

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 *long_double* 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 an unsigned character array named *data*, in the little-endian form. The *long_double* precision data type is defined as follows:

```
typedef struct {  
    unsigned char data[16];  
} long_double;
```

There are one type conversion and two arithmetic functions that return *long_double* typed values.

```
/* type conversion */  
long_double double_to_long_double(double op);  
  
/* multiplication of two doubles */  
long_double FP_mul(long_double op1, long_double op2);  
  
/* addition of two doubles */  
long_double FP_add(long_double op1, long_double op2);
```

Because the *printf* function does not know how to print a *long_double* typed variable, you

need two additional functions that produce strings out of a *long_double* typed variable. The first function returns the bit sequence of the operand (from MSB to LSB spaced in a group of 8 bits), and the other one returns the normalized form of the operand in binary (for example, 1.010010011100 x 2¹²⁰³).

```

/* returns the bit sequence string */
char *long_double_print_bitseq(long_double op);

/* returns the string of the normalized form in binary */
char *long_double_print_normalized(long_double 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[15]. 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

[illegible]

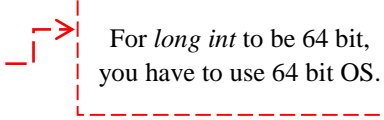
4. Hints

I would recommend to use type casting, which convert the values from one type to another explicitly using the cast operator. When you manipulate bits in data with bitwise operators and shift operators such as `&`, `|`, `^`, `<<`, `>>`, etc.

An example code is listed below.

```
#include <stdio.h>
...
    unsigned long *casted;
    double input;

    printf("Input number: ");
    scanf("%lf", &input);
    casted = (unsigned long*)( &input);
```



For *long int* to be 64 bit,
you have to use 64 bit OS.

5. Logistics

- A. 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. 2018310123.pdf).
- B. Compress and merge the source code file and PDF document file into a single tar file. Submit the tar file via i-Campus by the due date (Apr 13, 11:59pm).
- C. Only the assignments submitted before the deadline will receive the full credit. 10% of the credit will be deducted for every single day delay.

6. Readme

- A. How to decompress a tar file

```
$ tar xvzf PA1_ldfp.tgz
```

- B. How to build the program

```
$ make
```

- C. How to clean the program

```
$ make clean
```

- D. How to compress a tar file

```
$ tar cvzf studentid.tgz ldfp.h ldfp.c main.c Makefile studentid.pdf
```