Array

An array is a primitive linear data structure in which similar type of data elements are stored in contiguous memory locations i.e., one after another at one place.

Inserting an element at the given position in 1-D Array

The insert operation stores the given element at a desired location in an array.
Let us suppose we are given an array with a memory map as follows:

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
5	10	15	20	25
1000	1002	1004	1006	1008

Suppose we want to store the element 12 in the 3rd position.
At first, we must make space at the 3rd position in the array. To achieve this, elements 25, 20, 15 are shifted to the right one place from the end up to the fourth place as shown below.

Making space to insert 12 at third position

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]
5	10	~~15~~	15	20	25
1000	1002	1004	1006	1008	1010

Then, an element 12 is stored at the desired location as follows:

Inserting 12 at third position

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]
5	10	12	15	20	25
1000	1002	1004	1006	1008	1010

Finally, increase the length of an array by one, as an element is inserted into it.

Algorithm to Insert an Element at Desired Location is as follows:

Algorithm: Insertion of an element in an array

```
Insert [array, length, position, item]
1.  Start
2.  Repeat steps 3 and 4 for c = length down to position
3.  Set array [c + 1] = array [c]
4.  Set c = c-1
    [End of loop]
5.  Set array [position] = item
6.  Set length = length + 1
7.  End
```

/*Program to insert an element at a given position in an array*/

```
#include <stdio.h>
#define MAX 20

void input(int*,int);
void output(int*,int);
void insert_element(int*, int*, int, int);

int main()
{
		int array[MAX], position, n, item;
		printf("Enter number of elements in an array\t:");
		scanf("%d", &n);
		input(array, n);
		output(array, n);
		printf("Enter the location where you wish to insert an element\n");
		scanf("%d", &position);
		printf("Enter the value to insert\n");
		scanf("%d", &item);
		insert_element(array, &n, position, item);
		printf("\nThe modified array after inserting an element is as follows:\n");
		output(array, n);
	return 0;
}
```

```
void insert_element(int*a, int*l, int pos, int item)
{
                int i;
                for (i = *l - 1; i >= pos - 1; i--)
                {
                                 a[i+1] = a[i];
                 }
        a[pos-1] = item;
        *l = *l + 1;
}

void input(int *a, int n)
{
                int i;
                for (i=0; i<n; i++)
                {
                                printf("Enter an element at position %d", i+1);
                                scanf("%d", a+i);
                  }
 }

void output(int *a, int n)
{
                int i;
                printf("\nThe elements in an array are as follows:\n");
                for(i=0; i<n; i++)
                {
                                printf("%d", *(a+i));
                }
}
```

Deleting an Element from 1-D Array

Let us suppose we are given an array with a memory map as follows:

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]
5	10	12	15	20	25
1000	1002	1004	1006	1008	1010

Suppose we want to delete the element stored at the 4th position, i.e., 15. At first, we will store 15 in a temporary variable, say, *temp* as below:

<u>Copy an element 15 to a temporary variable</u>

temp = arr[3];

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]
5	10	12	~~15~~	20	25
1000	1002	1004	1006	1008	1010

Then, shift the elements 20, 25 to the left one place.

<u>Rearrange the elements</u>

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
5	10	12	20	25
1000	1002	1004	1006	1008

Finally, decrease the length of an array by one as one element is deleted from it.

Algorithm to Delete an Element from an Array is as follows:

Algorithm: Delete an element from array

```
Delete_element_array [array, position, length]
1.  Start
2.  Set item = array [position]
3.  Repeat for i = position to length
        i.  Set array [i] = array [i+1]
        ii. Set i= i+1
    [End of loop]
4.  Reset length = length – 1
5.  End
```

