EECS 114: Assignment 1

October 2, 2015

Due Tuesday 13 Oct 2015 at 12:00 AM

1 Linked List

Implement a singly-linked list class of type `char` named `MyList` in a source file named `MyList.java`. Define a `Node` class as a private data member of the `MyList` class. Updates or accesses to individual node instances are strictly made via the `Node` member methods. A sample `MyList` class file is provided below.

In a separate file named `Main.java` implement a `Main` class with main method. This will serve as a test harness to exhaustively test the functionality of the list's methods. One approach for testing is to have a test method for each list member method. Test for expected behavior. Test that erroneous behavior is caught. Test for special cases. For full credit you must have a test case for each method you implement.

A description of the required functionality is below. A reasonable plan of attack is to implement and test the more straight-forward methods prior to moving on to the more challenging ones. Your goal should be to complete a portion of the program, and the associated tests, each day.

NOTE: You may NOT add a `tail` pointer to the `MyList` class or make any changes to the required method signatures below. You may add a constructor that sets a node's `next` and `value`.

```
public class MyList {
    // Node class
   private class Node {
      private char value;
      private Node next;
      public Node() { this(null, null); }

      // Accessor methods
      public char getElement() { return value; }
      public Node getNext() { return next; }

      // Modifier methods
      public void setElement(char v) { value = v; }
      public void setNext(Node n) { next = n; }
    }
    // MyList class
    private Node head;
    // Implement required methods here
}
```

1.1 MyList Class Methods

The required `MyList` class methods are as follows.

- `MyList(MyList rhs)` - Constructor. Instantiates `this` list as a deep copy of `rhs`.
- `MyList (char[] charArray, int n)` - Constructor. Instantiates `this` list as a deep copy of `charArray` of length `n`.
- `boolean remove(int index)` - Removes the element at the specified position in `this` list. Returns `true` if this list contained the specified element.
- `boolean remove(char value)` - Removes the first occurrence of the specified element from `this` list, if it is present. Returns `true` if this list contained the specified element.
- `boolean removeAll(char value)` - Removes all occurrences of the specified element from `this` list, if it is present. Returns `true` if this list contained the specified element.
- `Node previous(Node curr)` - Returns the previous node for the specified node `curr`.
- `Node next(Node curr)` - Returns the next node for the specified node `curr`.
- `bool contains(char value)` - Returns `true` if this list contains the specified `value`.
- `char get(int index)` - Returns the element at the specified position in this list. Index of first element in list is `0`.
- `void set(int index, char value)` - Replaces the element at the specified position in this list with the specified element.
- `boolean equals(MyList llist)` - Compares the specified `MyList` with `this` list for equality. Equal lists have exactly identical items in same order in list.
- `void pushFront(char value)` - Inserts a node with specified `value` at the front of the list.
- `void pushBack(char value)` - Inserts a node with specified `value` at the back of the list.
- `void popFront()` - Removes the front item from the list.
- `void popBack()` - Removes the last item from the list.
- `void swap(int i, int j)` - Swaps the `value` of the node at position `i` in the list with `value` of the node at position `j`. Be sure you handle out-of-range calls.
- `void insertAtPos(int i, char value)` - Inserts a node with specified `value` at position `i` in the list, shifting elements starting at i to the right, if needed.
- `void insertAfter(int i, char value)` - Inserts a node with specified `value` at position `i+1` in the list.
- `void insertBefore(int i, char value)` - Inserts a node with specified `value` at position `i-1` in the list.
- `MyList subList(int fromIndex, int toIndex)` - Returns a list object that is a portion of `this` list between the specified fromIndex, inclusive, and toIndex, exclusive.
- `int find(char value)` - Returns the position of the first occurrence of `value` in `this` list. If the character is not in the list, the method returns `-1`.
- `int find(MyList queryStr)` - Returns the position of the first occurrence of `queryStr` in `this` list. If `queryStr` is not in the list, the method returns `-1`.

