Arrays

2.1 Introduction to Array:

- An array is a collection of similar data elements. These data elements have the same data type. The elements of the array are stored in consecutive memory locations and are referenced by an index (also known as the subscript). The subscript is an ordinal number which is used to identify an element of the array:

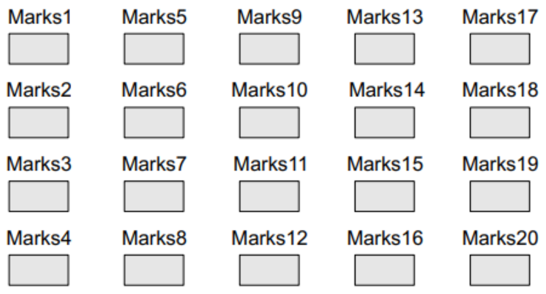


Figure 2.1: Twenty variables for 20 students.

2.2 DECLARATION OF ARRAYS

An array must be declared before being used. Declaring an array means specifying the following:

- Data type—the kind of values it can store, for example, int, char, float, double.
- Name—to identify the array.
- Size—the maximum number of values that the array can hold

Arrays are declared using the following syntax:

type name[size];

The type can be either int, float, double, char, or any other valid data type. The number within brackets indicates the size of the array, i.e., the maximum number of elements that can be stored in the array. For example, if we write,

int marks[10];

then the statement declares marks to be an array containing 10 elements. In C, the array index starts from zero. The first element will be stored in marks[0], second element in marks[1], and so on. Therefore, the last element, that is the 10th element, will be stored in marks[9]. Note that 0, 1, 2, 3 written within square brackets are the subscripts. In the memory, the array will be stored as shown in Fig. 2.2.

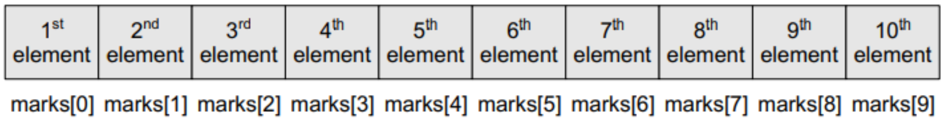


Figure 2.2: Memory representation of an array of 10 elements.

Figure 2.3 shows how different types of arrays are declared.

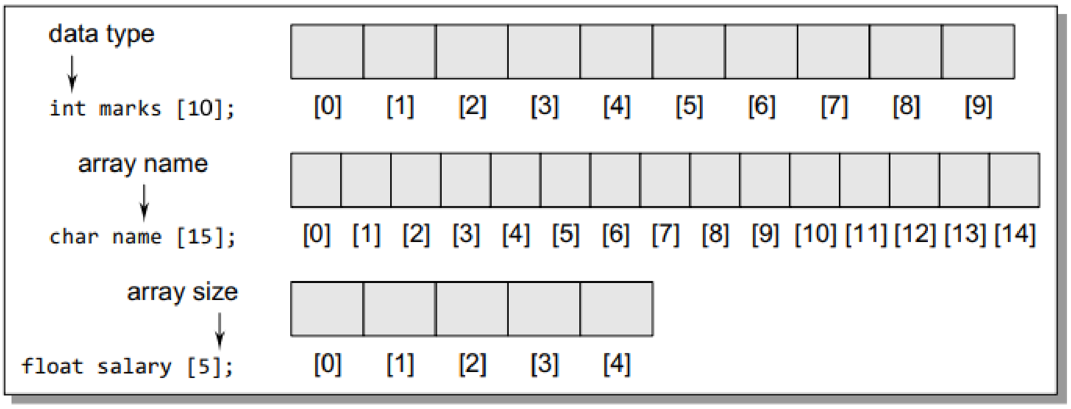


Figure 2.3: Declaring arrays of different data types and sizes.

2.3 ACCESSING THE ELEMENTS OF AN ARRAY

Storing related data items in a single array enables the programmers to develop concise and efficient programs.

To access all the elements, we must use a loop. Now to process all the elements of the array, we use a loop as shown in **Fig. 2.4.**

```
// Set each element of the array to -1
int i, marks[10];
for(i=0;i<10;i++)
        marks[i] = -1;
```

Figure 2.4: Declaring arrays of different data types and sizes.

