CMPSC 311 Assignment 2: Getting Started with C and Unix

Out: January 23, 2014 Due: February 4, 2014

Purpose

This assignment is designed to familiarize you with C programming in a Unix command-line environment. You will use C to perform some programming tasks that you would likely find straightforward in a more familiar high-level language. This will help you adjust to C idiom, working with arrays, passing by value and reference, and using a Unix command-line development toolset (Vim, GCC, and Make).

Description

In this assignment you will develop a small library of assorted functions to manipulate data and arrays, along with a program which uses this library. The program will read in a series of numbers from standard input and use the library functions to modify them in various ways, printing out the results at each step. You will also construct a basic Makefile for building the library and program.

Procedure

1. Log into your virtual machine and open a terminal emulator.
2. Create a directory in which you will keep all of your assignments for this semester, and change into that directory:

   ```
   $ mkdir cmpsc311
   $ cd cmpsc311
   ```

   (**Note:** command-line examples are commonly indicated by a `$` or `%` to represent the shell prompt. It's not part of the command, so don't type it in.)
3. From your virtual machine, download the source code tarball provided for this assignment. To do this, use the `wget` utility to download the file off the course website:

   ```
   http://www.cse.psu.edu/~djp284/cmpsc311-s15/docs/assign2.tar.gz
   ```
4. Extract the tarball using the `tar` utility.

   ```
   $ tar -xvzf assign2.tar.gz
   ```

   This will create a directory `assign2` containing these files:

   - `assign2.c`: C source file containing the main function of the program
   - `a2lib.h`: C header file containing declarations of the library functions
   - `a2lib.c`: C source file which will contain the implementation of the library functions

5. As you complete the remaining steps, use good, consistent coding style and write helpful comments that explain what the code is doing and why.

6. Complete the main function in the `assign2.c` file. A basic Makefile has been provided so that you can build the project by running `make` at the command line. (You will replace this with a more advanced Makefile at the end of this assignment.)

   The `main()` function of the program must do the following, calling the appropriate library function for each step:

   (a) *(This step has been done for you.)* Read a fixed number of integer values from the terminal and place them in an array.
   (b) Populate the double-precision array with the square roots of the integers by using your `root_array` function. If this function fails, print an error message and end the program by returning an error value from main.
   (c) Display the original integer array using your `print_int_table` function.
   (d) Seven-sort the integer array in place using your `seven_sort` function. Display the resulting array.
   (e) Replace each element of the integer array with the `weighted_bitcount` of its value. Display the resulting array.
   (f) Display the original double-precision array using your `print_dbl_table` function.
   (g) Calculate the up-down sum of the double-precision array using your `updown_sum` function and display the resulting array.
   (h) Round each element of the double-precision array to the nearest multiple of 1/4 using your `round_quarter` function. Display the resulting array.
   (i) Print the final double-precision array vertically using your `print_vertical` function.

7. Fill in the `a2lib.c` file with your implementations of the library functions, as specified in the next section. You will also need to modify the function declarations in `a2lib.h` to specify the appropriate return values and parameters.

8. Create a new Makefile for the source code in the `assign2` directory to replace the inefficient one provided with the starter code. This Makefile should first compile each out-of-date source file to an object file, then link those object files into an executable. Your Makefile should demonstrate the following: rules with correct prerequisites, defining and using variables, phony targets for `all` and `clean`, and at least one pattern rule with automatic variables (such as `$@`). The Makefile should include detailed comments explaining each rule, definition, variable, etc.

   (**Note:** for this assignment, the Makefile should *not* rely on implicit rules; i.e., the prerequisites and commands for each step should be explicitly specified.)

Library functions

These are the specifications for the functions you will be implementing in the `a2lib` library. You are to implement each function exactly as it is described here, including the parameter and return types. In addition, numeric quantities must be processed *as numbers* (it is generally inefficient to convert numbers to strings, do processing, and convert them back).

Important: Libraries and systems are meant to be used with many different programs. Your library code, therefore, must not make assumptions based on what your main function does, because it may also be used with other programs. This is important to understand when writing systems!

- `print_int_table`: Displays an integer array as a table with 5 columns. Each number should be right-aligned in a column that is 11 characters wide.
  - Parameters: a reference to an integer array, and the length of the array
  - Sample output:

```
      35962         81 1234567890       4444         10
    2221111      52571          1    1101017          0
      31415        942
```

- `print_dbl_table`: Displays a double-precision array as a table with 5 columns. The displayed number should be rounded to exactly 2 decimal places and right-aligned in a column that is 9 characters wide.
  - Parameters: a reference to a double-precision array, and the length of the array
  - Sample output:

```
     3.14 12345.78   156.00    42.42  1048.57
   113.09     0.01     0.00  1001.96
```

- `root_array`: Calculates the square root of each element in an integer array and places it in the corresponding element of a double-precision array of the same length. If any of the integers is negative, this function should return 1 immediately to indicate failure. Otherwise, it should return 0 to indicate success.
  - Parameters: a reference to an integer array, a reference to a double-precision array, and the (shared) length of the arrays
- `seven_sort`: Bubblesorts an array of integers so that all numbers ending in the digit 7 come first (in ascending order), followed by all numbers which do not end in 7 (again, in ascending order).

