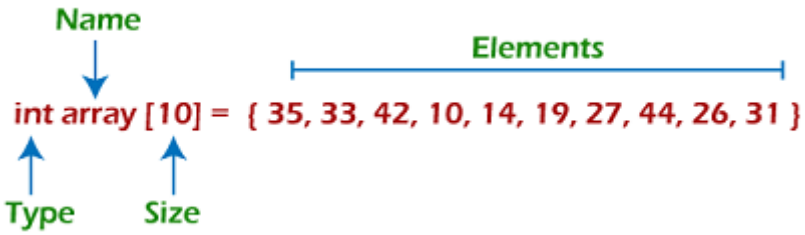
Properties of array

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 are arrays required?

Arrays are useful because -

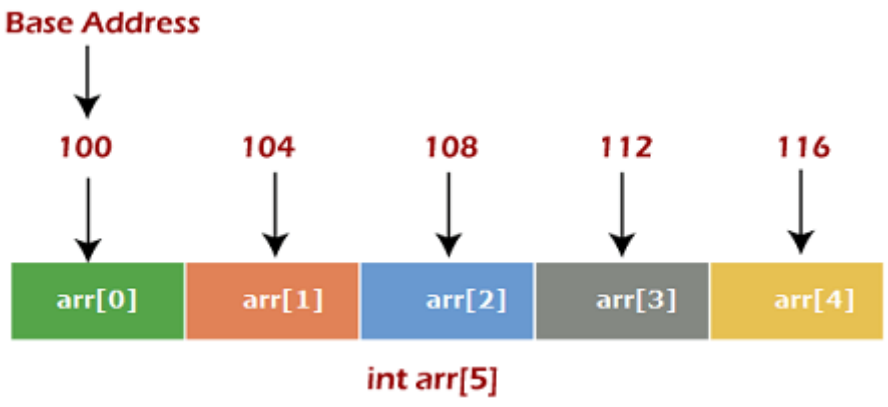
- Sorting and searching a value in an array is easier.
- Arrays are best to process multiple values quickly and easily.
- **Arrays are good for storing multiple values in a single variable -** In computer programming, most cases require storing a large number of data of a similar type. To store such an amount of data, we need to define a large number of variables. It would be very difficult to remember the names of all the variables while writing the programs. Instead of naming all the variables with a different name, it is better to define an array and store all the elements into it.

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 -

1. 0 (zero-based indexing): The first element of the array will be arr[0].
2. 1 (one-based indexing): The first element of the array will be arr[1].
3. n (n - based indexing): The first element of the array can reside at any random index number.



In the above image, 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 [illegible] memory.

⇧ SCROLL TO TOP

How to access an element from the array?

We required the information given below to access any random element from the array -

- Base Address of the array.
- Size of an element in bytes.
- Type of indexing, array follows.

The formula to calculate the address to access an array element -

> Byte address of element A[i] = base address + size * (i - first index)

Here, size represents the memory taken by the primitive data types. As an instance, **int** takes 2 bytes, **float** takes 4 bytes of memory space in C programming.

We can understand it with the help of an example -

Suppose an array, A[-10 ..... +2] having Base address (BA) = 999 and size of an element = 2 bytes, find the location of A[-1].

L(A[-1]) = 999 + 2 x [(-1) - (-10)]

= 999 + 18

= 1017

Basic operations

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

- Traversal - This operation is used to print the elements of the array.
- Insertion - It is used to add an element at a particular index.
- Deletion - It is used to delete an element from a particular index.
- Search - It is used to search an element using the given index or by the value.
- Update - It updates an element at a particular index.

