Announcements

- P0 due 11pm today; Hw1 due this Thur.
- P1 out today, due Oct 1st
- Midterm 1: Oct 3rd (overview – BST)
- Schedule site updated
  - All lecture slides before midterm 1 are up

CMSC 341

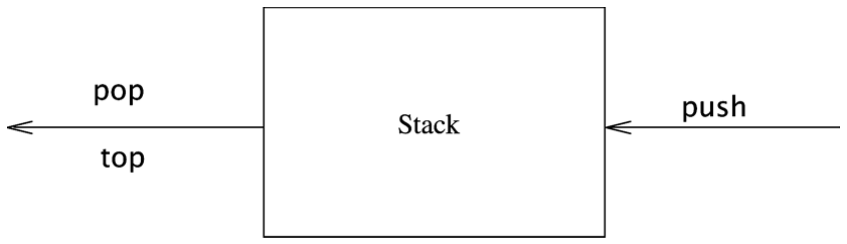
Stacks and Queues

Textbook Sections 3.6 - 3.7

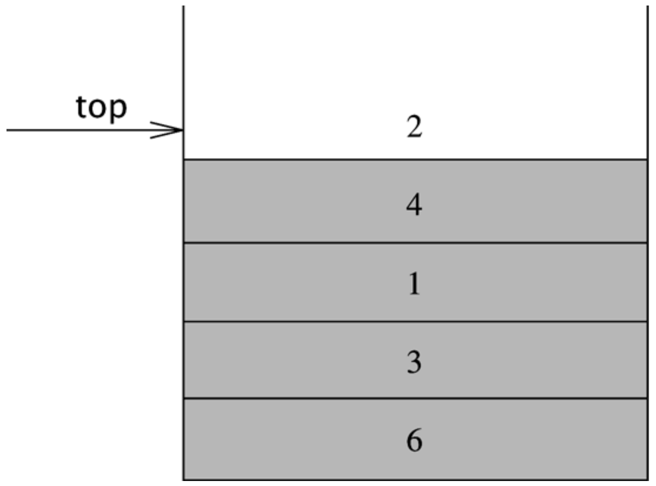
Stacks

Stack ADT

- Basic operations are push, pop, and top
- Why stack?
    - What is the running time for these operations?



Stack Model



Stacks

- A restricted list where insertions and deletions can only be performed at one location, the end of the list (top).
- LIFO – Last In First Out
  - Laundry Basket – last thing you put in is the first thing you remove
  - Plates – remove from the top of the stack and add to the top of the stack

Adapting Lists to Implement Stacks

- Adapter Design Pattern
- Allow a client to use a class whose interface is different from the one expected by the client
- Do not modify client or class, write adapter class that sits between them
- In this case, the List is an adapter for the Stack. The client (user) calls methods of the Stack which in turn calls appropriate List method(s).

Adapter Model for Stack

$$\begin{tikzcd}
\text{Client (Stack user)} \arrow[d, "\text{theStack.push(10)}"] \\
\text{Stack (adapter)} \arrow[d, "\text{theList.add(0, 10) ;}"] \\
\text{List (adaptee)}
\end{tikzcd}$$

Examples

- Balancing symbols
- Infix to postfix conversion
- Postfix expressions

Example 1: Balancing symbols

- Algorithm:

```
Make an empty stack
Read characters until the end of the file
    (1) If the character is an opening symbol, push it into the stack.
        else
    (2) If it is a closing symbol,
        (1) If the stack is empty, report an error, else
        (2) Pop the stack & check the popped
            (1) Error correspondence with the open symbol – error
            (2) Else continue
If the stack is not empty report an error.
```

Examples: [()], [(]) (please see lecture notes)

Example 2: Infix to postfix conversion

```
Make an empty stack
Read the characters until the end of the equation
  If read an operand, output
  If read a right parenthesis,
     pop till a corresponding left parenthesis
  If read + or * or (
     pop entries from the stack until we find an entry of lower priority
     exception: never remove a ( from the stack except when
     processing a )
     push the operator onto the stack
Pop the stack and output until it is empty
```

Example: a + b*c + (d*e + f)*g $\rightarrow$ abc*+de*f+g*+ (please see lecture notes)

Idea: the stack represents pending operators. When some of the operators on the stack that have high precedence are not known to be completed, and should be popped, because they are no longer pending.

