CIS 55 Data Structures
Programming Exercise 1, Due no later than 29-Oct-2021, 11:59pm Pacific Time

In this assignment, you will create a Linked List data structure variant called a "Circular Linked List". The Node structure is the same as discussed in the slides and defined as follows (we will use integers for data elements):

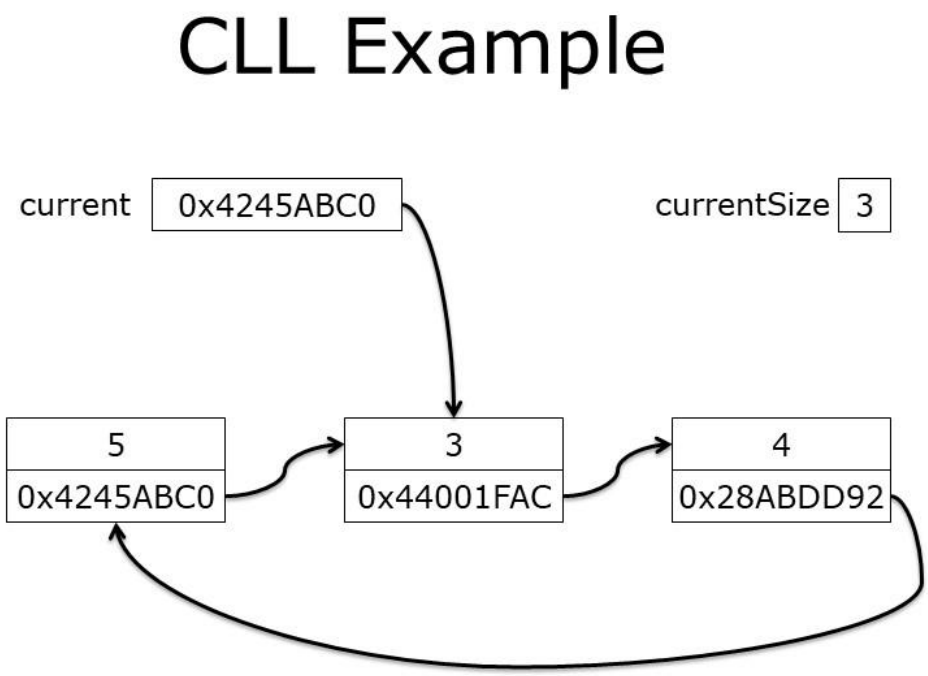
```
public class Node
{
	public int data;
	public Node next;
}
```

For the Circular Linked List, its class definition is as follows:

```
public class CircularLinkedList
{
	public int currentSize;
	public Node current;
}
```

In this Circular Linked List (CLL), each node has a reference to an existing next node. When Node elements are added to the CLL, the structure looks like a standard linked list with the last node's next pointer always pointing to the first. In this way, there is no Node with a "next" pointer in the CLL that is ever pointing to null.

For example, if a CLL has elements “5”, “3” and “4” and “current” is pointing to “3”, the CLL should look like:



Key observations with this structure:

1. The currentSize is 3 meaning there are 3 nodes in the CLL
2. The current node in the CLL is always pointing to an existing node. If current is null, the CLL is empty
3. All Nodes’ next pointers are pointing to an existing node
4. If there is only 1 Node in the CLL, its next pointer would be pointing to itself

Part 1) Using any programming language you prefer, at a minimum, complete the following functions for the CLL:

public CircularLinkedList()
public void insertAfterCurrent(int n)
public Node search(int n)
public void insertBeforeCurrent(int n)
public boolean update(int o, int n)
public boolean delete(int n)
public void printSize()
public void printCurrent()
public void printList()

The requirements for each function are as follows:

CircularLinkedList():

- This function is the default constructor for the CLL and creates an empty list with current pointing to null and currentSize set to 0

insertAfterCurrent(int n):

- This creates a new node and inserts it "after" the current node. If the list is empty, the new node becomes current and its next pointer points to itself. If the list has more than 1 element, the new node is placed "after" current and when complete, the new node that was inserted becomes the new "current" node
- For example, if the CLL is 3-5-4 and current is pointing to 3, after insertAfterCurrent(6), the CLL will be 6-5-4-3 and current will be pointing to 6

search(int n):

- This searches the CLL for a Node where its data property matches the given n. If a match is found, the Node is returned and the "current" Node in the CLL is now this Node. If a match is not found, the function returns null and the "current" Node should be the same as when the function started
- For example, if the CLL is 3-5-4 and current is pointing to 3, search(4) will find and return Node 4, and current will also be pointing to 4. If search(6) had been performed, search will return null, and current would still be pointing to 3

insertBeforeCurrent(int n):

- This creates a new node and inserts it "behind" the current node. If the list is empty, the new node becomes current and its next pointer points to itself. If

the list more than 1 element, the new node is placed "behind" current. When complete, the new node inserted is the new "current" node
- For example, if the CLL is 3-5-4 and current is pointing to 3, after insertBeforeCurrent(6), the CLL will be 6-3-5-4 and current will be pointing to 6

update(int o, int n):
- This searches the CLL for a Node where its data property matches the given o. If a match is found, the Node's data element is updated to the given n and the function returns true. The "current" Node in the CLL will also now be this Node. If a match is not found, the function returns false and the "current" Node should be the same as when the function started
- For example, if the CLL is 3-5-4 and current is pointing to 3, update(4,7) will find Node 4, change it to 7, set current to be pointing to 7 making the CLL 7-3-5, and update will return true. If update(6,7) had been performed, update will not be able to find 6, return false, and current would still be pointing to 3

delete(int n):
- This searches the CLL for a Node where its data property matches the given n. If a match is found, the Node's is removed from the CLL and the "current" Node is the Node that is "after" the Node that was removed and the function returns true. If a match is not found, the function returns false and the "current" Node should be the same as when the function started
- For example, if the CLL is 3-5-4 and current is pointing to 3, delete(5) will find Node 5, delete it and make the current point to 4 making the CLL 4-3, and delete will return true. If delete(7) had been performed, delete will return false, and current would still be pointing to 3

print functions:
- printSize: displays the number of elements in the CDLL
- printCurrent: displays the data value of the current element in the CDLL. If the list is empty, it prints "Empty List"
- printList: displays each element in the CLL. If the list is empty, it prints "Empty List"

Part 2) Write a main test program with a script of function calls that test all these functions. You can use whatever test data and function calls you like that covers as many practical scenarios you think are adequate. For each function call test, remember there may be different scenarios to consider:

1. Empty lists
2. Lists with only 1 Node
3. Lists with more than 1 Node

Submitting your work

When you are complete, email me a zip or rar file containing all of the following files:

1. All code files. If you are using Java, submit all .java files you coded. If you are using C++, submit all .h and .cpp files you coded
2. Take and submit screenshots of your program running and demonstrating all your function call tests
3. Include any other files you feel are relevant and want to submit. For example, if you want to create a separate document to describe any of your work, that is acceptable to include

Group work: I recommend working on your own, but you are welcome to work in pairs or groups of 3. If you work in a group, put all names on the work you submit. I also recommend all team members fully understand the details of all code developed.