Queues

Tuesday, September 25, 2007
11:00 PM

8.1 Introduction to Queues
- a queue is like a waiting line
  - enter one end
  - exit the other end
  - everyone exits in the same order that they entered
    - (no cutting into line)
- this is called a first-in, first-out (FIFO) data structure
- like a linked list with restrictions & improvements
  - only insert at the tail
  - only delete at the head
  - keep track of both head & tail (also called front & back)
- operations
  - create empty
  - check it empty
  - enqueue - add element
  - dequeue - delete element
  - front - retrieve value of 1st element

8.2 Array Implementation
- have two indices: front & back
  - front is index w/ 1st element
  - back is index of next available slot
- to enqueue - put value in back's slot, increment back
- to dequeue - increment front
- want to avoid shifting elements like we did w/ array based lists
  - have a "circular" array
  - if an index advances beyond end, wrap at back to 0
  - can be done with: (index +1) % capacity
- how to indicate empty w/circular array?
  - cannot use -1 index like lists & stacks
    - modulo formula will never evaluate to -1
  - look at behavior when last element dequeued
    - front & back are now the same
  - so look for front == back
- how about full?
  - if we fill up the array, we also get front == back
    - to prevent this, reserve one empty slot between front & back
  - full when only one empty slot remains
    - (back + 1) % capacity == front
- Pseudocode
  - Default constructor
    - set front & back to 0
  - empty()
    - if front == back
      - return true
    - else
      - return false
  - full()

```
        if (back + 1) % capacity == front
            return true
        else
            return false
    enqueue (elementType)
        if full()
            issue "full queue" error & return
        array[back] = element
        back = (back + 1) % capacity
    dequeue()
        if empty()
            issue "empty queue" error & return
        front = (front +1) % capacity
    elementType front()
        if empty()
            issue "empty queue" error & return
        return array[front]
```

Dynamic Array version

as w/ list & stack, must add functions to allocate & deallocate array

add:

- destructor
- copy constructor
- assignment operator
- constructor that takes an int for capacity

alter:

- default constructor to allocate default capacity

8.3 Linked Queues

like linked stack, linked queue is a specialized form of linked list

- only allows tail insertion & head deletion
- optimized to make both operations constant

to optimize insertion, add pointer to last element called tail or back

# of elements in queue

- 0 elements (empty)
  - head & tail are NULL
- 1 element
  - head & tail point to same node
- 2+ elements
  - head points to 1st element
  - tail points to last element

Operation Pseudocode

```
Default constructor
    set head & tail to NULL
Destructor
    while not empty()
        dequeue()
Copying method
    if source is empty()
        set head & tail to NULL
    else
        set ptr to source's head
        while ptr is not NULL
```

```
            enqueue(ptr->getData())
            set ptr to ptr->getNext()
Copy constructor
    call copying method
Assignment operator
    while not empty()
        dequeue()
    call copying method
empty()
    if head == NULL and tail == NULL
        return true
    else
        return false
enqueue(elementType)
    allocate new node & set data
    if allocation fails
        issue "out of mem" error & return
    if queue is empty()
        set new node's next to NULL
        set head & tail to new node
    else
        set tail's next to new node
        set tail to new node
dequeue()
    if empty()
        issue "empty queue" error & return
    set tmp to head
    if head == tail
        set head & tail to NULL
    else
        set head to head->getNext()
    delete tmp
elementType front()
    if empty()
        issue "empty queue" error & return
    return head->getData()
```