Stack and Queue

Data Structure: Stack

A stack is a simple data structure used for storing data

In a stack, the order in which the data arrives is important.

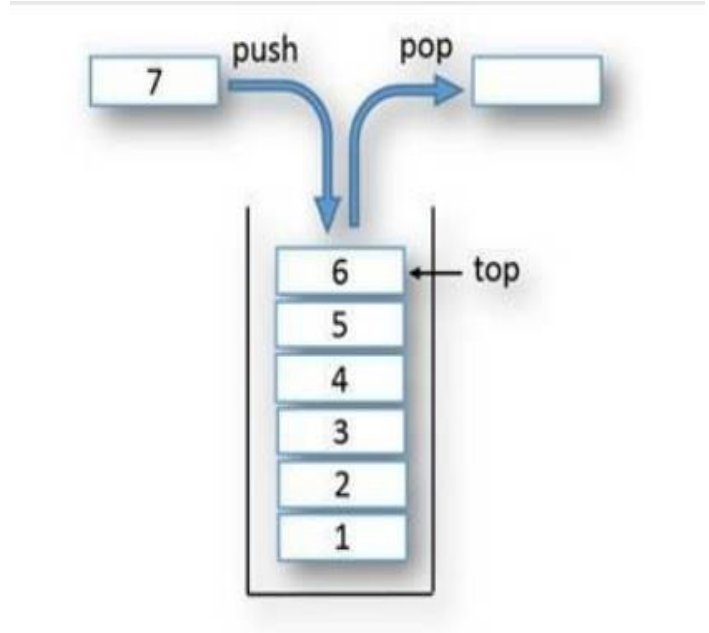
A stack is an ordered list in which insertion and deletion are done at one end, called top.

The last element inserted is the first one to be deleted. Hence, it is called the Last in First out (LIFO) or First in Last out (FILO) list.

Operations on stack

1. When an element is inserted in a stack, the operation is called push.
2. When an element is removed from the stack, the concept is called pop.
3. Returns the last inserted element without removing it, is called peek.
4. Trying to pop out an empty stack is called underflow.
5. Trying to push an element in a full stack is called overflow.

Generally, we treat underflow and overflow as exceptions.



Stack ADT

The following operations make a stack an ADT. (For simplicity, assume the data is an integer type.)

Main stack operations

- push (int data): Inserts data onto stack.
- int pop(): Removes and returns the last inserted element from the stack.

Auxiliary stack operations

- int peek(): Returns the last inserted element without removing it.
- int size(): Returns the number of elements stored in the stack.
- int IsEmptyStack(): Indicates whether any elements are stored in the stack or not.
- int IsFullStack(): Indicates whether the stack is full or not.

Applications of Stack

Following are some of the applications in which stack DS play an important role.

Direct applications

- Balancing of symbols
- Infix-to-postfix conversion
- Evaluation of postfix expression
- Implementing function calls (including recursion)
- Finding of spans (finding spans in stock markets, refer to Problems section)
- Page-visited history in a Web browser [Back Buttons]
- Undo sequence in a text editor
- Matching Tags in HTML and XML

Indirect applications

- Auxiliary data structure for other algorithms (Example: Tree traversal algorithms)

Implementation: There are many ways of implementing stack ADT.

- Simple array based implementation
- Dynamic array based implementation
- Linked lists implementation

Performance:

Space Complexity (for n push operations)	O(*n*)
Time Complexity of Push()	O(1)
Time Complexity of Pop()	O(1)
Time Complexity of Size()	O(1)
Time Complexity of IsEmptyStack()	O(1)
Time Complexity of IsFullStackf)	O(1)
Time Complexity of DeleteStackQ	O(1)

Data Structure: Queue

A queue is a data structure used for storing data (similar to Linked Lists and Stacks).

In queue, the order in which data arrives is important.

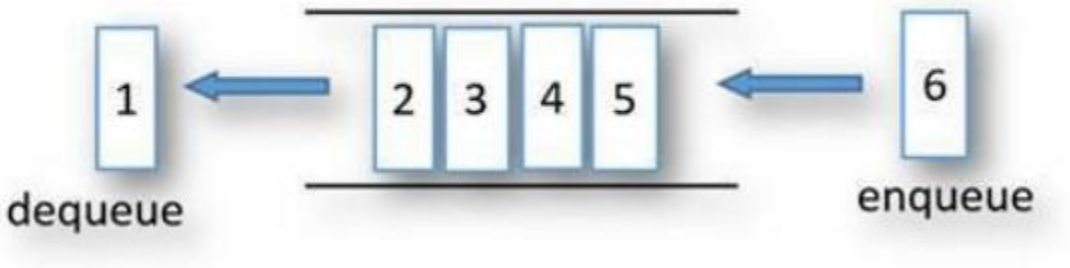
Definition: A queue is an ordered list in which insertions are done at one end (rear) and deletions are done at other end (front).

The first element to be inserted is the first one to be deleted. Hence, it is called First in First out (FIFO) or Last in Last out (LILO) list.

Operations on queue

1. When an element is inserted in a queue, the concept is called EnQueue.
2. when an element is removed from the queue, the concept is called DeQueue .
3. Returns the element at front without removing it, is called peek.
4. DeQueueing an empty queue is called underflow.
5. EnQueuing an element in a full queue is called overflow..

Generally, we treat underflow and overflow as exceptions.



Queue ADT

The following operations make a queue an ADT. (For simplicity, assume the data is an integer type.)

Main queue operations

- EnQueue(int data): Inserts an element at the end of the queue
- int DeQueue(): Removes and returns the element at the front of the queue

Auxiliary stack operations

- int Front(): Returns the element at the front without removing it
- int QueueSize(): Returns the number of elements stored in the queue
- int IsEmptyQueueQ: Indicates whether elements are stored in the queue or not

Applications of Queue

Following are some of the applications that uses queue.

Direct applications

- Operating systems schedule jobs (with equal priority) in the order of arrival (e.g., a print queue).
- Simulation of real-world queues such as lines at a ticket counter or any other first-come first-served scenario requires a queue.
- Multiprogramming.
- Asynchronous data transfer (file IO, pipes, sockets).
- Waiting times of customers at call center.
- Determining number of cashiers to have at a supermarket.

Indirect applications

- Auxiliary data structure for algorithms (Example: Tree traversal algorithms)

Implementation: There are many ways of implementing queue ADT.

- Simple array based implementation
- Dynamic array based implementation
- Linked lists implementation

Performance:

Space Complexity (for n push operations)	O(*n*)
Time Complexity of Push()	O(1)
Time Complexity of Pop()	O(1)
Time Complexity of Size()	O(1)
Time Complexity of IsEmptyStack()	O(1)
Time Complexity of IsFullStackf)	O(1)
Time Complexity of DeleteStackQ	O(1)

Circular Queue ADT

We reuse the array indexes in circular fashion to insert elements in queue.

Simple array implementation for queue is not efficient

We treat the last element and the first array elements as contiguous.