**MY DSA DOCUMENT**

**1.STACKS**

Mainly the following three basic operations are performed in the stack:

* **Push:**Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
* **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.
* **Peek or Top:** Returns top element of stack.
* **isEmpty:**Returns true if stack is empty, else false.



There are two ways to implement a stack:

* Using array
* Using linked list

**CODE:**

**1.Array**- <https://www.geeksforgeeks.org/stack-data-structure-introduction-program/>

**Pros:** Easy to implement. Memory is saved as pointers are not involved.   
**Cons:** It is not dynamic. It doesn’t grow and shrink depending on needs at runtime.

**2.Linked List**-https://www.geeksforgeeks.org/stack-data-structure-introduction-program/

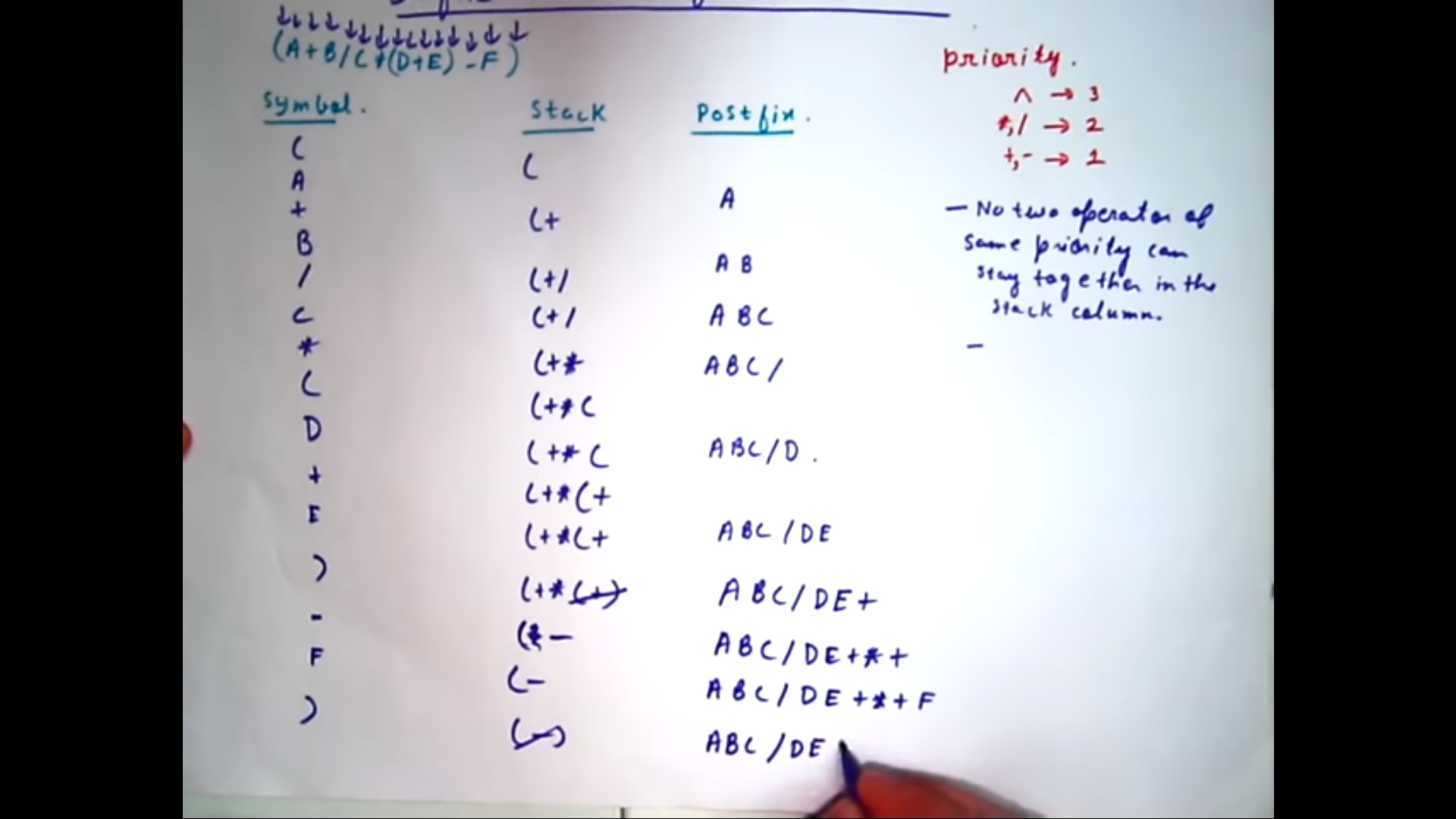
**Pros:** The linked list implementation of stack can grow and shrink according to the needs at runtime.   
**Cons:** Requires extra memory due to involvement of pointers.