Traversal operation

This operation is performed to traverse through the array elements. It prints all array elements one by one. We can understand it with the below program -

```
#include <stdio.h>
void main() {
  int Arr[5] = {18, 30, 15, 70, 12};
int i;
  printf("Elements of the array are:\n");
  for(i = 0; i<5; i++) {
    printf("Arr[%d] = %d,  ", i, Arr[i]);
  }
}
```

Output

```
Elements of the array are:
Arr[0] = 18,  Arr[1] = 30,  Arr[2] = 15,  Arr[3] = 70,  Arr[4] = 12,
```

Insertion operation

This operation is performed to insert one or more elements into the array. As per the requirements, an element can be added at the beginning, end, or at any index of the array. Now, let's see the implementation of inserting an element into the array.

```
#include <stdio.h>
int main()
{
   int arr[20] = { 18, 30, 15, 70, 12 };
   int i, x, pos, n = 5;
   printf("Array elements before insertion\n");
   for (i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");

   x = 50; // element to be inserted
   pos = 4;
   n++;

   for (i = n-1; i >= pos; i--)
```

```
    printf("Array elements after insertion\n");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

Output

```
Array elements before insertion
18 30 15 70 12
Array elements after insertion
18 30 15 50 70 12
```

Deletion operation

As the name implies, this operation removes an element from the array and then reorganizes all of the array elements.

```
#include <stdio.h>

void main() {
   int arr[] = {18, 30, 15, 70, 12};
   int k = 30, n = 5;
   int i, j;

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

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

   j = k;

   while( j < n) {
      arr[j-1] = arr[j];
      i = i + 1;
```

⇧ SCROLL TO TOP

```
  n = n -1;

  printf("\nElements of array after deletion:\n");

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

Output

```
Given array elements are :
arr[0] = 18,  arr[1] = 30,  arr[2] = 15,  arr[3] = 70,  arr[4] = 12,
Elements of array after deletion:
arr[0] = 18,  arr[1] = 30,  arr[2] = 15,  arr[3] = 70,
```

Search operation

This operation is performed to search an element in the array based on the value or index.

```
#include <stdio.h>

void main() {
  int arr[5] = {18, 30, 15, 70, 12};
  int item = 70, i, j=0 ;

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

  for(i = 0; i<5; i++) {
    printf("arr[%d] = %d,  ", i, arr[i]);
  }
   printf("\nElement to be searched = %d", item);
  while( j < 5){
    if( arr[j] == item ) {
      break;
    }
```

⇧ SCROLL TO TOP

```
    j = j + 1;
  }

  printf("\nElement %d is found at %d position", item, j+1);
}
```

Output

```
Given array elements are :
arr[0] = 18,  arr[1] = 30,  arr[2] = 15,  arr[3] = 70,  arr[4] = 12,
Element to be searched = 70
Element 70 is found at 4 position
```

Update operation

This operation is performed to update an existing array element located at the given index.

```
#include <stdio.h>

void main() {
  int arr[5] = {18, 30, 15, 70, 12};
  int item = 50, i, pos = 3;

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

  for(i = 0; i<5; i++) {
    printf("arr[%d] = %d,  ", i, arr[i]);
  }

arr[pos-1] = item;
  printf("\nArray elements after updation :\n");

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

⇧ SCROLL TO TOP

```
Given array elements are :
arr[0] = 18,  arr[1] = 30,  arr[2] = 15,  arr[3] = 70,  arr[4] = 12,
Array elements after updation :
arr[0] = 18,  arr[1] = 30,  arr[2] = 50,  arr[3] = 70,  arr[4] = 12,
```

Complexity of Array operations

Time and space complexity of various array operations are described in the following table.

Time Complexity

Operation	Average Case	Worst Case
Access	O(1)	O(1)
Search	O(n)	O(n)
Insertion	O(n)	O(n)
Deletion	O(n)	O(n)

Space Complexity

In array, space complexity for worst case is **O(n)**.

Advantages of Array

- Array provides the single name for the group of variables of the same type. Therefore, it is easy to remember the name of all the elements of an array.
- Traversing an array is a very simple process; we just need to increment the base address of the array in order to visit each element one by one.
- Any element in the array can be directly accessed by using the index.

Disadvantages of Array

- Array is homogenous. It means that the elements with similar data type can be stored in it.
- In array, there is static memory allocation that is size of an array cannot be altered.
- There will be wastage of memory if we store less number of elements than the declared

⇧ SCROLL TO TOP

Conclusion

In this article, we have discussed the special data structure, i.e., array, and the basic operations performed on it. Arrays provide a unique way to structure the stored data such that it can be easily accessed and can be queried to fetch the value using the index.

← Prev Next →