California State University, Sacramento
College of Engineering and Computer Science

Computer Science 130: Data Structures and Algorithm Analysis

Spring 2024 – Project #1 – A "Proper" Linked List

Overview

For this assignment, you are going to create a well-written, efficient, and versatile Linked-List class. You will use this in a future assignment to create an equally efficient Stack and Queue.

Part 1: Linked List Class

You are going to write a doubly linked-list class. You will use this again in the future. So, do a good job on it, or you will hurt your future assignments.

Your methods **must** be O(1) for all adds and removes. So, you **must** maintain a reference (link) to the tail. The following is the required interface for the class. This is not a deque, but it has many of the same features.

`LinkedList Class`		
`String`	`About()`	Returns text about you – the author of this class.
`void`	`AddHead(String value)`	Adds an object to the head of the list. **This must be O(1).**
`void`	`AddTail(String value)`	Adds an object to the tail of the list. **This must be O(1).**
`String`	`RemoveHead()`	Removed an object from the head of the list. **This must be O(1).** If the list is empty, return an empty string.
`String`	`RemoveTail()`	Removed an object from the tail of the list. **This must be O(1).** If the list is empty, return an empty string.
`Boolean`	`IsEmpty()`	Returns true if the linked list is empty. In other words, the head is null.
`String`	`ToString()`	Returns the contents of the linked list with comma separated values. **This must be O(n). Do not use recursion.**
`String`	`ToStringReverse()`	Returns the contents of the linked list, from last to first, with comma separated values. **This must be O(n). Do not use recursion.**

Part 2: Node Class

You are going to create a very, very basic node class. Make sure to come up with a nice, well-documented, and well-written solution. We may build upon it, later, to create more advanced data structures.

This class must be encapsulated (i.e. hidden) within the LinkedList class. In other words, it cannot be public.

Node Class		
	Node(String)	Constructor
Node	Prev	The previous node in the chain.
Node	Next	The next node in the chain.
String	Value	The value that the node contains. Do not declare it of type "object".

Part 3: Testing

A number of test files will be provided to you for testing your code. The format is designed to be easy to read in multiple programming languages. You need to use the classes, built in your programming language, to read the source files.

File Format

The first line of the data contains the total digits in the key. You might want to save this value – I can be used to separate the key from the value (using the substring function found in most programming languages).

```
Value 1
Value 2
...
Value n
END
```

The following is one of the most basic test files on the website.

File: years.txt

```
Sutter's Fort (1839)
Bear Flag Revolt (1846)
Sacramento State (1947)
Nachos (1953)
Buffalo wings (1964)
Great Toilet Paper Shortage (2020)
END
```

Reading the File

Different programming languages implement file IO in different ways. Some use classes and others use more primitive (or ingrained) features. This will take the form of either a While Loop or a Do Loop. In all cases, your main testing class will read the contents of the file and it to a instance of your linked-list class.

Do not make this a method in your LinkedList – keep it in main(). Why? It will make future assignments easier.

In the pseudocode below, I show the basic approach for both types of loops. Your programming language will use one or the other. Some programming languages will return Null if the end of the file was reached. Others may contain a method on the file reading class.

```
while the file has more data
    read a value
    list.AddTail(value)
end while

print list.ToString()
print list.ToStringReverse()
```

Allowed Programming Languages

You may use any of the following programming languages.

- C#
- C++ (not recommended)
- Java
- Swift
- Visual Basic .NET

The following **cannot** be used:

- Groovy
- JavaScript
- Kotlin
- Lua
- Nim
- Python
- Ruby
- Scala
- TypeScript

Proper Style

Well-formatted code

Points will be deducted if your program doesn't adhere to basic programming style guidelines. The requirements are below:

