ARRAY INSERTIONS

We have learnt how the insertion operation works. It is not always necessary that an element is inserted at the end of an array. Following can be a situation with array insertion –

- Insertion at the beginning of an array
- Insertion at the given index of an array
- Insertion after the given index of an array
- Insertion before the given index of an array

Insertion at the Beginning of an Array

When the insertion happens at the beginning, it causes all the existing data items to shift one step downward. Here, we design and implement an algorithm to insert an element at the beginning of an array.

Algorithm

We assume **A** is an array with **N** elements. The maximum numbers of elements it can store is defined by **MAX**. We shall first check if an array has any empty space to store any element and then we proceed with the insertion process.

```
begin

IF N = MAX, return
ELSE
   N = N + 1

   For All Elements in A
      Move to next adjacent location

   A[FIRST] = New_Element

end
```

Implementation in C

```
#include <stdio.h>
#define MAX 5

void main() {
   int array[MAX] = {2, 3, 4, 5};
   int N = 4;     // number of elements in array
   int i = 0;     // loop variable
   int value = 1; // new data element to be stored in array

   // print array before insertion
   printf("Printing array before insertion -\n");
```

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

   // now shift rest of the elements downwards
   for(i = N; i >= 0; i--) {
      array[i+1] = array[i];
   }

   // add new element at first position
   array[0] = value;

   // increase N to reflect number of elements
   N++;

   // print to confirm
   printf("Printing array after insertion -\n");

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

This program should yield the following output –

Output

```
Printing array before insertion -
array[0] = 2
array[1] = 3
array[2] = 4
array[3] = 5
Printing array after insertion -
array[0] = 0
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5
```

Insertion at the Given Index of an Array

In this scenario, we are given the exact location (**index**) of an array where a new data element (**value**) needs to be inserted. First we shall check if the array is full, if it is not, then we shall move all data elements from that location one step downward. This will make room for a new data element.

Algorithm

We assume **A** is an array with **N** elements. The maximum numbers of elements it can store is defined by **MAX**.

```
begin

IF N = MAX, return
```

```
ELSE
   N = N + 1

   SEEK Location index

   For All Elements from A[index] to A[N]
      Move to next adjacent location

   A[index] = New_Element

end
```

Implementation in C

```
#include <stdio.h>
#define MAX 5

void main() {
   int array[MAX] = {1, 2, 4, 5};

   int N = 4;         // number of elements in array
   int i = 0;         // loop variable
   int index = 2;     // index location to insert new value
   int value = 3;     // new data element to be inserted

   // print array before insertion
   printf("Printing array before insertion -\n");

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

   // now shift rest of the elements downwards
   for(i = N; i >= index; i--) {
      array[i+1] = array[i];
   }

   // add new element at first position
   array[index] = value;

   // increase N to reflect number of elements
   N++;

   // print to confirm
   printf("Printing array after insertion -\n");

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

If we compile and run the above program, it will produce the following result –

Output

```
Printing array before insertion -
array[0] = 1
array[1] = 2
array[2] = 4
array[3] = 5
Printing array after insertion -
array[0] = 1
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5
```

Insertion After the Given Index of an Array

In this scenario we are given a location (**index**) of an array after which a new data element (**value**) has to be inserted. Only the seek process varies, the rest of the activities are the same as in the previous example.

Algorithm

We assume **A** is an array with **N** elements. The maximum numbers of elements it can store is defined by **MAX**.

```
begin

IF N = MAX, return
ELSE
   N = N + 1

   SEEK Location index

   For All Elements from A[index + 1] to A[N]
      Move to next adjacent location

   A[index + 1] = New_Element

end
```

Implementation in C

```
#include <stdio.h>
#define MAX 5

void main() {
   int array[MAX] = {1, 2, 4, 5};

   int N = 4;        // number of elements in array
   int i = 0;        // loop variable
   int index = 1;    // index location after which value will be inserted
   int value = 3;    // new data element to be inserted

   // print array before insertion
   printf("Printing array before insertion -\n");
```

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

   // now shift rest of the elements downwards
   for(i = N; i >= index + 1; i--) {
      array[i + 1] = array[i];
   }

   // add new element at first position
   array[index + 1] = value;

   // increase N to reflect number of elements
   N++;

   // print to confirm
   printf("Printing array after insertion -\n");

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

If we compile and run the above program, it will produce the following result –

Output

```
Printing array before insertion -
array[0] = 1
array[1] = 2
array[2] = 4
array[3] = 5
Printing array after insertion -
array[0] = 1
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5
```

Insertion Before the Given Index of an Array

In this scenario we are given a location (**index**) of an array before which a new data element (**value**) has to be inserted. This time we seek till **index-1** i.e., one location ahead of given index, rest of the activities are same as in previous example.

Algorithm

We assume **A** is an array with **N** elements. The maximum numbers of elements it can store is defined by **MAX**.

```
begin

IF N = MAX, return
ELSE
```

```
   N = N + 1

   SEEK Location index

   For All Elements from A[index - 1] to A[N]
      Move to next adjacent location

   A[index - 1] = New_Element

end
```

Implementation in C

```
#include <stdio.h>
#define MAX 5

void main() {
   int array[MAX] = {1, 2, 4, 5};

   int N = 4;         // number of elements in array
   int i = 0;         // loop variable
   int index = 3;     // index location before which value will be inserted
   int value = 3;     // new data element to be inserted

   // print array before insertion
   printf("Printing array before insertion -\n");

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

   // now shift rest of the elements downwards
   for(i = N; i >= index + 1; i--) {
      array[i + 1] = array[i];
   }

   // add new element at first position
   array[index + 1] = value;

   // increase N to reflect number of elements
   N++;

   // print to confirm
   printf("Printing array after insertion -\n");

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

If we compile and run the above program, it will produce the following result –

Output

```
Printing array before insertion -
array[0] = 1
array[1] = 2
array[2] = 4
array[3] = 5
Printing array after insertion -
array[0] = 1
array[1] = 2
array[2] = 4
array[3] = 5
array[4] = 3
```