

## Getting started

It is assumed that you are comfortable programming in C or C++. If this is not the case, it is recommended you make yourself comfortable as quickly as possible. This lab will be done on [flip.engr.oregonstate.edu](http://flip.engr.oregonstate.edu), in C. Please ensure you can access this server and are comfortable navigating around.

In this lab, you will be exploring the numerical formats, including both integers and floating point values. You will be implementing addition, multiplication, subtraction, and division for both floating point and integer values.

## Part 1: Implement frexp Function

The standard C library provides a collection of functions for working with the parts of a floating point value. Specifically, you will be implementing the double form of `frexp`. Please see the man pages for details of implementation. Your version of the function should work identically to the supplied version. Feel free to use the example program from the man page as a test case. <https://web.engr.oregonstate.edu/~bridgewd/public/frexp.txt>

```
#include <stdio.h>
#include <math.h>
#include <float.h>
```

```
/*
```

```
note from Ryan:
```

```
I need to cite portions of this code if I end up using it...
```

```
I don't remember where I found the first foo off the top of my head,  
but a search should be able to find it.
```

```
*/
```

```
//pexp is a pointer to the exponent
```

```
double my_frexp(double value, int *pexp){
```

```
    double r;
```

```
    *pexp = 0;
```

```
    /*
```

```

    * return value must be strictly less than 1.0 and >=0.5 .
    */
    if ((r = fabs(value)) >= 1.0)
        for (; (r >= 1.0); (*pexp)++, r /= 2.0);
    else
        for (; (r < 0.5); (*pexp)--, r *= 2.0);

    return (value < 0 ? -r : r);
}

int main (){
    double result;
    int n;

    float float_in = 1.0;
    double double_in = 1.0;

    result = my_frexp (float_in , &n);
    printf ("%f = %f * 2^%d\n", float_in , result , n);

    //compare to frexp from math.h
    result = frexp (double_in , &n);
    printf ("%f = %f * 2^%d\n", double_in , result , n);

    printf("\n");
    return 0;
}

```

## Part 2: Feature Extraction

As we discussed in class, bit patterns have no meaning until such is assigned by the programmer. As such, a given bit pattern can be an integer, a floating point value, a 4 or 8 character string (depending on the size), etc. For this part of the lab, write code to treat a given value as each of these things.

For decimal value $-1.234$	
double mantissa	11101111100111011011001000000000111011000
double sign	1
double exponent	0111111100
long value	00111011111001110110110
long sign	1
char	SagFault

```
#include <stdio.h>
```

```
#include <math.h>
```

```
/*
```

```
(from wolfram alpha for 1.234)
```

```
sign digit | 0
```

```
exponent | 01111111
```

```
significand | 00111011111001110110110
```

```
from print_bits
```

```
byte 1 = 00111111
```

```
byte 2 = 10011101
```

```
byte 3 = 11110011
```

```
byte 4 = 10110110
```

```
they match!
```

```
00111111100111011111001110110110
```

```
00111111100111011111001110110110
```

```
*/
```

```
/*
```

```
note:
```

```
okay, this is weird. The lab instructions ask for the sign bit if a value is
```

*treated as a long, but long appears to use 2's complement instead of a sign bit*

*\*/*

```
void print_bits(signed char *ch, int max){
    int i;
    ch = ch + 3;
    for (i = 0; i < max; i++){

        int output[8];

        // converts hexadecimal byte to binary bit pattern
        printf("byte_%d=", i+1, *ch);
        int j;
        for (j = 7; j >= 0; j--) {
            output[j] = (*ch >> j) & 1;
            printf("%d", output[j]);
        }
        printf("\n");
        ch--; //next byte
    }
}
```

