Programming Lab 3: Resource Map, Pt. 1

EE 306: Introduction to Computing
Professor: Dr. Nina Telang
TAs: Vignesh Radhakrishnan, Jefferson Lint, Suhas Raja, Jerry Yang

Due: November 9, 2019

1 Overview

This lab is the first part of a two-part lab in which you will build a map of campus resources for UT students. In this lab, you will design the "back end" of the resource map by implementing a command-line interface (CLI) to a linked list of resources and their locations. By the end of this lab, you should be able to:

- access and modify linked lists in LC3 assembly.
- utilize subroutines in a program.
- perform basic input/output functionality with the LC3 console using TRAP routines.

Note: You may NOT talk to or show anyone other than the TAs or the professor your code. **Any violation of this rule constitutes academic dishonesty.**

2 Background

Command-line interfaces are the simplest way to interact with a computer. All computers have a command-line interface where users can enter commands and the computer can execute them. On Windows systems, you can search for the program "cmd" in your start menu and use it to create or delete folders, files, programs, etc (see Figure 1). You will learn all about CLIs and how to use them later in your EE career; for now, we will focus on implementing a basic CLI in LC3.

A CLI relies on a series of commands, similar to instructions, and arguments needed for the command, similar to inputs of an instruction. Every command of a CLI has the same structure: the command, followed by its (varying number of) arguments needed to execute the command. As seen in Figure 1, Windows has several commands such as `cd` (change current folder), `dir` (show contents of current folder), `mkdir` (make a folder). Each Windows command displayed here has one argument (except `dir`): a folder name. These arguments are the inputs to the command, and whatever actions the command does is the output. For example, the `mkdir test` command makes a folder with the name `test`. The command line therefore allows us to directly manipulate data and their locations in the computer at the lowest level possible.

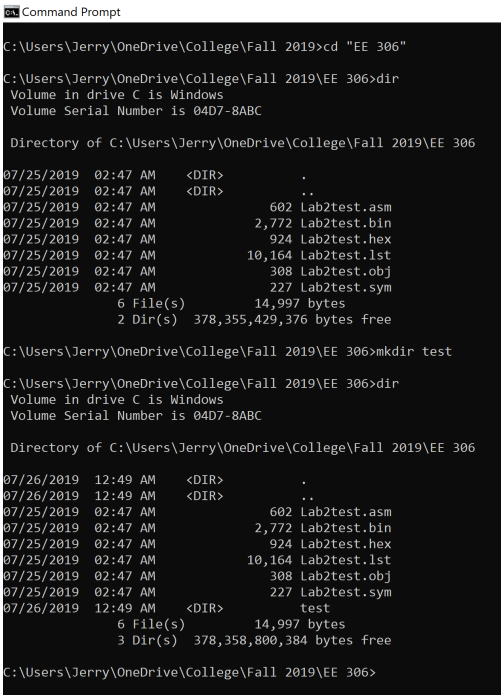


Figure 1: Windows command line interface (CLI)

3 Lab Specifications

In this lab, you will create a command-line interface to manage a database of UT student resources. The database is stored as an alphabetically sorted linked list in LC3 with the node structure in Table 1. The database will be comprised of resources from the provided list in `Lab3data.asm`. The location of the first node (a.k.a. head) is stored in x6000. The names of the resources are null-terminated strings, and their locations on a campus map are coordinates given by the superimposed grid (see Figure 2). The map location coordinates are stored with the top byte as the row letter in ASCII and the bottom byte as the column number. For example, the location of EER on the map is C5, which would be stored as x4305 in the linked list.

Pointer to resource name
Map location
Pointer to next node

Table 1: Node structure.

Your command-line interface should handle the following commands (see next section for an example run and formatting output):

1. `find [resource]` - Finds the resource in the linked list and prints its location on the console on the line below the entered command. If the resource is not in the linked list, print "Error: This resource doesn't exist!" to the console on the line under the command.

2. `find [location]` - Finds and prints all resources at the specified location, with one resource per line. If there are no resources at the specified location, do not print anything.

3. `add [resource] [location]` - Adds the resource and location into the linked list. The linked list must remain alphabetically sorted. When creating new nodes, only use memory

