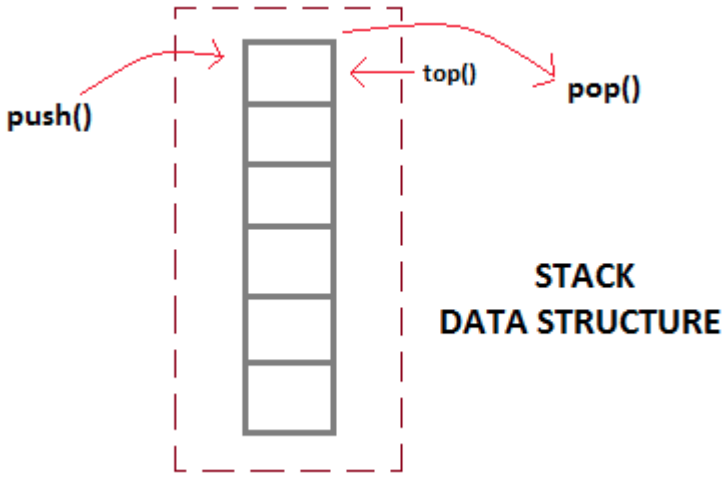
Stacks/Queues

If we take a linked list or arraylist and put some limitations on its operations. Limit it to only allow for insertion and deletion from only the front and rear. With these limitations, we get the stack and queue data structures.

Stack

- Last In, First Out (LIFO)

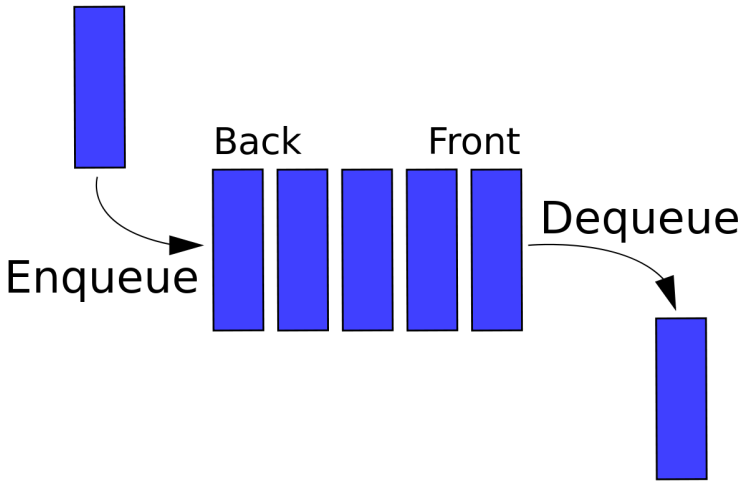


- Behaves like a stack of papers on your desk. You can ***push*** new paper on top when they arrive or you can ***pop*** the top paper off the stack.
- The Stack ADT has 2 major methods:
  - void push(T x)
  - T pop()
- Other methods that are helpful are:
  - T top() or T peak()
    - Allows to view the top without removing it
  - int size()
  - boolean isEmpty()
- For stacks the insertion and deletion are done at the ends

Method	Return Value	Stack Contents
push(5)	–	(5)
push(3)	–	(5, 3)
size()	2	(5, 3)
pop()	3	(5)
isEmpty()	false	(5)
pop()	5	()
isEmpty()	true	()
pop()	null	()
push(7)	–	(7)
push(9)	–	(7, 9)
top()	9	(7, 9)
push(4)	–	(7, 9, 4)
size()	3	(7, 9, 4)
pop()	4	(7, 9)
push(6)	–	(7, 9, 6)
push(8)	–	(7, 9, 6, 8)
pop()	8	(7, 9, 6)

Queue

- First In, First Out (FIFO)

