Self-Practice Week 2 - Fundamentals (part 2)

Data Structures

The goal of this assignment is to practice designing, implementing and testing basic data structures.

Exercise 1 – Circular Lists

A circular linked list is a linked list where all nodes are connected to form a circle. There is no *null* at the end. A circular linked list can be a singly circular linked list or doubly circular linked list. Design and implement the circular list data type (both singly and doubly linked), so to provide the following API:

```
// construct an empty circular list
CircularList()

// is the list empty?
isEmpty()→ bool

// return the number of items in the list
length() → int

// add the item at the end
append(item)

// add the item at the beginning
prepend(item)

// remove the item in position pos
delete(pos)→ item

// return the item in position pos
access(pos)→ int
```

Corner cases. When testing your implementation, consider the following corner cases:

- the client calls either `append()` or `prepend()` on an empty list
- the client calls `delete()` when the list is empty
- the client calls `access(-3)`

Exercise 2 – Deque

A *double-ended queue* or *deque* is a generalization of a stack and a queue that supports adding and removing items from either the front or the back of the data structure. Create a generic data type Deque that implements the following API:

```
// construct an empty deque
Deque()
```

```
// is the deque empty?
isEmpty()→ bool

// return the number of items on the deque
length()→ int

// add the item to the front
addFirst(item)

// add the item to the back
addLast(item)

// remove and return the item from the front
removeFirst()→ item

// remove and return the item from the back
removeLast()→ item
```

Corner cases. When testing your implementation, consider the following corner cases:

- the client calls either `removeFirst()` or `removeLast()` when the deque is empty
- the client calls either `addFirst()` or `addLast()` when the deque is empty

Exercise 3 – Palindrome Checker

Write an algorithm to check whether an input string is palindrome. A palindrome is a string that reads the same forward and backward, for example, "`radar`". Your checker should be case insensitive. That means the string "`Topspot`" should be considered palindrome.

Corner cases. When testing your implementation, consider the following cases:

- the string is empty
- the string is palindrome and has an odd number of characters
- the string is palindrome and contains a mix of upper/lower case characters