

Data Structures & Algorithms

Chapter - 03

Data Structures

Rakin Mohammad Sifullah
Computer Science Engineer

Overview

In this chapter, we will cover the basic understanding of Algorithms.

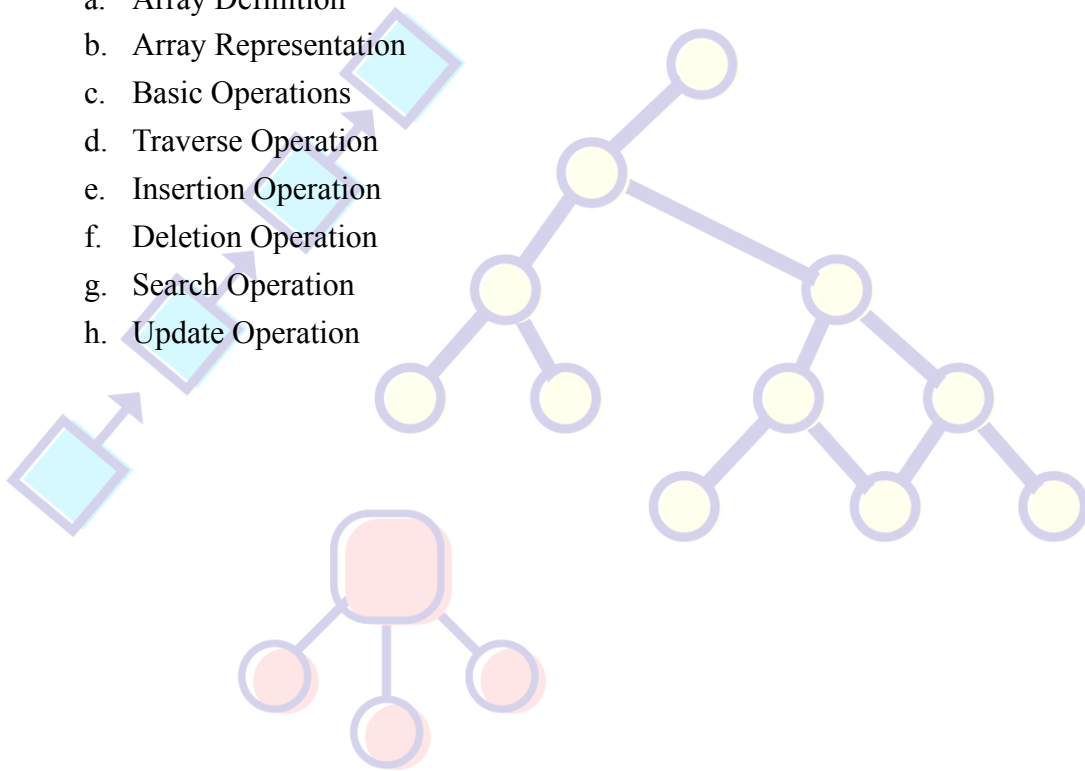
We will cover -

1. Data Structures Basics

- a. Data Definition
- b. Data Object
- c. Data Type
- d. Built-in Data Type
- e. Derived Data Type
- f. Basic Operations

2. Array Data Structure

- a. Array Definition
- b. Array Representation
- c. Basic Operations
- d. Traverse Operation
- e. Insertion Operation
- f. Deletion Operation
- g. Search Operation
- h. Update Operation



