CMPSC 311 Assignment 2: C Programming Basics

Out: January 27, 2014 Due: February 7, 2014

Purpose

This assignment is designed to familiarize you with working in a C programming environment. You will use C to perform some programming tasks that you would likely find straightforward in a language you already know. This will help you get used to C-specific idioms, working with arrays, passing by value and reference, and using the command-line development environment.

Description

In this assignment you will develop a program to manipulate data and arrays of various types. The program will read in a series of fifteen numbers from standard input, one per line, and modify them in various ways, printing out the results at each step.

Procedure

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

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

   (Note: in command-line examples, the prompt is often abbreviated as `$` or `%`. Don't actually type that character!)
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-s14/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`: Header file containing definitions of the various functions you will write.
5. Complete the code in the `assign2.c` file. Places where code needs to be added are indicated by `____`. See the comments in the files for detailed instructions and hints.

   The program shall perform the following functions as guided by the `main()` function:

(a) Read in fifteen double-precision values from the terminal and place them in an array. Note that the code to perform this step is already provided.

(b) Display the array using the `print_array_dbl` function.

(c) Compute the average (mean) of the inputs and display it.

(d) Subtract this average from each element of the array using the `reduce_value_by` function. Display the array again to show the changes.

(e) Compute a new array of fifteen integers from the double-precision inputs by using the `round75` function. Each index in the integer array should correspond to the same index in the double array; e.g., element zero of the integer array should contain the rounded value of the zeroth element of the double array.

(f) Display this new array using the `print_array_int` function.

(g) Now use the `reverse_int` function to reverse the digits of each element of the integer array. Display the array again to show the changes.

(h) Use the `parity_sort` function to move all the even numbers to the front of the array. Display the array again to show the result.

(i) Finally, print the numbers and their sum vertically using the `print_vertical_sum` function.

6. Create a new file called `a2lib.c`. This file will include the code for each of the functions defined in `a2lib.h`. You will also need to complete the function definitions in `a2lib.h`. These functions are defined in Table 1

   Note: You may refer to the "Bubble sort" page on Wikipedia for information on the bubble sort algorithm, but you *may not* copy any code!

7. Add comments to all of your files stating what the code is doing. Fill out a comment function header for each function you are implementing in the code. A sample header you can copy for this purpose is provided for the `main` function in the code.

8. Create a Makefile for the source code in the `assign2` directory. This Makefile should first compile the source to object files, then link the object files into an executable. (For this assignment, the Makefile should *not* rely on any implicit rules; i.e., you should explicitly define the dependencies and commands for each step.)

   Your Makefile must include at least: defining and using compile and link flag variables (such as `CFLAGS`), phony targets for `all` and `clean`, and one definition of a pattern rule with automatic variables (such as `$@`). The Makefile should include detailed comments explaining each rule, definition, variable, etc.

Submission

1. Create a tarball containing the `assign2` directory, complete with the source code and build files you have completed. Email the 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 syllabus for the policy on late submissions.

   The tarball should be named `LASTNAME-PSUEMAILID-assign2.tar.gz`, where LASTNAME is your last name in all capital letters and PSUEMAILID 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. Before sending the tarball, test it in a clean temporary directory using the following commands:

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

Bonus Points

Create a log utility for all output. This log utility should receive print data and print it out to `stderr` with a timestamp on each line. For example, a snippet of the sample output might read:

```
Mon Jan 27 10:21:10 EST 2014 Rounded integers:
Mon Jan 27 10:21:10 EST 2014 [-3, -38, -44, -47, -1, -44, -40, -40, -25, 178, -40, -821, -58, 634, 388]
Mon Jan 27 10:21:10 EST 2014 Reversed digits:
Mon Jan 27 10:21:10 EST 2014 [-3, -38, -44, -47, -1, -44, -40, -40, -25, 871, -40, -821, -58, 436, 883]
Mon Jan 27 10:21:10 EST 2014 After parity sort:
Mon Jan 27 10:21:10 EST 2014 [-38, -44, -44, -40, -40, -40, -58, 436, -3, -47, -1, -25, 871, -821, 883]
Mon Jan 27 10:21:10 EST 2014 Sum of values:
Mon Jan 27 10:21:10 EST 2014    -38
Mon Jan 27 10:21:10 EST 2014    -44
Mon Jan 27 10:21:10 EST 2014    -44
Mon Jan 27 10:21:10 EST 2014    -40
```

Reminder

As with all assignments in this class, you are prohibited from:

- Copying any content from the Internet.
- Discussing or sharing ideas, code, configuration, text, or anything else with others in the class.
- Seeking significant help from anyone inside or outside the class other than the instructor and TAs.

Likewise, you are responsible to protect your own 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 from a source, online, in person, etc., *document it in your submission.* This will go a long way!

Function	Parameters	Description
`print_array_int`	A reference to an array of integers, and the length of the array	Prints an array of integers on a single line. The numbers should be separated by commas, and the entire array should be surrounded by [square brackets]. Print a newline at the end.
`print_array_dbl`	A reference to an array of doubles, and the length of the array	Prints an array of double-precision numbers on a single line. Each number should display exactly two digits after the decimal (even if they are zeroes). The numbers should be separated by commas, and the entire array should be surrounded by [square brackets]. Print a newline at the end.
`array_average`	A reference to an array of doubles, and the length of the array	Returns the double-precision average (mean) of all the numbers in the array.
`reduce_value_by`	A reference to a double, and the double-precision amount to subtract from it	Subtracts the given amount from a double-precision number. This should modify the original value.
`round75`	A double-precision value	Returns the integer that results from rounding the value in the following way: if the decimal part is .75 or larger, round away from zero ("up"), and round toward zero ("down") otherwise.
`reverse_int`	A reference to the integer to be reversed	Reverses the digits of a positive integer. If the given integer is negative, this function should leave it alone and return 1. Otherwise, it should modify the original value and return 0.
`parity_sort`	A reference to an array of integers, and the length of the array	Rearranges an array of integers so that all the even numbers are at the beginning and the odd numbers are at the end. Within a given parity (even or odd), the numbers should be in the same order they were in the original array.
`get_num_length`	An integer value	Returns the number of characters it would take to print this number, taking into account the extra character '-' for negative numbers.
`print_sum`	A reference to an array of integers, and the length of the array	Prints a nicely formatted vertical-style sum of all the integers in an array (see sample output in Figure 2). There should be exactly one column of space between the '+' and the longest number, and the horizontal line should be just long enough so that nothing extends past it.

Figure 1: Functions to define and implement

```
Input values:
[40.00, 5.20, -1.00, -3.80, 42.00, -0.80, 3.14, 3.15, 18.45, 222.00, 3.33, -777.70, -15.00, 678.00, 432.00]
The average of the inputs is 43.26.
Recentered input values:
[-3.26, -38.06, -44.26, -47.06, -1.26, -44.06, -40.12, -40.12, -24.81, 178.74, -39.93, -820.96, -58.26, 634.74, 388.74]
Rounded integers:
[-3, -38, -44, -47, -1, -44, -40, -40, -25, 178, -40, -821, -58, 634, 388]
Reversed digits:
[-3, -38, -44, -47, -1, -44, -40, -40, -25, 871, -40, -821, -58, 436, 883]
After parity sort:
[-38, -44, -44, -40, -40, -40, -58, 436, -3, -47, -1, -25, 871, -821, 883]
Sum of values:
   -38
   -44
   -44
   -40
   -40
   -40
   -58
   436
    -3
   -47
    -1
   -25
   871
  -821
+  883
------
   989
```

Figure 2: Sample output