*// prints mantissa, sign, exponent*

```
void print_mse_as_float(signed char *ch, int max){

    printf("if_this_were_a_float...\n");

    int output[8*4];
    int k = 0;
    int i;
    ch = ch + 3;
    for (i = 0; i < max; i++){
        int j;
        for (j = 7; j >= 0; j--) {
            output[k] = (*ch >> j) & 1;
            k++;
        }
        ch--; //next byte
    }

    // now we split the string
    int sign[1];
    printf("sign_bit:");
    for (k = 0; k < 1; k++){
```

```

        sign[0] = output[k];
        printf("%d", output[k]);
    }
    int exponent[8];
    printf("\nexponent_bits: ");
    for (k = 1; k < 9; k++){
        exponent[k-1] = output[k];
        printf("%d", output[k]);
    }
    int mantissa[23];
    printf("\nmantissa_bits: ");
    for (k = 9; k < 32; k++){
        mantissa[k-9] = output[k];
        printf("%d", output[k]);
    }
    printf("\n");
}

// prints value and sign
void print_vs_as_long(signed char *ch){

    printf("if this were a long...\n");

    int output[8*4];
    int k = 0;
    int i;
    ch = ch + 3;
    for (i = 0; i < 4; i++){
        int j;
        for (j = 7; j >= 0; j--) {
            output[k] = (*ch >> j) & 1;
            k++;
        }
        ch--; //next byte
    }

    // now we split the string
    int sign[1];
    printf("sign_bit: ");
    for (k = 0; k < 1; k++){
        sign[0] = output[k];
        printf("%d", output[k]);
    }
    int exponent[32];
    printf("\nvalue_bits: ");

```

```

    for (k = 1; k < 32; k++){
        exponent[k-1] = output[k];
        printf("%d", output[k]);
    }
    printf("\n");
}

//1, 11, 52
void print_mse_as_double(signed char *ch){

    printf("if _this _were _a _double ... \n");

    int output[8*8];
    int k = 0;
    int i;
    ch = ch + 3;
    for (i = 0; i < 8; i++){
        int j;
        for (j = 7; j >= 0; j--) {
            output[k] = (*ch >> j) & 1;
            k++;
        }
        ch--; //next byte
    }

    // now we split the string
    int sign[1];
    printf("sign_bit:_");
    for (k = 0; k < 1; k++){
        sign[0] = output[k];
        printf("%d", output[k]);
    }
    int exponent[11];
    printf("\nexponent_bits:_");
    for (k = 1; k < 11; k++){
        exponent[k-1] = output[k];
        printf("%d", output[k]);
    }
    int mantissa[52];
    printf("\nmantissa_bits:_");
    for (k = 11; k < 52; k++){
        mantissa[k-11] = output[k];
        printf("%d", output[k]);
    }
    printf("\n");
}

```

```

}

void print_chars(signed char *ch){

    printf("if this were 8 chars...\n");

    int output[8*8];
    int i, j, k;
    ch = ch + 3;
    for (i = 0; i < 8; i++){
        int j;
        for (j = 7; j >= 0; j--) {
            output[k] = (*ch >> j) & 1;
            k++;
        }
        ch--; //next byte
    }

    // now we split the string...
    k = 0;
    for (i = 0; i < 8; i++){
        printf("char %d:", i);
        for (j = 0; j < 8; j++){
            printf("%d", output[k]);
            k++;
        }
        printf("\n");
    }

    printf("\n");
}

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

    float f = 1.234;
    int max = sizeof(sizeof(f));
    printf("\nfloat: %f\n", f);
    unsigned char *ch; //signed or unsigned chars... i still don't know?
    ch = (unsigned char *)(&f);

    print_mse_as_float(ch, max);
    print_mse_as_double(ch);
    print_vs_as_long(ch);
    print_chars(ch);
}

```

```

int i = 1;
printf("\nint: %d\n", i);
unsigned char *ch2;
ch2 = (unsigned char *)&i;

print_mse_as_float(ch2,max);
print_mse_as_double(ch2);
print_vs_as_long(ch2);
print_chars(ch2);

long int l = -1000;
printf("\nlong: %d\n", l);
unsigned char *ch3;
ch3 = (unsigned char *)&l;

print_mse_as_float(ch3,max);
print_mse_as_double(ch3);
print_vs_as_long(ch3);
print_chars(ch3);
}

```