CS 341L Fall 2009 Lab 2

Assigned: Monday, 31 August 2009

Due: 11:59pm, Tuesday 8 September 2009 by attachment as an email to labturnin.cs341l@gmail.com

You should attach your program with your last name and the lab number in the name of the file, e.g., "knockel-lab2.S".

No late assignments will be accepted unless circumstances warrant an extension for the whole class (e.g., server is down all weekend, etc.). You are expected to do your own work as an individual effort, copying the code of others is considered cheating.

The purpose of this lab is to make you more familiar with floating point numbers than you wanted to be, particularly the pitfalls of using them so that you can avoid mistakes in your future programs. In part 1, we will explore this interactively; your credit for this will just be your attendance. For parts 2 and 3, you will write and turn in C code to further explore an alternate way to represent real numbers and the concept of machine epsilon.

Part 1:

This portion of the lab will be interactive. We will go over blocks of C code with operations on floating point numbers and discuss what they will print out and then see if our intuitions were correct.

We want you to see this in the lab, but, if you missed it or want to go through it again, this will be posted to my blog after the Thursday lab is finished.

Part 2:

Hopefully, after part 1, you are terrified enough of floating point numbers to consider that they may not always be the best choice to represent real numbers. Also, many processors do not have floating point units, requiring you to represent real numbers in other ways.

Another way to represent real numbers is to store them as fixed-point decimals using integers. For example, we could store the value of pi in an integer as 3,141,592,653. If we wanted to subtract one from pi, then we would have to write 3,141,592,653 – 1,000,000,000. Note that this fixed-point decimal scheme has 9 decimal places and requires that we multiply numbers by one billion before we do operations with them.

We will explore both the floating point and fixed-point decimal representation of real numbers in the following exercise. We will write a pi calculator that will attempt to calculate an accurate approximation of pi. First we will do it using floating point numbers. Then we will do it using the fixed decimal representation with 9 decimal places and compare the accuracy.

Here is pseudo-code for calculating pi and printing each iteration:

```
sum = 0; denom = 1;
do {
     old_sum = sum;
     sum = sum + (1 / denom) – (1 / (denom + 2));
     denom = denom + 4;
     print(4 * sum);
} while (old_sum != sum);
```

See the end of this document for C code to get you started.

Part 3:

Fixed-point decimal representations work well in part 2 because our value of pi is always between 0 and 4, a relatively small range. However, often times, we need to express values that are both very small and very large in the same operations, for which fixed-point representations do not have enough digits to accurately represent. Floating point is excellent at representing a large range of numbers.

Recall that the set of floating point numbers that a machine can represent is both discrete and finite, simply because the machine can only represent so many numbers when given so many bits to do it with. To explore how expressive floating point numbers are for numbers of different magnitudes, we will introduce the concept of machine epsilon. Machine epsilon is the absolute difference between 1.0 and the next highest number that the machine can represent after 1.0. In general, eps(x) is the absolute difference between *x* and the next highest number after *x* that the machine can represent.

Write two functions, *float epsilonf(float x)* and *double epsilon(double x)* that return eps(x) of the given number. Use the library functions *nextafterf()* and *nextafter()* to implement *epsilonf()* and *epsilon()*, respectively. Consult the man pages for *nextafterf()* and *nextafter()* for usage details.

Compilation:

gcc -Wall -Wextra -lm knockel-lab2.c -o knockel-lab2

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

static void pi_float( void )
{
    float sum = ...;
    float pi;
    float denom = ...;
    float old_sum;
    do {
        old_sum = sum;
        sum += ...;
        denom += ...;
        pi = ...;
        printf ("%.9f\n", pi);
    } while (old_sum != sum);
}

static void pi_int( void )
{
    unsigned int sum = ...;
    unsigned int pi;
    unsigned int denom = ...;
    unsigned int old_sum;
    do {
        old_sum = sum;
        sum += ...;
        denom += ...;
        pi = ...;
        printf ("%u.%.9u\n", ..., ...);
    } while (old_sum != sum);
}

static float epsilonf( float f )
{
    ...
}
```

```
static double epsilon( double d )
{
    ...
}

int main( void )
{
    puts ("Press enter for floating point..."); while (getchar () != '\n');
    pi_float ();

    puts ("Press enter for integer..."); while (getchar () != '\n');
    pi_int ();

    puts ("Press enter for epsilon stuff..."); while (getchar () != '\n');
    printf ("Single precision machine eps: %e\n", epsilonf (1.0f));
    printf ("Double precision machine eps: %e\n\n", epsilon (1.0));

    printf ("Double precision eps(1.0e6): %e\n", epsilon (1.0e6));
    printf ("Double precision eps(1.0e12): %e\n", epsilon (1.0e12));
#if 0 /* added later -- not necessary */
    printf ("Double precision eps(1.0e18): %e\n\n", epsilon (1.0e18));
    printf ("Double precision eps(1.8): %e\n", epsilon (1.8));
    printf ("Double precision eps(1.9): %e\n", epsilon (1.9));
    printf ("Double precision eps(2.0): %e\n", epsilon (2.0));
    printf ("Double precision eps(2.1): %e\n\n", epsilon (2.1));

    printf ("Double precision eps(3.8): %e\n", epsilon (3.8));
    printf ("Double precision eps(3.9): %e\n", epsilon (3.9));
    printf ("Double precision eps(4.0): %e\n", epsilon (4.0));
    printf ("Double precision eps(4.1): %e\n\n", epsilon (4.1));

    printf ("Double precision eps(7.8): %e\n", epsilon (7.8));
    printf ("Double precision eps(7.9): %e\n", epsilon (7.9));
    printf ("Double precision eps(8.0): %e\n", epsilon (8.0));
    printf ("Double precision eps(8.1): %e\n", epsilon (8.1));
#endif
    return 0;
}
```