**ARRAY**

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.

**1.** C=1

2. Process LIST[C]

3. C= C+1

4. if (C<=N) then repeat 2 and 3

5. End.

**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.

**i) Insertion at the Beginning of an Array**

begin

IF N = MAX, return

ELSE

N = N + 1

For All Elements in A

Move to next adjacent location

A[FIRST] = New\_Element

end

**ii) Insertion at the given index of an array**

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

###### iii) Insertion after the given index of an array

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

###### iv) Insertion before the given index of an array

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

#### 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

##### **Advantages of an Array:**

1. Random access of elements using array index.
2. Use of less line of code as it creates a single array of multiple elements.
3. Easy access to all the elements.
4. Traversal through the array becomes easy using a single loop.
5. Sorting becomes easy as it can be accomplished by writing less line of code.

##### **Disadvantages of an Array:**

1. Allows a fixed number of elements to be entered which is decided at the time of declaration. Unlike a linked list, an array in C is not dynamic.
2. Insertion and deletion of elements can be costly since the elements are needed to be managed in accordance with the new memory allocation.

**STACK**

A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.



A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from the top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.

This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last, is accessed first. In stack terminology, insertion operation is called PUSH operation and removal operation is called POP operation.

**Stack Representation**

The following diagram depicts a stack and its operations −



A stack can be implemented by means of Array, Structure, Pointer, and Linked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays, which makes it a fixed size stack implementation.

**Basic Operations:**

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −

* push() − Pushing (storing) an element on the stack.
* pop() − Removing (accessing) an element from the stack.

When data is PUSHed onto stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks −

* peek() − get the top data element of the stack, without removing it.
* isFull() − check if stack is full.
* isEmpty() − check if stack is empty.

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named top. The top pointer provides top value of the stack without actually removing it.

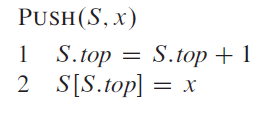
**Push Operation**

The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps −

* Step 1 − Checks if the stack is full.
* Step 2 − If the stack is full, produces an error and exit.
* Step 3 − If the stack is not full, increments top to point next empty space.
* Step 4 − Adds data element to the stack location, where top is pointing.
* Step 5 − Returns success.



If the linked list is used to implement the stack, then in step 3, we need to allocate space dynamically.



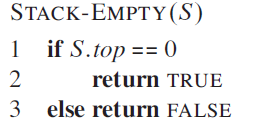
**Pop Operation**

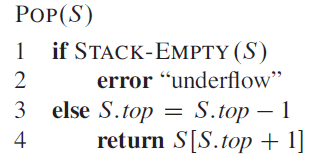
Accessing the content while removing it from the stack, is known as a Pop Operation. In an array implementation of pop() operation, the data element is not actually removed, instead top is decremented to a lower position in the stack to point to the next value. But in linked-list implementation, pop() actually removes data element and reallocates memory space.

A Pop operation may involve the following steps −

* Step 1 − Checks if the stack is empty.
* Step 2 − If the stack is empty, produces an error and exit.
* Step 3 − If the stack is not empty, accesses the data element at which top is pointing.
* Step 4 − Decreases the value of top by 1.
* Step 5 − Returns success.





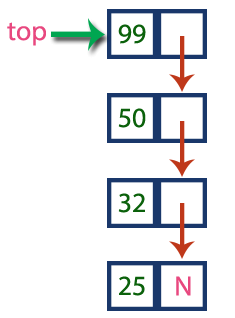


**Stack Using Linked List:**

The major problem with the stack implemented using an array is, it works only for a fixed number of data values. That means the amount of data must be specified at the beginning of the implementation itself. Stack implemented using an array is not suitable, when we don't know the size of data which we are going to use. A stack data structure can be implemented by using a linked list data structure. The stack implemented using linked list can work for an unlimited number of values. That means, stack implemented using linked list works for the variable size of data. So, there is no need to fix the size at the beginning of the implementation. The Stack implemented using linked list can organize as many data values as we want.