- `char[] toArray()` - Returns an array containing all of the items in this list in proper sequence (from first to last element).
- `void reverse()` - Reverses the items in the list.
- `int size()` - Returns the number of items in the list.
- `void print()` - Prints the contents of the list. Allows for printing a `MyList` instance `llist` to standard output in following way, `llist.print();`
- `String toString()` - Overrides the `toString()` method. Returns the formatted contents of `this` as a String. Allows for printing a `MyList` instance `llist` to standard output in following way, `System.out.println("llist contents = " + llist);`

1.2 MyList Analysis

This portion of your assignment must be typewritten and submitted as a text file named `assn1.txt` to be included in your `assn1.tgz` submission. The contents of the file should be an itemized list of the `MyList` class method and constructor signatures. For each individual entry provide running time (T(n)) analysis denoted in asymptotic notation for worst-case, best-case, and average-case running times. In no more than a sentence or two, justify each result.

2 Software and commands for remote login

Your Java implementation of programming assignment 1 will be tested and graded on the EECS Linux servers. Be sure you test your final submission on either `zuma.eecs.uci.edu` or `crystalcove.eecs.uci.edu`. Compile from the source on the server to ensure you are submitting code that compiles and runs. **Code that does not compile, will not be graded.** You can connect to the EECS Linux servers from virtually any computer anywhere that has internet access. What you need is a client program for *remote login*.

Use **`ssh`** as the primary way to connect to the server. **`ssh`** stands for *secure shell*, and it encrypts your network communication, so that your data cannot be understood by snoopers. For file transfers, use **`sftp`** or **`scp`**, which are secure. You could also set up an *ssh-tunnel* so that previously unencrypted communications can be encrypted. If you have a Windows machine you can download WinSCP (`https://winscp.net/eng/download.php`) which has a GUI interface for file transfer. Or use **`scp`** from the command line. Here are some examples on how to do this, `http://www.hypexr.org/linux_scp_help.php`. Depending on what computer you use, it may have a different *implementation* of **`ssh`**, but the basic function underneath are all the same.

- If you are logging in from a Windows machine, you can use **`PuTTY`** Telnet and SSH client. `http://www.chiark.greenend.org.uk`
- `MacOS X already has this built-in (use Terminal or X11 to run a unix shell). Most Linux distributions also bundle` **`ssh`**`.`
- `If you are logging in from an X terminal, you can use the command below (substitute your actual EECS username.`
  `% ssh -X username@zuma.eecs.uci.edu`
  `(Note: % is the prompt, not part of your command) It will prompt you for your password. Note that the -X option allows you to run programs that open X windows on your screen.`

3 Compiling your code

Compiling and running your program is done in two steps.
To test your program, it must be *compiled* with the **`javac`** command.
`% javac Main.java MyList.java`
Upon successful completion of the above command, a `.class` file is created with the same name as the sourcefiles.

There should also be a `.class` file for the `Node` class generated. To *run* your program, invoke the **`java`** command on the `Main.class`.

```
% java Main
```

Note that there is no `.class` extension required in order to run the java program.

4 Submit your work

You must turn-in a tar archive through the submission link on the EECS 114 Piazza webpage under Resources. The name of your submission must be `assn1.tgz`. Inside the archive there should be three files only, your two Java source code files `MyList.java, Main.java` and your running-time analysis `assn1.txt`. No other named files will be accepted. Do not include any directories or object (e.g., *.class, executable) files. Be sure to create your tar archive with the following command **`%tar -czvf lab1.tgz Main.java MyList.java assn1.txt`**. For testing and grading we will untar your submission with the following command, **`%tar -xzvf lab1.tgz`**.

Do not forget to put a class header on every file you create or modify. Files lacking a header will not be graded. As always, re-download and test your submission. Files that are corrupted or cannot be read cannot be graded. You should **ALWAYS** turn in what you have completed thus far of the program at least 6 hours before the due date (by 6 PM). Then continue to work on your program and turn in the most current version as you get it working.