Figure 3.5 shows the result of the code shown in Fig. 3.4. The code accesses every individual element of the array and sets its value to –1. In the for loop, first the value of marks[0] is set to –1, then the value of the index (i) is incremented and the next value, that is, marks[1] is set to –1. The procedure continues until all the 10 elements of the array are set to –1.

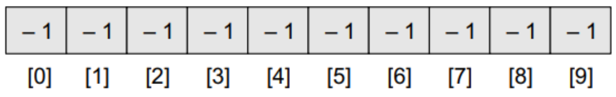


Figure 2.5: Array marks after executing the code given in Fig. 2.4.

Note There is no single statement that can read, access, or print all the elements of an array. To do this, we have to use a loop to execute the same statement with different index values.

2.3.1 Calculating the Address of Array Elements

You must be wondering how C gets to know where an individual element of an array is located in the memory. The answer is that the array name is a symbolic reference to the address of the first byte of the array.

When we use the array name, we are actually referring to the first byte of the array.

Since an array stores all its data elements in consecutive memory locations, storing just the base address, that is the address of the first element in the array, is sufficient. The address of other data elements can simply be calculated using the base address. The formula to perform this calculation is,

Address of data element, A[k] = BA(A) + w(k)

Here,

A is the array,

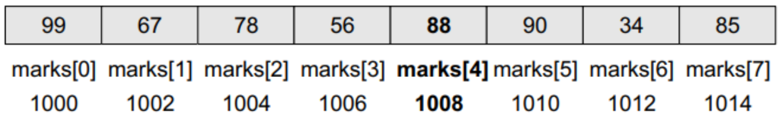
k is the index of the element of which we have to calculate the address,

BA is the base address of the array **A**,

and **w** is the size of one element in memory, for example, size of int is **2**.

Example 3.1 Given an array int marks[ ]={99,67,78,56,88,90,34,85}, calculate the address of marks[4] if the base address = 1000.

Solution



We know that storing an integer value requires 2 bytes, therefore, its size is 2 bytes.

marks[4] = 1000 + 2(4 – 0) = 1000 + 2(4) = 1008

2.4 STORING VALUES IN ARRAYS

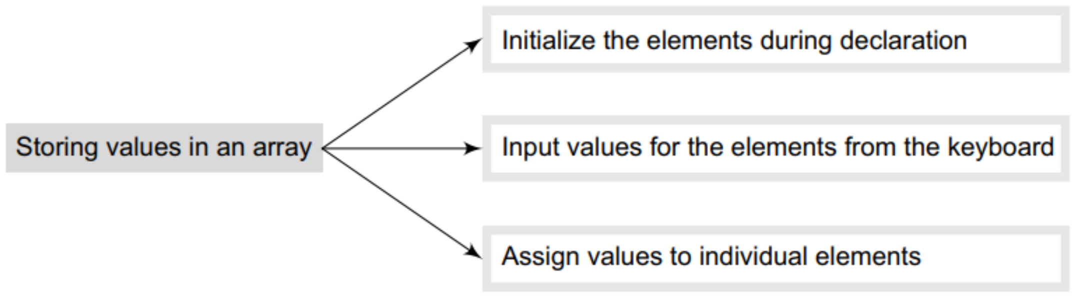

Storing values in an array
Initialize the elements during declaration
Input values for the elements from the keyboard
Assign values to individual elements

Figure 2.6: Storing values in an array.

Initializing Arrays during Declaration:

Arrays are initialized by writing,

type array_name[size]={list of values};

int marks[5]={90, 82, 78, 95, 88};