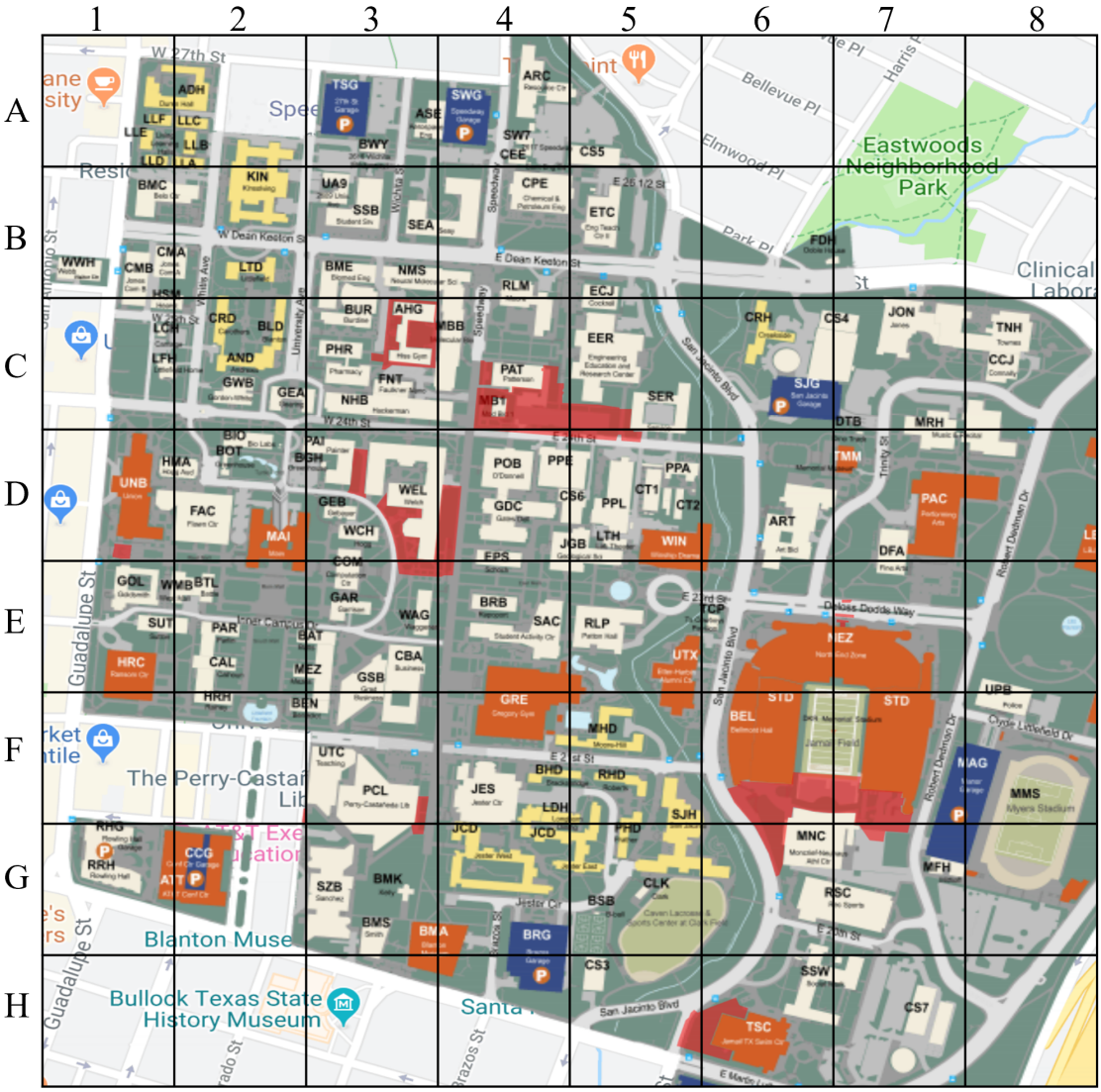


Figure 2: Campus map

between x8000-xE000. If a resource already exists in the database, print "Error: This resource already exists!" to the console on the line under the command.

4. `del [resource]` - Deletes the resource and its location from the linked list. If the resource is not in the linked list, print "Error: This resource doesn't exist!" to the console on the line under the command.

5. `list` - Lists all the resources and their respective locations on the console in alphabetical order, with each resource/location pair on its own line.

6. `exit` - Halts the program.

For all the commands above, `[resource]` is the case-sensitive name of a resource as a string enclosed in quotation marks, and `[location]` is the location of the resource on the 8x8 grid as a two-character word in the format XY with X a capital letter and Y a number (e.g. A2). Since arguments are case-sensitive, "IEEE" is not the same as "ieee"; uppercase letters come alphabetically

before lowercase letters. Commands are a maximum length of 256 characters and are all lowercase. You may assume that all arguments are entered correctly, without spelling errors/etc.

When the command-line interface starts, it should display the strings "UT Resource Map Database" and "Enter a command: " on separate lines and wait for the user to type in a command followed by "Enter". As the user types, you should echo the typed characters to the console so that the user can see what is being typed. (Note that GETC takes in a character but does not display it.) Once the user hits "Enter", you should print a newline, and your program should execute the command. If the command does not exist or is entered/spelled incorrectly, print "Error: Invalid command!" to the console on the line under the command. When it is finished or the user just presses "Enter" without typing a command, it should redisplay the "Enter a command: " prompt on the next line of the console and wait for the user to type in another command.

4 Hints & Tips

1. One way to conceptualize your program is as a series of subroutines that are called from a main routine; each command has its own subroutine. Your main routine will handle the input/output with the console and direct your program to a subroutine based on the entered command.

2. Start with a flowchart for the entire program, and break each command into its own flowchart. This will help break down the large program into smaller pieces.

3. Test each subroutine individually before integrating it into the main program. Therefore, if you encounter a bug, it is easier to pinpoint where it may be occurring.

