CSE/IT 122: Homework 0

Make sure your code is commented (use doxygen style) and follows the Linux Kernel Coding style (available on Canvas). Use a Makefile to compile. Make sure your code compiles on x86_64 Linux.

Important: Follow the naming scheme for files, sample input, and sample output. Your code is tested with diff and is unforgiving if your input/output or file names are wrong. If you don't follow the requested formats you will be marked off 25% even if your program is correct.

1. Write a program that prints the numbers from 1 to 100. But for multiples of three print Fizz instead of the number and for the multiples of five print Buzz. For numbers which are multiples of both three and five print FizzBuzz.

   Sample Output

   ```
   1
   2
   Fizz
   4
   Buzz
   Fizz
   7
   8
   Fizz
   Buzz
   11
   Fizz
   13
   14
   FizzBuzz
   ....
   ```

   Name your source code file `fizzbuzz.c`.

   Use redirection to capture the output to a file named `fizzbuzz.out`

   ```
   $ ./fizzbuzz > fizzbuzz.out
   ```

   This problem comes from
   `http://www.codinghorror.com/blog/2007/02/why-cant-programmers-program.html`.
   Before you read the blog post, solve the problem first.

2. Write a program that calculates the sum of $2+5+8+\cdots+(3k-1)$ using a `for` loop. Following the sample code posted on Canvas, time the function call and count the number of times the `for` loop executes. Name your file `sum.c` and your output file `sum.out`. Use redirection to capture the output (`$ ./sum > sum.out`). Use `unsigned` integers for the input and output. Calculate the sum for $k = 1, 10, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9$. The output is number of times the for loop executed (include the last test of the for loop), k, the sum, and how long the function took to run to 24 decimal places. The output is delimited by the vertical bar.

   Sample Output

```
2|1|2|0.000000000000000000000000
11|10|155|0.000000000000000000000000
101|100|15050|0.000000000000000000000000
...
```

3. To find the sum of $2 + 5 + 8 + \cdots + (3k - 1)$, you can rewrite it as

$$2+5+8+\cdots+(3k-1) = 3(1+2+3+\cdots+k)-(1+1+1+\cdots+1) = \frac{3k(k+1)}{2}-k \quad (1)$$

   Write a program that calculates the sum of $2 + 5 + 8 + \cdots + (3k - 1)$ using equation 1.

   Name your source file `sum_no_for.c` and your output file `sum_no_for.out`. Use redirection to capture the output (`$ ./sum_no_for > sum_no_for.out`). Use `unsigned` integers for the input and output. Calculate the sum for $k = 1, 10, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9$. The output is k, the sum, and how long the function took to run to 24 decimal places. The output is delimited by the vertical bar. The output is delimited by the vertical bar.

   Sample Output

```
1|2|0.000000000000000000000000
10|155|0.000000000000000000000000
100|15050|0.000000000000000000000000
...
```

4. For the previous two questions answer the following questions: What is the largest k you can use before overflow occurs? Is k the same for both functions? Which function is preferred to calculate the sum? Why? Write your answers in a text file named `to_for_or_not_to_for.txt`

5. Find the cube root of the numbers 1 to 100 using a binary search approach. Find the accuracy (epsilon) of a cube root to 9 decimal places. Name your source file `cube_binary.c` and the output `cube_binary.out`. Use redirection to capture the output (`$ ./cube_binary > cube_binary.out`). The output is the number of times it takes to find the cube root, the number, and the cube root of the number to 10 decimal places. The delimiter is the vertical bar.

Sample Output

```
32|1|0.9999999998
30|2|1.2599210497
33|3|1.4422495704
...
```

6. Using Newton's method $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ find the square root of the numbers from 1 to 100. Find the accuracy of the square root to 9 decimal places. Name your source file `newton_sqrt.c` and the output `newton_sqrt.out`. Use redirection to capture the output (`$ ./newton_sqrt > newton_sqrt.out`). The output is the number of times it takes to find the square root, the number, and the square root of the number to 10 decimal places. The delimiter is the vertical bar.

```
5|1|1.0000000000
4|2|1.4142135624
4|3|1.7320508076
...
```

7. Using Newton's method $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ find the cube root of the numbers from 1 to 100. Find the accuracy of the cube root to 9 decimal places. Name your source file `newton_cube.c` and the output `newton_cube.out`. Use redirection to capture the output (`$ ./newton_cube > newton_cube.out`). The output is the number of times it takes to find the cube root, the number, and the cube root of the number to 10 decimal places. The delimiter is the vertical bar.

```
6|1|1.0000000000
4|2|1.2599210500
3|3|1.4422495703
...
```

Submission

Do not include binaries in your tarball.

Tar your source code files, output files, Makefile and text files into a file named: `cse122_firstname_lastname_hw0.tar.gz`,

Upload the file to Canvas before the due date.