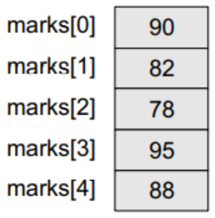
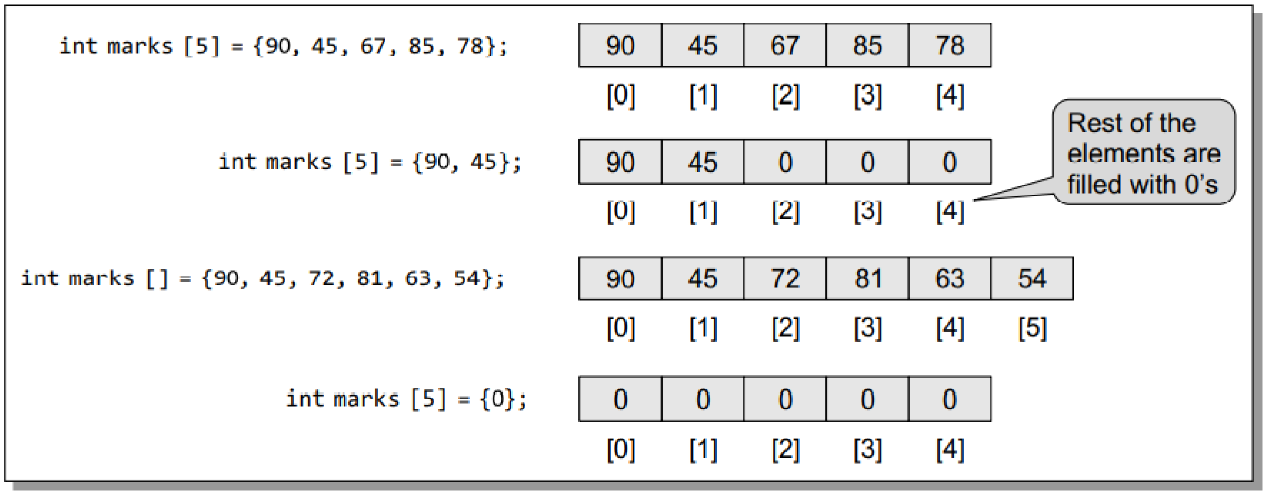


Figure 2.7: Initialization of array marks[5].

While initializing the array at the time of declaration, the programmer may omit the size of the array.
For example,

int marks[ ]= {98, 97, 90};

Figure 2.8 shows the initialization of arrays.



```
int marks [5] = {90, 45, 67, 85, 78};
int marks [5] = {90, 45};
int marks [] = {90, 45, 72, 81, 63, 54};
int marks [5] = {0};
```

Figure 2.8: Initialization of array elements.

Inputting Values from the Keyboard

An array can be initialized by inputting values from the keyboard. In this method, a while/do–while or a for loop is executed to input the value for each element of the array. For example, look at the code shown in **Fig. 2.9**

```
int i, marks[10];
for(i=0;i<10;i++)
      scanf("%d", &marks[i]);
```

Figure 2.9: Code for inputting each element of the array.

Assigning Values to Individual Elements

The third way is to assign values to individual elements of the array by using the assignment operator. Any value that evaluates to the data type as that of the array can be assigned to the individual array element. A simple assignment statement can be written as

marks[3] = 100;

```
int i, arr1[10], arr2[10];
arr1[10] = {0,1,2,3,4,5,6,7,8,9};
for(i=0;i<10;i++)
        arr2[i] = arr1[i];
```

Figure 2.10: Code to copy an array at the individual element level.

```
// Fill an array with even numbers
int i,arr[10];
for(i=0;i<10;i++)
      arr[i] = i*2;
```

Figure 2.11: Code for filling an array with even numbers.

2.5 OPERATIONS ON ARRAYS

There are a number of operations that can be preformed on arrays. These operations include:

- Traversing an array.
- Inserting an element in an array.
- Searching an element in an array.
- Deleting an element from an array.
- Merging two arrays.
- Sorting an array in ascending or descending order.

2.5.1 Traversing an Array

Traversing an array means accessing each and every element of the array for a specific purpose.

Traversing the data elements of an array A can include printing every element, counting the total number of elements, or performing any process on these elements. Since, array is a linear data structure (because all its elements form a sequence), traversing its elements is very simple.

The algorithm for array traversal is given in **Fig. 3.12.**

```
Step 1: [INITIALIZATION] SET I = lower_bound
Step 2: Repeat Steps 3 to 4 while I <= upper_bound
Step 3:      Apply Process to A[I]
Step 4:      SET  I = I + 1
        [END OF LOOP]
Step 5: EXIT
```

Figure 2.12: Algorithm for array traversal.

