Stacks:

->Datastructure in which the last element added to the structure is the first to be removed.

->LIFO (last in first out)

-Any type of stack must provide the following methods:

->push, pop, peek, size, isempty

There are two major implementations of a stack (language-agnostic):

-> array-based stack implementation

(1) an array

(2) an integer (usually called top) that references the next position in the array in add an element (i.e, top - 1 is the current top of the stack)

The variable top also tells he number of elements currently stored in the stack

-> linked-based implementation. For this implementation we need:

(1) a object which will represent a node to be linked in the data structure (think like boxes being tied together, or a chain),

usually called LinkedNode or LinearNode

LinkedNode (or LinearNode) has two references, one is to the element added in the LinkedNode, and a second reference to the next element in the chain

(2) a secibd object (called LinkedStack) which administers the LinkedNodes to emulate a stack. This object has two major variables:

A reference called top which points to the top of the stack

An int variable called length/size that states how many elements are in the stack.

Goal of any implementation is to support the stack operations in O(1)

Queue:

* Data structure in which the first element to be added is the first element to be removed(FIFO: first in-first out)

-Any type of queue must provide the following methods:

-> enqueue: add an element to the end of the queue (some books call this append)

-> dequeue: removes an element from the beginning of the queue

-> offer: returns the element at the beginning of the queue, without removing it from the queue.

-> size: returns the number of elements in the queue

->isEmpty: returns true if the queue is empty

There are three major implementations of queues (language agnostic):

1. Linked-based implementation
2. array-based implementation
3. circular array-based implementation

Goal of any implementation is to support the queue operations O(1)

Linked implementation of queues:

Every time you add you add the beginning, every time you remove it takes from the tail of the queue.

If head is null then tail is null. Head and tail do not reference the content but reference the order.