  Note: You may refer to the "Bubble sort" article on Wikipedia if you need a refresher on how bubble sort works, but you *may not* copy any code!
  - Parameters: a reference to an integer array, and the length of the array
  - Sample result: a seven-sort of the array $[7, 1, 23, 45, 67, 89, -87, -76]$ will change the order of its elements to $[-87, 7, 67, -76, 1, 23, 45, 89]$
- `weighted_bitcount`: Returns a weighted count of the bits in a given integer, such that bit 0 (the ones place) is worth 0, bit 1 (the twos place) is worth 1, bit 2 (the fours place) is worth 2, and so on. If the integer is negative, the result should be the weighted bit count of its absolute value.
  - Parameters: an integer
  - Sample result: given the integer $333 = 101001101_2$, bits 8, 6, 3, 2, and 0 are set (one), so this function should return $8 + 6 + 3 + 2 + 0 = 19$. Given the integer -333, it should give the same result.
- `updown_sum`: Returns the up-down sum of a double-precision array. This is calculated by taking the first element, minus the second, plus the third, minus the fourth, and so on, switching between addition and subtraction with each element.
  - Parameters: a reference to a double-precision array, and the length of the array

    - Sample result: given the array $[2.718, 3.14, 141.41, -1, 10.5]$, this function should return $2.718 - 3.14 + 141.41 - (-1.00) + 10.50 = 152.488$.

- `round_quarter`: Rounds the given double-precision number to the nearest multiple of 1/4. When the parameter is halfway between two multiples of 1/4, it should follow the convention of rounding up to the higher number. *This should modify the original value.*
    - Parameters: a reference to the double-precision number to modify
    - Sample results: $5.10 \to 5.00$, $5.15 \to 5.25$, $5.125 \to 5.25$, $-5.125 \to -5.00$.

- `print_vertical`: Displays a double-precision array such that the digits in each number are printed from top to bottom in a column instead of left to right. Negative signs should be in the top row, decimal points should be aligned on the same row, and the printed numbers should be *truncated* (not rounded) to show exactly two decimal places. Do not print any leading zeros before the decimal point.
    - Parameters: a reference to a double-precision array, and the length of the array
    - Sample output: given the array $[3.14, 141.41, -2.718, 0.01, -10.5]$, this function should output:

```
  - -
 1
 4  1
312 0
.....
14705
41110
```

- Don't forget to write a new Makefile!

Submission

1. Create a tarball containing the `assign2` directory, complete with the source code and build files you have completed. The tarball should be named `LASTNAME-uid123-assign2.tar.gz`, where LASTNAME is your last name in all capital letters and uid123 is your PSU email address without the "@psu.edu" part. For example, if the instructor were submitting this assignment, he would call the file `POHLY-djp284-assign2.tar.gz`.

2. Test your code in a clean temporary directory using the following commands:

```
$ mkdir /tmp/assign2test
$ cp LASTNAME-uid123-assign2.tar.gz /tmp/assign2test
$ cd /tmp/assign2test
$ tar -xvzf LASTNAME-uid123-assign2.tar.gz
$ cd assign2
$ make clean
$ make
$ # ... any commands needed to test the program
```

3. Email your tested tarball to `djpohly@cse.psu.edu` as well as the TA for your section (who is listed on the course website). This email should be sent by **11:59pm of the due date** of the assignment. Refer to the course website/syllabus for the policy on late submissions.

4. Any incorrect tarball (empty or containing something other than your completed code for this assignment) will be considered as not being submitted at all. We will try to notify you if we notice something is amiss, but it is up to you to confirm that your tarball is good by testing it as described above.

Important reminder

As with all assignments in this class, you are not permitted to:

- Copy any content from the Internet.
- Discuss or share ideas, code, configuration, text, or anything else related to the assignment with others in the class.
- Solicit help from anyone inside or outside the class other than the instructor, TAs, or assistants explicitly permitted by the instructor.

You may be asked at random by the instructor or TA to explain a portion of your code. If you cannot adequately explain how or why your own submission works, it will be considered highly suspect.

You are also responsible to protect your own ideas and code from falling into the hands of others. Consulting online sources is acceptable, but under no circumstances should *anything* be copied. Failure to abide by this requirement will result in dismissal from the class as described in the course syllabus.

Above all: when in doubt, be honest. If you do for any reason get significant help online, in person, etc., or use code from a source such as previous projects or other courses' lecture notes, please *document it in your submission.* This will go a long way!