CS101 - Data Abstraction Stacks and Queues

Aravind Mohan

Allegheny College

May 6, 2021

Stack

A plate dispenser is like a **Stack**.

What is a Stack ADT?

- A stack is a container of objects that are inserted and removed according to the last-in-first-out (**LIFO**) principle.
- Objects can be inserted at any time, but only the last (the most-recently inserted) object can be removed.
- Inserting an item is known as ”pushing” onto the stack. ”Popping” off the stack is synonymous with removing an item.

Stack ADT Operations

- A stack is an User Defined Data Type that supports four main methods:
  1. **new():** - Creates a new stack.
  2. **push(S, o:element):** - Inserts object o onto top of stack S.
  3. **pop(S)** - Removes the top object of stack S unless the stack is empty.
  4. **top(S)** - Returns the top object of the stack, without removing it unless stack is empty.

Stack ADT Supporting Operations

- **size(S)** - Returns the number of objects in stack S.
- **isEmpty(S)** - Indicates if stack S is empty.

Queue

A line of people standing in a ticket counter is similar to a **Queue**.

What is a Queue ADT?

- A queue differs from stack in that its insertion and removal routines follows first in first out (FIFO) principal.
- Elements can be inserted at any time, but only the element which has been in the queue longest can be removed.
- Elements are inserted in the rear (enqueued) and removed from the front (dequeued).

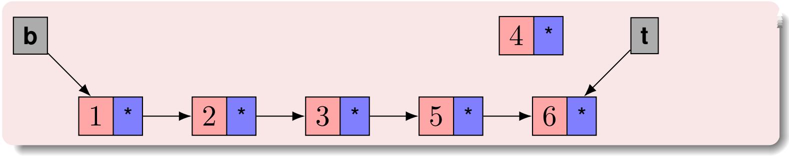
Queue ADT Operations

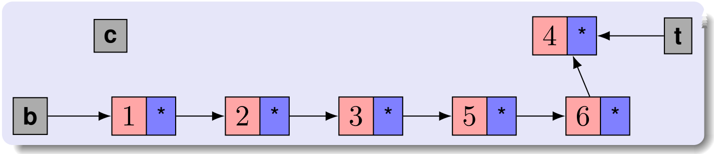
- A Queue is an Abstract Data Type that supports four main methods:
  - **new()** - Creates a new queue.
  - **enqueue(Q, o)** - Inserts object o at the rear of the queue Q.
  - **dequeue(Q)** - Removes the object from the front of the queue unless the queue is empty.
  - **front(Q)** - returns, but does not remove ,the front element unless the queue is empty.

Queue ADT Supporting Operations

- **size(Q)** - Returns the number of objects in queue Q.
- **isEmpty(Q)** - Indicates if queue Q is empty.

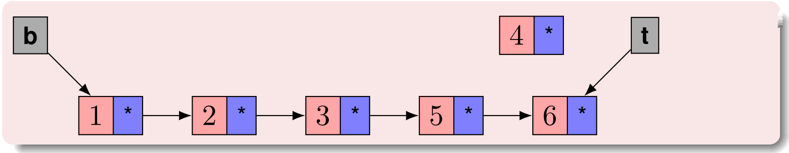
Stack Push Operation Approach 1 (Inefficient)

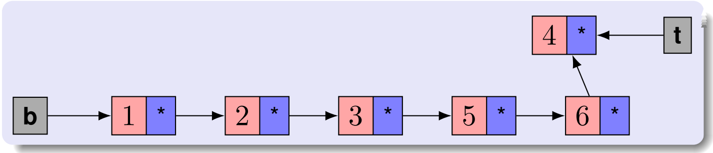




1. Create a new temp node with the value to be inserted.
2. Traverse through the list till last node using current pointer **c**.
3. Assign next of the last node to the temp node.
4. Assign top pointer **t** to the temp node.