Example 3: Postfix expressions

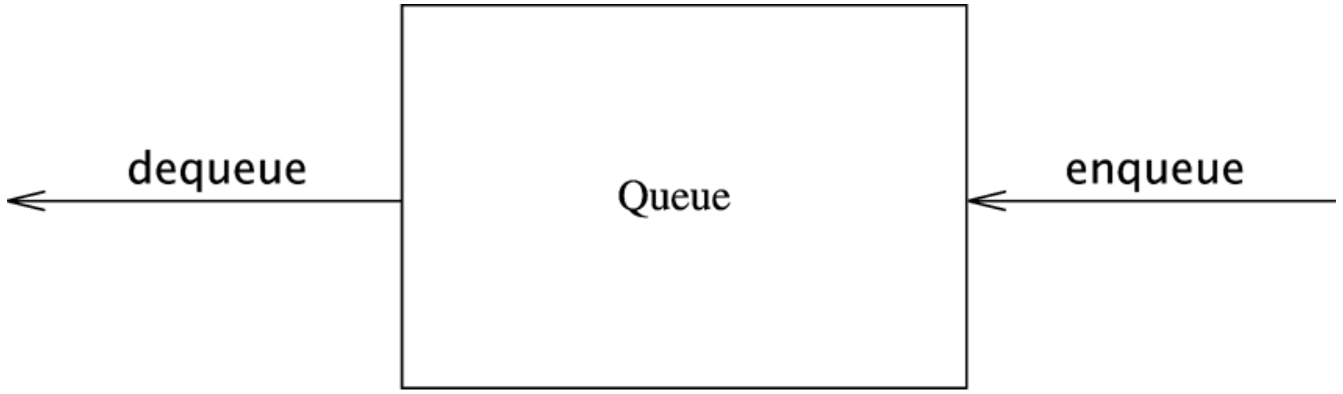
```
Make an empty stack
Read the characters until the end of the equation
  If read a number, push
  If read an operator,
     pop two numbers and apply the operator
     push the result to the stack
```

Examples: compute abc*+de*f+g*+ (please see lecture notes)
compute 6523+8*+3+*

Queues

Queue ADT

- Basic Operations are enqueue and dequeue



Queues

- Restricted List
  - *enqueue()*: only add to tail (or the rear)
  - *dequeue()*: only deletes (and returns) the element from the head (or the front)
- Examples
  - line waiting for service; jobs waiting to print
  - network access to a file server
- Implement as an adapter of List
  - Both arrayList and linkedList work
  - Running time O(1) for *enqueue* and *dequeue: is this possible for linkedList and arrayList?*

linkedList implementation: Adapter Model for Queue

$$\begin{tikzcd}
\text{Client (Queue user)} \arrow[d, "\text{theQ.enqueue(10)}"] \\
\text{Queue (adapter)} \arrow[d, "\text{theList.add(theList.size() -1, 10)}"] \\
\text{List (adaptee)}
\end{tikzcd}$$

ArrayList implementation: Circular Queue

- Adapter pattern may be impractical
  - Overhead for creating, deleting nodes
  - Max size of queue is often known
- A circular queue is a fixed size array
  - Slots in array reused after elements dequeued

Circular Queue Data

- A fixed size array
- Control Variables

  arraySize

    the fixed size (capacity) of the array

  currentSize

    the current number of items in the queue

    Initialized to 0

  front

    the array index from which the next item will be dequeued.

    Initialized to 0

  back

    the array index last item that was enqueued

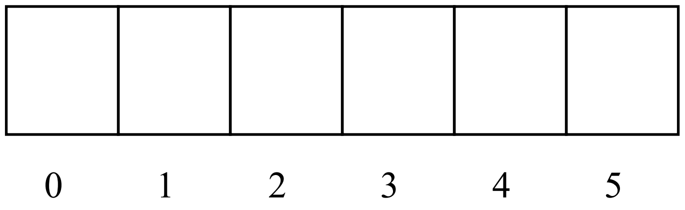
    Initialized to -1

Circular Queue Psuedocode

```
void enqueue( Object x ) {
   if currentSize == arraySize, throw exception  // Q is full
   back = (back + 1) % arraySize;
   array[ back ] = x;
   ++currentSize;
}

Object dequeue( ) {
   if currentSize == 0, throw exception          // Q is empty
   --currentSize;
   Object x = array[ front ];
   front = (front + 1) % arraySize
   return x;
}
```

Circular Queue Example



Trace the contents of the array and the values of currentSize, front and back after each of the following operations.

1. enqueue(12)
2. enqueue(17)
3. enqueue(43)
4. enqueue(62)
5. dequeue( )
6. dequeue( )
7. enqueue(42)
8. dequeue( )
9. enqueue(33)
10. enqueue(18)
11. enqueue(99)