CSC 143 Java

Stacks and Queues

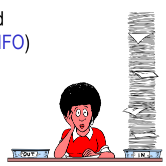
What Data Structure Do We Want?

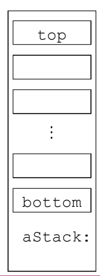
- We need to store a sequence of characters
  - The order of the characters in the sequence is significant
  - Characters are added at the end of the sequence
  - We only can remove the most recently entered character
- Discard pile in a card game
- Sequence of activation records

We need a data structure that is *Last In, First Out*, or LIFO – a *stack*

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(E): Adds an element to top of stack, increasing stack height by one
- E pop( ): Removes topmost element from stack and returns it, decreasing stack height by one.
  precondition: Stack is not empty
- E top( ): Returns the topmost element of stack, leaving stack unchanged. Also known as peek()
  precondition: Stack is not empty
- boolean isEmpty(): returns true if Stack is empty
- *Usually* no "direct access"
  - cannot index to a particular data item
- *Usually* no way to traverse the collection

Stack Practice

Example 1

```
Stack<String> s = new Stack<String>();
String v1,v2,v3,v4,v5;
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( );
```

Example 2

```
Stack<String> s= new Stack<String>();
String obj;
s.push("abc");
s.push("xyzzy");
s.push("secret");
obj = s.pop( );
obj = s.top( );
s.push("swordfish");
s.push("kugel");
```

How do we implement this class (efficiently?)

Stack Applicaton: Postfix Evaluation

- Read in the next "token" (operator or data)
  - If data, push it on the data stack
  - If (binary) operator (call it "op"):
    Pop off the most recent data (B) and next most recent (A)
    Perform the operation R = *A op B*
    Push R on the stack
- Continue with the next token
- When finished, the answer is the stack top.
- Simple, but works like magic!
- What are the error conditions?

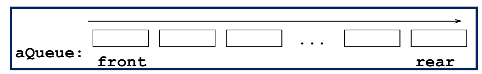
Stack App: Converting in-fix to post-fix

- Algorithm -- uses a stack of operators:
  - Read next token from list of infix tokens
  - If operand, add it to the postfix list of tokens (PE)
  - If '(', push it on stack
  - If operator:
    - if stack top is an op of >= precedence than current op then pop and add to PE.
    - repeat test until '(' is on top or stack empty
    - push the new operator
  - If ')', pop ops and add to PE until '(' has been popped
  - Repeat until end of input
    - pop rest of stack at end

What Structure Do We Want?

- 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 Operations

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

Queue Practice

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

```
Queue<String> q= new Queue<String>();   String 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( );
```

How do we implement this class (efficiently?)

Queue Application: Simulations

- Computer programs are often used to "simulate" some aspect of the real world
  - movement of people and things; economic trends; weather forecasting; physical, chemical, industrial processes
- Common starting data
  - Time between arrivals
  - Service time
  - Number of servers

- Often want to investigate/predict
  - Time spent waiting in queue
  - Effect of more/fewer servers
  - Effect of different arrival rates

Types of Simulations

Time-based: use a loop to count time
- Look and see what happens at every "tick" of the clock
- What happens during that tick?
  - Might "throw dice" to determine what happens
  - Might query all objects to see if anything was "scheduled to happen" at this time
- Size of time step? A day, a millesecond, a millenia, etc. depending on application

Event-based: use a loop to jump to the next event
- *Event list* (priority queue) holds the events waiting to happen; stored in chronological order
- Skip over time where nothing happens
- External events might come from a file, user input, number generator, etc.
  - example: Customer arrives
- Internal events are generated by other events within the system
  - example: Customer finishes transaction