Tutorial 6
Linked Lists

Learning Objectives

After this tutorial, you will be able to:

- Create and manipulate singly linked lists

Tutorial

In order to receive full marks, you must complete everything in the "Tutorial" section. Part marks may be provided if good progress was made toward completing the tutorial, at TA discretion.

1. Download the file `T06.tar.bz2` from the tutorial module on Brightspace. Extract and read through the tutorial files.

   This week's tutorial documents are compressed using another common compression tool: `bzip2`. Read the man page for `bzip2` (and `bunzip2`) and extract the files needed for this tutorial.

   *Remember when compiling that there are more than one source file. You may also want to include the -g flag if you plan on using valgrind to debug (**highly** recommended). Be sure to also use the -Wall flag to alert you to any additional warnings in your code.*

2. Create a `StudentNode` type for use in a linked list which is compatible with what is already written.

3. Draw a memory map diagram (similar to those seen in class) which represents a call and return from the `addStudent()` function. Your diagram should include the function call stack, heap, and all relevant pointers. You may use paper, paint, spreadsheets, a text file, or anything else that lets you visualize the memory.

4. Implement the functions `addStudent()` and `printList()` as prototyped.

   When adding a student, it should be added to the front of the list.

5. Write a function `appendStudent(StudentList* stuList, StudentNode* a)` which adds a student to the end of a list, always in the last position. Remember to consider all possible cases (empty list, inserting at the beginning, at the end, etc.)

Exercises

These exercises are optional, but are provided to give you some additional practice working with common operations that you will encounter while using pointers that might be tricky when you first work with them.

1. Write a function `popStudent(StudentList* stuList)` which deletes the student in the first position from the list.
2. Write a function `deleteStudent(StudentList* stuList, int pos)` which deletes the student in the given position from the list.
3. Alter the `StudentList` and `StudentNode` types so that the list becomes doubly linked. Make any necessary changes to the list functions so that they work correctly.

Saving and Submission

In-Person Attendance: Submission is optional, as you can be checked off in class, but you must still complete the required portion of the tutorial for marks.
Asynchronous: If you are completing the work on your own at home, make sure to follow the submission requirements.

1. Create an archive to submit your files, and include all of the files that you worked with in the tutorial.
    a. They may be submitted as a .tar, .tar.gz, bz2, or .tgz file
    b. Make sure to include all of the code needed to run your program
    c. Make sure to include your memory map, which should visualize what was specified

2. For **full marks** you should have completed all problems in the Tutorial section (and they should be seen upon running the program) without any memory leaks on exit. The exercises are optional, but highly recommended for the extra challenge and exploration of these concepts.

3. For **part-marks** you should have attempted to complete most of the tutorial. Grades for part-marks will be at TA discretion.