4. Your program may become too large to where you may not be able to access some of your labels from other places in your program. If this happens, you can use .FILLs to place the address of the desired label closer.

5. There are over a trillion possible testcases that can be formed from the provided resource list (2^{40}, to be precise). As in Lab 2, be sure to chose your testcases judiciously.

6. Extending the previous hint, use the assumptions that are stated in the lab manual to generate testcases. For example, you may assume that arguments are spelled correctly, but not commands, which means you should test misspelled commands as testcases.

5 Testcase & Example Run

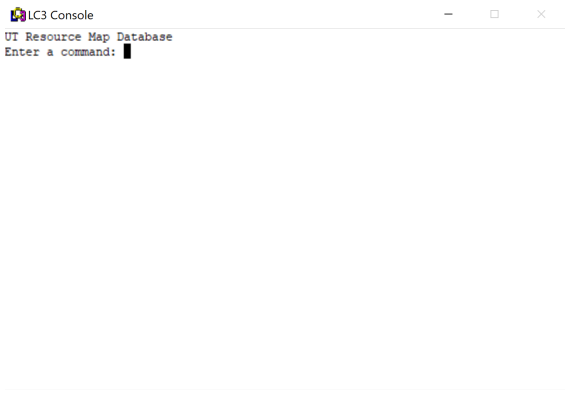
Figure 3 shows the code for the provided testcase. As in lab 2, assemble the file in the LC3 editor, and load the .obj file into LC3 before loading your program. An example run is displayed in Figure 4. Note that it does not show any edge cases that may cause your program to fail; be sure to test your program before submitting.

6 Submission

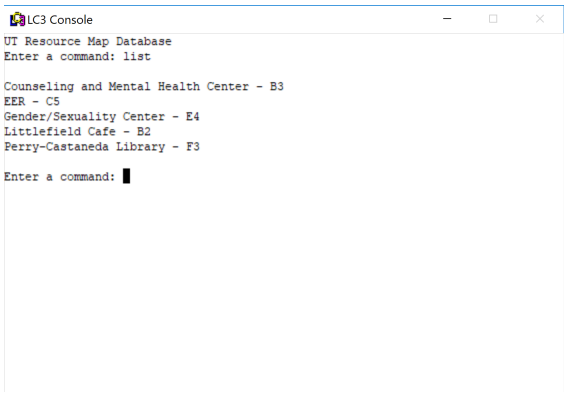
Create a Github repository using the provided link on the Canvas assignment, and push your code to the repository.

```
.ORIG x6000
.FILL HEAD
.BLKW x1000

HEAD .FILL RES3
     .FILL x4203
     .FILL N2

     .BLKW 3

N1   .FILL RES4
     .FILL x4504
     .FILL N3

     .BLKW 2

N2   .FILL RES1
     .FILL x4305
     .FILL N1

     .BLKW 5

N3   .FILL RES5
     .FILL x4202
     .FILL N4

     .BLKW 6

N4   .FILL RES2
     .FILL x4603
     .FILL 0

     .BLKW x100

RES1 .STRINGZ "EER"
RES2 .STRINGZ "Perry-Castaneda Library"
RES3 .STRINGZ "Counseling and Mental Health Center"
RES4 .STRINGZ "Gender/Sexuality Center"
RES5 .STRINGZ "Littlefield Cafe"

.END
```

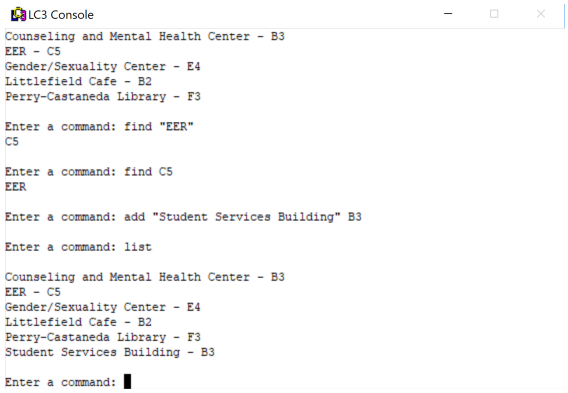
Figure 3: Testcase



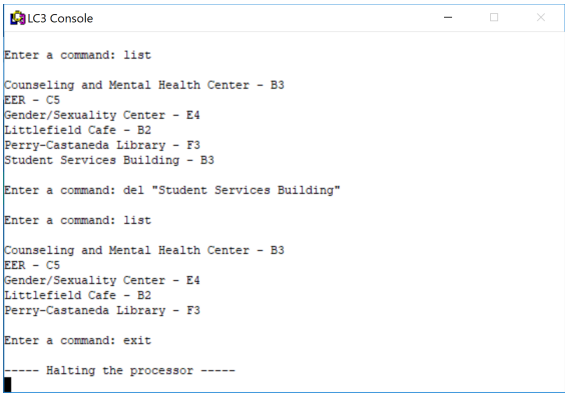
(a) Initial start



(b) `list` command



(c) `find` and `add` commands



(d) `del` and `exit` commands

Figure 4: Example run