/*Program to delete an element from an array*/

```
#include <stdio.h>
#define MAX 20
void input(int*,int);
void output(int*,int);
void del_element(int*, int*, int);

int main()
{
		int array[MAX], n, item;
		printf("Enter number of elements to be stored\t:");
		scanf("%d", &n);
		input(array, n);
		output(array, n);
		printf("Enter an element to delete\t");
		scanf("%d", &item);
		del_element(array, &n, item);
		printf("\nThe modified array after deleting an element is as follows:\n");
		output(array, n);
   getch();
   return 0;
}

void del_element(int*a, int*lp, int item)
{
	int i, pos, found;
	 for (i = 0; i < *lp; i++)
	 {
			 if (a[i] == item)
			{
					found = 1;
					pos = i;
					break;
			}
	 }
```

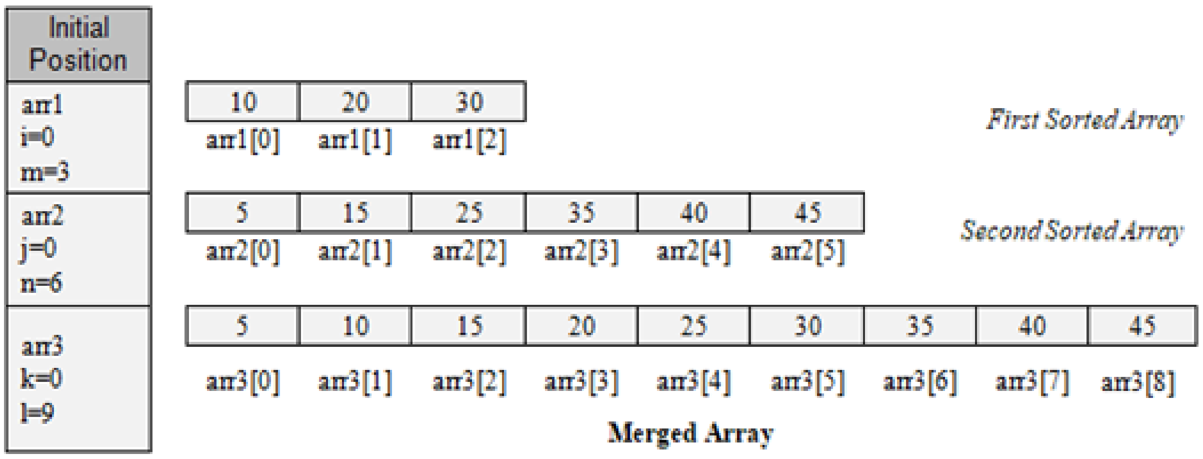
```
            if (found == 1)
             {
                          for (i = pos; i <  *lp - 1; i++)
                          {
                          a[i] = a[i + 1];
                          }
                                 *lp = *lp - 1;
             }
             else
             printf("Item %d is not found\n", item);
   }
}

void input(int *a, int n)
{
                int i;
                for (i=0; i<n; i++)
                {
                                printf("Enter an element at position %d", i+1);
                                scanf("%d", a+i);
               }
 }

void output(int *a, int n)
{
                int i;
                printf("\nThe elements in an array are as follows:\n");
                for(i=0; i<n; i++)
                {
                                printf("%d", *(a+i));
                }
}
```

Traversing an Array
The operation traverse is to access each element of array only once.

The algorithm to traverse an array is as below:
Algorithm: Traversal through an array

```
Traverse_array [arr, LB, UB]
1.  Start
2.  Set i=LB
3.  Repeat for i = LB to UB
     i. Display array[i]
    ii. Set i = i + 1
    [End of loop]
4.  End
```

Merging Two Arrays
(Merging after sorting the arrays)



We compare the elements of *arr1* and *arr2* one by one. If the element of *arr1* is less than the element of *arr2*, we store it in the resultant array. Once any of the arrays gets exhausted, we stop comparing the elements. The remaining elements of the unexhausted array are copied one by one in the resultant array until that array gets exhausted. We can understand the concept of the merge algorithm as given below.

