**Standard Stack Operations**

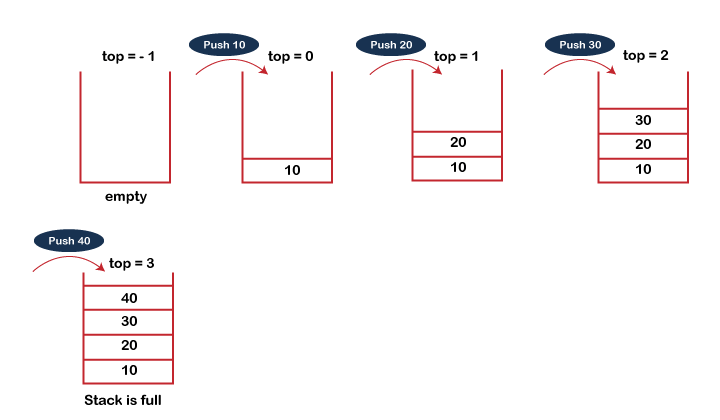
**The following are some common operations implemented on the stack:**

* **push():** When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
* **pop():** When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.
* **isEmpty():** It determines whether the stack is empty or not.
* **isFull():** It determines whether the stack is full or not.'
* **peek():** It returns the element at the given position.
* **count():** It returns the total number of elements available in a stack.
* **change():** It changes the element at the given position.
* **display():** It prints all the elements available in the stack.

**PUSH operation**

**The steps involved in the PUSH operation is given below:**

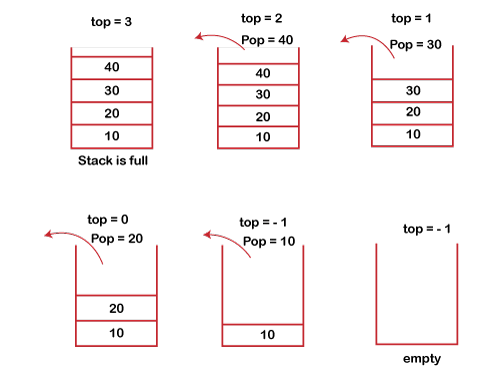
* Before inserting an element in a stack, we check whether the stack is full.
* If we try to insert the element in a stack, and the stack is full, then the ***overflow*** condition occurs.
* When we initialize a stack, we set the value of top as -1 to check that the stack is empty.
* When the new element is pushed in a stack, first, the value of the top gets incremented, i.e., **top=top+1,** and the element will be placed at the new position of the **top**.
* The elements will be inserted until we reach the ***max*** size of the stack.



**POP operation**

**The steps involved in the POP operation is given below:**

* Before deleting the element from the stack, we check whether the stack is empty.
* If we try to delete the element from the empty stack, then the ***underflow*** condition occurs.
* If the stack is not empty, we first access the element which is pointed by the ***top***
* Once the pop operation is performed, the top is decremented by 1, i.e., **top=top-1**.



**Applications of Stack**

**The following are the applications of the stack:**

* **Balancing of symbols:** Stack is used for balancing a symbol. For example, we have the following program:

1. int main()
2. {
3. cout<<"Hello";
4. cout<<"javaTpoint";
5. }

As we know, each program has *an opening* and *closing* braces; when the opening braces come, we push the braces in a stack, and when the closing braces appear, we pop the opening braces from the stack. Therefore, the net value comes out to be zero. If any symbol is left in the stack, it means that some syntax occurs in a program.

* **String reversal:** Stack is also used for reversing a string. For example, we want to reverse a "**javaTpoint**" string, so we can achieve this with the help of a stack.  
  First, we push all the characters of the string in a stack until we reach the null character.  
  After pushing all the characters, we start taking out the character one by one until we reach the bottom of the stack.
* **UNDO/REDO:** It can also be used for performing UNDO/REDO operations. For example, we have an editor in which we write 'a', then 'b', and then 'c'; therefore, the text written in an editor is abc. So, there are three states, a, ab, and abc, which are stored in a stack. There would be two stacks in which one stack shows UNDO state, and the other shows REDO state.  
  If we want to perform UNDO operation, and want to achieve 'ab' state, then we implement pop operation.
* **Recursion:** The recursion means that the function is calling itself again. To maintain the previous states, the compiler creates a system stack in which all the previous records of the function are maintained.
* **DFS(Depth First Search):** This search is implemented on a Graph, and Graph uses the stack data structure.
* **Backtracking:** Suppose we have to create a path to solve a maze problem. If we are moving in a particular path, and we realize that we come on the wrong way. In order to come at the beginning of the path to create a new path, we have to use the stack data structure.
* **Expression conversion:** Stack can also be used for expression conversion. This is one of the most important applications of stack. The list of the expression conversion is given below:
* Infix to prefix
* Infix to postfix
* Prefix to infix
* Prefix to postfix

Postfix to infix

* **Memory management:** The stack manages the memory. The memory is assigned in the contiguous memory blocks. The memory is known as stack memory as all the variables are assigned in a function call stack memory. The memory size assigned to the program is known to the compiler. When the function is created, all its variables are assigned in the stack memory. When the function completed its execution, all the variables assigned in the stack are released.