1. If programming C++, Java, or C#, I don't care where you put the starting curly bracket. Just be consistent.
2. Indentation must be used.
3. Indentation must be consistent. Three or four spaces works. Beware the tab character. It might not appear correctly on my computer (tabs are inconsistent in size).
4. Proper commenting. Not every line needs a comment, but sections that contain logic often do. Add a comment before every section of code – such as a loop or If Statement. Any complex idea, such as setting a link, must have a comment. Please see below:

The following code is well formatted and commented.

```
void foo()
{
    int x;

    //Print off the list
    x = 0;
    while (x < this.count)
    {
       items.Add(this[x]);  //Add the item to the temporary list
       x++;
    }
}
```

Poorly-formatted code

The following code is poorly formatted and documented.

```
void foo()
{
    int x;

x = 0;
while (x < this.count)   {
     items.Add(this[x]);  //Call add on items.
       x++;
}
}
```

Requirements

You must write your linked-list yourself.

Do NOT use any built-in collection classes such as lists, arraylists, templates, etc…

If you use any of these, you will receive a zero. No exceptions. No resubmissions.

The following are the requirements for this assignment:

- This **must** be completely all your code. If you share your solution with another student or re-use code from another class, you will receive a zero.
- Do not use any built-in collection classes (many programming languages provide them).
- You **must** write your own Linked-List. It must all be **O(1)** for all add/removes. This means your Linked-List class must be O(1).
- But, any assignment attempting to implement pools will automatically receive a zero. We haven't covered them yet.
- You may use any of the programming languages listed below.
- Create some excellent testing for your class. I want at least 10 examples.
- Proper style (see below).

Grading

1	All add/remove functions are O(1)	20%
2	All remove functions are O(1).	20%
3	Correct interface for the LinkedList	10%
4	Correct interface for the Node	10%
5	Node class is encapsulated in the LinkedList class (private)	10%
6	Proper Style	10%
7	Testing by Reading the File	10%
8	Test RemoveHead() and RemoveTail()	10%

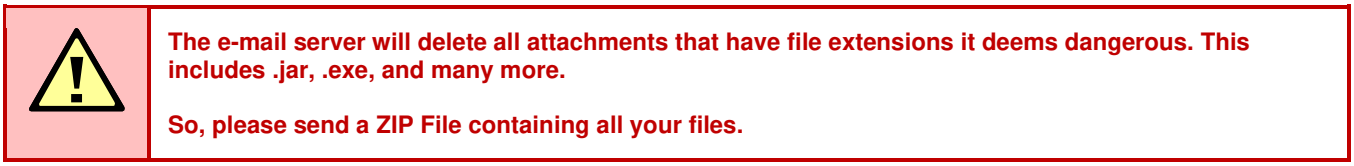
Due Date

Due **February 16, 2024** by 11:59 pm.

Given you already have developed excellent programming skills in CSc 20, this shouldn't be a difficult assignment. **Do not send it to canvas. I will not read nor grade Canvas e-mails.**

E-Mail the following to **dcook@csus.edu**:

- The source code for the classes. Just send the source code files (.java, .cs, .cpp,, etc….)
- The source code containing your tests.
- Output generated by your tests. Either send a text file (saved from the output) or a screenshot.



The e-mail server will delete all attachments that have file extensions it deems dangerous. This includes .jar, .exe, and many more.

So, please send a ZIP File containing all your files.

Some Helpful Pseudocode

AddHead Pseudocode

```
procedure AddHead(String value)
    node = new Node(value)     ... Create a new node.

    if the list is empty      ... is head null?
        head = node            ... Link both head and tail to the new node
        tail = node
    else
        node.next = head       ... Link the new node to the head
        head.prev = node       ... Link the old head to the new node
        head = node            ... The head is now the new node
    end if
end procedure
```

AddTail Pseudocode

```
procedure AddTail(String value)
    node = new Node(value)   ... Create a new node.

    if list is empty        ... is head/tail null?
        head = node
        tail = node
    else
        tail.next = node      ... Link the old tail to the new node
        node.prev = tail      ... Link new node to the old tail
        tail = node           ... The new node is the tail
    end if
end procedure
```