Programming Examples:

1. Write a program to read and display *n* numbers using an array.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int i, n, arr[20];
        clrscr();
        printf("\n Enter the number of elements in the array : ");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
               printf("\n arr[%d] = ", i);
               scanf("%d",&arr[i]);
        }
        printf("\n The array elements are ");
        for(i=0;i<n;i++)
               printf("\t %d", arr[i]);
        return 0;
}
```

Output

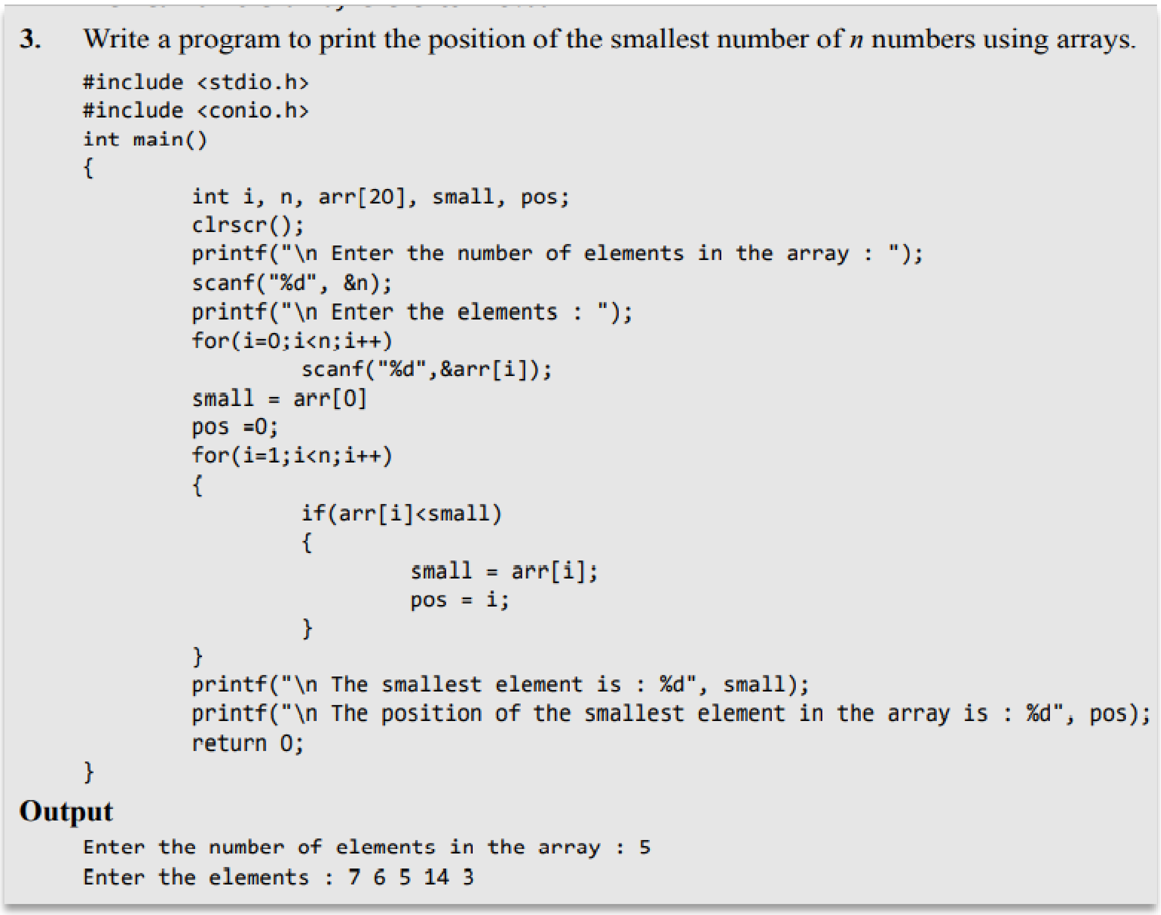
```
Enter the number of elements in the array : 5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
The array elements are    1    2    3    4    5
```

2. Write a program to find the mean of *n* numbers using arrays.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int i, n, arr[20], sum =0;
        float mean = 0.0;
        clrscr();
        printf("\n Enter the number of elements in the array : ");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
                printf("\n arr[%d] = ", i);
                scanf("%d",&arr[i]);
        }
        for(i=0;i<n;i++)
                sum += arr[i];
        mean = (float)sum/n;
        printf("\n The sum of the array elements = %d", sum);
        printf("\n The mean of the array elements = %.2f", mean);
        return 0;
}
```