Data Definition

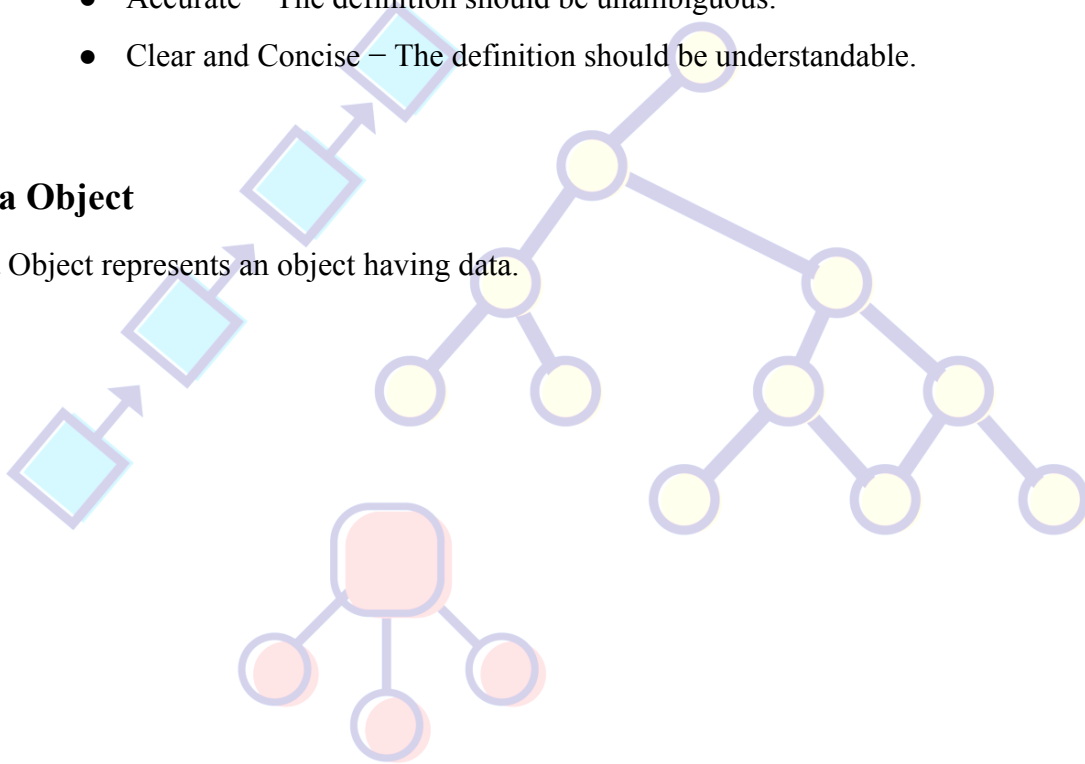
Data Definition

Data Definition defines a particular data with the following characteristics.

- Atomic – Definition should define a single concept.
- Traceable – The definition should be able to be mapped to some data element.
- Accurate – The definition should be unambiguous.
- Clear and Concise – The definition should be understandable.

Data Object

Data Object represents an object having data.



Data Type

The data type is a way to classify various types of data such as integer, string, etc. which determines the values that can be used with the corresponding type of data, the type of operations that can be performed on the corresponding type of data. There are two data types –

1. Built-in Data Type
2. Derived Data Type

Built-in Data Type

Those data types for which a language has built-in support are known as Built-in Data types. For example, most of the languages provide the following built-in data types.

1. Integers
2. Boolean (true, false)
3. Floating (Decimal numbers)
4. Character and Strings

Derived Data Type

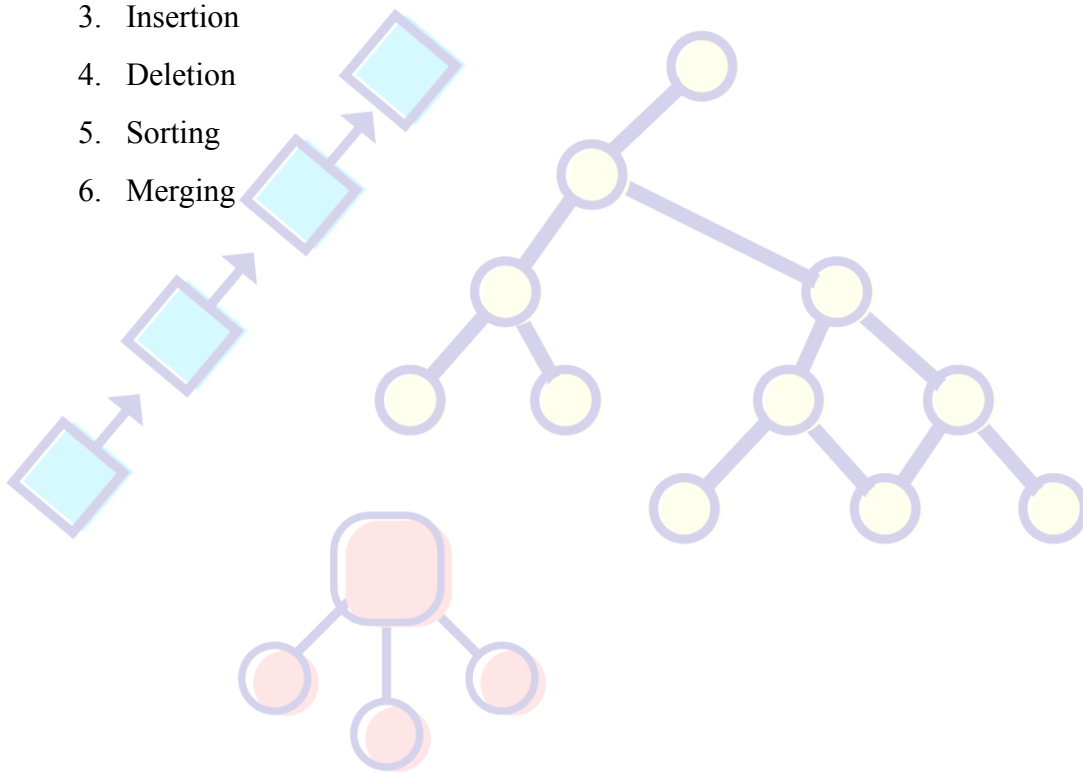
Those data types which are implementation independent as they can be implemented in one or the other way are known as derived data types. These data types are normally built by the combination of primary or built-in data types and associated operations on them. For example –

1. List
2. Array
3. Stack
4. Queue

Basic Operations

The data in the data structures are processed by certain operations. The particular data structure was chosen largely depends on the frequency of the operation that needs to be performed on the data structure.

