Task Sheet 1: Basic List Operations

Level – Pass/Credit (This task would require the students to go a little beyond the concepts of Basic List Operations to further explore Circular Singly Linked List.)

SIT221 Data Structures and Algorithms - Practical Task

Task Objective

In this task you will learn about and create a Circular Singly Linked List, a data structure that can store an arbitrary amount of data components and supports a variety of common operations such as read, write, and delete.

Background

The goal of this task is to investigate the development of a Circular Singly Linked List (CSLL), a data structure that can store an arbitrary number of data components and supports a variety of common read, write, and delete operations. Unlike a typical singly linked list, a CSLL connects the last node to the first node, making a circle. This circular link enables efficient looping across the list.

A CSLL has various advantages over other data structures, such as arrays or doubly linked lists. For example, a CSLL does not need a contiguous memory space to hold data and can expand dynamically. Furthermore, the CSLL's circular design provides for more efficient operations, particularly when cycling around the list several times.

Task Details

To complete the task, follow the instructions below:

Initial Setup:

Create a new Microsoft Visual Studio project and include the CircularSinglyLinkedList.cs file. This file includes a template for the CircularSinglyLinkedList class. The task's goal is to add needed functionality to the class in order to create a fully working data structure. Import the Tester.cs file into the project to activate the prepared Main function, which is required for debugging and testing the anticipated program class and interfaces for runtime and logical errors.

Node class:

Learn about the CircularSinglyLinkedList's nested Node class and its structure. This is a generic class designed to represent a node in a circular singly linked list. A node contains a data record (payload) and a reference to the next node.

CircularSinglyLinkedList Class:

Complete the CircularSinglyLinkedList class using the following methods:

- INode AddFirst(T value)

- INode AddAfter(INode node, T value)
- INode Find(T value)
- void Remove(INode node)
- void RemoveFirst()
- void RemoveLast()
- void Clear()

Testing:

Use the Tester class to extensively test the CircularSinglyLinkedList, addressing all potential logical flaws and runtime errors. Extend the test cases to guarantee full coverage.

Task Requirements

1. INodeAddFirst(T value)

   Description: Inserts a new node at the beginning of the Circular Singly Linked List with the provided value. If the list is empty, the new node is both the head and the tail, with the Next pointer pointing to itself. Otherwise, the new node's Next pointer is set to the current head while the tail's Next pointer is updated to the new node. This action preserves the list's circularity.
   Returns: The newly formed node is an instance of INode.

2. INode AddAfter (INode node, T value)

   Description: Inserts a new node with the supplied value directly after the given node in the list. If the previous node is the tail, the new node becomes the new tail, with the Next pointer pointing to the head. If the provided node is in the middle of the list, set the new node's Next pointer to the given node's Next, and update the given node's Next pointer to the new node.
   Returns: The newly formed node is an instance of INode.

3. INode Find (T value)

   Description: Searches the list for the first node that contains the provided value. The method walks the list, beginning at the head, and compares each node's value to the specified value. If a match is detected, the associated node is returned. If no matches are discovered, null is returned.
   Returns the detected node as an instance of INode, or null if no node with the supplied value exists.

4. void Remove (INode node)

   Description: Deletes the selected node from the list. The procedure searches the list to locate the node and keeps track of the preceding node to update the Next pointer. If the head is the node to be removed, it is relocated to the next node, and the tail's Next pointer is set to the new head. If the node to be deleted is the tail, the previous node is

replaced with the new tail, and the Next pointer is set to the head. This action preserves the list's circularity.
Returns: Nothing.

5. void RemoveFirst()

   Description: Removes the list's first node (the head). If the list is empty, an InvalidOperationException is raised. If there is just one node, the head and tail are set to null. Otherwise, the head is moved to the next node, while the tail's Next pointer is set to the new head.
   Returns: Nothing.

6. void RemoveLast()

   Removes the list's final node (the tail). If the list is empty, an InvalidOperationException is raised. If there is just one node, the head and tail are set to null. Otherwise, the list is explored to locate the node preceding the tail. This node's Next pointer is moved to the head, and it becomes the new tail.
   Returns: Nothing.

7. void Clear()

   Clears the list by setting the head and tail to null, deleting all nodes from it. The count is set to 0.
   Returns: Nothing.

Expected Printout

Appendix A provides an example printout for your program. If you are getting a different printout then your implementation is not ready for submission. Please ensure your program prints out Success for all tests before submission.

