**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 deque(){

Class ListNode(){

DataType Data;

ListNode Next;

ListNode Previous;

}

Private ListNode currentNode;

Public void insertLeft(Datatype item){

If (currentNode == null){

currentNode = new ListNode();

currentNode.Data = item;

currentNode.Previous = null;

currentNode.Next = null;

} else {

While (currentNode.Previous != null){

currentNode = currentNode.Previous

}

currentNode.Previous = new ListNode();

currentNode.Previous.Data = item;

currentNode.Previous.Previous = null;

currentNode.Previous.Next = currentNode;

}

}

Public DataType deleteRight(){

DataType Item;

If(currentNode == null){

Return error;

} else {

While (currentNode.Next != null){

currrentNode = currentNode.Next;

}

Item = currentNode.Data;

currentNode = currentNode.previous;

currentNode.next.previous = null;

currentNode.next = null;

return Item;

}

}

}

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

Class deque(){

Class ListNode(){

DataType Data;

ListNode Next;

ListNode Previous;

}

Class ListHeaderNode(){

ListNode head;

ListNode tail;

Int size;

}

Private ListHeaderNode Header;

Public Deque(){

Header = new ListHeaderNode();

Header.head = null;

Header.tail =null;

Header.size = 0;

}

Public void insertRight(DataType item){

ListNode currentNode = new ListNode();

currentNode.item = obj;

if(header.head == null){

currentNode.Previous = currentNode;

currentNode.Next = currentNode;

Header.head = currentNode;

Header.tail = currentNode;

Header.size = 1;

} else {

currentNode.previous = header.tail;

currentNode.next = header.head;

Header.tail = currentNode;

Header.size = Header.size + 1;

}

}

Public DataType deleteLeft (){

DataType item;

If (header.head = null){

Return error;

} else {

item = header.head.Data;

Header.head.previous = null;

Header.tail.next = header.head.next;

Header.head.next = null;

Header.head = header.tail.next;

header.head.previous = header.tail;

header.size = header.size – 1;

return obj; }

}

}

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.**

Class HybridImplementation(){

Int rows = 100;

String hybridArray[][] = new String[rows][2];

For (int I=0; I< rows ; I++){

hybridArray[I][2] = I+1;

}

hybridArray[rows-1][2] = -1;

//Assuming standard stack implementation already declared

Stack freeSpace = new Stack;

For (I = rows-1; I >=0; I--){

freeSpace.push(I);

}

Class arrayStack{

Public arrayStack(String item){

if(freeSpace.isEmpty() == true){

return array full error;

}

Int header = freeSpace.pop();

hybridArray[header][1] = item;

hybridArray[header][2] = -1;

Int tail = header;

}

Public void push(String item){

if(freeSpace.isEmpty() == true){

return array full error;

}

Int I = freeSpace.pop();

hybridArray[I][1] = item;

hybridArray[I][2] = -1;

hybridArray[tail][2]= I; //finds and sets the previous next value to the new value, then sets the tail to the new value

this.tail = I;

if(this.header == null){

this.header=I; // in case of empty stack

}

}

Public String pop(){

If (header == null){

Return empty stack error;

}

Int tempTail =tail;

String tempReturn = hybridArray[tempTail][1]; //sets the temp value for later return

Boolean found;

For(I = 0; I < rows; i++){

If(hybridArray[i][2] == tempTail){ //goes through the “Next” portion of the array, finds the value that points to the tail, then changes it to -1

hybridArray[i][2] = -1;

tail = I;

found = true;

break;

}

If (found == false){

Header = null;

Tail = null; //if it wasn’t found, it means that this was the last value in the stack, and the stack is now empty

}

freeSpace.push(tail); //return the tail value into free space.

Return tempReturn;

}

}

Class arrayQueue{

Public arrayQueue(String item){

if(freeSpace.isEmpty() == true){

return array full error;

}

Int header = freeSpace.pop();

hybridArray[header][1] = item;

hybridArray[header][2] = -1;

Int tail = header;

}

Public void push(String item){

if(freeSpace.isEmpty() == true){

return array full error;

}

Int I = freeSpace.pop();

hybridArray[I][1] = item;

hybridArray[I][2] = -1;

this.tail = I;

hybridArray[tail][2]= I; //finds and sets the previous next value to the new value

if(this.header == null){

this.header=I; // in case of empty stack

}

}

Public void pop(){

String tempReturn = hybridArray[this.header][1];

Int tempIndex = header;

If(tempIndex != this.tail){

Int nextIndex = hybridArray[tempIndex][2];

header = nextIndex;

hybridArray[tempIndex][2] = -1; //deletes the reference to any other rows so the array is not confused. Not necessary with stacks since the tail is popped and it is already -1

} else {

Header = null;

Tail = null;

}

freeSpace.push(tempIndex);

return tempReturn;

}

}