1. Traversing
2. Searching
3. Insertion
4. Deletion
5. Sorting
6. Merging



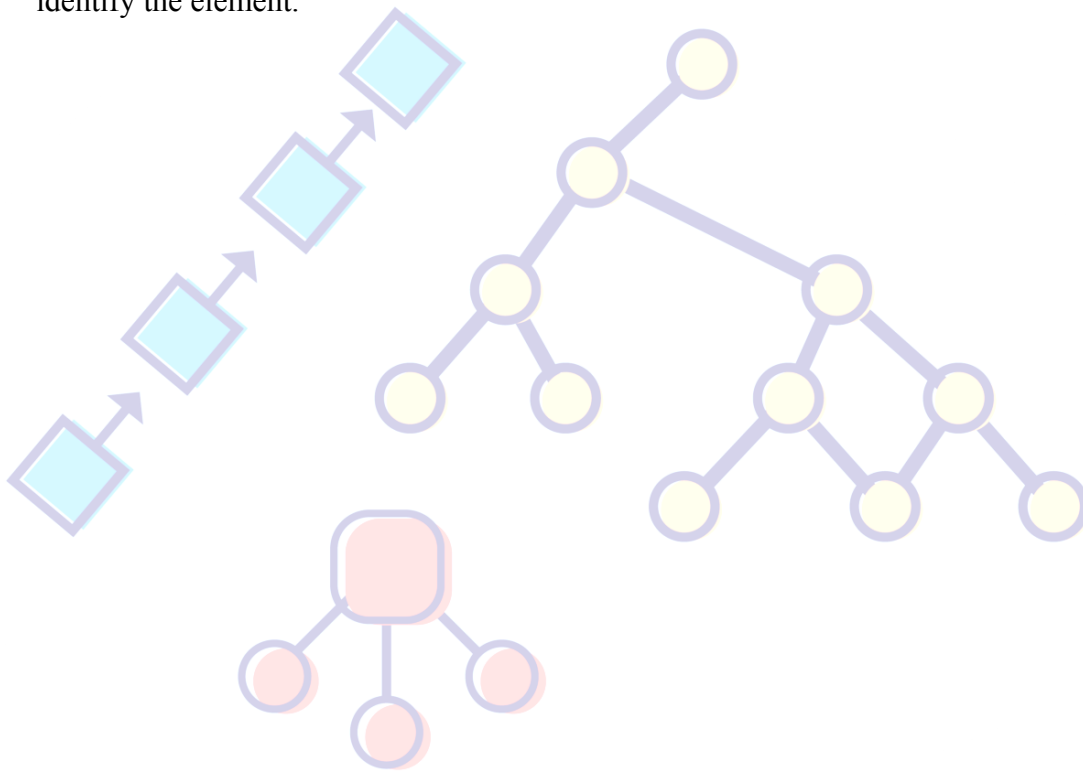
Array Data Structure

Definition

The array is a container that can hold a fixed number of items and these items should be of the same type. Most of the data structures make use of arrays to implement their algorithms.