Output

```
Enter the number of elements in the array : 5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
The sum of the array elements = 15
The mean of the array elements = 3.00
```

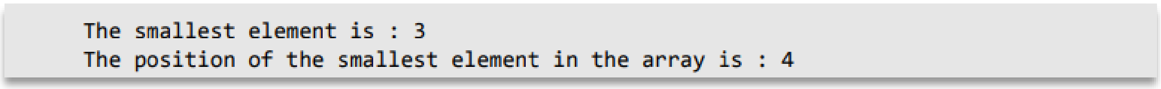


3. Write a program to print the position of the smallest number of *n* numbers using arrays.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int i, n, arr[20], small, pos;
        clrscr();
        printf("\n Enter the number of elements in the array : ");
        scanf("%d", &n);
        printf("\n Enter the elements : ");
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        small = arr[0]
        pos =0;
        for(i=1;i<n;i++)
        {
                if(arr[i]<small)
                {
                        small = arr[i];
                        pos = i;
                }
        }
        printf("\n The smallest element is : %d", small);
        printf("\n The position of the smallest element in the array is : %d", pos);
        return 0;
}
```

Output

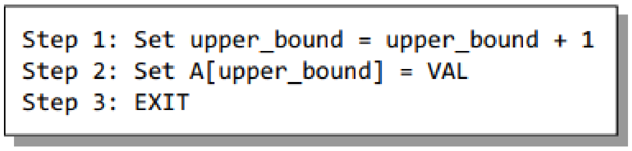
```
Enter the number of elements in the array : 5
Enter the elements : 7 6 5 14 3
```



```
The smallest element is : 3
The position of the smallest element in the array is : 4
```

2.5.2 Inserting an Element in an Array

If an element has to be inserted at the end of an existing array, then the task of insertion is quite simple. We just have to add 1 to the upper_ bound and assign the value. Here, we assume that the memory space allocated for the array is still available. For example, if an array is declared to contain 10 elements, but currently it has only 8 elements, then obviously there is space to accommodate two more elements. **But if it already has 10 elements, then we will not be able to add another element to it.**



```
Step 1: Set upper_bound = upper_bound + 1
Step 2: Set A[upper_bound] = VAL
Step 3: EXIT
```

Figure 2.13: Algorithm to append a new element to an existing array.

Figure 2.13 shows an algorithm to insert a new element to the end of an array. In Step 1, we increment the value of the upper_bound. In Step 2, the new value is stored at the position pointed by the upper_bound. For example, let us assume an array has been declared as

```
int marks[60];
```

The array is declared to store the marks of all the students in a class. Now, suppose there are 54 students and a new student comes and is asked to take the same test. The marks of this new student would be stored in marks[55]. Assuming that the student secured 68 marks, we will assign the value as

```
marks[55] = 68;
```

- **Algorithm to Insert an Element in the Middle of an Array**

The algorithm INSERT will be declared as INSERT (**A, N, POS, VAL**). The arguments are:

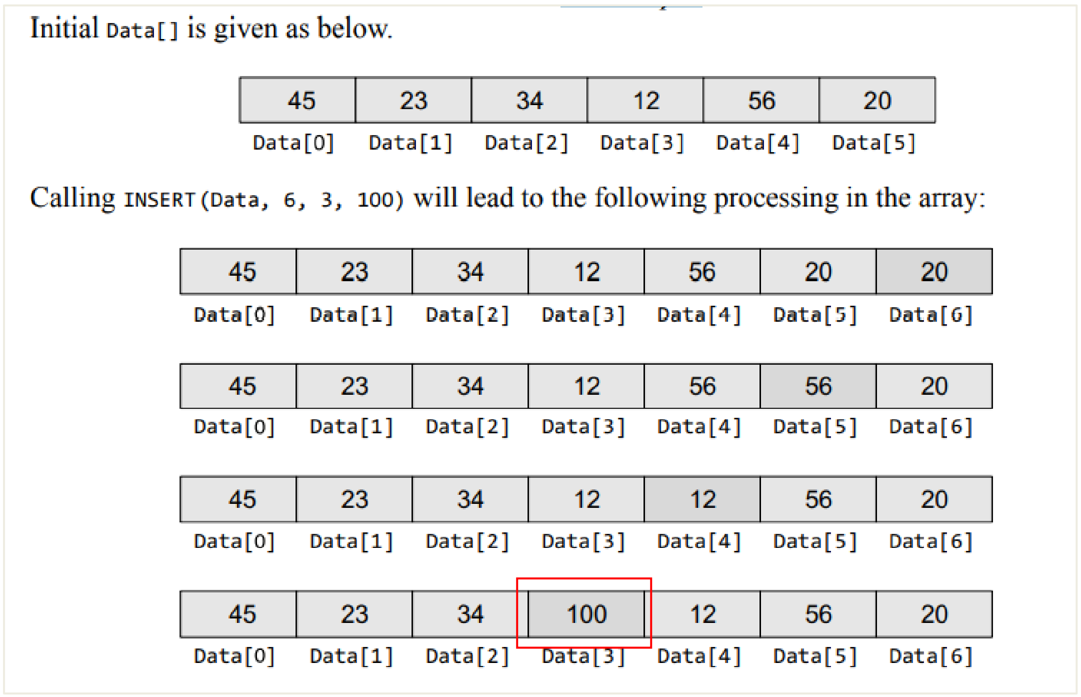
a) **A**, the array in which the element has to be inserted.
b) **N**, the number of elements in the array.
c) **POS**, the position at which the element has to be inserted.
d) **VAL**, the value that has to be inserted.

