ELEC 278: Fundamentals of Information Structures
Lab 3: Advanced Lists and Stacks

Fall 2023-Instructors: Ni & Mertin

October 1, 2023

1 Objectives

The objectives of this lab are to further develop your understanding of data structures and their implementation in C by implementing parts of data structures and the algorithms to manipulate them.

2 Instructions

Download the file `lab3.zip` from OnQ and unzip it. Open the `lab3` folder that you extracted in either CLion or VS Code (note: you need to make sure you open the correct folder, i.e., the one that directly contains the code files). Then, complete the following tasks.

2.1 Task 1: Linked Stack Implementation

The file `linked-stack.c` contains starter code for an implementation of linked-list-based stacks containing characters. The function declarations and documentation are given `stack.h`. Fill in each "todo" in the file:

1. Complete the data structure definition by specifying the fields of `struct stack_node`.
2. Finish the implementation of `stack_free` to make sure that, when the stack is no longer needed, it is properly destructed and all allocations are freed.
3. Implement `stack_push` to enable entries to be added to the stack.
4. Implement `stack_pop` to enable entries to be removed from the stack.

2.2 Task 2: Array Stack Implementation

The file `array-stack.c` contains starter code for an implementation of array-based stacks containing characters. The function declarations and documentation are given `stack.h`; this implements the same interface as the linked-list-based stacks! Fill in each "todo" in the file:

1. Implement `stack_push` to enable entries to be added to the stack.
2. Implement `stack_pop` to enable entries to be removed from the stack.

You must use the definition of `struct stack` exactly as it is provided in the starter code when implementing these functions; it does not require any modification.

2.3 Task 3: Checking Bracket Balancing

The file `demo.c` contains starter code for a function `check_brackets`, which takes a string and checks that the brackets in the string are balanced. It should look for normal parentheses `()`, square brackets `[]`, and curly braces {}. It should return `false` if there are any mismatched or missing closing brackets, and `true` otherwise.

3 Running the Code

There are two executables you can choose from when running the project. Both will prompt you to enter a line of text and report the result of your bracket-checking algorithm. However, `linked-demo` will do so using the linked stack implementation from task 1, while `array-demo` will do so using the array stack implementation from task 2. You should make sure that your algorithm works with either stack implementation; they should be functionally equivalent!

You can also run the "All CTest" option in CLion, or press the "Run CTest" button in the bottom bar of VS Code to automatically run the algorithm against each of a set of test inputs (in the `test-cases` folder), using each implementation of stacks. This will show, for each combination, whether it produced the correct answer. This does not exhaustively check that your code is perfectly correct, but it is a good starting point. Note that, initially, the algorithm fails for cases 2 and 3, since it never returns `false`.

4 Marking Criteria

After completing all tasks, call over a graduate TA to mark the lab. Lab 3 has 9 marks in total:

- Is your completion of the linked-list-based stack structure and corresponding `stack_free` function correct? Does it correctly represent a stack and correctly free all memory? (1 mark)
- Does your implementation of `stack_push` for linked-list-based stacks correctly add the stack entry? (1 mark)
- Does your implementation of `stack_pop` for linked-list-based stacks correctly remove the stack entry and provide the removed value? (1 mark)
- Does your implementation of `stack_push` for array-based stacks correctly add the stack entry? (1 mark)
- Does your implementation of `stack_pop` for array-based stacks correctly remove the stack entry and provide the removed value? (1 mark)
- Does your code correctly make use of stacks to check for balanced brackets? (2 marks)
- Is your code sufficiently well-formatted and commented so that the purpose of each variable/field/parameter and the reason for each function call or data structure manipulation is clear? Refer to the guidance on the course assignment for expected levels of commenting. (2 marks)