**Array Operation**

**Sample Declaration of an array:**

// A sample program for Array Declaration

#include <stdio.h>

int main()

{

int one\_dim [10];    # declaration of 1D array

int two\_dim [2][2];  #declaration of 2D array

int three\_dim [2][3][4] = {

{ {3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2} },

{ {13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9} }

}; #declaration of 3D array. Here the elements are also defined.

return 0;

}

**various operations that can be performed on arrays.**

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

For(c=lb;c<=ub;c++)

Printf(“%d”, A[c]);

**Insertion**

It is not always necessary that an element is inserted at the end of an array. Various situation for insertion in Array:

* 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.

### Example:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Initial | Itr 1 | Itr 2 | Itr 3 | Itr 4 | Itr 4 | Itr5 | Insert |
| |  | | --- | | 10 | | 20 | | 30 | | 40 | | 50 | | 60 | |  | | |  | | --- | | 10 | | 20 | | 30 | | 40 | | 50 | |  | | 60 | | |  | | --- | | 10 | | 20 | | 30 | | 40 | |  | | 50 | | 60 | | |  | | --- | | 10 | | 20 | | 30 | |  | | 40 | | 50 | | 60 | | |  | | --- | | 10 | | 20 | |  | | 30 | | 40 | | 50 | | 60 | | |  | | --- | | 10 | |  | | 20 | | 30 | | 40 | | 50 | | 60 | | |  | | --- | |  | | 10 | | 20 | | 30 | | 40 | | 50 | | 60 | | |  | | --- | | X | | 10 | | 20 | | 30 | | 40 | | 50 | | 60 | |

### Algorithm

**Insert(A, New\_Element)**

* 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, display “Error”, return

ELSE

N = N + 1

FIRST = 1

// For All Elements in A

// Move to next adjacent location

For C = 1 to N-1

A[C+1] = A[C]

A[FIRST] = New\_Element

end

Will the loop Work? (New Element = 5)

For C = 1 to N-1

A[C+1] = A[C]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Initial | C = 1 | C=2 | C=3 | C=4 | C=5 | New Value |
| |  | | --- | | 10 | | 20 | | 30 | | 40 | | 50 | | 60 | |  | | |  | | --- | | 10 | | 10 | | 30 | | 40 | | 50 | | 60 | |  | | |  | | --- | | 10 | | 10 | | 10 | | 40 | | 50 | | 60 | |  | | |  | | --- | | 10 | | 10 | | 10 | | 10 | | 50 | | 60 | |  | | |  | | --- | | 10 | | 10 | | 10 | | 10 | | 10 | | 60 | |  | | |  | | --- | | 10 | | 10 | | 10 | | 10 | | 10 | | 10 | |  | | |  | | --- | | 5… | | 10 | | 10 | | 10 | | 10 | | 10 | | 10 | |

Actual Expected Loop

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Initial | C=5 | C=4 | Itr 3 | Itr 4 | Itr 4 | Itr5 | Goal |
| |  | | --- | | 10 | | 20 | | 30 | | 40 | | 50 | | 60 | |  | | |  | | --- | | 10 | | 20 | | 30 | | 40 | | 50 | | 60 | | 60 | | |  | | --- | | 10 | | 20 | | 30 | | 40 | | 50 | | 50 | | 60 | | |  | | --- | | 10 | | 20 | | 30 | | 40 | | 40 | | 50 | | 60 | | |  | | --- | | 10 | | 20 | | 30 | | 30 | | 40 | | 50 | | 60 | | |  | | --- | | 10 | | 20 | | 20 | | 30 | | 40 | | 50 | | 60 | | |  | | --- | | 10 | | 10 | | 20 | | 30 | | 40 | | 50 | | 60 | | |  | | --- | | X.. | | 10 | | 20 | | 30 | | 40 | | 50 | | 60 | |

begin

IF N = MAX, return

ELSE

N = N + 1

FIRST = 1

// For All Elements in A

// Move to next adjacent location

For C = N-1 to 1 N C A

A[C+1] = A[C] 5 - 10,20,30,40,50,0

6 5 10,20.30,40,50,50

// For C = N to 2 6 4 10,20,30,40,40,50

// A[C] = A[C-1]

A[FIRST] = New\_Element

end

### Implementation in C

[Live Demo](http://tpcg.io/sEnxiW)

#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

* 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.

### Example: (with insertion POS = 3)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Initial | Itr 1 | Itr 2 | Itr 3 | Itr 4 | Insert |
| |  | | --- | | 10 | | 20 | | 30 | | 40 | | 50 | | 60 | |  | | |  | | --- | | 10 | | 20 | | 30 | | 40 | | 50 | |  | | 60 | | |  | | --- | | 10 | | 20 | | 30 | | 40 | |  | | 50 | | 60 | | |  | | --- | | 10 | | 20 | | 30 | |  | | 40 | | 50 | | 60 | | |  | | --- | | 10 | | 20 | | ….. | | 30 | | 40 | | 50 | | 60 | | |  | | --- | | 10 | | 20 | | X | | 30 | | 40 | | 50 | | 60 | |

### Algorithm

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

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

For C = N to POS Step -1

A[C+1] = A[C]

A[POS] = New\_Element

end

### Implementation in C

[Live Demo](http://tpcg.io/K3ydXf)

#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

For C = N to POS+1

A[C+1] = A[C]

A[POS+1] = New\_Element

// Or I can write A[C-1] = New Element

end

## 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

If Location Index <1

return

// For All Elements from A[index - 1] to A[N]

// Move to next adjacent location

For C = N to POS-1

A[C+1] = A[C]

A[POS-1] = New\_Element

end

## 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 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

## 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.

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

## 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 Kth position of LA.

1. Start

2. Set LA[K-1] = ITEM

3. Stop