**Project 4: Float Analysis**

**Brandon Horner**

**March 4, 2013**

**CS 200**

**Purpose:**

The purpose of this project is to analyze the bit pattern of an inputted float using bitwise operators including masks.

**Approach:**

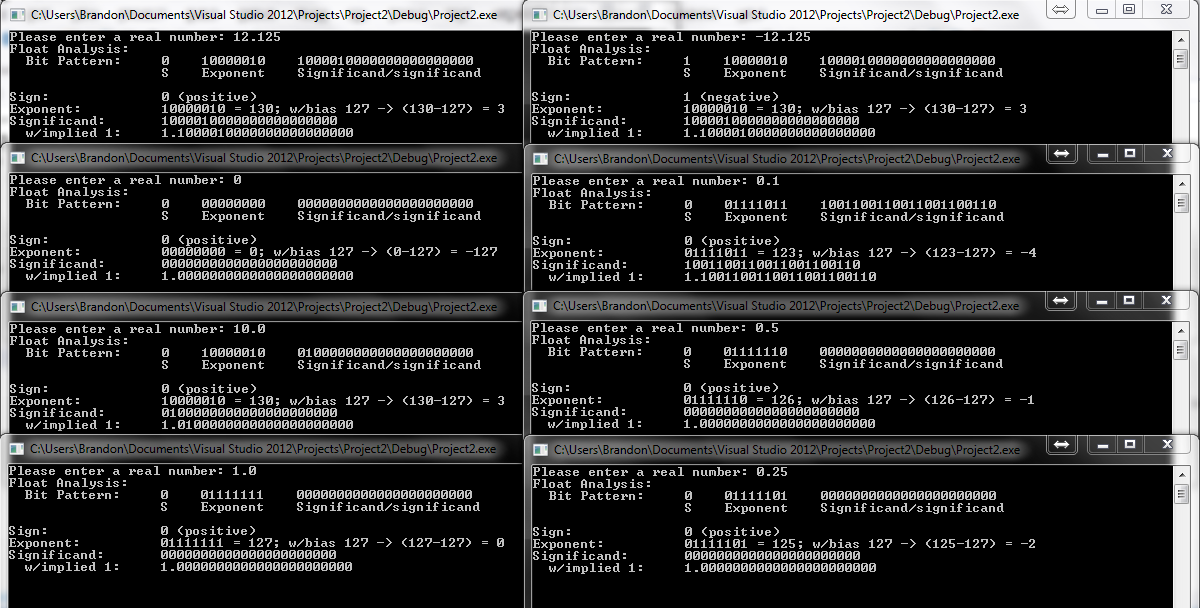
To start this project I actually looked at the sample output and broke it down into what variables I was going to need. I then first attempted to obtain the floating point number through converting the whole number of a float and the fraction number. This turned out to be a little trickier than I hoped so I turned to Google. I found out about Union in C++ and implemented it into my program with help from scattered comments on the internet.

I used bitset to handle 32-bit integers, but this presented a problem when I went to split the 32 bit unsigned number into the sign, exponent and significand. Instead of dealing with integers, I converted the 32-bit unsigned number to a string using – ‘bits.to\_string()’ and for looped over the strings to extract the desired string for my output. Using the shift also required me to change the 32-bit unsigned number to a long int, but the process of shifting and masking seemed to work smoothly.

I used the windows 7 calculator application to come up with hex mask ‘0x7FFFFFF’ for the significand. Once I came up with all the numbers, I attempted to format it as it was formatted in the assignment sheet.

**Conclusion:**

This project (at first) was challenging to convert the float to a 32-bit binary number. Attempting to do the conversion on my first couple of tries did not seem to give the right output, so turning to the internet I found union and it helped a lot. This project turned out to be good practice for C++ syntax, which I am a lot more comfortable with now.

**Sample Output:**

**Source Code (also in attachments):**

#include <iostream>

#include <bitset>

#include <string>

using namespace std;

/\* Author: Brandon Horner

This program takes input as a float, converts it to a 32-bit binary number,

and displays multiple outputs to represent features of IEEE 754 floating point

format.

\*/

int main(){

float input;

cout << "Please enter a real number: ";

cin >> input;

union{

float input; // assumes sizeof(float) == sizeof(int)

int output;

} data;

data.input = input;

bitset<sizeof(float) \* CHAR\_BIT> bits(data.output);

int bias = 127;

long int sign = (bits.to\_ulong() >> 31); // converts 'bitset<32U> bits' to 'long int bits' and shift

long int exponent = (bits.to\_ulong() >> 23) & 0xFF; // mask 11111111

long int significand = (bits.to\_ulong() & 0x7FFFFF); // mask with 23 1's padded with 0's

int bn\_sign = bits[31]; // sign is in the 31 index of bits

string bits\_string = bits.to\_string();

//Finding exponent slice of binary with string output

string bn\_exponent = "";

for (int i = 1; i < 9; i++){ // for bits 1-9, determine if char is '1' or not

if (bits\_string.at(i) == '1'){

bn\_exponent += "1";

}

else{

bn\_exponent += "0";

}

}

//Finding significand slice of binary with string output

string bn\_significand = "";

for (int i = 9; i < 31; i++){ // for bits 9-31, determine if char is '1' or not

if (bits\_string.at(i) == '1'){

bn\_significand += "1";

}

else{

bn\_significand += "0";

}

}

//Formatting multiple outputs

cout << "Float Analysis: " << endl;

cout << " Bit Pattern: " << bn\_sign << " " << bn\_exponent << " " << bn\_significand <<endl;

cout << " " << "S Exponent Significand/significand\n" << endl;

if (sign == 1){ // value 1 indicates negative number

cout << "Sign: " << bn\_sign << " (negative)" << endl;

}

else{

cout << "Sign: " << bn\_sign << " (positive)" << endl;

}

cout << "Exponent: " << bn\_exponent << " = " << exponent <<"; w/bias 127 -> (" << exponent << "-127) = " << exponent - bias << endl;

cout << "Significand: " << bn\_significand << endl;

cout << " w/implied 1: " << "1." << bn\_significand << endl;

int a;

cin >> a;

}