CS 455 Extra Credit Assignment

Fall 2021 [Bono]
Due: Tuesday, Dec. 7, 11:59pm
No work accepted after this date.

Introduction and Background

This is an extra credit assignment worth 3%. (So, someone with a perfect score in the course who also got full marks on this could have 103 weighted points at the end, out of 100 points of required work.) It is open to anyone in the class, but is completely optional.

The assignment is to implement and test some functions that operate on C++ linked lists. It's like a lab, although a bit bigger than a lab; but unlike a lab you are required to do the work independently (i.e., no partnerships).

Like PA5, this is a multi-file program with a Makefile that uses separate compilation. We did that primarily to give us the freedom to easily test your code using a different test driver than the one you're writing here. Hopefully between the Makefile handout (readings for Week 15), the recent lectures, and the last assignment, you are feeling comfortable with multi-file C++ programs.

The assignment files

The files in **`bold`** below are ones you modify and submit. The ones not in bold are ones that you will use, but not modify.

- `ecListFuncs.h` This contains the Node struct definition for our linked lists (but not the method implementations) and commented prototypes for the functions you are required to write.
- **`ecListFuncs.cpp`** Implementation file for the the list functions required for this assignment. Stub versions of the functions are already provided so you can compile the program right from the start.
- **`ectest.cpp`** A test program for your list functions.
- `Makefile` A file with rules for the "make" command. There are comments at the top of the file telling you how to use it.
- **`README`** See section on Submitting your program for what to put in it. Before you start the assignment please read the following statement which you will be "signing" in the README:

> "I certify that the work submitted for this assignment does not violate USC's student conduct code. In particular, the work is my own, not a collaboration, and does not involve code created by other people, with the exception of the resources explicitly mentioned in the CS 455 Course Syllabus. And I did not share my solution or parts of it with other students in the course."

The assignment

Complete the implementation of the four functions `listToString`, `buildList`, `insertAt` and `partitionBy`, described in more detail in `ecListFuncs.h`. None of the functions depend on each other, and we will test each of them separately, so you can still get some extra credit by just implementing some of the functions. However, you will not get credit for a function unless you have also included your own tests of the function in `ectest.cpp` (read on for more details about that).

The complete specification for these functions (i.e., prototypes + function comments) are given in `ecListFuncs.h`. You put your implementations of these functions in `ecListFuncs.cpp`.

Complete the implementation of the test program `ectest.cpp`. It should be a non-interactive program that thoroughly tests the four functions you are going to write. It should compare the actual results to the expected results for each case you test, and print out the message "FAILED" for any test cases where the expected result does not match the actual result. You should also print out something about what each test case is (input and results), but it's up to you what that will look like. More about this test program in the next section.

We provided a Makefile; to compile the code use the command:

```
make ectest
```

Suggested implementation and test plan

Since you are implementing functions to convert a string to a linked list, and vice versa, once you test listToString, it should be easy to use those functions to quickly whip up test cases and code to compare expected results to actual results (you can compare two strings in C++ with ==). Here is a suggested plan for implementation:

1. Implement `listToString` and test it on a few hard-coded linked lists you create. These tests could be in a function, `testListToString` that you call in `ectest.cpp`. Once you are convinced listToString works...
2. Implement `buildList` and test it on some hard-coded strings, using `listToString` to see whether the correct list was built: you can compare that resulting string to a hard-coded expected results string. Note: since the format for the input to buildList is different from the format for the return value of listToString, you can't directly compare those two strings.
3. Now implement and test `insertAt`, adding test cases using buildList and listToString in the same way as before.
4. Finally, implement and test `partitionBy`, similarly.

C++ library Hints

- The string class has operators defined on it, such as +, +=, ==, and != that may come in handy.
- "+" on a `string` and an `int` in C++ will not do the automatic conversion, like it does in Java: the easiest way to convert an int to a string is using the function `std::to_string(int)` (include file `<string>`). Its part of C++11: the makefile calls the compiler with the option set to use C++11 features.
- the `std::istringstream` class may be useful for `buildList` (include file `<sstream>`). It is very similar to using a Scanner on a String in Java: you can use the input interface you are familiar with, but where the source is a string, instead of cin.

Grading criteria

Unlike other programming assignments this one will be graded primarily on correctness. You will also be evaluated on whether you have a test program that meets the spec given, and tests all of the list functions you implemented.

The other evaluation criteria is not wasting resources. For example, you will lose credit if you implement what should be an O(n) function in $O(n^2)$, or if you have memory leaks, e.g., by doing unnecessary calls to `new`, or not reclaiming memory no longer needed. There is an additional restriction on how you must complete `partitionBy` -- see function comments for details.

We will not be evaluating your code on style, except that if your code is so messy/unreadable that we can't evaluate the aspects mentioned in the previous paragraph you will just lose credit for that part. (I would hope things like using good names and consistent indenting are automatic for you by now.) You are not required to write helper functions, but you are welcome to do so to increase code readability and reuse. (In particular, your test program will probably be a mess without a functional decomposition.)

For the four functions in ecListFuncs.cpp, you will receive no credit for the function if it prints anything.

You will also receive no credit for any of the four functions implemented but with no accompanying code to test that function in ectest.cpp.

README file / Submitting your program

Your `README` file must document which of the functions you successfully completed and known bugs in your program (this might be in the form of test cases you tried, but that didn't work), and any special instructions or other information for the grader. It must also contain the signed certification shown near the top of this document.

The submit script will check that the necessary files are present, will compile them, and will make sure there are no `cout` statements in your `ecListFuncs.cpp` code.
