Announcements

- Written homework available on assignment page (due next Friday in class).

CS200: Linear Structures

- Stacks: Walls Ch. 7
- Queues: Walls Ch. 8

Linear, time-ordered structures

- Data structures that reflect a temporal relationship
  - order of removal based on order of insertion
- We will consider:
  - "first come, first serve"
  - first in first out - FIFO (queue)
  - "take from the top of the pile"
  - last in first out - LIFO (stack)

Stacks

- Last In First Out (LIFO) structure
  - A stack of pancakes or dishes
  - Add/Remove done from same end

Stack ADT

- Create
  - Query
    - empty?
    - peek - returns the most recently added element
  - Add
    - push - add (to the top of the stack)
- Remove
  - pop - remove (from the top of the stack)
  - popAll - remove all

Applications - the run-time stack

- Nested method calls tracked on call stack (aka run-time stack)
  - First method that returns is the last one invoked
  - Element of call stack - activation record
    - parameters
    - local variables
    - return address: pointer to next instruction to be executed in calling method

Applications - Depth First Search

- Looking for a path in a maze
- Strategy:
  - Prioritize directions: right, straight or left.
  - At a dead end backtrack and try a different direction
- Recursive solution?
- Stack-based solution?

Digression: Expressions

- Types of Algebraic Expressions
  - Prefix
  - Postfix (RPN)
  - Infix
- Prefix and postfix are easier to parse. No ambiguity.
- Postfix: operator applies to the operands that immediately precede it.
- Examples:
  1. 5 4 3 * -
  2. 5 * 4 - 3
  3. 5 * 4 3 -
  4. * - 5 4 3

Parsing a Postfix Expression

while there are input tokens left
read the next token
if the token is a value
  push it onto the stack.
else
  the token is a function taking n arguments
  pop the top n values from the stack and evaluate the function
  push the result on the stack
If there is only one value in the stack return it as the result
else
  throw an exception

Checking for balanced braces

- How can we use a stack to determine whether the braces in a string are balanced?
  abc{def}{ijk}{l{mn}}op}qr
  abc{def} } {ghi{[kl]}m

Stack Methods

push(in newItem:StackItemType) throws StackException
- adds a new item to the top of the stack
- Exception when insertion fails
pop():StackItemType throws StackException
- deletes the item at the top of the stack and returns it
- Exception when deletion fails
peek():StackItemType {query} throws StackException
- returns the top item from the stack, but does not remove it
- Exception when retrieval fails

Array-Based Implementation

public class StackArrayBased implements StackInterface {
  final int MAX_STACK = 50; // maximum size of stack
  private Object items[];
  private int top;
  public StackArrayBased() {
    items = new Object[MAX_STACK];
    top = -1; }
  public void push(Object newItem) throws StackException{
    if (!isFull()) {
      items[++top] = newItem; ...
    } ...
  }

  What's missing here?
ArrayList Implementation

- push
  - use `add` to add to the end of the ArrayList
  - worst case complexity: $O(n)$ (but ONLY when capacity needs to be extended)
- pop
  - use `get(size-1)` to obtain the return value
  - use remove(size-1) to remove the element at the top
- peek
  - use `get(size-1)`
- What if we push and pop at index 0?

```
public Stack implements StackInterface {
private ArrayList<Object> items;
...
```

Linked List Implementation

- push
  - worst case complexity: $O(1)$
- pop
  - worst case complexity: $O(1)$
- peek
  - Same as pop

```
public class StackReferenceBased implements StackInterface {
private Node top;
public StackReferenceBased() {
top = null;
}
public boolean isEmpty() {
return top == null;
}
public void push(Object newItem) {
top = new Node(newItem, top);
}
```

Stack API in Java

```
public class Stack<E> extends Vector<E> {
    public StackReferenceBased implements StackInterface {
        private StackReferenceBased() {
top = null;
        }
        public boolean isEmpty() {
return top == null;
        }
        public void push(Object newItem) {
            top = new Node(newItem, top);
        }
```

Comparison of implementations

- What are the advantages and disadvantages of each implementation?
- Array based implementation
- ArrayList based implementation
- Linked-list based implementation
Queues
- First In First Out (FIFO) structure
- Imagine a checkout line
- So removing and adding are done from opposite ends of structure.
  - add to tail, remove from head
  - Used in operating systems (e.g., print queue).

Queue Methods
- `enqueue(item)`: Adds an item to the tail of the queue.
  - Throws exception if not successful.
- `dequeue()`: Deletes the item from the head of the queue and returns it.
  - Throws exception if not successful.
- `peek()`: Returns the item from the head of the queue.
  - Throws exception if not successful.

Queue Implementation - Using ArrayList
- `enqueue(item)`: Use add - adds to the end of the ArrayList.
- `dequeue()`: Use get(0) to get return value.
- `peek()`: Use get(0).

A Naïve Array-Based Implementation
- Drift can cause the queue to appear full.
  - How do we initialize front and back?
    (Hint: what does a queue with a single element look like?)

An Array-Based Implementation
- A wrap-around array eliminates the problem of drift.
  - `enqueue()`: `back = (back+1) % MAX_QUEUE; items[back] = newItem; ++count;`
  - `dequeue()`: `frontItem = items[front];
    front = (front+1) % MAX_QUEUE;
    --count;`
  - `peek()`: return item at front

An Array-Based Implementation
- The effect of some queue operations:
An Array-Based Implementation

- Issue with the circular array implementation:
  - `front` and `back` cannot be used to distinguish between queue-full and queue-empty conditions

Reference-Based Implementation

- A linked list with two external references
  - A reference to the front
  - A reference to the back

- A circular linked list with one external reference
  - `lastNode` references the back of the queue
  - `lastNode.getNext()` references the front

Variations:
- Use a flag `full` to distinguish between the full and empty conditions
- Declare `MAX_QUEUE + 1` locations for the array items, but use only `MAX_QUEUE` of them for queue items

Solution: count # of items
Reference-Based Implementation

Inserting an item into a non-empty queue

Reference-Based Implementation

Inserting an item into an empty queue

Reference-Based Implementation

Deleting an item from a queue of more than one item

Queue implementations

- What are the advantages/disadvantages of the circular array / linked list implementations?