CSE3442 (Spring 2020)
Lab #5

In this lab, you will complete the terminal interface. The following steps will guide you through this process:

1. Start with the lab 4 code, renaming the file lab5_*your_name*.c, where *your_name* is replaced with your name as it appears in MyMav.
2. Code the parseFields() function implementing this algorithm:
    a. Decide on 3 sets of characters – alpha, numeric, and delimiter. Alpha is a-z and A-Z, numeric is 0-9 and optionally hyphen and period (or comma in some localizations), and everything else is a delimiter.
    b. Assume that the previous character type is a delimiter when starting to search the buffer.
    c. Go through the buffer from left to right, looking for the start of a field (a transition from a delimiter to a alpha or numeric character). For each field (at the transition), record the type of field (alpha or numeric – you can use 'a' or 'n' if you wish) in the type array, and the offset of the field within the buffer of the field in the position array, and increment the field count. Make the previous character stored equal to the new character and keep moving through the buffer string until the end is found. If the field count equals MAX_FIELDS, return from the function.
    d. Before returning, convert all delimiters in the string to NULL characters to aid in getter functions to follow.
3. Code the char* getFieldString(USER_DATA* data, uint8_t fieldNumber) function to return the value of a field requested if the field number is in range or NULL otherwise.
4. Code the int32_t getFieldInteger(USER_DATA* data, uint8_t fieldNumber) function to return a pointer to the field requested if the field number is in range and the field type is numeric or 0 otherwise.
5. Code the bool isCommand(USER_DATA* data, const char strCommand[], uint8_t minArguments) function which returns true if the command matches the first field and the number of arguments (excluding the command field) is greater than or equal to the requested number of minimum arguments.
6. Test the functions with code similar to the pseudo-code below:

```
// Get the string from the user
getsUart0(&info);
// Echo back to the user of the TTY interface for testing
#ifdef DEBUG
putsUart0(info.buffer);
putcUart0("\n")
#endif
// Parse fields
parseFields(&info);
// Echo back the parsed field information (type and fields)
#ifdef DEBUG
uint8_t i;
```

```
for (i = 0; i < info.fieldCount; i++)
{
        putcUart0(info.fieldType[i]);
        putcUart0("\t");
        putsUart0(&info.buffer[fieldPosition[i]]);
        putcUart0("\n");
}
#endif
// Command evaluationbool valid = false;
// set add, data → add and data are integers
if (isCommand(&info, "set", 2))
{
        int32_t add = getFieldInteger(&info, 1);
        int32_t data = getFieldInteger(&info, 2);
        valid = true;
        // do something with this information
}
// alert ON|OFF → alert ON or alert OFF are the expected commands
if (isCommand(&info, "alert", 1))
{
        char* str = getFieldString(&info, 1);
        valid = true;
        // process the string with your custom strcmp instruction, then do something
}
// Process other commands here
// Look for error
if (!valid)
        putsUart0("Invalid command\n");
```

7. If the user types "set 1, 2" followed by a carriage return, then the code should process the string as a set command with add=1 and data=2. If the code above was run with DEBUG defined and the user inputs the string "sen<back_space>t 1, 2", the output would be:
   set 1, 2
   a
   set
   n
   1
   n
   2
8. Demonstrate your code and e-mail the file to the grader.