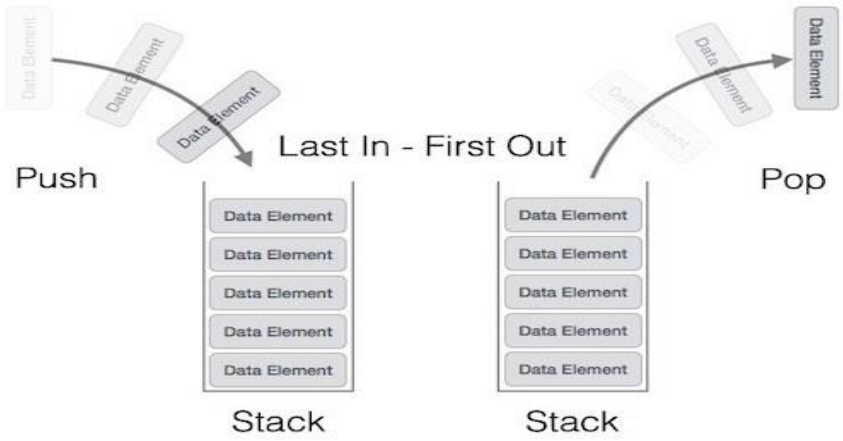
STACK DATA STRUCTURE

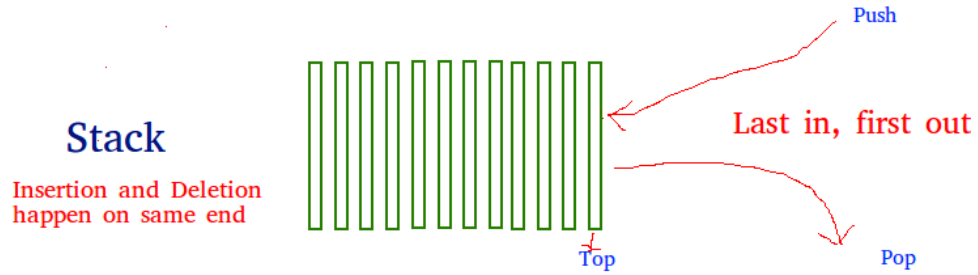
- Stack is a linear data structure which follows a particular order in which the operations are performed.
- It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

- 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 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.
- The element which is placed (inserted or added) last, is accessed first.

Stack Representation

The following diagram depicts a stack and its operations –





- Stack can either be a fixed size one or it may have a sense of dynamic resizing.
- We are implementing stack using arrays, which makes it a fixed size stack implementation.

Basic Operations

- .A stack is used for the following two primary operations

I. **PUSH()**

- Adds an item in the stack.
- If the stack is full, then it is said to be an Overflow condition
- In stack terminology, insertion operation is called PUSH operation.

II. **POP()**

- Removing (accessing) an element from the stack.
- In stack terminology removal operation is called POP operation.
- The items are popped in the reversed order in which they are pushed. (LIFO/FILO)

FILO-First In Last Out

- If the **stack is empty**, then it is said to be an **Underflow condition**

TOP Pointer

- 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 −

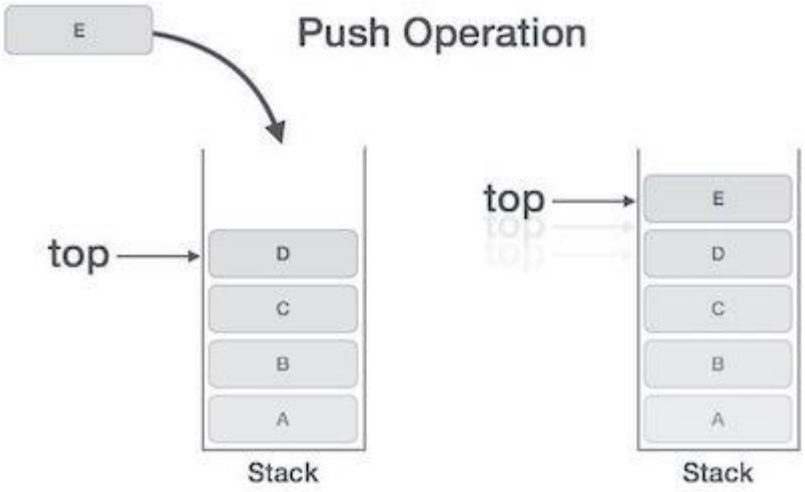
Step 1 − Checks if the stack is full.

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.



Algorithm for PUSH Operation

```
begin procedure push: stack, data

   if stack is full
      return null
   endif

   top ← top + 1
   stack[top] ← data

end procedure
```

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.

A Pop operation may involve the following steps −

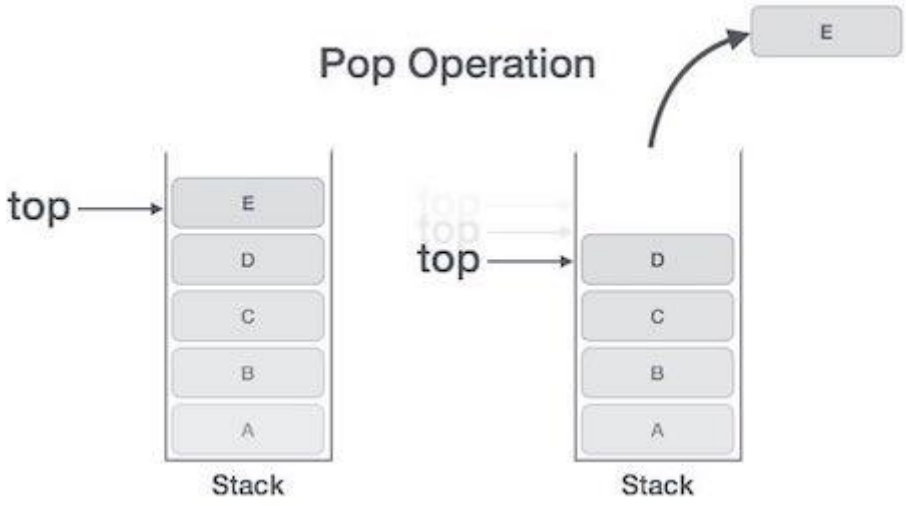
Step 1 − Checks if the stack is empty.

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.



Algorithm for Pop Operation

```
begin procedure pop: stack

   if stack is empty
      return null
   endif

   data ← stack[top]
   top ← top - 1
   return data

end procedure
```

- When elements are added (PUSH Operation) to stack it grow at one end.
- Similarly, when elements are deleted from a stack (POP Operation), it shrinks at the same end.