**3.STL CODE**: <https://www.geeksforgeeks.org/stack-in-cpp-stl/>

The functions associated with stack are:  
[empty()](https://www.geeksforgeeks.org/stack-empty-and-stack-size-in-c-stl/) – Returns whether the stack is empty – Time Complexity : O(1)  
[size()](https://www.geeksforgeeks.org/stack-empty-and-stack-size-in-c-stl/) – Returns the size of the stack – Time Complexity : O(1)  
[top()](https://www.geeksforgeeks.org/stack-top-c-stl/) – Returns a reference to the top most element of the stack – Time Complexity : O(1)  
[push(g)](https://www.geeksforgeeks.org/stack-push-and-pop-in-c-stl/) – Adds the element ‘g’ at the top of the stack – Time Complexity : O(1)  
[pop()](https://www.geeksforgeeks.org/stack-push-and-pop-in-c-stl/) – Deletes the top most element of the stack – Time Complexity : O(1)

**4.Infix to Postflix**

Algoritthm:



<https://www.youtube.com/watch?v=vXPL6UavUeA>

CODE:<https://www.geeksforgeeks.org/stack-set-2-infix-to-postfix/?ref=lbp>

**2.QUEUE**

**Enqueue:**Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.  
**Dequeue:** Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.  
**Front:**Get the front item from queue.  
**Rear:** Get the last item from queue.



1.Array:<https://www.geeksforgeeks.org/queue-set-1introduction-and-array-implementation/>

2.Linked List:<https://www.geeksforgeeks.org/queue-linked-list-implementation/>

3.QUEUE STL

**The functions supported by queue are :**

1. [empty()](https://www.geeksforgeeks.org/queueempty-queuesize-c-stl/) – Returns whether the queue is empty.
2. [size()](https://www.geeksforgeeks.org/queueempty-queuesize-c-stl/) – Returns the size of the queue.
3. [queue::swap() in C++ STL](https://www.geeksforgeeks.org/queue-swap-cpp-stl/): Exchange the contents of two queues but the queues must be of same type, although sizes may differ.
4. [queue::emplace() in C++ STL](https://www.geeksforgeeks.org/queueemplace-c-stl/): Insert a new element into the queue container, the new element is added to the end of the queue.
5. [queue::front() and queue::back() in C++ STL](https://www.geeksforgeeks.org/queuefront-queueback-c-stl/)– **front()** function returns a reference to the first element of the queue. **back()** function returns a reference to the last element of the queue.
6. [push(g) and pop()](https://www.geeksforgeeks.org/queuepush-and-queuepop-in-cpp-stl/) – **push()** function adds the element ‘g’ at the end of the queue. **pop()** function deletes the first element of the queue.

CODE:<https://www.geeksforgeeks.org/queue-cpp-stl/>

4.Priority Queue

Priority Queue is an extension of [queue](http://quiz.geeksforgeeks.org/queue-set-1introduction-and-array-implementation/)with following properties.

1. Every item has a priority associated with it.
2. An element with high priority is dequeued before an element with low priority.
3. If two elements have the same priority, they are served according to their order in the queue.
4. A typical priority queue supports following operations.  
   **insert(item, priority):**Inserts an item with given priority.  
   **getHighestPriority():** Returns the highest priority item.  
   **deleteHighestPriority():**Removes the highest priority item.
5. **How to implement priority queue?**  
   ***Using Array:***A simple implementation is to use array of following structure.

struct item {

int item;

int priority;

}

1. insert() operation can be implemented by adding an item at end of array in O(1) time.
2. getHighestPriority() operation can be implemented by linearly searching the highest priority item in array. This operation takes O(n) time.
3. deleteHighestPriority() operation can be implemented by first linearly searching an item, then removing the item by moving all subsequent items one position back.
4. We can also use Linked List, time complexity of all operations with linked list remains same as array. The advantage with linked list is deleteHighestPriority() can be more efficient as we don’t have to move items.

5.Dequeue

allows insert and delete at both ends.

