ME 477 Embedded Computing

Laboratory Experiment #1
Introduction to myRIO C Programming
& the keypad/LCD device driver

Objectives

In this exercise you will gain experience with:

1. C programming for myRIO.
2. The beginning of a device driver for the keypad/lcd.
3. On-line debugging techniques.

Introduction

In addition to the `main()` program, you will write two related C functions, and carry out typical procedures with them. At this point, you are expected to have only an elementary knowledge of C, but you should become familiar with the procedures, such as debugging, that you will need in future.

Pre-Laboratory Preparation

Part #1 User Input: double_in()—Very often in an interaction between a computer and a user, a message or "prompt" is written on the LCD display and the user is expected to respond by entering an appropriate decimal number through the keypad. In this laboratory exercise you will write a C function, called `double_in()`, to perform the complete keypad/LCD procedure.

This function will be used here, and in later exercises, to obtain numerical information through interaction with the terminal. The function will execute the following steps each time it is called:

1. A user prompt (a string of ASCII characters) is written on Line-1 of the LCD display. A pointer to the string corresponding to this prompt is the only parameter of the `double_in()` function.
2. A floating point number is accepted from the keypad in response to the prompt. If an error occurs in the input string, the display is cleared, an error message is written on Line-2 of the display, and the prompt is issued again on the first line.

   The number is entered as a string of ASCII characters that may include the decimal digits 0 - 9, a decimal point, and a minus sign, and is terminated by ENTR.
3. The entered string is interpreted as a floating point number.
4. The floating point number (C data type "double") is returned from `double_in()` function to the calling program.

The prototype of the `double_in()` function is

```
double  double_in(char *prompt);
```

For example, a call to `double_in()` might be:

```
vel = double_in("Enter Velocity:  ");
```

The variable `vel` would be assigned the value entered.

The LCD interaction would look like:

```
Enter Velocity:  -50.75
```

Or, if an error occurs: (e.g. user enters: -50..75)

```
Enter Velocity: _
Bad Key.  Try Again.
```

Allow for four possible user errors:

Error Type	Error Message Displayed on Line-2
No digits are entered (e.g. ENTR only)	Short. Try Again.
↑ or ↓	Bad Key. Try Again.
"–" other than first character (e.g."––")	Bad Key. Try Again.
".." double decimal point	Bad Key. Try Again.

Our goal here is that the user must enter a valid number before the `double_in()` function can exit. Notice that the errors are detected in the string that the user enters.

Here is a possible strategy for `double_in()`:
Begin by using the `printf_lcd()` function to display the prompt on the LCD screen. Then,

1. Use `fgets_keypad()` "Get String" to obtain the string from the keypad. Its prototype is:

   ```
   char * fgets_keypad(char *buf, int buflen);
   ```

   Note: If no digits are entered, `fgets_keypads()` returns a NULL, *not* a string of zero length.
2. Use the `strpbrk()` "String Pointer Break" to detect ↑ or ↓. Note: ↑ is returned by `fgets_keypad()` as the ASCII character "[", and ↓ as "]".
3. Use the `strpbrk()` to detect minus signs ("-") *beyond* the first character.
4. Use the `strstr()` to detect double "." (i.e. "..").
5. Use `sscanf()` "Scan Formatted from String" to perform the ASCII-string-to-double conversion. Hint: Because `sscanf()` is converting to a variable of type `double`, you need to use the format `%lf` (long float).

Note: `printf_lcd()` and `fgets_keypad()` work like the standard C functions `printf()` and `fgets()`, and are linked to your program from `me477LIbrary`.

Write a main program that tests your `double_in()` function by calling it twice from the `main()` program, assigning each result to a different (double) variable. Then, as check, print the values of both variables on the console using `printf()`.

Part #2 Display on LCD: printf_lcd()—Our second task is to write the `printf_lcd()` function used by `double_in()`. The C function `printf()` prints to the standard output device, in our case the Console pane of the Eclipse IDE. We want `printf_lcd()` to operate exactly as `printf()`, except that it will print to the LCD screen. Refer to your C text. To do this, we want `printf_lcd()` to accept a format string with variable number of arguements. Therefore, the prototype for `printf_lcd()` is

```
  int printf_lcd(const char *format, ...);
```

where `format` is a string specifying how to interpret the data, and the ellipsis (`...`) represents the variable list of arguments specifying data to print. The return value is an `int` equal to the number of characters written if successful or a negative value if an error occurred.

For example,

```
    n = printf_lcd("\fa = %f, b = %f", a, b);
```

Here is a possible strategy for `printf_lcd()`:

a) Use the C function `vsnprintf()` to write the data to a C string. Then,

b) Use the LCD driver function `putchar_lcd()` to successively write each character in the string to the LCD display. Note: It is strongly suggested that you use an incremented pointer to access the string, rather than an array index.

The C function `vsnprintf()` writes formatted data from variable argument list to a buffer (the string) of a specified size.

The tricky part is passing the variable argument list of `printf_lcd()` to `vsnprintf()`. Here is an example fragment of code. From your C text, study the data type `va_list`, and the C macros `va_start()` and `va_end()` to see how this works.

```
int printf_lcd(char *format, ...) {
   va_list args;

   va_start(args, format);
      n = vsnprintf(string, 80, format, args);
   va_end(args);
```

As usual, you must allocate storage for the C `string` of length 80.

The `main()` program, the `double_in()` function, and the `printf_lcd()` function should all be in the same file: `main.c`. Be sure to `#include` the header files `me477.h`, `<stdio.h>`, `<stdarg.h>`, and `<string.h>` in the code.

Once you have defined `printf_lcd()` within your `main.c`, your code will supersede the version in `me477LIbrary`.

Some Background

The C function `putchar_lcd()` places the single character corresponding to its argument on the LCD screen. Its prototype is

```
    int  putchar_lcd(int c);
```

where both the input parameter and the returned value are the character to be sent to the display. A character constant is an integer, written as one character within single quotes, such as `'x'`.

For example, calls to `putchar_lcd(()` might be:

```
 ch = putchar_lcd('m');   or   putchar_lcd('\n');
```

To write both parts of your program you also need to know how the escape sequences used in the `putchar_lcd()` function affect the LCD screen. This concerns the important matter of I/O (input/output), which we will consider in detail later. For now the following table explains the escape sequences:

Escape Sequence	Function
\f	Clear Display
\b	Move cursor left one space
\v	Move cursor to the start of line-1
\n	Move cursor to the start of the next line

Laboratory Procedure

C Program—Debug and test your C program. As necessary, use breakpoints and single-stepping to find errors.

For this lab, your report should include the parts required for all reports.