Assignment 7 — Functions and Recursion

Due: Monday, March 5th

C supports recursive functions, and since all high-level code must eventually be compiled into low-level instructions, such functions must be translated into assembly languages. For this assignment, you'll write implement recursive functions in both C and LC-3 assembly.

Deliverables:

Demo — Be prepared to show your recursive LC-3 assembly implementation to your instructor.

Write-up — Answer the questions throughout this assignment. Fill out your answers *at the end* of this assignment, where the questions have been copied with space for answers.

Handin — `countFuncs.c`, `charCount.asm`
GitHub Classroom Link: `https://classroom.github.com/a/R4Pows1W`

For this assignment, you may work alone or with *one* partner. However, the handin portion must be completed individually. Remember to write both of your names on the write-up.

Part 1: Recursive Functions

Begin by accepting the GitHub Classroom assignment: `https://classroom.github.com/a/R4Pows1W`.

Complete the `countBackwardsFrom` and `countForwardsTo` functions in `countFuncs.c`, implementing them recursively:

```
void countBackwardsFrom(int n);
```

- Prints out positive integers, counting backwards from n to 1.
- Takes one argument:
  - `n` — A positive integer at which to start

```
void countForwardsTo(int n);
```

- Prints out positive integers, counting forwards from 1 to n.
- Takes one argument:
  - `n` — A positive integer at which to stop

Requirements:

- Do not alter the existing prototypes or depend on code in other files.
- The functions must be implemented recursively.
- You may not use any global or static variables.
- You may not use any helper functions.
- Your code must compile on Cal Poly's Unix servers using "`gcc -Wall -Werror -ansi -pedantic`".

In order to test your functions, a driver has been provided. Compile and run `countFuncs.c` together with `countDriver.c`.

Sample Run:

```
Enter an integer: 5
5, 4, 3, 2, 1
1, 2, 3, 4, 5
```

(Where italicized characters are typed by the user. Your program will be tested using `diff`, so your output must match *exactly.*)

Part 2: Recursive Character Counting

Read through `charCount.c`. This program contains two functions: `main`, which prompts the user to enter a string and a character, and `charCount`, which recursively finds the number of occurrences of that character within that string.

Sample Run:

```
Enter an string: foobar
Enter a character for which to search: o
'o' occurs 2 times!
```

Now, read through `main.asm`. This is the LC-3 implementation of the `main` function from `charCount.c`.

Recall that, in order to support recursive function calls, a program must save data in stack frames on the runtime stack; it cannot simply call subroutines. Every time a function is called, both the caller and callee functions must perform specific setup and teardown steps.

Since the `main` function calls the `charCount` function, `main.asm` implements caller setup and teardown.

1. Draw the runtime stack as it appears after `charCount`'s callee setup is complete. Include all known addresses and values. Indicate any changes to the stack and frame pointers.

   [*Note:* To ease implementation and debugging, I suggest you confirm the correctness of your drawing with your instructor before continuing.]

Part 3: The LC-3 Implementation

Complete the LC-3 implementation by translating the `charCount` function into an LC-3 assembly subroutine.

Requirements:

- Define your subroutine in a file named "`charCount.asm`".
- Your subroutine must begin at memory location `0x3300`.
- Do not modify `main.asm`. Your subroutine should be compatible with `main.asm` as-is.
- You must store all data on the runtime stack, using the setup and teardown procedures presented in lecture. You should not attempt to preserve data in registers across function calls.
- Your implementation must be recursive. Note that this means that the subroutine will be both a callee (since it's called by `main`) *and* a caller (since it calls itself).

Once `charCount.asm` is complete, you should be able to assemble and run it together with `main.asm`. The output should match the sample run of `charCount.c` shown above.

- Your subroutine, together with `main.asm`, must be able to be run multiple times by resetting the PC to `0x3000`. It should not require that the LC-3 be reinitialized, that any files be reloaded, or that any registers be reset.

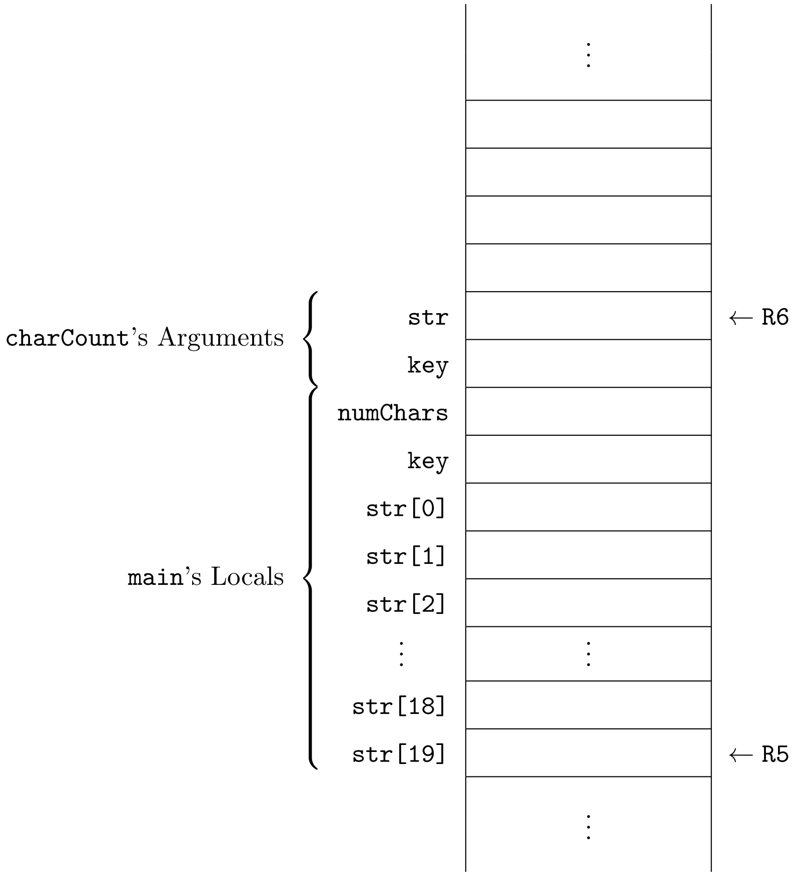
Part 2: Recursive Character Counting

1. Draw the runtime stack as it appears after `charCount`'s callee setup is complete. Include all known addresses and values. Indicate any changes to the stack and frame pointers.

   Shown below is an outline of the runtime stack after `main`'s caller setup finishes. Supposing:

   - The user enters "Hi" as the string in which to search.
   - The user enters 'i' as the character for which to search.
   - The stack pointer is initialized at `0x3000`.

   Complete this drawing by adding any known values to the existing stack, as well as the results of `charCount`'s callee setup.



   [*Note:* To ease implementation and debugging, I suggest you confirm the correctness of your drawing with your instructor before continuing.]