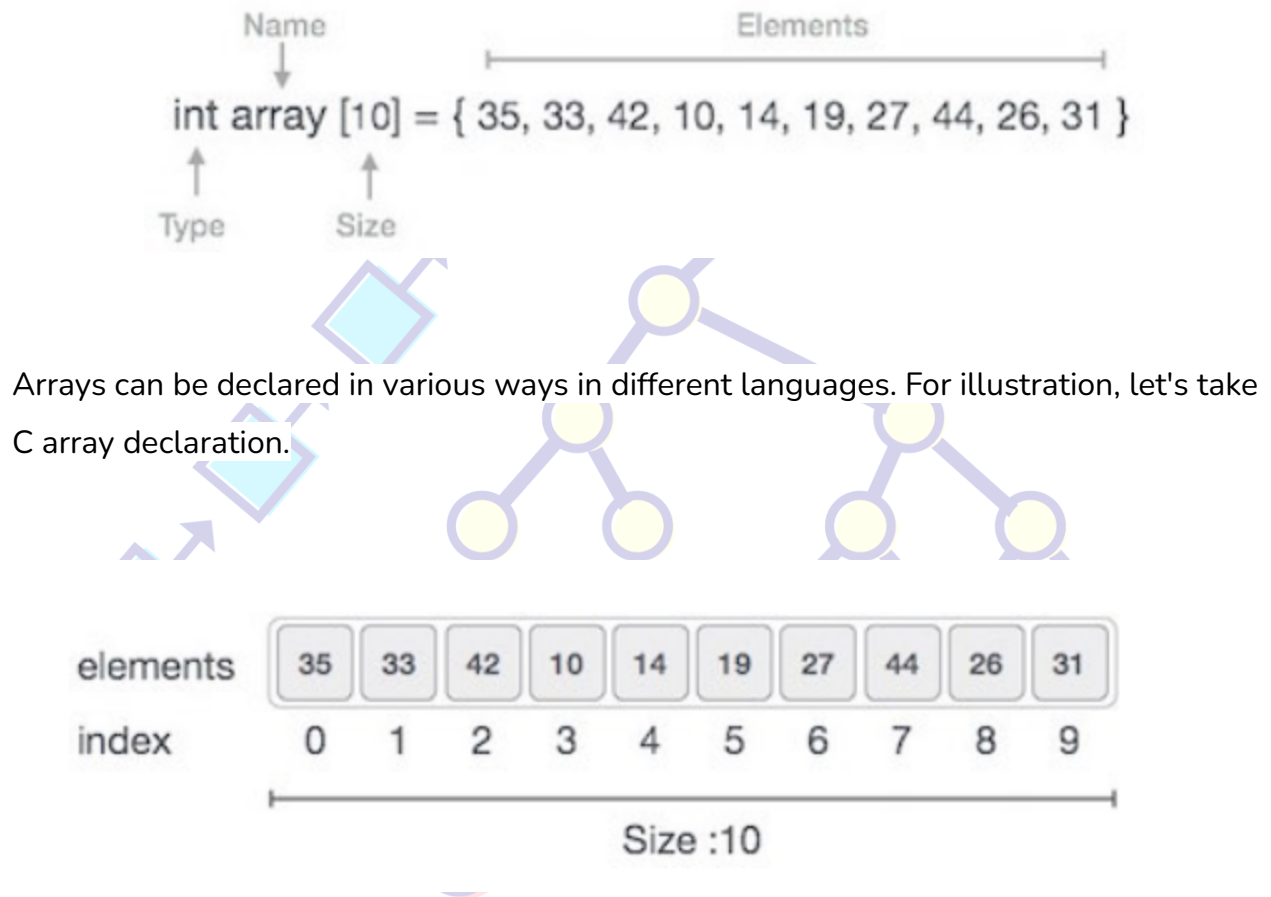
Following are the important terms to understand the concept of Array.

1. **Element** – Each item stored in an array is called an element.
2. **Index** – Each location of an element in an array has a numerical index, which is used to identify the element.



Array Representation

Arrays can be declared in various ways in different languages. For illustration, let's take C array declaration.



As per the above illustration, following are the important points to be considered.

1. Index starts with 0.
2. Array length is 10 which means it can store 10 elements.
3. Each element can be accessed via its index. For example, we can fetch an element at index 6 as 9.

Basic Operations

Following are the basic operations supported by an array.

- **Traverse** – print all the array elements one by one.
- **Insertion** – Adds an element at the given index.
- **Deletion** – Deletes an element at the given index.
- **Search** – Searches an element using the given index or by the value.
- **Update** – Updates an element at the given index.

In C, when an array is initialized with size, then it assigns default values to its elements in following order.

Data Type	Default Value
bool	false
char	0
int	0
float	0.0
double	0.0f
void	
whar_t	0

