Joanna Klukowska
joannakl@cs.nyu.edu

Project 1: Working with Bits.

Due date: Feb. 28, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but **all work for all assignments must be entirely your own**. Any sharing or copying of assignments will be considered cheating.

Things to remember for all of the programs:

- If you are allocating memory using `malloc`, you need to deallocate it using `free`. Proper memory management is part of the grade for each problem.
- Do not return dangling pointers (pointer to a local variable that was created in a function) - it may take a while before you see the results and these are things that are hard to debug.
- Use `sizeof` operator to determine sizes of types, do not hardcode the actual numerical values. The only exceptions to this rule if for problems 4 and 5 in which you have to assume that the single precission floating point numbers are represented using 4 bytes (32 bits).
- Use parenthesis to avoid ambiguity of operator precedence (this will save you a lot of debugging time).

The problems below are listed in order of difficulty. You should attempt to solve them in that order.

Problem 1 (20 points): Break a Secret Messages

A beginner programmers was attempting to print text to an output file. Something went wrong and the output file contains numbers instead of the text. Your job is to read the file and figure out the text that your fellow programmer started with.

The numbers in the output file resulted from incorrect type of the pointer that was used to iterate through the data, not by corruption of the data, so all the correct bits are there. You simply need to interpret them using correct types.

You can assume that the text file contains at most 1000 lines of numbers (one number per line). You can also assume that the numbers fit in the integer range.

Your program should work with any file name. The file name should be specified as the command line argument to the program.

You need to define a function with the following signature

```
char * convert_to_string ( int text[] , int size );
```

in a file called `proj1.c`. This function should take an array of integers (these are the values read from your file) and the number of elements in the array. It should return a string that corresponds to the original message.

Problem 2 (20 points): Create a Secret Message

After you solved probelm 1, you decide that it would actually be useful to store secret messages in this fashion (and distribute them to your yournger collegues who did not yet have a pleasure of taking CSO and cannot possibly figure out what they mean).

Write a program that reads regular text (containing ASCII characters) from an input file (provided as argument 1 on the command line) and writes its numberical equivalent to an output file (provided as argument 2 on the command line). The output file should contain numbers (in the range of int type), one per line.

You need to define a function with the following signature

```
int convert_to_int ( char * text );
```

in a file called `proj1.c`. This function should take a c-string as its parameter and convert the first `sizeof(int)` bytes of the c-string to an integer that is returned. Your function should work even if the c-string passed to it has fewer than `sizeof(int)` characters (the missing bits should be filled with zeroes.

Problem 3 (20 points): Extract bits and bytes

Implement two functions: one that extracts the most significant byte out of an integer variable and one that extracts the least significant byte out of an integer variable.

Joanna Klukowska
joannakl@cs.nyu.edu

You need to define two functions with the following signatures in a file called `proj1.c`.

`int get_most_significant_byte ( int x );`

This function should take an integer as its parameter and return an integer that stores the value of the most significant byte of that number.

`int get_least_significant_byte ( int x );`

This function should take an integer as its parameter and return an integer that stores the value of the least significant byte of that number.

For example:
when called with 123 (0x0000007b), the first fucntion should return 0 (0x00) and the second should return 123 (0x7b);
when called with -123 (0xffffff85), the first function should return 255 (0xff) and the second should return 133 (0x85);
when called with 128974848 (0.07b00000), the first function should return 7 (0x07) and the second should return 0 (0x00).

RESTRICTION: You are not allowed to use any loops or conditional statements in the implementation of this function. Your implementation should use `sizeof(int)` to determine the number of bytes in an integer, not a hardcoded 4.

Problem 4 (20 points): *Split* Bits of a Single Precission IEEE Floating Point

Imeplement functions that extract the bits from the single precission IEEE floating point number. Your need to write three different functions: one that extracts the expontent bits (`exp` in the textbook), one that extracts the fraction bits (`frac` in the textbook), and one that extracts the sign bit (`s` in the textbook). Use section 2.4.2 in the textbook as a reference regarding which bits belong to which of the three parts.

