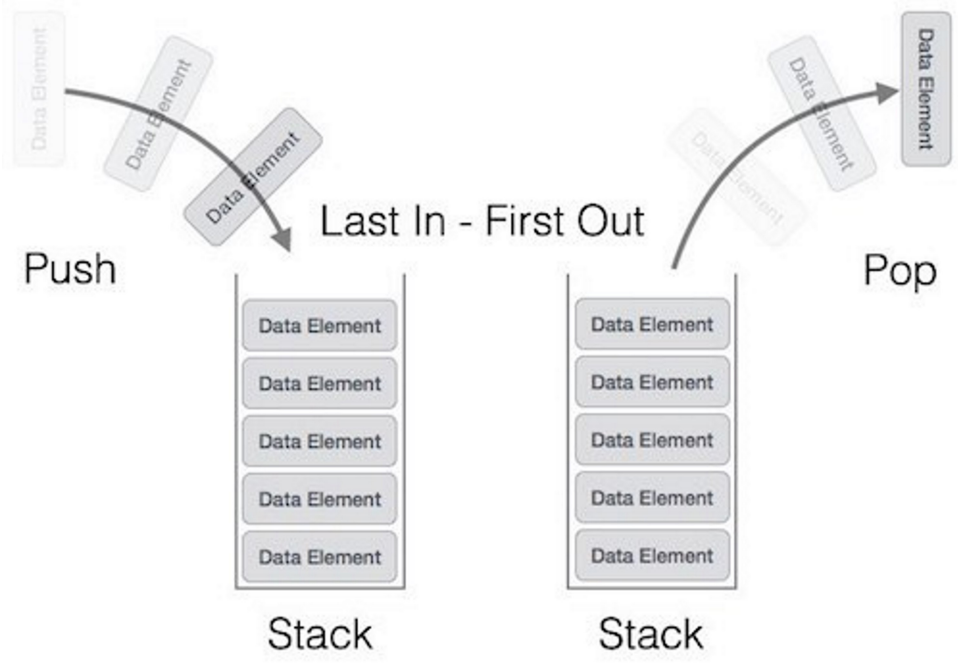
Stacks

05 February 2021 12:23



Stack:

Basic Operations:

1) **push()**
2) **pop()**
3) **peek()**
4) **isFull()**
5) **isEmpty()**
6) **Top Pointer**
7) **Overflow and Underflow Conditions**

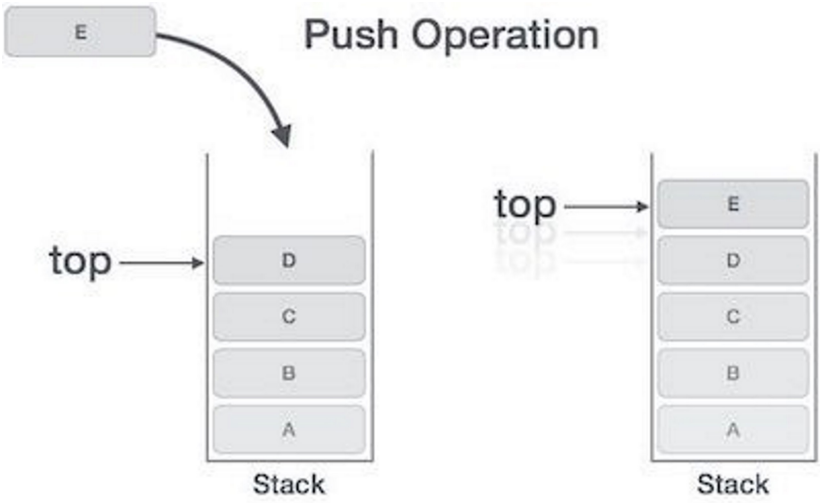
Stack Implementation:

1) **Array**
2) **Linked List**

1) push():

Steps:

- Checks if the stack is full.
- If the stack is full, produces an error and exit.
- If the stack is not full, increments top to point next empty space.
- Adds data element to the stack location, where top is pointing.
- Returns success.



Algorithm:

```
begin procedure push: stack, data

  if stack is full
    return null
  endif

  top ← top + 1
  stack[top] ← data

end procedure
```

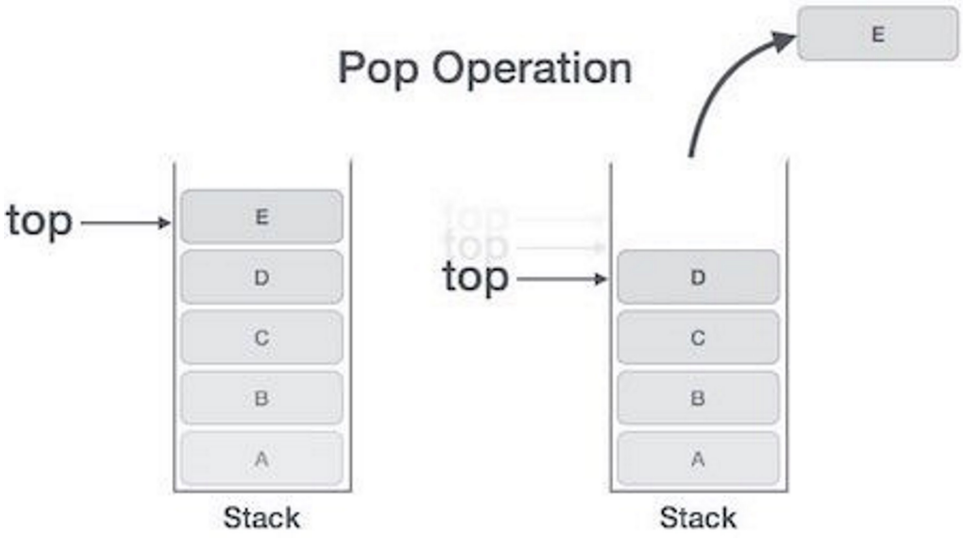
Code:

```
void push(int value) {
  if(top == SIZE-1)
    printf("\nOverflow. Stack is Full");
  else{
    top++;
    stack[top] = value;
    printf("\nInsertion was successful");
  }
}
```

2) pop():

Steps:

- Checks if the stack is empty.
- If the stack is empty, produces an error and exit.
- If the stack is not empty, accesses the data element at which top is pointing.
- Decreases the value of top by 1.
- Returns success.



Algorithm:

```
begin procedure pop: stack

  if stack is empty
    return null
  endif

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

end procedure
```

Code:

```
void pop() {
  if(top == -1)
    printf("\nUnderflow. Stack is empty");
  else{
    printf("\nDeleted : %d", stack[top]);
    top--;
  }
}
```

3) peek():

Steps:

- Check whether stack is EMPTY (top == -1).
- If it is EMPTY, then terminate the function and throw an error.
- If it is NOT EMPTY, then return stack[top].

Code:

```
void peek() {
  if(top == -1)
  {
    printf("\n The stack is empty");
    break;
  }
  else
    printf("%d", stack[top]);
}
```

Applications:

1) UNDO functionality in text editors
2) Valid Parenthesis
3) Next Greater Elements

Complexity:

1) Access: O(n)
2) Search: O(n)
3) Insertion: O(1)
4) Deletion: O(1)
5) Space: O(n)