Traverse Operation

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

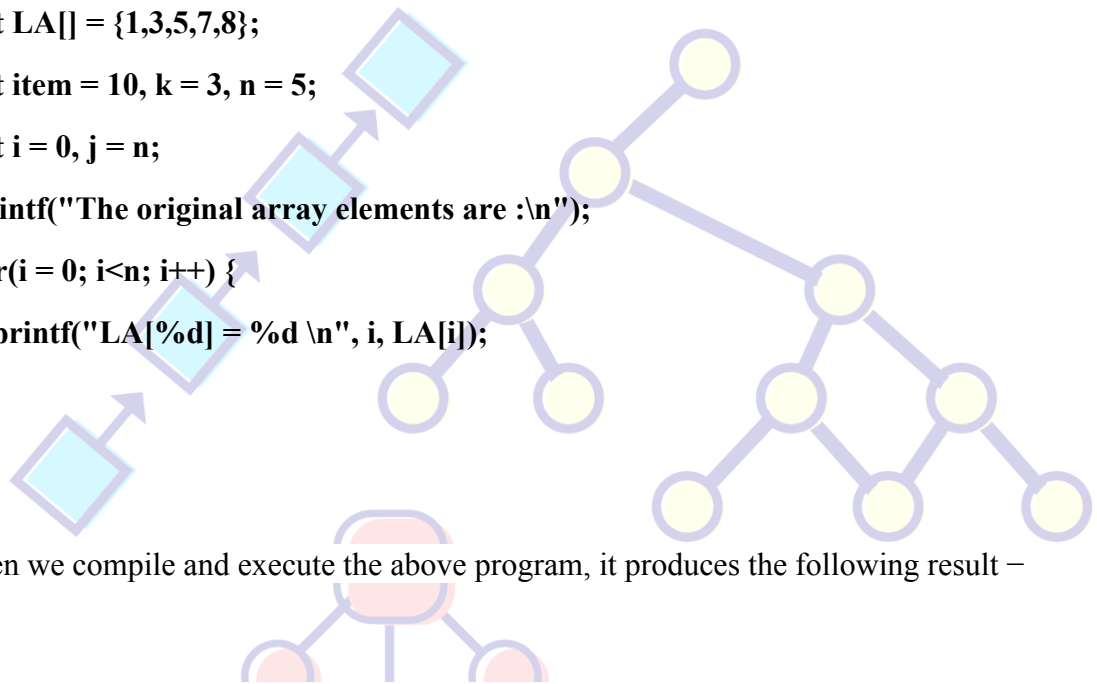
Example

Following program traverses and prints the elements of an array:

```
#include <stdio.h>

int 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]);
    }
}
```

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



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

Process returned 0 (0x0)   execution time : 0.008 s
Press any key to continue.
```

Insertion Operation

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

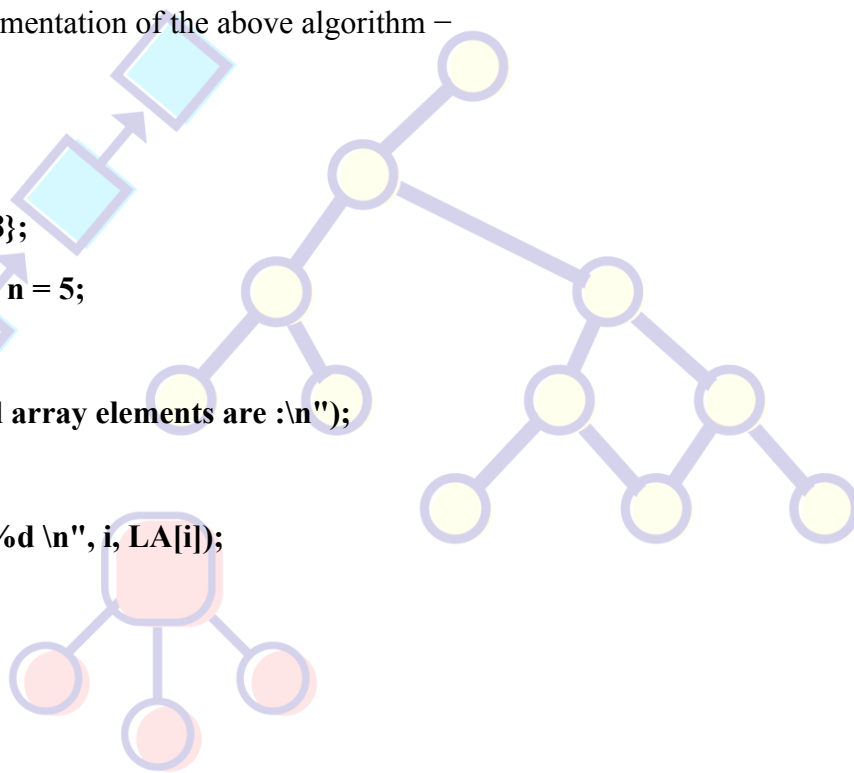
Here, we see a practical implementation of insertion operation, where we add data at the end of the array –

Example

Following is the implementation of the above algorithm –

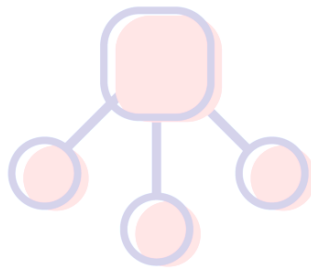
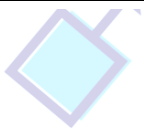
```
#include <stdio.h>

int 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 we compile and execute the above program, it 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] = 7  
LA[4] = 8  
LA[5] = 6  
  
