CSED 211, Fall 2021
Data Lab: Manipulating Bits
Assigned: Sept. 16, Due: Sept. 23, 11:59PM

SeongKu Kang (seongku@postech.ac.kr) is the lead person for this assignment.

1 Introduction

The purpose of this assignment is to become more familiar with floating point numbers. You'll solve five problems in the presentation.

2 Logistics

This is an individual project. All handins are electronic. Clarifications and corrections will be posted on the course Web page.

3 Handout Instructions

Start by copying bits.c to a (protected) directory on a Linux machine in which you plan to do your work. Then give the command

```
unix> cp bits.c ./path/your/directory
```

You will be modifying and turning in is `bits.c`.

The `bits.c` file contains a skeleton for each of the 5 programming puzzles. See the comments in `bits.c` for detailed rules and a discussion of the desired coding style.

4 The Puzzles

This section describes the puzzles that you will be solving in `bits.c`. You will implement some common single-precision floating-point operations. You are allowed to use standard control structures (conditionals,

loops), and you may use both `int` and `unsigned` data types, including arbitrary unsigned and integer constants. You may not use any unions, structs, or arrays. Most significantly, you may not use any floating point data types, operations, or constants. Instead, any floating-point operand will be passed to the function as having type `unsigned`, and any returned floating-point value will be of type `unsigned`. Your code should perform the bit manipulations that implement the specified floating point operations.

Table 1 describes a set of functions that operate on the bit-level representations of floating-point numbers. Refer to the comments in `bits.c` for more information.

Name	Description	Rating
`float_neg(uf)`	Compute `-f`	2
`float_i2f(x)`	Compute `(float) x`	4
`float_twice(uf)`	Compute `2*f`	4
`float_abs(uf)`	Compute absolute value of `f`	2
`float_half(uf)`	Compute `0.5*f`	4

Table 1: Floating-Point Functions. Value `f` is the floating-point number having the same bit representation as the unsigned integer `uf`.

Functions `float_neg(uf)`, `float_twice(uf)`, `float_abs(uf)`, `float_half(uf)` must handle the full range of possible argument values, including not-a-number (NaN) and infinity. The IEEE standard does not specify precisely how to handle NaN's, and the IA32 behavior is a bit obscure. We will follow a convention that when of these functions is given a NaN value as argument, it returns the same value for the result (Refer to the comments in `bits.c`).

5 Evaluation

Your score will be computed out of a maximum of 16 points. Correctness points. We will evaluate your functions. You will get full credit for a puzzle if it passes all of the tests, and no credit otherwise.

6 Handin Instructions

Upload your source file `bits.c` and report in plms. You need to explain your answer in the report. The format of file is (student number)_(your name).c / .doc.

7 Advice

- Don't include the `<stdio.h>` header file in your `bits.c` file, as it confuses `dlc` and results in some non-intuitive error messages. You will still be able to use `printf` in your `bits.c` file for debugging without including the `<stdio.h>` header, although `gcc` will print a warning that you can ignore.

- The `dlc` program enforces a stricter form of C declarations than is the case for C++ or that is enforced by `gcc`. In particular, any declaration must appear in a block (what you enclose in curly braces) before any statement that is not a declaration. For example, it will complain about the following code:

```
int foo(int x)
{
  int a = x;
  a *= 3;      /* Statement that is not a declaration */
  int b = a;  /* ERROR: Declaration not allowed here */
}
```