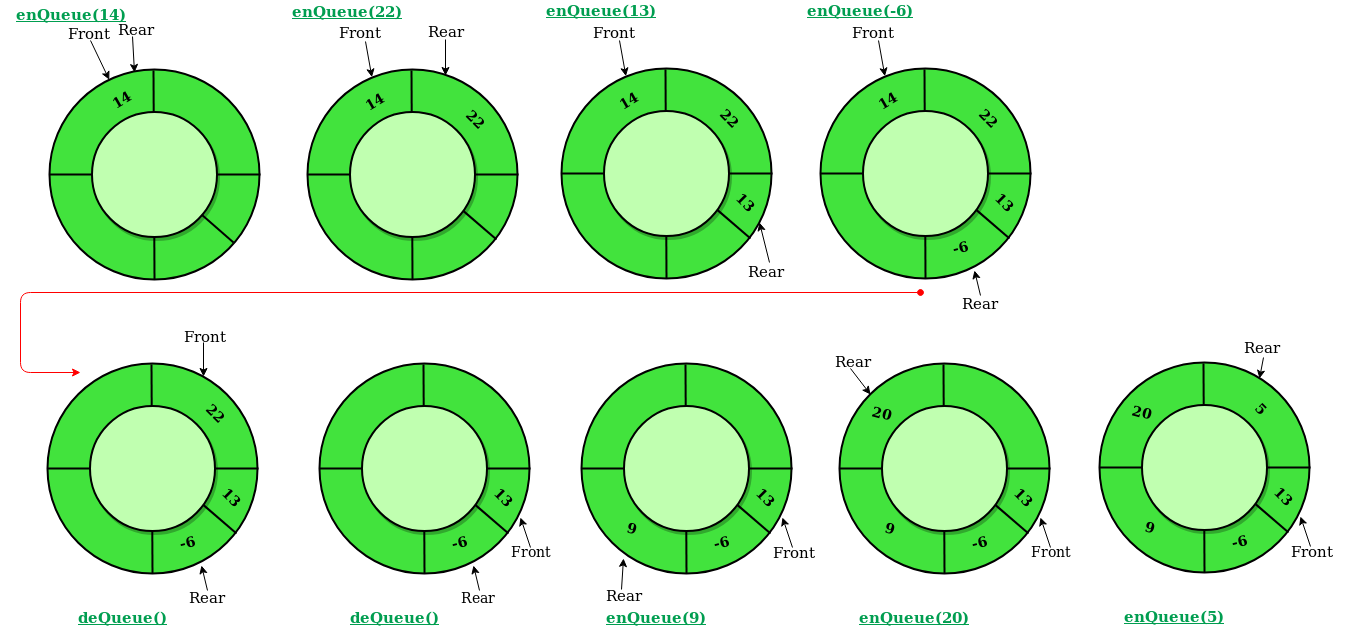
**Operations on Deque:**  
Mainly the following four basic operations are performed on queue:

***insertFront()***: Adds an item at the front of Deque.  
***insertLast()***: Adds an item at the rear of Deque.  
***deleteFront()***: Deletes an item from front of Deque.  
***deleteLast()***: Deletes an item from rear of Deque.

In addition to above operations, following operations are also supported  
***getFront()***: Gets the front item from queue.  
***getRear()***: Gets the last item from queue.  
***isEmpty()***: Checks whether Deque is empty or not.  
***isFull()***: Checks whether Deque is full or not.

**Applications of Deque:**  
Since Deque supports both stack and queue operations, it can be used as both. The Deque data structure supports clockwise and anticlockwise rotations in O(1) time which can be useful in certain applications.  
Also, the problems where elements need to be removed and or added both ends can be efficiently solved using Deque.

6.CIRCULAR QUEUE



Operations on Circular Queue:

* **Front:** Get the front item from queue.
* **Rear:** Get the last item from queue.
* **enQueue(value)**This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.

**Steps:**

* 1. Check whether queue is Full – Check ((rear == SIZE-1 && front == 0) || (rear == front-1)).
  2. If it is full then display Queue is full. If queue is not full then, check if (rear == SIZE – 1 && front != 0) if it is true then set rear=0 and insert element.
* **deQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position.

**Steps:**

* 1. Check whether queue is Empty means check (front==-1).
  2. If it is empty then display Queue is empty. If queue is not empty then step 3
  3. Check if (front==rear) if it is true then set front=rear= -1 else check if (front==size-1), if it is true then set front=0 and return the element.

**CODE:**

[**https://www.geeksforgeeks.org/circular-queue-set-1-introduction-array-implementation/**](https://www.geeksforgeeks.org/circular-queue-set-1-introduction-array-implementation/)