Process returned 0 (0x0)    execution time : 0.022 s  
Press any key to continue.
```



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 $K \leq N$. Following is the algorithm to delete an element available at the Kth position of LA.

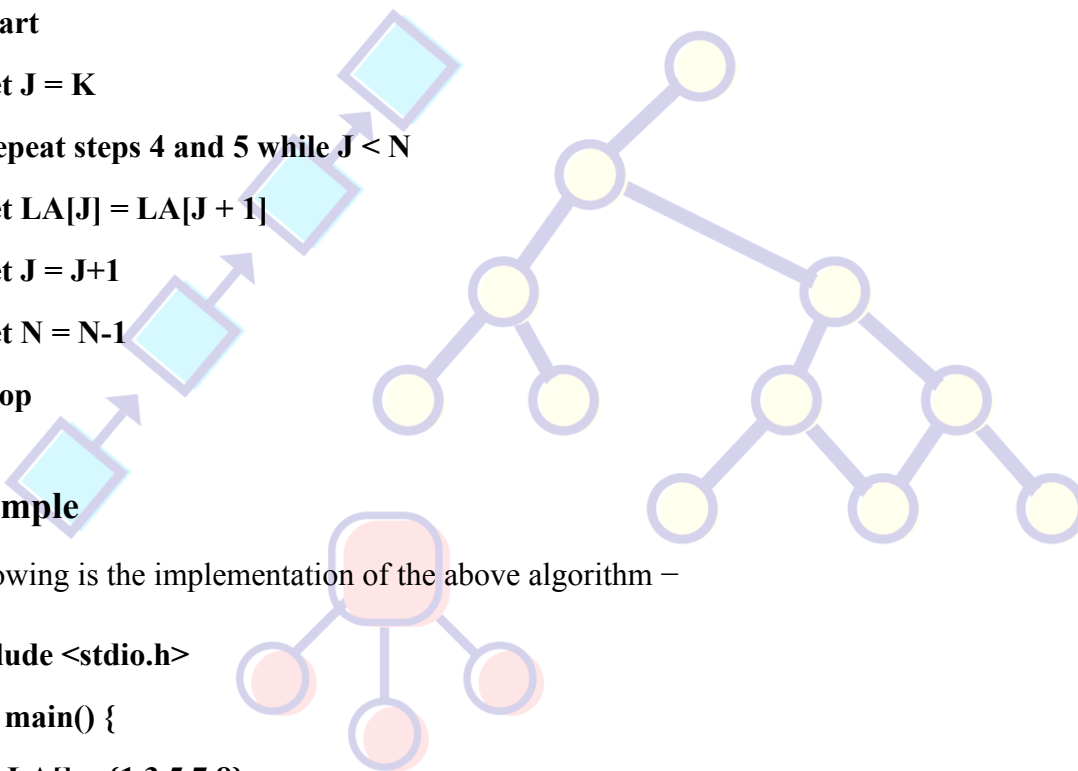
1. Start
2. Set $J = K$
3. Repeat steps 4 and 5 while $J < N$
4. Set $LA[J] = LA[J + 1]$
5. Set $J = J + 1$
6. Set $N = N - 1$
7. Stop

Example

Following is the implementation of the above algorithm –

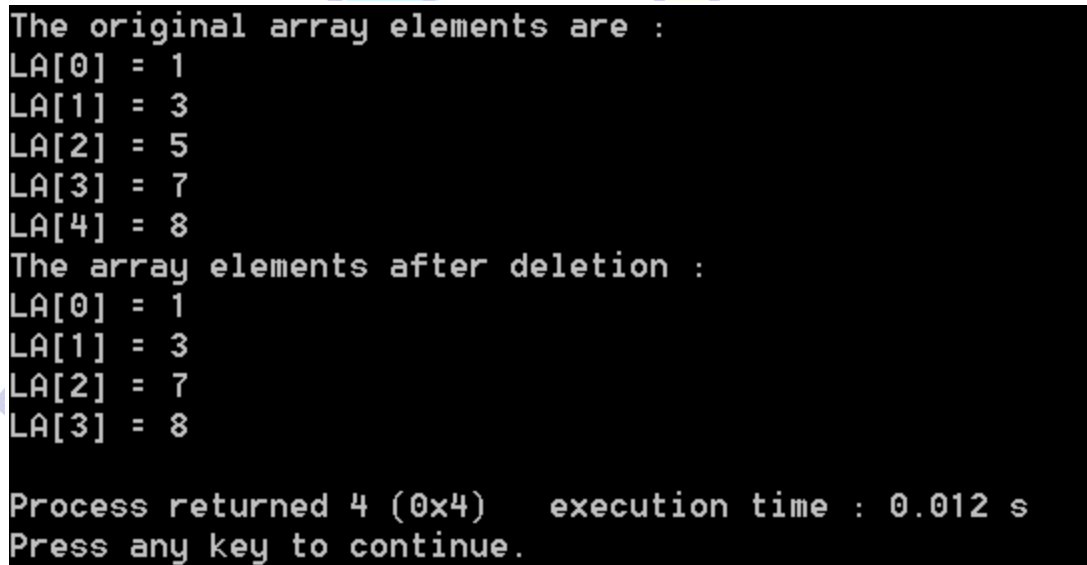
```
#include <stdio.h>

