Adam Rich

EN.605.202.87.SP18 Data Structures

Homework Assignment 6

March 13, 2018

**Assignment 6 – More on Lists**

*Write pseudo-code not Java for problems requiring code. You are responsible for the appropriate level of detail.*

*The questions in this assignment give you the opportunity to explore a new data structure and to experiment with the hybrid implementation in Q3.*

1. **A deque (pronounced deck) is an ordered set of items from which items may be deleted at either end and into which items may be inserted at either end.  Call the two ends left and right. This is an access-restricted structure since no insertions or deletions can happen other than at the ends. Implement the deque as a doubly-linked list (not circular, no header). Write InsertLeft and DeleteRight.**

class DequeNode

node pred

node succ

object value

end-class DequeNode

class Deque

node left

node right

method insertLeft(curr as DequeNode)

if left is null (and right is null)

left = curr

right = curr

curr.pred = null // not circular

curr.succ = null

else

curr.succ = left

curr.pred = null

left = curr

end-if

end-method

method deleteRight

object out // return value

if right is null

do nothing or throw error as desired

else

node old\_right = right

right.pred.succ = null

right = right.pred

// shred old\_right

out = old\_right.value

old\_right.pred = null

old\_right.succ = null

old\_right.value = null

end-if

return out

end-method

end-class Deque

1. **Implement a deque from problem 1 as a doubly-linked circular list with a header. Write InsertRight and DeleteLeft.**

Note: the lectures did not discuss whether the head node should be part of the circular structure . . . I am assuming that is should. Lecture said that header should be a different kind of object, but I am confused on how this might be implemented as the pointers should be all of the same type? This is assuming that is not something to worry about.

class DequeNode

node pred

node succ

object value

end-class DequeNode

class DequeHeader

node left

node right

// other info as desired, maybe length

end-class DequeHeader

class Deque

DequeHeader head

method insertRight

if head.right = head //head points to itself

head.left = curr

head.right = curr

curr.pred = head

curr.succ = head

else

node old\_right = head.right

curr.pred = old\_right

curr.succ = head

old\_right.succ = curr

head.right = curr

end-if

end-method

method deleteLeft

if head.left = head

do nothing or throw error as desired

else

node old\_left = head.left

head.left = old\_left.succ

head.left.pred = head

// shred old\_left

old\_left.pred = null

old\_left.succ = null

old\_left.value = null

end-if

end-method

end-class Deque

1. **Write a set of routines for implementing several stacks and queues within a single array. Hint: Look at the lecture material on the hybrid implementation.**

// Not sure what the type of value is going to be, just call it Object

// for now. Could be anything!

class MonsterNode

Object value

int next

int prev

method init(Object v)

value = v

next = -1

prev = -1

end-method

end-class

class MonsterArray

// An array that actually is a dynamic number of stacks and/or queues

// Use position 0 to be the NULL sub-array, i.e. the "linked" sub-array

// of all unused slots in the array

// Pos zero will never be used to store actual data

int N = a predefined constant

init arr = array of MonsterArrayNodes, with "N + 1" elements

method init

// Everything is part of the NULL sub-array on init

// link everything back to slot zero

for i = 0 to N // Remember! Has N+1 slots

arr[i].next = i + 1

arr[i].prev = i - 1

end-for

// Correct last slot's next

arr[N].next = -1

end-method

method getNode(int pos)

return arr[pos]

end-method getNode

method delete(int pos)

// Returns the deleted node

// Assume pos is valid

int pos\_next = arr[pos].next

int pos\_prev = arr[pos].prev

arr[pos\_prev].next = pos\_next if pos\_prev != -1

arr[pos\_next].prev = pos\_prev if pos\_next != -1

// Deep copy for return

MonsterNode out = new MonsterNode

out.prev = pos\_prev

out.next = pos\_next

out.value = arr[pos].value

me.returnToNull(pos)

return out

end-method delete

method insertAfter(int pos, Object v)

// returns the position of the added node

int free = getFromNull

if pos = -1 then // means we are adding the first element of the sub-array

arr[free].prev = -1

arr[free].next = -1

else

int free\_prev = pos

int free\_next = arr[pos].next

arr[free].prev = free\_prev

arr[free].next = free\_next

arr[free\_prev].next = free if not free\_prev = -1

arr[free\_next].prev = free if not free\_next = -1

arr[free].value = v

return free

end-method

private method getFromNull

// returns int pos of free slot

int free = arr[0].next

if free = -1 throw nast error

else

int free\_next = arr[free].next

arr[free\_next].prev = 0 if free\_next != -1

arr[0].next = free\_next

return free

end-method

private method returnToNull(int pos)

// Will want to make sure that pos

// is not already in the NULL sub-array

int zero\_next = arr[0].next

arr[0].next = pos

arr[pos].next = zero\_next

arr[pos].prev = 0

arr[zero\_next].prev = pos

out = arr[pos].value

clear arr[pos].value

return out

end-method returnToNull

private method findLast(int pos)

do while arr[pos].next != -1

pos = arr[pos].next

end-loop

return pos

end-method findLast

end-class

class Stack

MonsterArray monster // ref to the monster array

int first // bottom of stack

int last // top of the stack

method init

arr = ref to MonsterArray

first = -1

last = -1

end-method init

method pop

if last = -1 throw nasty error

else

MonsterNode ret = monster.delete(last)

last = ret.prev

return ret.value

end-method pop

method push(Object v)

// insertAfter returns pos of insterted object

last = monster.insertAfter(last, v)

if first = -1 then first = last

end-method push

method isEmpty

return first == -1?

end-method isEmpty

method peek

return arr.getNode(last).value

end-method peek

end-class

class Queue

MonsterArray monster // ref to the monster array

int first // front of queue

int last // back of queue

method init

arr = ref to MonsterArray

first = -1

last = -1

end-method init

// Take element off front

method delete

if first = -1 throw nasty error

else

MonsterNode ret = monster.delete(first)

first = ret.next

return ret.value

end-method pop

// Add element to end

method insert(Object v)

// insertAfter returns pos of insterted object

last = monster.insertAfter(last, v)

if first = -1 then first = last

end-method push

method isEmpty

return first == -1?

end-method isEmpty

method peek

return arr.getNode(first).value

end-method peek

end-class