```
Step 1: [INITIALIZATION] SET I = N
Step 2: Repeat Steps 3 and 4 while I >= POS
Step 3:         SET A[I + 1] = A[I]
Step 4:         SET I = I - 1
        [END OF LOOP]
Step 5: SET N = N + 1
Step 6: SET A[POS] = VAL
Step 7: EXIT
```

Figure 2.14: Algorithm to insert an element in the middle of an array.

In the algorithm given in **Fig. 2.14**, in Step 1, we first initialize I with the total number of elements in the array. In Step 2, a while loop is executed which will move all the elements having an index greater than POS one position towards right to create space for the new element. In Step 5, we increment the total number of elements in the array by 1 and finally in Step 6, the new value is inserted at the desired position.

Now, let us visualize this algorithm by taking an example.



Initial `Data[]` is given as below.

45	23	34	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

Calling `INSERT(Data, 6, 3, 100)` will lead to the following processing in the array:

45	23	34	12	56	20	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

45	23	34	12	56	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

45	23	34	12	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

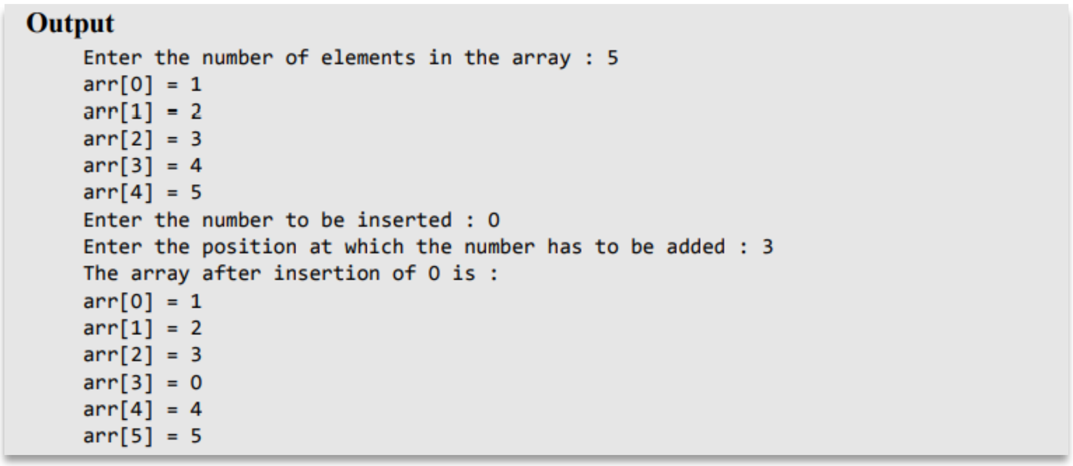
45	23	34	100	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

Programming Examples:

Write a program to insert a number at a given location in an array.

