1 CSE 251 Project 1

CSE 251, Timothy H. Brom Instructor
`http://www.cse.msu.edu/~cse251`
Project 1 is due: February 20th 2012 at 11:55pm on Handin. Name your program integral.c.
This project is an individual project. You are not to work with or discuss the project with any other students. You may ask questions of the course staff, of course. Think of the project as you would a test, you can't collaborate while taking a written test in class, just like you are not allowed to collaborate on this assignment.

1.1 Project Description

You are to write a program that requests values of a and b (as doubles) and computes a numerical estimate of the integral equation in Equation (1):

$$\int_a^b \left(\frac{\sin(\pi x)}{\pi x} \right) dx \tag{1}$$

The function we are computing the integral over is the normalized sinc function listed in Equation (2):

$$f(x) = \frac{\sin(\pi x)}{\pi x} \tag{2}$$

See the page on Wikipedia for more details on this function. It is an important function in signal processing.

You will compute an estimate of the integral using the rectangle method presented in Equation (3):

$$\int_a^b f(x)\, dx \approx \sum_{i=1}^{n} f(a + (i - 0.5)\,\Delta)\,\Delta \tag{3}$$

Where:

$$\Delta = \frac{(b-a)}{n} \tag{4}$$

You can also see the Rectangle Method Wikipedia page for more details. The method works by breaking the interval into n equal size regions of width Δ and estimating the integral over the region as a rectangle where the value

of the function at the midpoint of the region is the height and Δ is the width of the region. This is, of course, a numerical approximation. As n grows larger, the approximation gets better and the result approaches the exact value of the integral as $n \to \infty$

If I compute the estimate with $n = 12$, Ill get a slightly better estimate than if I used $n = 11$. Your program should compute the result for every value starting at 10 and increasing by 1 until a stopping condition is met. There are two possible stopping conditions:

a) Stop when n reaches 100,000.

b) Stop when the decrease in error becomes less than $1 * 10^{-10}$ You can represent $1 * 10^{-10}$ in a C program as 1e-10.

Stop when either condition is reached.

The sinc function has a *removable singularity* at $x = 0$ where it is understood that the value is the limit value of 1. So, in your program, assume:

$$f(0) = \frac{\sin(\pi 0)}{\pi 0} = 1 \tag{5}$$

This is important! Otherwise your program may fail due to a divide by zero.

If the user enters values such that $a \geq b$, present an appropriate error message and ask the user to enter both values again.

All counters should be implemented using integer variables (the *int* type). Do all math other than counters using double precision floating point (the *double* type).

1.2 Example

Suppose I enter the values: $a = -1$ and $b = 1$. The program begins with $n = 10$, which means:

$$\Delta = \frac{(b-a)}{n} = \frac{(1--1)}{10} = 0.2 \tag{6}$$

It then computes the value of:

$$\sum_{i=1}^{n} f(a + (i - 0.5)\Delta)\Delta \tag{7}$$

My example program computed a value of 1.182328209. It then computed the same equation with $n = 11$. This gave a value of 1.181744890. The decrease in the error from the first computation to the second is $1.181744890 - 1.182328209 = 5.833186x10^{-4}$. Next, it tries $n = 12$, which decreases the error by $4.430462x10^{-4}$. The following is the output of my example program for the first 12 values computed:

```
Enter a value for a: -1
Enter a value for b: 1
Integral evaluation
10:  1.182328209
11:  1.181744890 5.833186e-004
12:  1.181301844 4.430462e-004
13:  1.180957417 3.444270e-004
14:  1.180684353 2.730641e-004
15:  1.180464206 2.201472e-004
16:  1.180284129 1.800768e-004
17:  1.180134952 1.491767e-004
18:  1.180009987 1.249652e-004
19:  1.179904262 1.057250e-004
20:  1.179814020 9.024236e-005
21:  1.179736377 7.764251e-005
```

I highly recommend that you output the values as they are computed. It will make it easier to tell if the program is working. I also outputted the decrease in error from the previous iteration. The change in the error continues to decrease as the approximation approaches the result. Eventually, when $n = 1883$ in this example, the decrease in the error goes below $1 * 10^{-10}$ and we are done:

```
1874:  1.178979839 1.013800e-010
1875:  1.178979839 1.012121e-010
1876:  1.178979839 1.010594e-010
1877:  1.178979839 1.008906e-010
1878:  1.178979839 1.007312e-010
1879:  1.178979839 1.005740e-010
1880:  1.178979839 1.004128e-010
1881:  1.178979839 1.002500e-010
```

```
1882:  1.178979839 1.000922e-010
1883:  1.178979838 9.992607e-011
The integral result is 1.178979838
```

1.3 Details

In your program source code, include the following comment block before the main function in your program, filling in your name and the n and estimates for the given a, b values.

```
/*
 * CSE 251 Project 1
 * By:  <your name here>
 *
 * Complete the following table:
 *
 *     a    b     n estimate
 *    -1    1  1883 1.178979838
 *     0   10
 * -1000 1000
 *    10   15
 *   0.1  0.2
 */
```

Name your program integral.c and turn in on the handin system.

1.4 Grading Criteria

The project will be graded as follows:

- Up to ten points will be deducted for badly formatted/documented code
- Ten points will be deducted for each incorrect answer computed for the above a and b values (Watch out for $a = -1000$ and $b = 1000$ - this one is very easy to get wrong. Be sure to think about the answer you get, and whether or not it makes sense mathematically).
- Up to 30 points will be deducted for logic errors in code.

- Remaining 20 points will only be deducted if the submitted code doesn't even compile.

1.5 Suggestions and Comments

Consider adding code to stop the computation after a smaller number of steps so you can see how it starts and the initial values it computes rather than sitting through 18,000 lines of output. I added this line to make mine stop so I could see the first dozen or so results:

```
if(n > 21)
    break;
```

I, of course, removed this line later so the program could finish the computation.

You can output in a scientific notation using %e.

The approach in this assignment is not the most efficient, but has been chosen as a good exercise in the initial C programming skills we have learned.