Programming Assignment # 1

Mediocre Precision Floating Point Data Type

1. Objectives

Design and implement the 48-bit floating point data type and its associated multiplication and addition operations. Write a program that gets two mediocre data through keyboard input and prints both multiplication and addition of them in the bit-level representation.

2. Details

32-bit *float* is too coarse for scientific computation. 64-bit *double* is too big. We want a compromise between float and double. Therefore, here we design and implement a new data type, named "mediocre precision floating point data type".

A mediocre precision variable is 48-bit wide. In C, a mediocre precision variable is represented with a structure holding a character array of six as follows:

```
struct mediocre {
    char data[6];
};
typedef struct mediocre mediocre;
```

The internal structure of the mediocre type consists of 1 bit for sign, 9 bit for exponent and 38-bit for significand, from MSB to LSB. They are organized in the array, *data*, in little-endian form.

There are two arithmetic operation functions associated with this data type.

```
/* multiplication of two mediocre variables */
mediocre mult(mediocre op1, mediocre op2);

/* addition of two mediocre variables */
mediocre add(mediocre op1, mediocre op2);
```

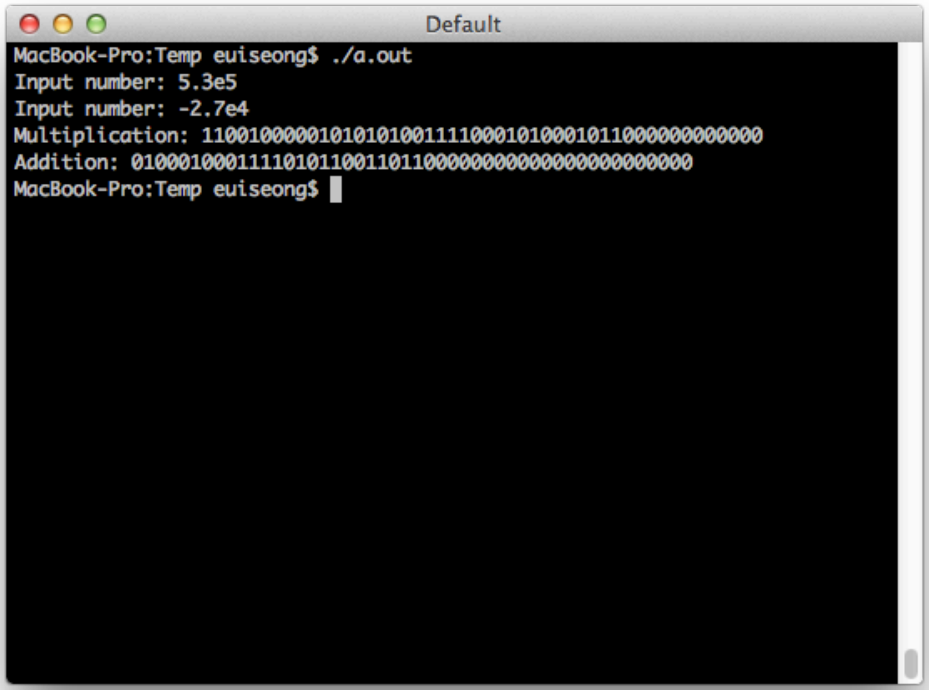
Because your program needs to obtain keyboard input for two mediocre operands and print the result of multiplication and addition operations, you have to write following functions for input/output:

```
/* obtaining a mediocre operand through keyboard input */
int key_in(mediocre *op);

/* printing bit-level representation of a mediocre variable */
int print_bits(mediocre op);
```

Restriction: **A.** You can't use *double* or *float* typed variables in your code except the *key_in* function. **B.** Your data must be aligned in little endian so that the LSB must be accommodated in data[0] and the MSB in data[5].

3. Example



```
MacBook-Pro:Temp euiseong$ ./a.out
Input number: 5.3e5
Input number: -2.7e4
Multiplication: 110010000010101010011110001010001011000000000000
Addition: 010001000111101011001101100000000000000000000000
MacBook-Pro:Temp euiseong$ 
```

4. Hints

A. There are no standard ways to reinterpret “double precision” as “long long int” (casting will change the bit sequence). So, I would recommend to use “memcpy” function, which copies the contents of a memory area to another memory area. An example code is listed below.

```
#include <string.h>
........
long long int casted;
double input;

printf(“Input number: “);
scanf(“%lf”, &input);
memcpy(&casted, &input, sizeof(double));
```

B. The basic algorithm of the *key_in* function is getting the input as a *double* typed data and casting it to the mediocre type manually. During the procedure, you supposedly round significand of the double-typed data. The easiest round to implement in C is “round toward zero”. Use “round toward zero” as your rounding method.

5. Logistics

A. Make sure that you have included your name and ID in the header comment of your code.

B. All three functions must be packed in a single file. The source file name should be "studentid.c" (e.g. 2015310123.c).

C. Submit the source file and document file via i-Campus by the due date. The due is a hard deadline. No delayed submissions are allowed.

D. Your C file must be compiled with "-c" option to an object file. The object file will be linked together with the auto-tester program. So, NEVER put a function named "main" in your C file. Compilation will fail if there is a main function in your program.

E. NEVER include the declarations of the "mediocre" data type and "mult", "add", "key_in" and "print_bits" functions in the C file to submit because they will also make troubles during the compilation. Use a separate header file for them while you are programming and testing.