Further Notes

- Learn the material of chapters 3.4 and especially that of section 7.3.3 of the SIT221 course book “Data structures and algorithms in Java” (2014) by M. Goodrich, R. Tamassia, and M. Goldwasser. You may access the book on-line for free from the reading list application in CloudDeakin available in Resources ® Additional Course Resources ® Resources on Algorithms and Data Structures ® Course Book: Data structures and algorithms in Java. As a complementary material, to learn more about a singly linked and doubly linked lists, you may refer to Chapter 2 of SIT221 Workbook available in CloudDeakin in Resources

  ® Additional Course Resources ® Resources on Algorithms and Data Structures ® SIT221 Workbook.

- If you still struggle with such OOP concepts as Generics and their application, you may wish to read Chapter 11 of SIT232 Workbook available in Resources ® Additional Course Resources ® Resources on Object-Oriented Programming. You may also have to read Chapter 6 of SIT232 Workbook about Polymorphism and Interfaces as you need excellent understanding of these topics to progress well through the practical tasks of the unit. Make sure that you are proficient with them as they form a basis to design and develop programming modules in this and all the subsequent tasks. You may find other important topics required to complete the task, like exceptions handling, in other chapters of the workbook.
- We will test your code in Microsoft Visual Studio 2017. Find the instructions to install the community version of Microsoft Visual Studio 2017 available on the SIT221 unit web-page in CloudDeakin at Resources ® Additional Course Resources ® Software ® Visual Studio Community 2017. You are free to use another IDE if you prefer that, e.g. Visual Studio Code. But we recommend you to take a chance to learn this environment.

Marking Process and Discussion

To get your task completed, you must finish the following steps strictly on time.

1. Work on your task either during your allocated lab time or during your own study time.
2. Once the task is complete you should make sure that your program implements all the required functionality, is compliable, and has no runtime errors. Programs causing compilation or runtime errors will not be accepted as a solution. You need to test your program thoroughly before submission. Think about potential errors where your program might fail. Note we can sometime use test cases that are different to those provided so verify you have checked it more thoroughly than just using the test program provided.
3. Submit your solution as an answer to the task via the OnTrack submission system. This first submission must be prior to the submission "S" deadline indicated in the unit guide and in OnTrack.
4. If your task has been assessed as requiring a "Redo" or "Resubmit" then you should prepare a new submission. You will have 1 (7 day) calendar week from the day you receive the assessment from the tutor. This usually will mean you should revise the lecture, the readings indicated, and read the unit discussion list for suggestions. After your submission has been corrected and providing it is still before the due deadline you can resubmit.
5. If your task has been assessed as correct, either after step 3 or 4, you can "discuss" with your tutor. This first discussion must occur prior to the discussion "D".
6. Meet with your to demonstrate/discuss your submission. Be on time with respect to the specified discussion deadline.
7. The tutor will ask you both theoretical and practical questions. Questions are likely to cover lecture notes, so attending (or watching) lectures should help you with this compulsory interview part. The tutor will tick off the task as complete, only if you provide a satisfactory answer to these questions.
8. If you cannot answer the questions satisfactorily your task will remain on discussion and you will need to study the topic during the week and have a second discussion the following week.
9. Please note, due to the number of students and time constraints tutors will only be expected to mark and/or discuss your task twice. After this it will be marked as a "Exceeded Feedback".
10. Note that we will not check your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this

reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work through the unit.

11. Final note, A “Fail” or “Exceeded Feedback” grade on a task does not mean you have failed the unit. It simply means that you have not demonstrated your understanding of that task through OnTrack.

Appendix A – Sample output of Test Cases

```
Test A: Create a new list

:: SUCCESS: list's state []

Test B: Add a sequence of numbers 1, 2, 3, 4, 5 using AddLast()

:: SUCCESS: list's state [1 -> 2 -> 3 -> 4 -> 5]

Test C: Remove the first number using RemoveFirst()

:: SUCCESS: list's state [2 -> 3 -> 4 -> 5]

Test D: Add a number at the start using AddFirst(0)

:: SUCCESS: list's state [0 -> 2 -> 3 -> 4 -> 5]

Test E: Find the node containing 3

:: SUCCESS: Node found: 3

Test F: Remove the node containing 3

:: SUCCESS: list's state [0 -> 2 -> 4 -> 5]

Test G: Clear the list

:: SUCCESS: list's state []

---------- SUMMARY ----------

Tests passed: ABCDEFG
```

All codes (Answered and Implemented) for this Task sheet:

https://drive.google.com/drive/folders/1WvUD9ZCL9VN7mpn-f-GVR75DgXs57LnX?usp=share_link