MAT 4370: Programming Assignment 15

Due: Wednesday, April 24

Learning Objectives

This assignment is an extension of the programming project found on page 527 of the King textbook. By completing this assignment, you will gain experience with the following:

- bit fields
- the `struct` and `union`
- floating point representation

Although you will use "low level" programming techniques, your program will not need to use any bitwise or shift operators.

Background

The IEEE standard for floating point arithmetic (also known as IEEE 754) provides a specification for how floating point values are stored on modern computers. Single precision floating point values consist of a 4-byte quantity (i.e., 32 bits) subdivided into three fields:

- 1 bit for the **sign**
- 8 bits for the **exponent**
- 23 bits for the **fraction** (also known as the significand or mantissa)

The website `http://babbage.cs.qc.cuny.edu/IEEE-754/` provides an online converter, which I encourage you to use during the development of your program.

As an example of a single precision floating point value, consider the 32 bit quantity `0x40600000`. Writing this in binary, we get:

0100 0000 0110 0000 0000 0000 0000 0000

Subdividing this 32-bit quantity into the fields described above, we get:

sign : 0

exponent : 1000 0000

fraction : 110 0000 0000 0000 0000 0000

What is the meaning of these bit patterns?

The sign is simple: 0 is positive; 1 is negative

The exponent uses "excess 127", so the actual exponent is $e - 127$ where e is the exponent value viewed as an unsigned value

The fraction has an implied "hidden bit", so if the 23-bit fraction is f, the actual value is $1.f$, base 2.

For this example,

- the sign is positive,
- the exponent is $128 - 127 = 1$, and
- the fraction is $(1.110\ 0000\ 0000\ 0000\ 0000\ 0000)_2 = 1 + 2^{-1} + 2^{-2} = 1.75$

Therefore, the value represented is $+(1.11)_2 \times 2^1 = 3.5$.

Desired Program

We want a C program which will allow us to see the machine representation of single precision floating point values. Given a floating point value in "human form" (e.g., 3.5), we would like a program to display each of the three fields as hexadecimal quantities. For example, given input 3.5, we wish to see:

```
sign = 0, expo = 0x80, fraction = 0x600000 (0x40600000)
```

Similarly, we would also like to program to accept the three fields of a floating point value and output the "human form" in response. For example, given input `0:0x80:0x600000`, the program should determine the equivalent value, 3.5.

Your program should include the following components:

- a `struct` with bit fields for each of the three floating point fields
- a `union` which allows three different viewpoints for a given floating point value: a collection of 32 bits, the three IEEE fields, and as a `float`.
- Two functions, with the following prototypes:

```
void
display_hex(float value);

void
display_float(unsigned int sign, unsigned int exponent, unsigned int fraction);
```

  The first function will display a given floating point value in hexadecimal, showing the individual IEEE fields as well as the 8-digit hexadecimal value.

  The second function should display the "human form" of a floating point value given its three IEEE fields, given in hexadecimal.

Your program should demonstrate each of these functions by providing its user the ability to enter an arbitrary number of test cases. See the sample program execution shown in Figure 1.

What to Submit

Put your source in code in a folder named `hw15` and submit your work in the usual manner.

```
Demo #1:  sign:exponent:fraction -> floating point value
  sign    : 0 or 1
  exponent: 2 digit hexadecimal value
  fraction: 6 digit hexadecimal value (0x000000 ... 0x7FFFFF)

Enter desired number of test cases: 2
> 0:0x80:0x600000
Floating point value =   3.50000000

> 1:0x81:0x700000
Floating point value =  -7.50000000


Demo #2:  floating point value -> hexadecimal representation
Enter desired number of test cases: 3
> -7.5
sign = 1, expo = 0x81, fraction = 0x700000 (0xc0f00000)

> 3.5
sign = 0, expo = 0x80, fraction = 0x600000 (0x40600000)

> 0.75
sign = 0, expo = 0x7e, fraction = 0x400000 (0x3f400000)
```

Figure 1: Sample program execution. All program inputs occur in response to the > prompt.