You need to define three functions with the following signatures in a file called `proj1.c`

`unsigned int * get_exp ( float f );`

This function should take a single precission floating point number and return bits conrresponding to the exponent bits as a pointer to `unsigned int`. All bits other than the exponenet bits should be zeroed in the returned value.

`unsigned int * get_frac ( float f );`

This function should take a single precission floating point number and return bits conrresponding to the fraction bits as a pointer to `unsigned int`. All bits other than the fraction bits should be zeroed in the returned value.

`unsigned int * get_sign ( float f );`

This function should take a single precission floating point number and return the bit conrresponding to the sign bit as a pointer to `unsigned int`. All bits other than the sign bit should be zeroed in the returned value. (Notice that this means that the returned value will be either or zero bits or a single 1 bit in the most significant position followed by all zero bits.)

Here are sample outputs that your functions should produce (assuming the bits of the values that you are returnding are printed):

```
value    1.200000e+38
all bits              01111110101101001000111001010010
exp      2122317824   01111110100000000000000000000000
frac        3444306   00000000001101001000111001010010
sign              0   00000000000000000000000000000000
```

```
value    201.0
all bits              01000011010010010000000000000000
exp      1124073472   01000011000000000000000000000000
frac        4784128   00000000010010010000000000000000
sign              0   00000000000000000000000000000000
```

```
value    -1024.25
all bits              11000100100000000000100000000000
exp      1149239296   01000100100000000000000000000000
frac           2048   00000000000000000000100000000000
sign     2147483648   10000000000000000000000000000000
```

```
value    1.399995e-40
all bits              00000000000000011000011001000011
exp               0   00000000000000000000000000000000
frac          99907   00000000000000011000011001000011
sign              0   00000000000000000000000000000000
```

```
value    inf
all bits              01111111100000000000000000000000
```

Joanna Klukowska
joannakl@cs.nyu.edu

```
exp      2139095040   01111111100000000000000000000000
frac              0   00000000000000000000000000000000
sign              0   00000000000000000000000000000000
```

Note that for this problem you are simply writing the functions, not a program that displays the above values (this will come in the next problem).

Problem 5 (20 points): Deciphering Single Precission IEEE Floating Point

In this problem you will build on the functions that you created for problem 4. The goal is to be able to write a program that determines the values of E, M and s variables in the formula

$$V = (-1)^s \times M \times 2^E$$

used for encoding the floating point numbers. See section 2.4.2 in the textbook for reference.

You need to implement the following functions in a file called `proj1.c`:

`int is_normalized ( float f );`
This function should take a single precission floating point number and return zero (a.k.a., false) when the value of `f` is not normalized, and any other number (a.k.a., true) when the value of `f` is normalized.

`int is_denormalized ( float f );`
This function should take a single precission floating point number and return zero (a.k.a., false) when the value of `f` is not denormalized, and any other number (a.k.a., true) when the value of `f` is denormalized.

`int is_special ( float f );`
This function should take a single precission floating point number and return zero (a.k.a., false) when the value of `f` is not special, and any other number (a.k.a., true) when the value of `f` is special.

`int get_E ( float f );`
This function should take a single precission floating point number and return the numerical value of E from the above formula. The value of E is related to the `exp` bits.

`char * get_M ( float f )`
This function should take a single precission floating point number and return the numerical value of M from the above formula. The value of M should be returned as a string with the appropriate leading character and containing the decimal point. The value of M is related to the `frac` bits.

`int get_s ( float f );`
This function should take a single precission floating point number and return the numerical value of s from the above formula, i.e. `0` for positive values of `f`, `1` for negative values of `f`. The value of E is related to the `exp` bits.

Write a program that uses all of the above functions decipher the binary encoding of single precision IEEE floating point numbers. Your program should prompt the user for a floating point number and display the information about that number as shown in the exaples below.