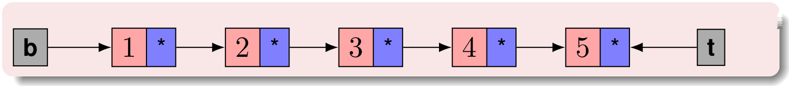
Stack Push Operation Approach 2 (Efficient)

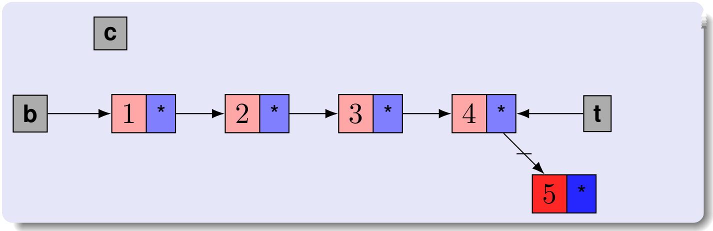




1. Create a new temp node with the value to be inserted.
2. Assign next of the node pointed by top pointer to the temp node.
3. Assign top pointer **t** to the temp node.

Stack Pop Operation



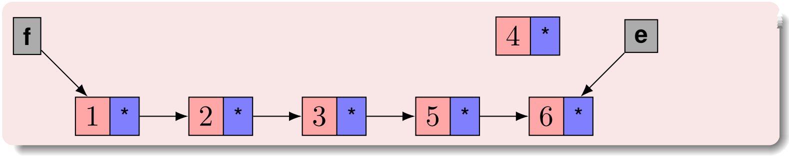


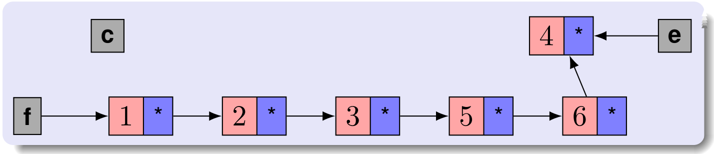
1. Traverse through the list till the previous of last node using current pointer **c**.
2. Assign next of the previous of last node to null reference.
3. Assign top pointer to reference the previous of last node.
4. Free up the last node.

Implementation ...

- **PS** the stack folder in course repo.

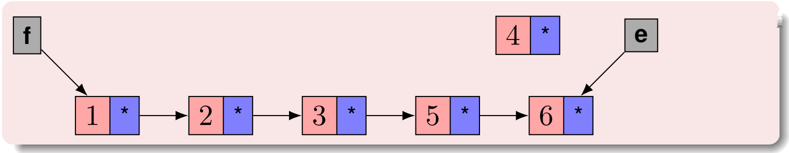
Queue Enqueue Operation Approach 1 (Inefficient)

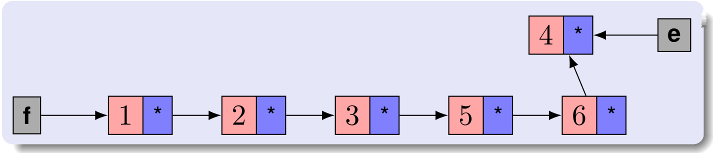




1. Create a new temp node with the value to be inserted.
2. Traverse through the list till last node using current pointer **c**.
3. Assign next of the last node to the temp node.
4. Assign end pointer **e** to the temp node.

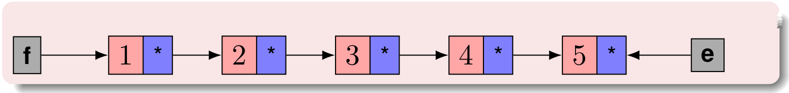
Queue Enqueue Operation Approach 2 (Efficient)

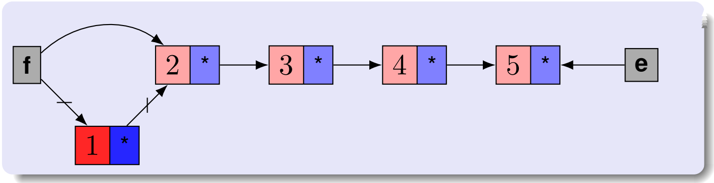




1. Create a new temp node with the value to be inserted.
2. Assign next of the node pointed by end pointer to the temp node.
3. Assign end pointer **e** to the temp node.

Queue Dequeue Operation





1. Assign the front pointer to the node N(0), that is the node pointed by the next of the first node.
2. Free up the first node.

Implementation ...

- **PS** the queue folder in course repo.

Reading Assignment

- **GT** Chapter 6 - 6.1, 6.2, 6.3

Questions?

Please ask if there are any Questions!