CSC 143

Stacks and Queues:

Concepts and Implementations

Overview

- Topics
  - Stacks
  - Queues
  - Simulations
- Readings
  - Textbook sec. 25.2 & 25.3

Typing and Correcting Chars

- What data structure would you use for this problem?
  - User types characters on the command line
  - Until she hits enter, the backspace key (<) can be used to "erase the previous character"

Sample

Action	Result
type h	**h**
type e	**he**
type l	**hel**
type o	**helo**
type <	**hel**
type l	**hell**
type w	**hellw**
type <	**hell**
type <	**hel**
type <	**he**
type <	**h**
type i	**hi**

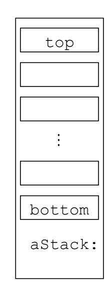
Analysis

- We need to store a sequence of characters
- The order of the characters in the sequence is significant
- Characters are added at the end of the sequence
- We only can remove the most recently entered character

- We need a data structure that is *Last in, first out*, or LIFO – a *stack*
  - Many examples in real life: stuff on top of your desk, trays in the cafeteria, discard pile in a card game, …

Stack Terminology

- *Top*: Uppermost element of stack,
  - first to be removed
- Bottom: Lowest element of stack,
  - last to be removed
- Elements are always inserted and
  removed from the top (LIFO)

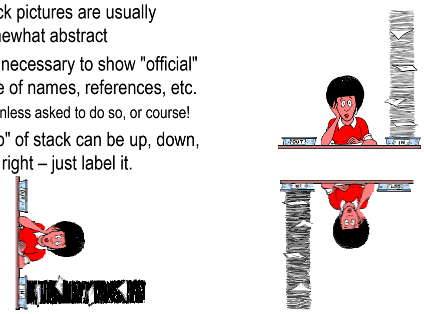


Stack Operations

- push(Object): Adds an element to top of stack, increasing stack height by one
- Object pop( ): Removes topmost element from stack and returns it, decreasing stack height by one
- Object top( ): Returns a copy of topmost element of stack, leaving stack unchanged
- No "direct access"
  - cannot index to a particular data item
- No convenient way to traverse the collection
  - Try it at home!

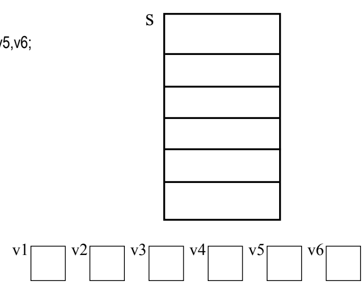
Picturing a Stack

- Stack pictures are usually somewhat abstract
- Not necessary to show "official" style of names, references, etc.
  - Unless asked to do so, or course!
- "Top" of stack can be up, down, left, right – just label it.



What is the result of...

```
Stack s;
Object v1,v2,v3,v4,v5,v6;
s.push("Yawn");
s.push("Burp");
v1 = s.pop( );
s.push("Wave");
s.push("Hop");
v2 = s.pop( );
s.push("Jump");
v3 = s.pop( );
v4 = s.pop( );
v5 = s.pop( );
v6 = s.pop( );
```



Stack Practice

- Show the changes to the stack in the following example:

```
Stack s;
Object obj;
s.push("abc");
s.push("xyzzy");
s.push("secret");
obj = s.pop( );
obj = s.top( );
s.push("swordfish");
s.push("terces");
```


Stack Implementations

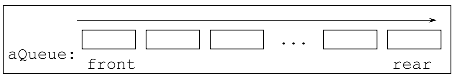
- Several possible ways to implement
  - An array
  - A linked list
    Useful thought problem: How would you do these?
- Java library does not have a Stack class
- Easiest way in Java: implement with some sort of List
  - push(Object) :: add(Object)
  - top( ) :: get(size( ) –1)
  - pop( ) :: remove(size( ) -1)
  - Precondition for top( ) and pop( ): stack not empty

What is the Appropriate Model?

- waiting line at the movie theater...
- job flow on an assembly line...
- traffic flow at the airport....
- "Your call is important to us. Please stay on the line. Your call will be answered in the order received. Your call is important to us...
  - ...
- Characteristics
  - Objects enter the line at one end (rear)
  - Objects leave the line at the other end (front)
- This is a "*first in, first out*" (FIFO) data structure.

Queue Definition

- Queue: Ordered collection, accessed only at the front (remove) and rear (insert)
  - Front: First element in queue
  - Rear: Last element of queue
- FIFO: First In, First Out
- Footnote: picture can be drawn in any direction



Abstract Queue Operations

- insert(Object) : Adds an element to rear of queue
  - succeeds unless the queue is full (if implementation is bounded)
  - often called "enqueue"
- Object front( ) : Return a copy of the front element of queue
  - precondition: queue is not empty
- Object remove( ) : Remove and return the front element of queue
  - precondition: queue is not empty
  - often called "dequeue"

Queue Example

- Draw a picture and show the changes to the queue in the following example:

```
Queue q; Object v1, v2;

q.insert("chore");
q.insert("work");
q.insert("play");
v1 = q.remove();
v2 = q.front();
q.insert("job");
q.insert("fun");
```


What is the result of:

```
Queue q; Object v1,v2,v3,v4,v5,v6
q.insert("Sue");
q.insert("Sam");
q.insert("Sarah");
v1 = q.remove( );
v2 = q. front( );
q.insert("Seymour");
v3 = q.remove( );
v4 = q.front( );
q.insert("Sally");
v5 = q.remove( );
v6 = q. front( );
```


Queue Implementations

- Similar to stack
  - Array – trick here is what do you do when you run off the end
  - Linked list – ideal, if you have both a *first* and a *last* pointer.
- No standard Queue class in Java library
- Easiest way in Java: use LinkedList class
  - insert(Object):: addLast(Object) [or add(Object)]
  - getFront( ):: getFirst( )
  - remove( ):: removeFirst( )

    Interesting "coincidence" that a Java LinkedList supports exactly the operations you want to implement queues.

Bounded vs Unbounded

- In the abstract, queues and stacks are generally thought of as "unbounded": no limit to the number of items that can be inserted.
- In most practical applications, only a finite size can be accommodated: "bounded".
- Assume "unbounded" unless you hear otherwise.
  - Makes analysis and problem solution easier
  - Well-behaved applications rarely reach the physical limit
- When the boundedness of a queue is an issue, it is sometimes called a "buffer"
  - People speak of bounded buffers and unbounded buffers
  - Frequent applications in systems programming

    E.g. incoming packets, outgoing packets

Summary

- Stacks and Queues
  - Specialized list data structures for particular applications

- Stack
  - LIFO (Last in, first out)
  - Operations: push(Object), top( ), and pop( )

- Queue
  - FIFO (First in, first out)
  - Operations: insert(Object), getFront( ), and remove( )
- Implementations: arrays or lists are possibilities for each
- Next up: applications of stacks and queues