```
#include <stdio.h>
```

```
#include <conio.h>
int main()
{
        int i, n, num, pos, arr[10];
        clrscr();
        printf("\n Enter the number of elements in the array : ");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
                printf("\n arr[%d] = ", i);
                scanf("%d", &arr[i]);
        }
        printf("\n Enter the number to be inserted : ");
        scanf("%d", &num);
        printf("\n Enter the position at which the number has to be added :");
        scanf("%d", &pos);
        for(i=n-1;i>=pos;i--)
                arr[i+1] = arr[i];
        arr[pos] = num;
        n = n+1;
        printf("\n The array after insertion of %d is : ", num);
        for(i=0;i<n;i++)
            printf("\n arr[%d] = %d", i, arr[i]);
        getch();
        return 0;
}
```

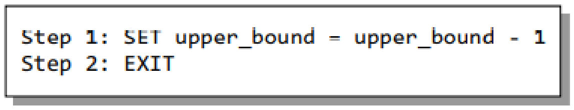


Output

```
Enter the number of elements in the array : 5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
Enter the number to be inserted : 0
Enter the position at which the number has to be added : 3
The array after insertion of 0 is :
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 0
arr[4] = 4
arr[5] = 5
```

2.5.3 Deleting an Element from an Array

Deleting an element from an array means removing a data element from an already existing array. If the element has to be deleted from the end of the existing array, then the task of deletion is quite simple. We just have to subtract 1 from the upper_bound. Figure 3.15 shows an algorithm to delete an element from the end of an array.



```
Step 1: SET upper_bound = upper_bound - 1
Step 2: EXIT
```

Figure 2.15: Algorithm to delete the last element of an array.

The array is declared to store the marks of all the students in the class. Now, suppose there are 54 students and the student with roll number 54 leaves the course. The score of this student was stored in marks[54]. We just have to decrement the upper_bound. Subtracting 1 from the upper_bound will indicate that there are 53 valid data in the array.

• Algorithm to delete an element from the middle of an array

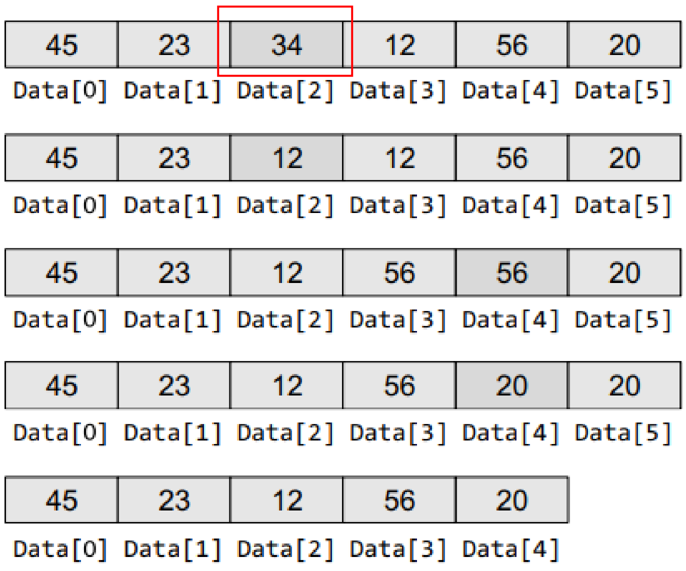
The algorithm DELETE will be declared as DELETE(A, N, POS). The arguments are:

a) **A**, the array from which the element has to be deleted.
b) **N**, the number of elements in the array.
c) **POS**, the position from which the element has to be deleted.