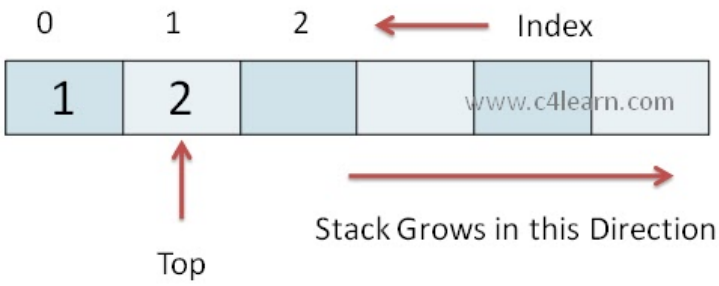


- Behaves like a line at Starbucks, the first person in the line is the first to get served. The last person in the line will be the last person to get served.
- When the first person leaves the line, ***dequeue***, they leave from the front of the line. When a new person arrives, the get on the line from the back, ***enqueue***
- The Queue ADT has 2 main methods:
  - void enqueue(T x)
  - T dequeue()
- Other methods that are helpful are:
  - int size()
  - boolean isEmpty()
- For Queue, the insertion and deletion occur on opposite ends

Method	Return Value	first ← Q ← last
enqueue(5)	–	(5)
enqueue(3)	–	(5, 3)
size( )	2	(5, 3)
dequeue( )	5	(3)
isEmpty( )	false	(3)
dequeue( )	3	( )
isEmpty( )	true	( )
dequeue( )	null	( )
enqueue(7)	–	(7)
enqueue(9)	–	(7, 9)
first( )	7	(7, 9)
enqueue(4)	–	(7, 9, 4)

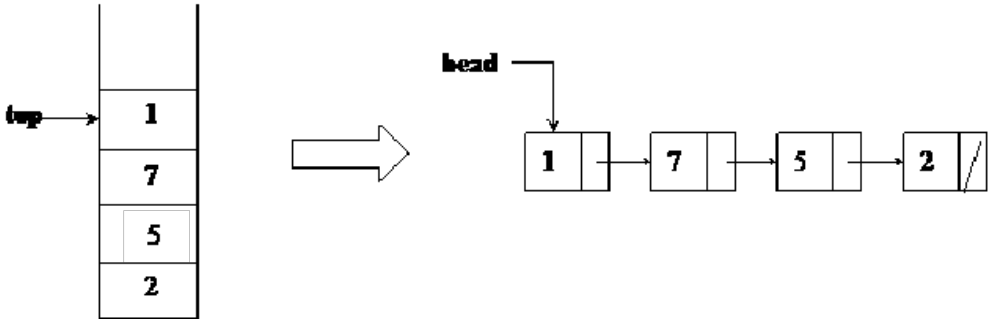
Stack Implementation

1. Array Based Implementation
   - Use a large array to store the data
   - Removing/adding to the rear of an array is efficient and easy
   - Data must occupy the left part of the array
   - The TOP is on the rightmost value
   - TOP is the value of the index where the top elements is stored in the array



Source: http://www.c4learn.com/data-structure/stack-array-representation/

2. Linked Implementation
    - Uses nodes and their links
    - The TOP nodes is the head node
    - We remove and add to the top of the list since it is easy and efficient



Source: http://www1.widgetserver.com/?subid4=1553534880.0676442566&kw=video&KW1=Dedicated%20Server%20USA&KW2=Dedicated%20Server%20Asia&KW3=Dedicated%20Server%20Europe&searchbox=0&domainname=0&backfill=0

3. LinkedList Adapter
    - Adapt a Linked List to implement number 2

Queue Implementation

- Linked Implementation
    - Uses nodes and their links
    - The front node is the head node
    - The rear node is the tail node
    - Removing from the head and adding to tail is easy and efficient
- LinkedList Adapter
    - Adapt a Linked List to implement number 1
- Array Based Implementation
    - There is a issue is we use a standard array
    - Suppose:

    - Now as we remove/add data, the elements start to move left.

- Eventually, we will run out of space to add in the array even though the entire array could be empty
- Solution: Use a circular array
    - Index[i] is next to index[(i+1) % capacity]
    - Front is the index to remove from
    - Rear is the index to add to
    - Size is the number of elements in the queue
    - Capacity it the total number of space available