**What is Stack?**

*A stack is a linear data structure in which the insertion of a new element and removal of an existing element takes place at the same end represented as the top of the stack.*

To implement the stack, it is required to maintain the **pointer to the top of the stack**, which is the last element to be inserted because **we can access the elements only on the top of the stack.**

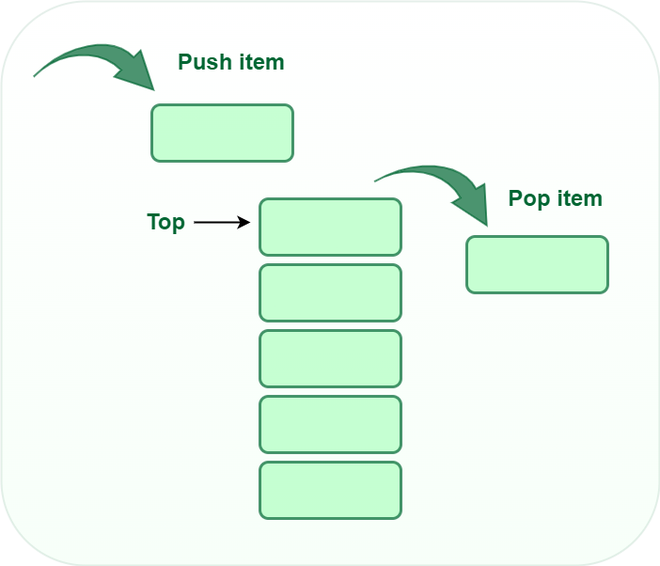
**LIFO( Last In First Out ):**

*This strategy states that the element that is inserted last will come out first. You can take a pile of plates kept on top of each other as a real-life example. The plate which we put last is on the top and since we remove the plate that is at the top, we can say that the plate that was put last comes out first.*

**Basic Operations on Stack**

To make manipulations in a stack, there are certain operations provided to us.

* **push()** to insert an element into the stack
* **pop()**to remove an element from the stack
* **top()** Returns the top element of the stack.
* **isEmpty()**returns true if the stack is empty else false.
* **size()** returns the size of the stack.



**Push:**

Adds an item to the stack. If the stack is full, then it is said to be an**Overflow condition.**

**Algorithm for push:**

begin

if stack is full

return

endif

else

increment top

stack[top] assign value

end else

end procedure

**Pop:**

Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an **Underflow** **condition.**

**Algorithm for pop:**

begin

if stack is empty

return

endif

else

store value of stack[top]

decrement top

return value

end else

end procedure

**Top:**

Returns the top element of the stack.

**Algorithm for Top:**

begin

return stack[top]

end procedure

**isEmpty:**

Returns true if the stack is empty, else false.

**Algorithm for isEmpty**:

begin

if top < 1

return true

else

return false

end procedure

**Types of Stacks:**

* **Fixed Size Stack**: As the name suggests, a fixed size stack has a fixed size and cannot grow or shrink dynamically. If the stack is full and an attempt is made to add an element to it, an overflow error occurs. If the stack is empty and an attempt is made to remove an element from it, an underflow error occurs.
* **Dynamic Size Stack**: A dynamic size stack can grow or shrink dynamically. When the stack is full, it automatically increases its size to accommodate the new element, and when the stack is empty, it decreases its size. This type of stack is implemented using a linked list, as it allows for easy resizing of the stack.

In addition to these two main types, there are several other variations of Stacks, including:

1. **Infix to Postfix Stack**: This type of stack is used to convert infix expressions to postfix expressions.
2. **Expression Evaluation Stack**: This type of stack is used to evaluate postfix expressions.
3. **Recursion Stack**: This type of stack is used to keep track of function calls in a computer program and to return control to the correct function when a function returns.
4. **Memory Management Stack**: This type of stack is used to store the values of the program counter and the values of the registers in a computer program, allowing the program to return to the previous state when a function returns.
5. **Balanced Parenthesis Stack**: This type of stack is used to check the balance of parentheses in an expression.
6. **Undo-Redo Stack**: This type of stack is used in computer programs to allow users to undo and redo actions.