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]);  
}  
}
```

When we compile and execute the above program, it 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  
  
Process returned 4 (0x4)    execution time : 0.012 s  
Press any key to continue.
```

Search Operation

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

Algorithm

Consider LA is a linear array with N elements and K is a positive integer such that $K \leq N$.

Following is the algorithm to find an element with a value of ITEM using sequential search.

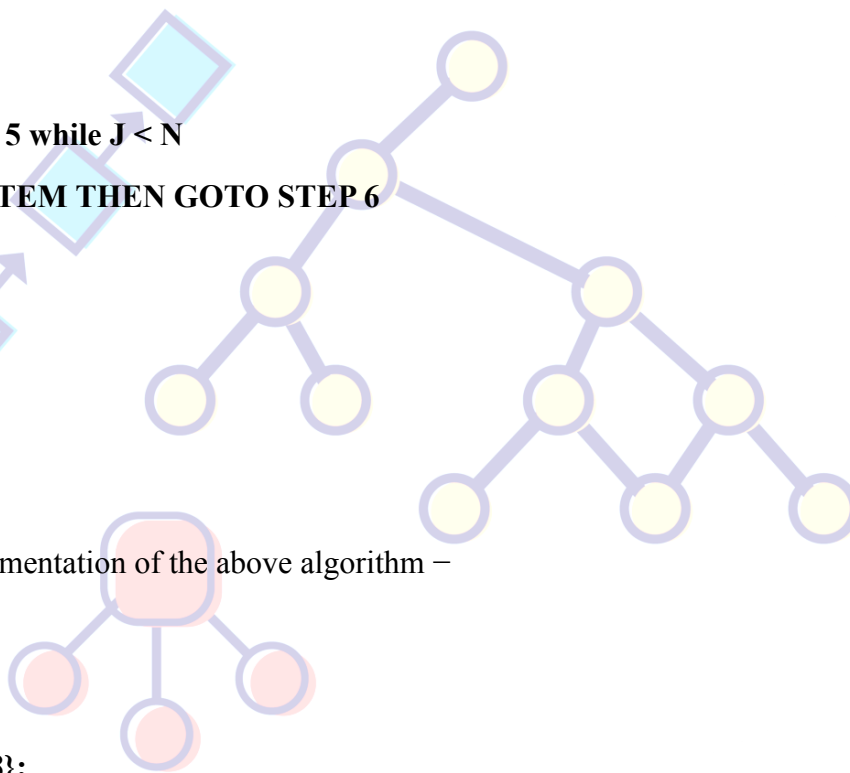
1. Start
2. Set $J = 0$
3. Repeat steps 4 and 5 while $J < N$
4. IF $LA[J]$ is equal ITEM THEN GOTO STEP 6
5. Set $J = J + 1$
6. PRINT J, ITEM
7. Stop