User enters `1.2e+38`

```
value    1.200000e+38
all bits              01111110101101001000111001010010
exp      2122317824   01111110100000000000000000000000
frac        3444306   00000000001101001000111001010010
sign              0   00000000000000000000000000000000
1.200000e+38 is normalized
E = -127
M = 1.01101001000111001010010
s = 0
```

User enters `201.0`

```
value    2.010000e+02
all bits              01000011010010010000000000000000
exp      1124073472   01000011000000000000000000000000
frac        4784128   00000000010010010000000000000000
sign              0   00000000000000000000000000000000
2.010000e+02 is normalized
E = -127
M = 1.10010010000000000000000
```

Joanna Klukowska
joannakl@cs.nyu.edu

```
s = 0
```

User enters `-1024.25`

```
value    -1.024250e+03
all bits              11000100100000000000100000000000
exp      1149239296   01000100100000000000000000000000
frac           2048   00000000000000000000100000000000
sign     2147483648   10000000000000000000000000000000
-1.024250e+03 is normalized
E = -127
M = 1.00000000000100000000000
s = 1
```

User enters `1.4e-45`

```
alue    1.401298e-45
all bits              00000000000000000000000000000001
exp               0   00000000000000000000000000000000
frac              1   00000000000000000000000000000001
sign              0   00000000000000000000000000000000
1.401298e-45 is denormalized
E = -127
M = 0.00000000000000000000001
s = 0
```

This one cannot be entered from the keyboard, but you can test your program by setting manually the value of the floating point number of `1.0/0.0`

```
value    inf
all bits              01111111100000000000000000000000
exp      2139095040   01111111100000000000000000000000
frac              0   00000000000000000000000000000000
sign              0   00000000000000000000000000000000
  inf is special
E = -127
M =
s = 0
```

This one cannot be entered from the keyboard either, but you can test your program by setting manually the value of the floating point number of `1.0/0.0 - 1.0/0.0`

```
value    -nan
all bits              11111111110000000000000000000000
exp      2139095040   01111111100000000000000000000000
frac        4194304   00000000010000000000000000000000
sign     2147483648   10000000000000000000000000000000
 -nan is special
E = -127
M =
s = 1
```

Accessing and submitting this project

You will be given access to a private repository called `YOUR_GITHUB_USERNAME_proj1` on GitHub and, of course, `YOUR_GITHUB_USERNAME` should be replaced by yout GitHub username). **DO NOT FORK THIS REPOSITORY! You should be working directly with that repository.**

The repository contains an empty file `proj1.c` file and a corresponging `proj1.h` header file. You need to implement most of your code in `proj1.c`.

It also contains empty `prob1.c`, `prob2.c` and `prob5.c` files. You need to implement the programs for these problems in those files.

It contains a textfile `message` that contains the scrambled message for problem 1. You should try to decipher that input file. You'll know when you get the correct text.

Joanna Klukowska
joannakl@cs.nyu.edu

It contains a file called `Makefile` - you should not need to edit this file, unless you are adding additional source code and header files. You can create executables for programs for problems 1, 2 and 5 by running one of the following from your terminal

```
make prob1
make prob2
make prob5
```

This will either produce errors resulting from compilation errors (preprocessing, compilation or linking), or produce an executable files called `prob1`, `prob2` and `prob5`, respectively. Each of the commands will also produce the object file `proj1.o`.

If you make any changes to the `Makefile` make sure that you submit the modified `Makefile` with your assignment

Do not add and commit the binaries produced by the makefile. You can run

```
make clean
```

to remove all the binary files from the directory before committing changes.

To submit the homework, **push** the final version to that repository. You should push intermediate versions as well - this is a way to make sure that you have a backup of the files. We will collect your files from your repository at the due date. (You may make further changes to the code, but they will not be graded.)

NOTE: you should write test code for the functions that you implement for problems 3 and 4. But these files should not be submitted with the assignment.

Questions

Post any questions you have regarding this assignment to Piazza under the ”homeworks” topic.