National Taiwan Normal University
CSIE Programming II

Instructor: Po-Wen Chi
Due Date: 2021.03.30 PM 11:59

Assignment 2

Policies:

- Zero tolerance for late submission.
- You need to prepare a README file about how to make and run your program. Moreover, you need to provide your name and your student ID in the README file.
- For the writing assignment, I only accept pdf. MS. doc/docx format is not acceptable. Moreover, please use Chinese instead of English except foreign students.
- Do not forget your Makefile. For your convenience, each assignment needs only one Makefile.
- The executable programs should be hw0201, hw0202 ....
- You should pack your homework in one zip file. The file name should be StudentId_hw02.zip.

2.1 Mixed Number Arithmetic (20 pts)

A mixed number is a whole number, and a proper fraction represented together. It generally represents a number between any two whole numbers. There are some examples here.

$$1\frac{3}{4}, 5\frac{5}{7}, -2\frac{11}{19}, \ldots$$

Given $a\frac{b}{c}$, a is the whole number, b is the numerator and c is the denominator.

Please develop a structure for the mixed number and implement some arithmetic operations. You need to provide a header file **mixed.h** and its .c file. TA will prepare hw0201.c which include your header file. Note that you should prepare a Makefile to build hw0201.c.

```
typedef struct _sMixedNumber {
}sMixedNumber;

```

```
4  int mixed_set( sMixedNumber *pNumber, int32_t a, int32_t b,
       int32_t c);
5  // return -1 if invalid; otherwise, return 0.
6  int mixed_print( const sMixedNumber number);
7  // in the form of (a,b,c)
8  void mixed_add( sMixedNumber *pNumber, const sMixedNumber r1,
       const sMixedNumber r2);
9  // pNumber = r1 + r2
10 void mixed_sub( sMixedNumber *pNumber, const sMixedNumber r1,
       const sMixedNumber r2);
11 // pNumber = r1 - r2
12 void mixed_mul( sMixedNumber *pNumber, const sMixedNumber r1,
       const sMixedNumber r2);
13 // pNumber = r1 * r2
14 void mixed_div( sMixedNumber *pNumber, const sMixedNumber r1,
       const sMixedNumber r2);
15 // pNumber = r1 / r2
```

Note:

- The fraction part of the mixed number should be in the irreducible form and its absolute value should be less than 1. Both the numerator and the denominator should be integers.
- If there is no fraction part, you should use $(x, 0, 0)$ to present the number.
- The signed bit exists in the whole number. The numerator can be negative if and only if the whole number is 0.

2.2 IEEE 754 (20 pts)

You all know what IEEE 754 is, right? The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point arithmetic established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE). If you forget what it is, I hope figure 2.1 can arouse your memory.

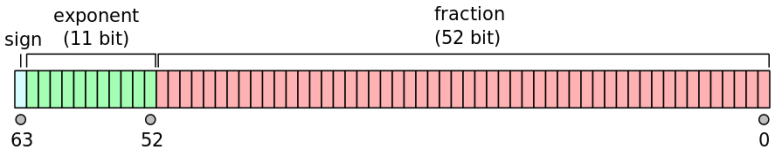

FIGURE 2.1: Double-precision floating-point format.

Now I want you to develop a program for the user to input a double floating-point number and display the number as **sign, exponent, fraction**.

```
$ ./hw0202
Please enter a floating-point number (double precision): 1.75
Sign: 0
Exponent: 01111111111
Fraction: 1100000000000000000000000000000000000000000000000000
1.75 = (-1)^0 * (1 + 2^-1 + 2^-2) * 2^(1023-1023)
```

2.3 Bit Modifier (20 pts)

Please look at the following code.

```
typedef union
{
    struct {
        unsigned char b1:1;
        unsigned char b2:1;
        unsigned char b3:1;
        unsigned char b4:1;
        unsigned char b5:1;
        unsigned char b6:1;
        unsigned char b7:1;
        unsigned char b8:1;
    } bits;
    unsigned char byte;
} uByte;
```

Please use this union type to write the following program.

```
$ ./hw0203
Please enter a byte (0-255): 255
Data: 255 11111111
Flip bit (1-8, 0: exit): 1
Data: 127 01111111
Flip bit (1-8, 0: exit): 1
Data: 255 11111111
Flip bit (1-8, 0: exit): 0
Bye
```

Note that you cannot use the bit operator.