**Applications of the stack:**

* [Infix to Postfix](https://www.geeksforgeeks.org/stack-set-2-infix-to-postfix/) /Prefix conversion
* Redo-undo features at many places like editors, photoshop.
* Forward and backward features in web browsers
* Used in many algorithms like [Tower of Hanoi,](https://www.geeksforgeeks.org/recursive-functions/)[tree traversals](https://www.geeksforgeeks.org/618/), [stock span problems](https://www.geeksforgeeks.org/the-stock-span-problem/), and [histogram problems](https://www.geeksforgeeks.org/largest-rectangular-area-in-a-histogram-set-1/).
* Backtracking is one of the algorithm designing techniques. Some examples of backtracking are the Knight-Tour problem, N-Queen problem, find your way through a maze, and game-like chess or checkers in all these problems we dive into someway if that way is not efficient we come back to the previous state and go into some another path. To get back from a current state we need to store the previous state for that purpose we need a stack.
* In Graph Algorithms like [Topological Sorting](https://www.geeksforgeeks.org/topological-sorting/) and [Strongly Connected Components](https://www.geeksforgeeks.org/strongly-connected-components/)
* In Memory management, any modern computer uses a stack as the primary management for a running purpose. Each program that is running in a computer system has its own memory allocations
* String reversal is also another application of stack. Here one by one each character gets inserted into the stack. So the first character of the string is on the bottom of the stack and the last element of a string is on the top of the stack. After Performing the pop operations on the stack we get a string in reverse order.
* Stack also helps in implementing function call in computers. The last called function is always completed first.
* Stacks are also used to implement the undo/redo operation in text editor.

**Implementation of Stack:**

A stack can be implemented using an array or a linked list. In an array-based implementation, the push operation is implemented by incrementing the index of the top element and storing the new element at that index. The pop operation is implemented by decrementing the index of the top element and returning the value stored at that index. In a linked list-based implementation, the push operation is implemented by creating a new node with the new element and setting the next pointer of the current top node to the new node. The pop operation is implemented by setting the next pointer of the current top node to the next node and returning the value of the current top node.

Stacks are commonly used in computer science for a variety of applications, including the evaluation of expressions, function calls, and memory management. In the evaluation of expressions, a stack can be used to store operands and operators as they are processed. In function calls, a stack can be used to keep track of the order in which functions are called and to return control to the correct function when a function returns. In memory management, a stack can be used to store the values of the program counter and the values of the registers in a computer program, allowing the program to return to the previous state when a function returns.

In conclusion, a Stack is a linear data structure that operates on the LIFO principle and can be implemented using an array or a linked list. The basic operations that can be performed on a stack include push, pop, and peek, and stacks are commonly used in computer science for a variety of applications, including the evaluation of expressions, function calls, and memory management.There are two ways to implement a stack –

* Using array
* Using linked list

# Python program for implementation of stack

# import maxsize from sys module

# Used to return -infinite when stack is empty

from sys import maxsize

# Function to create a stack. It initializes size of stack as 0

def createStack():

stack = []

return stack

# Stack is empty when stack size is 0

def isEmpty(stack):

return len(stack) == 0

# Function to add an item to stack. It increases size by 1

def push(stack, item):

stack.append(item)

print(item + " pushed to stack ")

# Function to remove an item from stack. It decreases size by 1

def pop(stack):

if (isEmpty(stack)):

return str(-maxsize -1) # return minus infinite

return stack.pop()

# Function to return the top from stack without removing it

def peek(stack):

if (isEmpty(stack)):

return str(-maxsize -1) # return minus infinite

return stack[len(stack) - 1]

# Driver program to test above functions

stack = createStack()

push(stack, str(10))

push(stack, str(20))

push(stack, str(30))

print(pop(stack) + " popped from stack")