Algorithm: Merge two sorted arrays

```
Merge_two _sorted_arrays [array1, array2, array3, length1, length2]
1.  Start
2.  Set i=0, j=0, k= 0
3.  Repeat the steps 4 to 10 while (i<length1 and j<length2)
4.   Check if array1[i] < array2[j]
5.         Set array3[k] = array1[i]
6.         Set i = i+1
      [End of if statement]
7.  Else
8.        Set array3[k] = array2[j]
9.        Set j = j+1
      [End of Else statement]
10. Set k = k + 1
    [End of while loop]
11. Assign the remaining elements of either array1 or array2 not exhausted yet in array3.
12. End
```

/*Program to merge two sorted arrays*/

```
#include <stdio.h>
void input(int*,int);
void output(int*,int);
void merge_arrays (int*, int*, int*,int,int);

int main()
{
        int array1[20], array2[20], array3[40], l, m;
        printf("Enter number of elements in first array\t:");
        scanf("%d", &l);
        input(array1, l);
        output(array1, n);
        printf("Enter number of elements in second array\t:");
        scanf("%d", &m);
        input(array2, m);
        output(array2, m);
```

```
          merge_arrays(array1, array2, array3, l, m);
          printf("\nThe merged array is as follows:\n");
          output(array3, l+m);
  getch();
  return 0;
}
void merge_arrays(int *a1, int *a2, int *a3, int l, int m)
{
          int i=0, j=0, k=0;
          while(i<l && j<m)
          {
                if (a1[i] < a2[j])
                {
                                a3[k] = a1[i];
                        i++;
                }
                        else
                {
                                a3[k] = a2[j];
                                j++;
                }
           k++;
         }
        if (l == 1)                     //The first list is exhausted
        {
                while (j < m)
                {
                        a3[k] = a2[j];
                        j++;
                        k++;
                }
        }
        if (j >= m)             //The second list is exhausted
        {
                while (i < l)
                {
                        array3[k] = array1[i];
                        i++;
                        k++;
                }
        }
}

void input(int *a, int n)
{
        int i;
        for (i=0; i<n; i++)
        {
                printf("Enter an element at position %d", i+1);
                scanf("%d", a+i);
        }
}
void output(int *a, int n)
{
        int i;
        printf("\nThe elements in an array are as follows:\n");
        for(i=0; i<n; i++)
        {
                printf("%d", *(a+i));
        }
}
```

Assignment Data Structures: 03

1. Define an array? How is it represented in a memory?
2. What are the applications of arrays? Give suitable examples to show the usefulness of an array.
3. Write a menu driven 'C' program having the following functionalities:

    i. Input an array
    ii. Output an array
    iii. Insert an element at any given position.
    iv. Delete any given element
    v. Quit from a program

4. Write a 'C' program to bubble sort and then merge two arrays.
5. For a single dimensional array, if the base address is 2000 and a data element stored in this array needs only 3 bytes. Find the address of the 6th indexed element?
6. What is the output of following code:

```
int main()
{
    int array[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    printf("%u %u %u\n", array[0]+1, *(array[0]+1), *(*(array+0)+1));
  return 0;
}
```

a) 4678   3456   2 b) 2   2   2
c) 3756   2   2 d) None of the options

7. int main()

```
{
    int a[2][3][4] = { {1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 1, 2},
                       {2, 1, 4, 7, 6, 7, 8, 9, 0, 0, 0, 0} };
    printf("%u, %u, %u, %d\n", a, *a, **a, ***a);
  return 0;
} //addresses are arbitrarily taken
```

a) 1002, 2004, 4008, 2 b) 2004, 4008, 8016, 1
c) 1002, 1002, 1002, 1 d) Error

8. What is the output of the following code?

```
power(int**);
int main()
{
    int a=5, *ptr; /* Address of 'a' is 100 */
    ptr = &a;
    a = power(&ptr);
    printf("%d\n", a);
  return 0;
}
power(int **ptr)
{
    int p;
    p = **ptr***ptr;
  return (p);
}
```

a) 10000 b) 5
c) 15 d) 25