2.4 Big Number Library (40 pts)

Most of you complain that you can get higher scores if you can have more time in the last semester. Congratulations! Now, your wish comes true. This problem is exactly the same with the problem in the final exam, except that this time, you should design your own structure.

As you all know, the native C language provides at most 64-bits integer support. However, in computer science, sometimes we need larger numbers. So in this problem, I want you to develop a library for big number

computation. You need to design your own structure with the following functions. You should prepare a header file called **bignum.h**. **This is very important because our TAs will prepare hw0204.c that includes bignum.h. So do not use different names.**

```
typedef struct _sBigNum sBigNum;

void print( const sBigNum num );

int32_t set( sBigNum *pNum, char *str );

int32_t compare( const sBigNum num01, const sBigNum num02 );

int32_t digits( const sBigNum num );

void add( sBigNum *pResult,const sBigNum num01, const sBigNum
    num02 );

void subtract( sBigNum *pResult, const sBigNum num01, const
    sBigNum num02 );

void multiply( sBigNum *pResult, const sBigNum num01, const
    sBigNum num02 );

void divide( sBigNum *pQuotient, sBigNum *pRemainder, const
    sBigNum num01, const sBigNum num02 );

int32_t power( sBigNum *pResult, int32_t n, int32_t k );

int32_t combine( sBigNum *pResult, int32_t n, int32_t k );
```

The functions are described as follows:

- **print**: Print the number on the screen.
- **set**: Set the number with the input string.
- **compare**: If Num01 is larger than Num02, return 1; If Num01 is equal to Num02, return 0; If Num01 is less than Num02, return -1.
- **digits**: Return the decimal digit number of Num.
- **add**: Result = Num01 + Num02.
- **subtract**: Result = Num01 - Num02.
- **multiply**: Result = Num01 * Num02.
- **divide**: Num01 / Num02 = Quotient ... Remainder.
- **power**: Result = n^k.

- **combine**:

$$\text{Result} = C_k^n = C_{k-1}^{n-1} + C_k^{n-1}.$$

The hw0204.c is provided by TA. For your convenience, I give you a sample here and I also put it on my site.

```
#include <stdio.h>
#include <stdint.h>
#include "bignum.h"

int main()
{
    sBigNum a, b;

    set( &a, "123" );
    set( &b, "99" );

    print( a );
    print( b );

    if( compare( a, b ) == 1 )
    {
        printf( "Comparison Pass.\n" );
    }
    else
    {
        printf( "Comparison Fail.\n" );
    }

    if( digits( a ) == 3 )
    {
        printf( "Digits Pass.\n" );
    }
    else
    {
        printf( "Digits Fail.\n" );
    }

    sBigNum ans, q, r;

    add( &ans, a, b );
    print( ans );
    subtract( &ans, a, b );
    print( ans );
    multiply( &ans, a, b );
    print( ans );

    divide( &q, &r, a, b );
    print( q );
    print( r );

    power( &ans, 20, 10 );
```

```
47      print( ans );
48
49      combine( &ans, 20, 10 );
50      print( ans );
51
52      return 0;
53  }
```

2.5 Bit Operation (5 pts)

I want to write a program to display a 32-bit integer in the binary form. So I write the following code. However, this code has some problem.

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  int main()
5  {
6      int32_t number = 0;
7
8      scanf( "%d", & number );
9
10     int32_t bit = 1;
11     bit = bit << 31;
12
13     for( int i = 0 ; i < 32 ; i++ )
14     {
15         if( bit & number )
16             printf( "1" );
17         else
18             printf( "0" );
19         bit = bit >> 1;
20     }
21     return 0;
22 }
```

Please explain the problem of this code and show how to fix it.