In linked list implementation of a stack, every new element is inserted as '**top**' element. That means every newly inserted element is pointed by '**top**'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by '**top**' by moving '**top**' to its previous node in the list. The **next** field of the first element must be always **NULL**.

**Example**



In the above example, the last inserted node is 99 and the first inserted node is 25. The order of elements inserted is 25, 32,50 and 99.

**Stack Operations using Linked List**

To implement a stack using a linked list, we need to set the following things before implementing actual operations.

* **Step 1 -**Include all the **header files** which are used in the program. And declare all the **user defined functions**.
* **Step 2 -**Define a '**Node**' structure with two members **data** and **next**.
* **Step 3 -**Define a **Node** pointer '**top**' and set it to **NULL**.
* **Step 4 -**Implement the **main** method by displaying Menu with list of operations and make suitable function calls in the **main** method.

**push(value) - Inserting an element into the Stack**

We can use the following steps to insert a new node into the stack...

* **Step 1 -**Create a **newNode** with given value.
* **Step 2 -**Check whether stack is **Empty** (**top** == **NULL**)
* **Step 3 -**If it is **Empty**, then set **newNode → next** = **NULL**.
* **Step 4 -**If it is **Not Empty**, then set **newNode → next** = **top**.
* **Step 5 -**Finally, set **top** = **newNode**.

**pop() - Deleting an Element from a Stack**

We can use the following steps to delete a node from the stack...

* **Step 1 -**Check whether **stack** is **Empty** (**top == NULL**).
* **Step 2 -**If it is **Empty**, then display **"Stack is Empty!!! Deletion is not possible!!!"** and terminate the function
* **Step 3 -**If it is **Not Empty**, then define a **Node** pointer '**temp**' and set it to '**top**'.
* **Step 4 -**Then set '**top** = **top → next**'.
* **Step 5 -**Finally, delete '**temp**'. (**free(temp)**).

**display() - Displaying stack of elements**

We can use the following steps to display the elements (nodes) of a stack...

* **Step 1 -**Check whether stack is **Empty** (**top** == **NULL**).
* **Step 2 -**If it is **Empty**, then display **'Stack is Empty!!!'** and terminate the function.
* **Step 3 -**If it is **Not Empty**, then define a Node pointer **'temp'** and initialize with **top**.
* **Step 4 -**Display '**temp → data** --->' and move it to the next node. Repeat the same until **temp** reaches to the first node in the stack. (**temp → next** != **NULL**).
* **Step 5 -**Finally! Display '**temp → data** ---> **NULL**'.

**RELEVANT READING MATERIAL AND REFERENCES:**

**Source Notes:**

1. <https://www.tutorialspoint.com/data_structures_algorithms/array_data_structure.htm>
2. <https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm#:~:text=A%20stack%20is%20an%20Abstract,a%20pile%20of%20plates%2C%20etc.&text=LIFO%20stands%20for%20Last%2Din,)%20last%2C%20is%20accessed%20first.>
3. <http://www.btechsmartclass.com/data_structures/stack-using-linked-list.html>

**Lecture Video:**

1. <https://www.youtube.com/watch?v=hT1dIbFQoGE&list=PLGdMwVKbjVQ8Ew7KUp65sRL9_k2_3xlKE&index=9>
2. <https://www.youtube.com/watch?v=3XCGzrCxVkw&list=PLGdMwVKbjVQ8Ew7KUp65sRL9_k2_3xlKE&index=10>
3. <https://youtu.be/RBSGKlAvoiM>
4. <https://www.youtube.com/watch?v=SE-RkUlheoc>

**Online Notes:**

1. <http://www.crectirupati.com/sites/default/files/lecture_notes/ds%20ln.pdf>
2. <http://www.vssut.ac.in/lecture_notes/lecture1428550942.pdf>

**Text Book Reading:**

1. Cormen, Leiserson, Rivest, Stein, “*Introduction to Algorithms*”, Prentice Hall of India, 3rd edition 2012. problem, Graph coloring.
2. Lipschutz, S., “*Data Structures, Schaum's Outline Series*”, Tata McGraw Hill.

**Online Book Reference:**

1. <https://www.edutechlearners.com/download/books/DS.pdf>

**In addition: PPT can be also be given.**