Lecture 6:
Queues
(Chapter 5)

Oct 10, 2016

Data Structures, CS102

Chee Yap

Today's Lecture

I. Warmup, Review, Homework, etc.

II. Queues

Next...

I. Prelim, Review, etc

II. Queues

III. Evaluation Arithmetic Expressions

[Start] [End]

I. Prelim, Review, etc

Prelim, Review, Comments

Midterm (Tuesday October 18)

Homework 4 (Also due October 18?)

Prelim, Review, Comments

Recursive Makefile

Prelim, Review, Comments

Recursive Makefile

Pocket Calculator

Valid Arithmetic Expressions

Prelim, Review, Comments

Recursive Makefile

Pocket Calculator

Valid Arithmetic Expressions

Generating Random Arithmetic Expressions

Prelim, Review, Comments

Recursive Makefile

Pocket Calculator

Valid Arithmetic Expressions

Generating Random Arithmetic Expressions

Evaluating Arithmetic Expressions

Next...

I. Prelim, Review, etc

II. Queues

III. Evaluation Arithmetic Expressions

[Start] [End]

II. Queues

Queues

Queue Concepts:

- Enqueue: add to end-of-queue
- Dequeue: remove from front-of-queue

Queues

Example: consider the sequence of operations

EnQ(”Jane”), EnQ(”John”), EnQ(”Jill”),

DeQ(), EnQ(”Jack”), DeQ().

* What is the result at each step?

* What is the state of the Queue at each step?

Queues

Base:

```
public interface QueueInterface <T> {
    T dequeue() throws QueueUnderflowException;
    boolean isEmpty();
}
```

Queues

Base:

```
public interface QueueInterface <T> {
    T dequeue() throws QueueUnderflowException;
    boolean isEmpty();
}
```

Either extend to:

```
public interface UnboundedQueueInterface <T>
            extends QueueInterface <T> {
    void enqueue();
}
```

Queues

Base:

```
public interface QueueInterface <T> {
    T dequeue() throws QueueUnderflowException;
    boolean isEmpty();
}
```

Or extend to:

```
public interface BoundedQueueInterface <T>
            extends QueueInterface <T> {
    void enqueue() throws QueueOverflowException;
    boolean isFull();
}
```

Rich Man's Queue

Queues and Stacks are in Java standard library

java.util.Queue.

(similarly, `java.util.Stack`)

Part of Java's Library Collection Framework

`java.util.Collection`

Rich Man's Queue

```
interface java.util.Queue <E> extends Collection
        + offer(item:  E): boolean
        + poll():  E
        + remove():  E
        + peek():  E
        + element():  E
```

* Why "offer" instead of "add"?

  If fail to add, we do not throw exception!

* poll() remove front item, but peek does not.

Rich Man's Queue

```
interface java.util.Collection <E>
    + add(item:  E): boolean
    + addAll(c:  Collection<?  extends E>):  boolean

    + clear():  void
    + contains(o:  Object):  boolean
    + containsAll(c:  Collection<?>):  boolean
    + equals(o:  Object):  boolean
    + hashCode():  int
    + isEmpty():  boolean
    + iterator():  Iterator<E>
    + remove(o:  Object):  boolean
    + removeAll(c:  Collection<?>):  boolean
    + retainAll(c:  Collection<?>):  boolean
    + size():  int
    + toArray():  Object[]
```

Rich Man's Queue

```
interface java.util.Iterator <E>
    + hasNext():  boolean
    + next():  E
    + remove():  void
```

* What are iterators?
* What does remove() remove?

Rich Man's Queue

Iterator Demo

Eclipse (see Lectures/src/testQueue.java)

Imported interfaces:

java.util.Iterator, java.util.Queue,

Imported class:

java.util.LinkedList

(implementing java.util.Deque, hence java.util.Queue)

Next...

I. Prelim, Review, etc

II. Queues

III. Evaluation Arithmetic Expressions

[Start] [End]

III. Evaluation Arithmetic Expressions

Expressions: Infix to Postfix Conversion

Pocket Calculator

Demo

1 - 2 + 3 - 4 = ?

1 - 2 * 3 - 4 = ?

1 - (2 * 3) - 4 = ?

Expressions: Infix to Postfix Conversion

(Arithmetic) Expressions

operators :: $"+|-|*"$

operands :: $"[0-9]+"$

parens :: $"()"$

Expressions: Infix to Postfix Conversion

Valid Expressions

Parenthesis are balanced

"...(...((...)...(..)..))"

4 Transition rules satisfied

* OPERATOR $\rightarrow$ LPAREN | OPERAND
* OPERAND $\rightarrow$ RPAREN | OPERATOR
* LPAREN $\rightarrow$ LPAREN | OPERAND
* RPAREN $\rightarrow$ RPAREN | OPERATOR

Expressions: Infix to Postfix Conversion

Generating Random Arithmetic Expressions

- Sequential (use transition rules)
- Recursive
  - `expr(n)` $\rightarrow$ `expr(n-m) op expr(n-m)`
  - where `expr(n)` has `m` operands

Expressions: Infix to Postfix Conversion

Evaluating Expressions: 2 steps

* Convert: from Infix Notation to Polish Notation
* Evaluate: Polish Notation

WHAT is needed for conversion?

The order of operands are preserved!

This suggests a queue

The order of operator may be reversed!

This suggests a stack

Expressions: Infix to Postfix Conversion

Study of `convert` code

(found in `expr/Expr.java` of hw4)

–INPUT: input string `ss`

–OUTPUT: output queue `outQ` of tokens

–Data Structures:

`inQ` stores tokens from input `ss`

`stk` is a stack to help the conversion

Expressions: Infix to Postfix Conversion

Aside: Use De Morgan's law to simplify the code snippet in red

Transform an infix expression to postfix notation

Suppose Q is an arithmetic expression in infix notation. We will create an equivalent postfix express
contain any parentheses.

We will use a stack in which each item may be a left parenthesis or the symbol for an operation.

```
Start with an empty stack.  We scan Q from left to right.

While (we have not reached the end of Q)
   If (an operand is found)
      Add it to P
   End-If
   If (a left parenthesis is found)
      Push it onto the stack
   End-If
   If (a right parenthesis is found)
      While (the stack is not empty AND the top item is
             not a left parenthesis)
         Pop the stack and add the popped value to P
      End-While
      Pop the left parenthesis from the stack and discard it
   End-If
   If (an operator is found)

      If (the stack is empty or if the top element is a left
          parenthesis)
         Push the operator onto the stack
      Else
         While (the stack is not empty AND the top of the stack
                is not a left parenthesis AND precedence of the
                operator <= precedence of the top of the stack)
            Pop the stack and add the top value to P
         End-While
         Push the latest operator onto the stack
      End-If
   End-If
End-While
While (the stack is not empty)
```

Expressions: Infix to Postfix Conversion

Solution:

```
while (stack non-empty
            AND stack.peek() ≠ LPAREN
            AND token ≤ stack.peek() )
      outQ.enqueue( stack.pop())
stack.push(token)
```

Thanks for Listening!

“Algebra is generous,
she often gives more than is asked of her.”

— JEAN LE ROND D’ALEMBERT (1717-83)