```
/*****************************************************************/
/*  Assign 2 template */
/*****************************************************************/

#include <stdio.h>

typedef union float_32{
        float   floating_value_in_32_bits;
        int     arg_32;
        struct  sign_exp_mantissa{
                unsigned  mantissa:23;
                unsigned  exponent:8;
                unsigned      sign:1;
        } f_bits;
        struct single_bits{
                unsigned  b0 :1;
                unsigned  b1 :1;
                unsigned  b2 :1;
                unsigned  b3 :1;
                unsigned  b4 :1;
                unsigned  b5 :1;
                unsigned  b6 :1;
                unsigned  b7 :1;
                unsigned  b8 :1;
                unsigned  b9 :1;
                unsigned  b10:1;
                unsigned  b11:1;
                unsigned  b12:1;
                unsigned  b13:1;
                unsigned  b14:1;
                unsigned  b15:1;
                unsigned  b16:1;
                unsigned  b17:1;
                unsigned  b18:1;
                unsigned  b19:1;
                unsigned  b20:1;
                unsigned  b21:1;
                unsigned  b22:1;
                unsigned  b23:1;
                unsigned  b24:1;
                unsigned  b25:1;
                unsigned  b26:1;
                unsigned  b27:1;
                unsigned  b28:1;
                unsigned  b29:1;
                unsigned  b30:1;
                unsigned  b31:1;
        }bit;
} FLOAT_UN;

// A function to print out bits from a 32 bit container
// provided by the union FLOAT_UN above, and using
// a text string as a label for the bit string
// as passed in the second argument

int print_bits(FLOAT_UN float_32, char * text){
        char bit_string[43];
        int i,j;

        for(i=0; i<42; i++){
                bit_string[i] = ' ';

        }
        bit_string[42] = '\0';

bit_string[0] = float_32.bit.b31?'1':'0';
```

```
bit_string[2] = float_32.bit.b30?'1':'0';
bit_string[3] = float_32.bit.b29?'1':'0';
bit_string[4] = float_32.bit.b28?'1':'0';
bit_string[5] = float_32.bit.b27?'1':'0';

bit_string[7] = float_32.bit.b26?'1':'0';
bit_string[8] = float_32.bit.b25?'1':'0';
bit_string[9] = float_32.bit.b24?'1':'0';
bit_string[10] = float_32.bit.b23?'1':'0';

bit_string[12] = float_32.bit.b22?'1':'0';
bit_string[13] = float_32.bit.b21?'1':'0';
bit_string[14] = float_32.bit.b20?'1':'0';

bit_string[16] = float_32.bit.b19?'1':'0';
bit_string[17] = float_32.bit.b18?'1':'0';
bit_string[18] = float_32.bit.b17?'1':'0';
bit_string[19] = float_32.bit.b16?'1':'0';

bit_string[21] = float_32.bit.b15?'1':'0';
bit_string[22] = float_32.bit.b14?'1':'0';
bit_string[23] = float_32.bit.b13?'1':'0';
bit_string[24] = float_32.bit.b12?'1':'0';

bit_string[26] = float_32.bit.b11?'1':'0';
bit_string[27] = float_32.bit.b10?'1':'0';
bit_string[28] = float_32.bit.b9?'1':'0';
bit_string[29] = float_32.bit.b8?'1':'0';

bit_string[31] = float_32.bit.b7?'1':'0';
bit_string[32] = float_32.bit.b6?'1':'0';
bit_string[33] = float_32.bit.b5?'1':'0';
bit_string[34] = float_32.bit.b4?'1':'0';

bit_string[36] = float_32.bit.b3?'1':'0';
bit_string[37] = float_32.bit.b2?'1':'0';
bit_string[38] = float_32.bit.b1?'1':'0';
bit_string[39] = float_32.bit.b0?'1':'0';

printf("%23s  %s\n",text, bit_string);
return 0;
}

int main(int argc, char * argv[])
{

FLOAT_UN float_32_s1, float_32_s2, float_32_rslt, fun_arg;

/**local helper variables**/

float   the_hardware_result;
int     mant_s1, mant_s2, mant_res, exp_s1, exp_s2;
int     i, j, k, shift_count;

/* Request two FP numbers */

printf("please enter your first floating point number and new-line: ");
scanf("%g", &float_32_s1.floating_value_in_32_bits);

printf("please enter your second floating point number and new-line: ");
scanf("%g", &float_32_s2.floating_value_in_32_bits);

/* generate floating point hardware result */


/* Get the mantissa and exponent components */
/*   into the helper variables */
```

```
/*    Init the helper variables  */

mant_s1 = float_32_s1.f_bits.mantissa;
mant_s2 = float_32_s2.f_bits.mantissa;
exp_s1  = float_32_s1.f_bits.exponent;
exp_s2  = float_32_s2.f_bits.exponent;

/** check for normalization and slam in the **/
/** hidden bit if normalized **/

/** The rest is left to you */

}
```