Programming Assignment # 1

Double Precision Addition/Multiplication without Overflow

1. Objectives

Design and implement the multiplication and addition functions between two double precision operands. The return values from your functions must be 128-bit floating point data typed, and thus no overflow occurs. Write a test program that obtains two double precision operands through the standard input and prints both the sum and multiplication of the two operands by using the implemented functions.

2. Details

As learned in our class, overflow, which produces "infinity", may occur when a computer conducts multiplication or addition over two floating point variables unless the result is stored in a larger data typed variable. Therefore, if you do not want infinity outcomes from arithmetic operations over 32-bit *float* variables, you can cast the operands to 64-bit *double* and conduct the operation. However, this workaround cannot be applied to arithmetic between two double precision operands since the standard C does not provide a 128-bit floating point data type. Therefore, you must define your own 128-bit floating point data type and supporting functions for it.

The internal structure of the 128-bit floating point data type, which we call *super* precision from now on, consists of 1 bit for sign, 15 bits for exponent and 112 bits for significand, from MSB to LSB. They are organized in a character array named data, in the little-endian form. The super precision data type is defined as follows:

```
typedef struct {
    char data[16];
} super;
```

There are two arithmetic functions that return super typed values.

```
/* multiplication of two doubles */
super super_mult(double op1, double op2);

/* addition of two doubles */
super super_add(double op1, double op2);
```

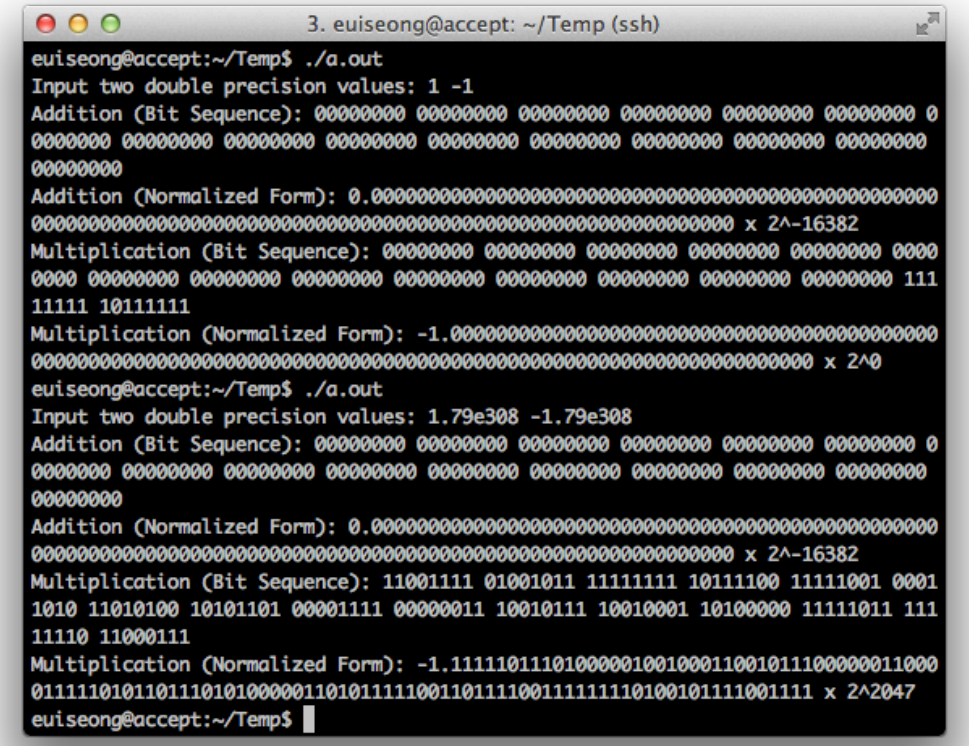
Because the *printf* function does not know how to print a super typed variable, you need two additional functions that produce strings out of a super typed variable. The first function returns the bit sequence of the operand, and the other one returns the normalized form of the operand in binary (for example, 1.010010011100 x 2^1203).

```
/* returns the bit sequence string */
char *super_print_bitseq(super op);

/* returns the string of the normalized form in binary */
char *super_print_normalized(super op);
```

Restriction: Your data must be aligned in little endian so that the LSB must be accommodated in data[0] and the MSB in data[3]. Do not use long double or any non-standard floating point data types. Also, use of any arbitrary precision arithmetic libraries are prohibited.

3. Evaluation



4. Hints

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));
```

5. Logistics

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

B. The source file name should be "studentid.c" (e.g. 2012310123.c)

C. Prepare a separate document in PDF format, which explains the design and implementation of your code. The document file name should be "studentid.pdf" (e.g. 2012310123.pdf).

E. Compress and merge the source code file and PDF document file into a single zip file. Submit the zip file via i-Campus by the due date.

F. Only the assignments submitted before the deadline will receive the full credit. 25% of the credit will be deducted for every single day delay.