Assg 13: Process List of Integers

CSci 515 Spring 2015

2015-04-21

Dates:

Due: Tuesday May 5, by Midnight

Objectives

- More practice processing linked lists
- More practice with dynamic memory allocation
- More practice with pointers

Description

In this lab you will create a couple of functions to process linked lists of integer values. You should use your generation function from the lab in order to create randomly generated linked lists to test these functions.

Perform the following tasks:

1. Implement a function called `displayList()`. This function should take a list of integers as its input and display the list on standard output. Use this function to demonstrate your implementations of the following functions.

2. Create a function called `averageList()`. This function should take a pointer to the head node in a linked list of integers as its only parameter. This function should process all elements in the list, to find their average value and return the result. Use floating point arithmetic to sum up and average the values, and return a floating point value as the

result of this function. In your `main()` function, create a randomly generated list with 10 elements, display the list, and demonstrate calling this function to find the average of the 10 values in your list.

3. Implement a function called `concatenateLists()`. This function will take two linked lists as its input parameters. This function should cause the second linked list that is passed in to be concatenated on to the end of the first linked list. To do this, you need to find the last `Node` in the first linked list, then set its next pointer to point to the head of the second linked list. If I were writing this, I might create a small helper function that returns the last node in a linked list, called something like `findLastNode()`, and use this result to implement the concatenate operation. The result of calling this function is that the first list will be modified. Thus this function does not return a new result nor create a new list. But after calling the function, the pointer to the first list will now point to the concatenated result of appending the second list. In your `main()` function, create two lists of integers of size 4 and 3, display the two lists. Then use the concatentate function and show that the resulting lists are concatenated again by displaying the first list.

4. Implement a function called `reverseList()`. This function will take a list of integers as input (a singly linked list), and it will return a new list. This function should construct a new list that will have all of the items of the original list, but in reverse order. For example, given the list `5 -> 3 -> 2 -> NULL` this function would return a new list `2 -> 3 -> 5 -> NULL`. This function should not destroy the original list, it needs to dynamically create new `Node` items and create a new list, and copy the integer values from the original list nodes to the new list nodes. This function will return the pointer to the head of this new list as the result of calling this function. In your `main` function create a list of 8 integers, reverse it, and display the original and the reversed list.

5. Demonstrate all of your functions in your `main()` function. Make sure you adequately test and demonstrate the correct working of all of your functions.

Here is an example output from running a correct implementation of this assignment:

```
List of 10 integers:
```

```
4 -> 7 -> 18 -> 16 -> 14 -> 16 -> 7 -> 13 -> 10 -> 2 -> EOL
Average of list: 10.7

List 1:
3 -> 8 -> 11 -> 20 -> EOL
List 2:
4 -> 7 -> 1 -> EOL
After concatenating:
3 -> 8 -> 11 -> 20 -> 4 -> 7 -> 1 -> EOL

List before reverse:
7 -> 13 -> 17 -> 12 -> 9 -> 8 -> 10 -> 3 -> EOL
List after reverse:
3 -> 10 -> 8 -> 9 -> 12 -> 17 -> 13 -> 7 -> EOL
```

NOTE: Now that our programs have more functions than just the `main()` function, the use of the function headers becomes meaningful and required. Make sure that all of your functions have function headers preceding them that document the purpose of the functions, and the input parameters and return value of the function.

Assignment Submission

An eCollege dropbox has been created for this assignment. You should upload your version of the assignment to the eCollege dropbox named `Assg 13 Process List of Integers` created for this submission. Work submitted by the due date will be considered for evaluation.

Requirements and Grading Rubrics

Program Execution, Output and Functional Requirements

1. Your program must compile, run and produce some sort of output to be graded. 0 if not satisfied.
2. 10+ pts. For correct implementation of the `displayList()` function.
3. 20+ pts. For correct implementation of the `averageList()` function.
4. 30+ pts. For correctly implementing the `concatenateLists()` function.

5. 30+ pts. For correctly implementing the `reverseList()` function.

6. 10+ pts. For demonstrating your functions adequately in your `main()` function.

Program Style

Your programs must conform to the style and formatting guidelines given for this course. The following is a list of the guidelines that are required for the assignment to be submitted this week.

1. The file header for the file with your name and program information and the function header for your main function must be present, and filled out correctly.

2. A function header must be present for all functions you define. You must document the purpose, input parameters and return values of all functions. Your function headers must be formatted exactly as shown in the style guidelines for the class.

3. You must indent your code correctly and have no embedded tabs in your source code. (Don't forget about the Visual Studio Format Selection command).

4. You must not have any statements that are hacks in order to keep your terminal from closing when your program exits (e.g. no calls to system()).

5. You must have a single space before and after each binary operator.

6. You must have a single blank line after the end of your declaration of variables at the top of a function, before the first code statement.

7. You must have a single blank space after , and ; operators used as a separator in lists of variables, parameters or other control structures.

8. You must have opening `{` and closing `}` for control statement blocks on their own line, indented correctly for the level of the control statement block.

9. All control statement blocks (if, for, while, etc.) must have `{ }` enclosing them, even when they are not strictly necessary (when there is only 1 statement in the block).

10. You should attempt to use meaningful variable and function names in your program, for program clarity. Of course, when required, you must name functions, parameters and variables as specified in the assignments. Variable and function names must conform to correct `camelCaseNameingConvention` .

11. Put the `*` for pointer variable declarations next to the type declaration, with no space between the type and the `*`. Also please follow the convention of using `Ptr` at the end of names for pointer variables.

Failure to conform to any of these formatting and programming practice guidelines for this assignment will result in at least 1/3 of the points (33) for the assignment being removed for each guideline that is not followed (up to 3 before getting a 0 for the assignment). Failure to follow other class/textbook programming guidelines may result in a loss of points, especially for those programming practices given in our Deitel textbook that have been in our required reading so far.