```
Step 1: [INITIALIZATION] SET I = POS
Step 2: Repeat Steps 3 and 4 while I <= N - 1
Step 3:             SET A[I] = A[I + 1]
Step 4:             SET I = I + 1
            [END OF LOOP]
Step 5: SET N = N - 1
Step 6: EXIT
```

Figure 2.16: Algorithm to delete an element from the middle of an array.

Figure 2.16 shows the algorithm in which we first initialize I with the position from which the element has to be deleted. In Step 2, a while loop is executed which will move all the elements having an index greater than POS one space towards left to occupy the space vacated by the deleted element. When we say that we are deleting an element, actually we are overwriting the element with the value of its successive element. In Step 5, we decrement the total number of elements in the array by 1.


45 23 34 12 56 20
Data[0] Data[1] Data[2] Data[3] Data[4] Data[5]
45 23 12 12 56 20
Data[0] Data[1] Data[2] Data[3] Data[4] Data[5]
45 23 12 56 56 20
Data[0] Data[1] Data[2] Data[3] Data[4] Data[5]
45 23 12 56 20 20
Data[0] Data[1] Data[2] Data[3] Data[4] Data[5]
45 23 12 56 20
Data[0] Data[1] Data[2] Data[3] Data[4]

Programming Examples:

9. Write a program to delete a number from a given location in an array.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int i, n, pos, arr[10];
        clrscr();
        printf("\n Enter the number of elements in the array : ");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
                printf("\n arr[%d] = ", i);
                scanf("%d", &arr[i]);
        }
        printf("\nEnter the position from which the number has to be deleted : ");
        scanf("%d", &pos);
        for(i=pos; i<n-1;i++)
                arr[i] = arr[i+1];
        n--;
        printf("\n The array after deletion is : ");
        for(i=0;i<n;i++)
                printf("\n arr[%d] = %d", i, arr[i]);
        getch();
        return 0;
}
```

Output

```
Enter the number of elements in the array : 5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
Enter the position from which the number has to be deleted : 3
The array after deletion is :
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 5
```

2.5.4 Merging Two Arrays

Merging two arrays in a third array means first copying the contents of the first array into the third array and then copying the contents of the second array into the third array. Hence, the merged array contains the contents of the first array followed by the contents of the second array.

If the arrays are unsorted, then merging the arrays is very simple, as one just needs to copy the contents of one array into another. But merging is not a trivial task when the two arrays are sorted and the merged array also needs to be sorted. Let us first discuss the merge operation on unsorted arrays. This operation is shown in **Fig 3.17**.

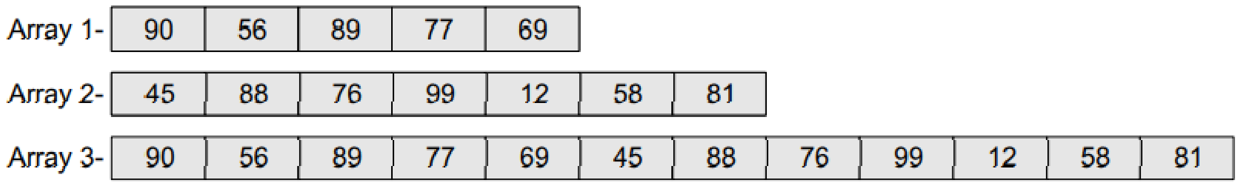

Array 1- 90 56 89 77 69
Array 2- 45 88 76 99 12 58 81
Array 3- 90 56 89 77 69 45 88 76 99 12 58 81

Figure 2.17: Merging of two unsorted arrays.

Programming Examples:

Write a program to merge two unsorted arrays.

```
#include <stdio.h>
#include <conio.h>
int main()
```

```
{
        int arr1[10], arr2[10], arr3[20];
        int i, n1, n2, m, index=0;
        clrscr();
        printf("\n Enter the number of elements in array1 : ");
        scanf("%d", &n1);
        printf("\n\n Enter the elements of the first array");
        for(i=0;i<n1;i++)
        {
                printf("\n arr1[%d] = ", i);
                scanf("%d", &arr1[i]);
        }
        printf("\n Enter the number of elements in array2 : ");
        scanf("%d", &n2);
        printf("\n\n Enter the elements of the second array");
        for(i=0;i<n2;i++)
        {
                printf("\n arr2[%d] = ", i);
                scanf("%d", &arr2[i]);
        }
        m = n1+n2;
        for(i=0;i<n1;i++)
        {
            arr3[index] = arr1[i];
            index++;
        }
        for(i=0;i<n2;i++)
        {
                arr3[index] = arr2[i];
                index++;
        }
        printf("\n\n The merged array is");
        for(i=0;i<m;i++)
                printf("\n arr[%d] = %d", i, arr3[i]);
        getch();
        return 0;
}
```

Output

```
Enter the number of elements in array1 : 3
Enter the elements of the first array
arr1[0] = 1
arr1[1] = 2
arr1[2] = 3
Enter the number of elements in array2 : 3
Enter the elements of the second array
arr2[0] = 4
arr2[1] = 5
arr2[2] = 6
The merged array is
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
arr[5] = 6
```