Example

Following is the implementation of the above algorithm –

```
#include <stdio.h>

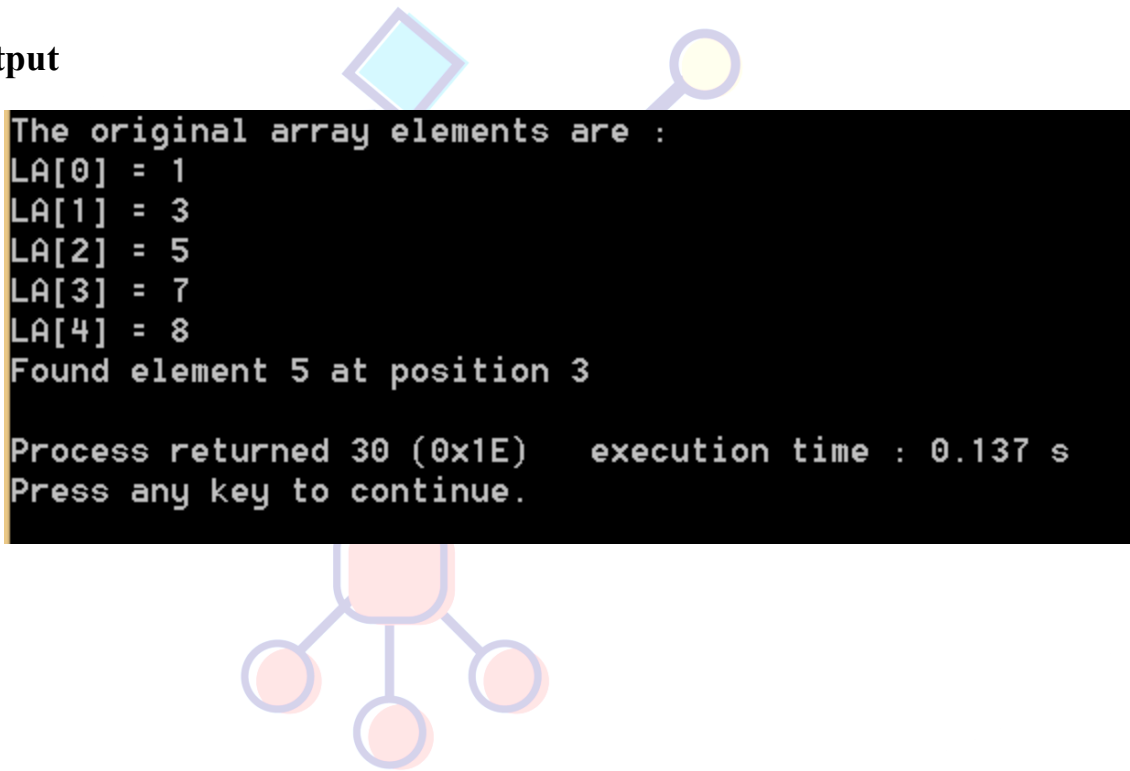
void main() {
    int LA[] = {1,3,5,7,8};
    int item = 5, n = 5;
    int i = 0, j = 0;
    printf("The original array elements are :\n");
    for(i = 0; i<n; i++) {
        printf("LA[%d] = %d \n", i, LA[i]);
    }
    while( j < n){
        if( LA[j] == item ) {
```



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

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

Output



```
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

Process returned 30 (0x1E)   execution time : 0.137 s
Press any key to continue.
```

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 $K \leq N$.

Following is the algorithm to update an element available at the Kth position of LA.

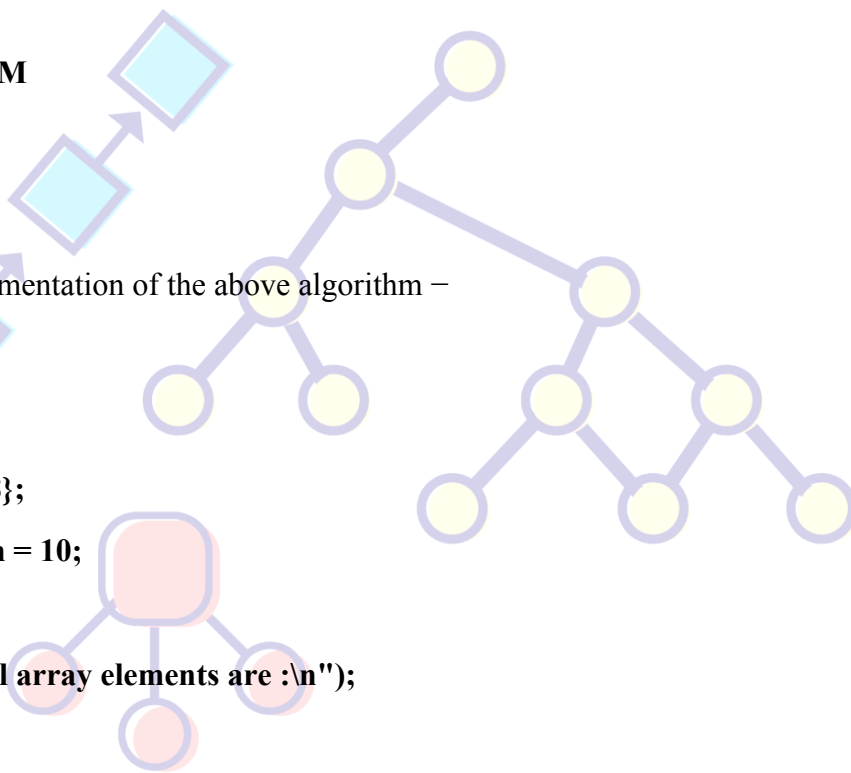
1. Start
2. Set $LA[K-1] = \text{ITEM}$
3. Stop

Example

Following is the implementation of the above algorithm –

```
#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]);
    }
}
```



When we compile and execute the above program, it 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

