IMPLEMENTING ARRAYS

Learning Outcomes

After the completion of this lesson, the student must be able to:

- define arrays;
- understand the different array operations;
- simulate algorithms and basic codes on arrays;
- implement array operations;

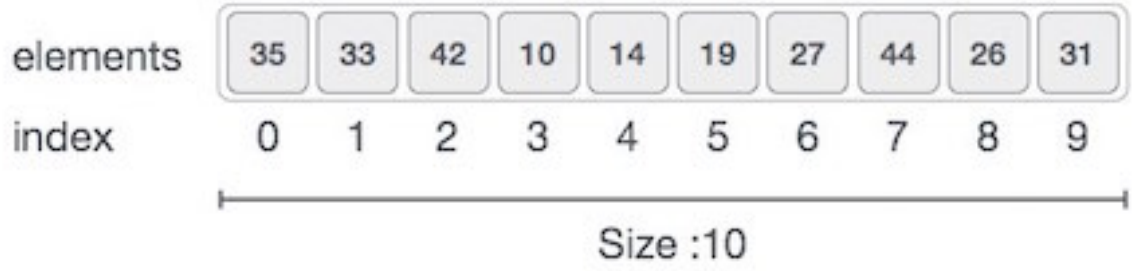
Array is a container which can hold a fix number of items and these items should be of the same type. Most of the data structures make use of arrays to implement their algorithms. Following are the important terms to understand the concept of Array.

- **Element** − Each item stored in an array is called an element.
- **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.

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.

- Index starts with 0.
- Array length is 10 which means it can store 10 elements.
- 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.

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>
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]);
   }
}
```

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 −

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

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**. Following 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] = LA[J + 1]
5. Set J = J+1
6. Set N = N-1
7. Stop
```

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

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

```
#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);
}
```

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**. Following is the algorithm to update an element available at the K^{th} position of LA.

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

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

Activity # 2

Part 1 – Simulate and give the output of the following codes on array implementation:

```
import java.util.Arrays;
class Test
{
    public static void main (String[] args)
    {
        int arr1[] = {1, 2, 3};
        int arr2[] = {1, 2, 3};
        if (Arrays.equals(arr1, arr2))
            System.out.println("Same");
        else
            System.out.println("Not same");
    }
}
```

Activity # 2

Part 1 – Simulate and give the output of the following codes on array implementation:

```
import java.util.Arrays;
class solution
{
  public static int diff_between_two_elemnts(int[] nums)
    {
        int diff_two_elemnts = Integer.MIN_VALUE;

        for (int i = 0; i < nums.length - 1; i++) {
            for (int j = i + 1; j < nums.length; j++) {
                diff_two_elemnts = Integer.max(diff_two_elemnts,nums[j] - nums[i]);
            }
        }

        return diff_two_elemnts;
    }

    public static void main(String[] args)
    {
        int[] nums = { 2, 3, 1, 7, 9, 5, 11, 3, 5 };
        System.out.println("\nOriginal array: "+Arrays.toString(nums));

        System.out.print("The maximum difference between two elements of the said array elements\n" + diff_between_two_elemnts(nums));
    }
}
```

Activity # 2

Part 2 – Write a program that will produce the following output:

Write a Java program to find minimum sub-array sum of specified size in a given array of integers.

Example:
Input :
num = { 1, 2, 3, 4, 5, 6, 7, 8, 9,10}

Output:
Sub-array size: 4
Sub-array from 0 to 3 and sum is: 10

Activity # 2

Part 2 – Write a program that will produce the following output:

Write a Java program to sort a given array of distinct integers where all its numbers are sorted except two numbers.
Example:

Input :
num1 = { 3, 5, 6, 9, 8, 7 }
num2 = { 5, 0, 1, 2, 3, 4, -2 }

Output:
After sorting new array becomes: [3, 5, 6, 7, 8, 9]
After sorting new array becomes: [-2, 0, 1, 2, 3, 4, 5]

Activity # 2

Part 2 – Write a program that will produce the following output:

Write a Java program to arrange the elements of a given array of integers where all positive integers appear before all the negative integers.

Note:

User must input 10 integers.
Output must be sorted in ascending order

Reference/s:

https://www.tutorialspoint.com/data_structures_algorithms/index.htm