Announcements
• Homework 3 quiz due Monday
• Program 2 due 11/01

Material
• Discussion of Project 2
• Implementation of stacks
• Queues
Queue

• Definition:
  • Data Structure that allows adding elements (enqueue) on one end and removal (dequeue) from the other end.

• Dynamic FIFO Storage Structure
  • Size and Contents can change during execution
  • First in First Out
  • Elements are inserted (enqueue) into the rear and retrieved (dequeue) from front.
• Operations
  • enqueue - add element at end
  • dequeue - remove element from front
  • first - examine first element (similar to stack.peek())
  • isEmpty
  • size
Radix Sort

- Algorithm uses an array of queue
- Divide numbers into 10 different queues by looking at each digit at a time
  - 1st loop: use the single digits and place 0s on queue 0, 1s on queue 1, etc.
  - 2nd loop: take numbers in order of queue 0, queue 1, etc. and organize them by tens digit
- repeat for all positions
Radix sort

- One of few sorting algorithms not based on comparison of keys
  - Code is in the book (L& C: Chap. 7)

- What is Big O of this algorithm?
- What are the drawbacks?
Infix to Postfix

- Algorithm uses a string with an infix expression and produces a corresponding postfix expression stored in a queue. Internally, the algorithm uses a queue and a stack to store its results. The queue is returned and the stack is discarded after completion.

- Starts with an empty queue, (the postfix queue), uses an operator stack, & returns the queue.

- Operators have precedence which must be considered, any number of parenthesis are possible.
Infix to Postfix: Steps

- Parse the input infix expression:
  - if you find an operand, add it to the postfix queue
  - if you find an operator, pop off the stack all operators (not parens) that have equal or higher precedence and add them to the postfix queue. Then push the operator found in the input infix expression onto the stack.
  - if you find a left parenthesis, push it onto the stack
  - if you find a right parenthesis, pop operators from the stack and add them to the queue until you find the matching left parenthesis, which you can ignore
- When you find the end of the infix expression, copy all remaining operators in the stack to the queue
- Return the queue
Implementations of Queues

• Array Implementation
  • similar to queues
  • need index to “head” and index to “tail”
  • on insert, shift all up
  • Add to end, remove from front

• What is the inefficiency here?

• How can it be solved?
What if we did not shifting?

- Add to the [rear]
  - increment rear
- Remove from the [front]
  - increment front
Circular Array

• Circular Array Queue
  • Code operations to force array indicies to ‘wrap-around’
    front = (front + 1) % MAXQUE ;
    rear = (rear + 1) % MAXQUE ;

• front and rear indicies delimit the bounds of the queue contents

• Enqueue:
  Move the rear index 1 position clockwise & write the element in that position.

• Dequeue:
  Return element at front and move front one position clockwise

• Count (queue size) is stored and maintained or boolean full status flag maintained.
Circular Array Queue

- Special cases
  - Empty or Full Queue?
    Assume queue has 1 element.
    Dequeue the element.
    Where are the indicies?
    Fill up the queue.
    Where are the indicies?

- Solution
  design implementation to ensure that different states of the queue are represented distinctly
  eliminates need to maintain a queue size count.
Circular Array Queue

• Solution
  • *Front* refers to the position preceding actual front element
    full queue: contains \((\text{max} - 1)\) elements.

• Tradeoff:
  one memory location saves
  processing (maintaining queue size count)

• Distinct States
  Full Queue:
  \[(\text{que}. \text{rear} + 1) \mod \text{MAXQUE} == \text{que}. \text{front}\]
  Empty Queue:
  \[\text{que}. \text{rear} == \text{que}. \text{front}\]
  One-element Queue:
  \[(\text{que}. \text{front} + 1) \mod \text{MAXQUE} == \text{que}. \text{rear}\]