CS270 Programming Assignment 1
"Number Detective"

Due Monday, February 7 (via RamCT by 4:59pm)
Homework and programming assignments are to be done individually.

Goals

In this project you will learn C programming and reinforce your understanding of number representation by implementing several C functions for converting decimal and floating point numbers.

The Assignment

Make a subdirectory called PA1 for the programming assignment; all files must reside in this subdirectory. We provided you with a number of files to complete the assignment. Copy the following files into your PA1 directory:

```
http://www.cs.colostate.edu/~cs270/Assignments/PA1/main.c
http://www.cs.colostate.edu/~cs270/Assignments/PA1/myfunctions.h
http://www.cs.colostate.edu/~cs270/Assignments/PA1/myfunctions.c
http://www.cs.colostate.edu/~cs270/Assignments/PA1/Makefile
```

You will need to implement the following functions inside "myfunctions.c". The skeleton structure has already been provided for you.

```
int bits2Integer(char bitString[], int length);
int decimal2Integer(char decimalString[], int length);
void float2Bits(char floatString[], char bitString[]);
```

Function #1:

Convert a string array of 1's and 0's in 2's complement form (max 32 digits) to a 32-bit signed integer form of a machine. The bit string is stored in `bitString[]` argument, and the number of 1's and 0's in this bit string is passed via the `length` argument. The bit string itself is input in "natural" form, i.e., the MSB is on the left side and LSB to the right. You can access individual elements of the bit string array using an array index, like so:

```
if (bitString[index] == 1) {intvalue += pow(2, position);}
```

You need to apply the techniques you learned in HW-1 for the conversion (i.e., multiply by the weight of a digit). Alternate techniques (such as library calls) will not be accepted. Failure to follow these guidelines will result in loss of points. Note that the function returns the 32-bit signed int value resulting from the conversion.

Function #2:

Convert a signed decimal number string to a 32-bit signed integer form of a machine. The decimal digits (in string form) are stored in `decimalString[]` array, and `length` corresponds to number of digits in the decimal number plus the leading sign character. For example, for an input number "-123", the length will be 4.

You need to apply the techniques you learned in HW-1 for the conversion. Alternate techniques (such as library calls) will not be accepted. Failure to follow these guidelines will result in loss of points. Similar to function #1, this function returns the 32-bit signed int value resulting from the conversion.

Function #3:

In this function you will be converting from a floating point number string to a bit string of 1's and 0's. You need to use a technique similar to the paper and pencil technique you leaned to convert floating point numbers to binary digits. We will not accept alternate techniques (such as using pointer type casting, library calls and techniques found on the net). Failure to stick to these guidelines will result in loss of points.

Format of input floating point number:

For your convenience, we are limiting the input floating numbers to a fixed format with a total of 10 characters. The numbers will start with a '+' or a '-' sign, followed by an integer part, then a '.' character, and then followed by a fractional part (that means the floating point number can only have a max of 8 digits/numbers). The fraction will be 0 padded as necessary.

Examples:

+12.200000, -1234567.8 … etc.

The technique:

- Start by converting the integer part of the floating point string found in `floatString[]` to a binary string of 1's and 0's. You need to use `bitString[]` to store your results.
- If the number starts with a '-', the first character in `bitString[]` (i.e., `bitString[0]`) will be '1', and '0' otherwise (i.e., incase the number starts with a '+').
- Convert the fraction part to a float value.
    - (Note: this part has already been filled for you in the code).
- Now use this float variable to figure out the 1's and 0's of the floating point number, just like you did in the homework (i.e., successive multiplication).
- Normalize the result and compute the exponent.
- Convert the exponent to a bit string and fill into the appropriate location inside `bitString[]` array.

Compile and Run:

We provided you with a Makefile to compile the program. Run the program first using the following command:

```
%> pa1 --help
```

This will produce a small usage message with the command line arguments to run the program.

--bits2int will invoke the bits2Interger() function.

--decimal2int will invoke the decimal2Integer() function.

--float2bits will invoke the float2Bits() function.

Run the program with the following test cases:

```
%> pa1 --bits2int 1010000110011
%> pa1 --decimal2int +1234
%> pa1 --float2bits -22.62500
```

Calculate the results by hand and verify if you are getting the correct output. We will run your program using different inputs, so DO NOT HARDCODE values!

For this assignment you must also submit a README file with your name and answers to the following questions. Copy the question into the file and then type in the answer after the question.

Question 1: Are you doing your assignments on the school machines or at home? If at school what is the name of the machine you are using to answer these questions?

Question 2: Type gcc --version on the command line and write down the output. This assumes Linux, if you are running on another operating system, then write down the compiler version.

Question 3: Does the machine you are on use the little endian or big endian representation? How did you determine that?

Submission Instructions

When you are done, your directory should have main.c, myfunctions.c, myfunctions.h, Makefile and a README file. To package the files into a single compressed file, type the following command from inside PA1 directory:

```
%> cd PA1
%> make pack
```

This will create a file called PA1.tar.gz one directory above PA1. All assignments will be submitted directly to RamCT, which will be explained and demonstrated in class. A sanity check of your PA1.tar.gz will ensure that your submission has all the required files:

```
%> mkdir ~/Temp
%> cp PA1.tar.gz ~/Temp
%> cd ~/Temp
%> tar -zxvf PA1.tar.gz
%> ls PA1
```

Grading Criteria

Points will be awarded as follows: functionality - 75 points (25 for each function), coding style and comments - 10 points, following assignment directions - 5 points, and supplying answers to the README questions - 10 points. The grading factors we consider for coding style include having clear and concise comments, consistent indentation, and the minimal amount of code to solve the problem. You will also need to ensure that every function (declared in myfunctions.h) has a properly formatted description header.

Late Policy

Late assignments will be accepted up to 24 hours past the due date and time for a deduction of 25% and will not be accepted past this period. RamCT will report assignments as late the second the deadline passes, so make sure and leave yourselves time to submit to RamCT. Please contact the instructor or teaching assistant if you have RamCT problems.