CSE 3100 Systems Programming
Homework #2

For instructions on how to work on the assignment on Mimir, please review materials in Lab 1. Do not forget to submit your work before the deadline.

1 Counting Mondays (40 points)

In this problem, we will implement a program that counts the number of Mondays that fall on the first of the month, between January 1 of year y_1 and December 31 of year y_2. I "stole" this problem from Project Euler: https://projecteuler.net/problem=19. You're given four files: `calendar.c`, `calendar.h`, `daysInMonthCounter.c` and `firstMondaysCounter.c`. In this assignment, you will only need to modify the first two files. We'll start by modifying `calendar.h` to lay the groundwork for solving this problem.

1.1 Defining the Months

Open the `calendar.h` file with your favorite text editor. At the top, you'll notice `define` statements that define the days of the week as being between 0 and 6. This is useful, because the programmer can refer to a day by its name, instead of its number. We want to do something similar for the months. I've defined the months for `JAN`, `FEB` and `DEC`. You need to insert defines into here for the other months.

1.2 Computing Days in Month

Now, open the `calendar.c` file. You'll notice a `numDaysInMonth` function. It takes a month and a year, and computes the number of days in that month. Here are the rules:

1. January, March, May, July, August, October and December have 31 days
2. April, June, September, and November have 30 days
3. February has 29 days if the year is not a centrury and is divisble by 4, or, if it is a century, if it is divisble by 400. Otherwise, it has 28 days

You should be able to see why `switch` would be useful here. I've filled in part of the function, you need to fill in the rest. Once you're done, you can compile and test it as follows:

```
$ cc -o daysInMonthCounter calendar.c daysInMonthCounter.c
$ ./daysInMonthCounter
Enter month (1 = jan,..., 12 = dec): 2
Enter year: 2000
Number of days in month 2 of year 2000: 29
Enter month (1 = jan,..., 12 = dec): 1
Enter year: 1980
Number of days in month 1 of year 1980: 31
Enter month (1 = jan,..., 12 = dec): 0
```

Note that to exit the program, you enter `0` month. **I recommend that you test each month to make sure this works, as it is critical for our final goal**.

1.3 Counting First Mondays

Now, you need to implement the `numFirstMondays` function. This function takes three arguments:

- `startYear`: The starting year
- `endYear`: The ending year
- `janFirstWeekday`: The weekday of January 1 of `startYear`

You'll need a loop that iterates through the month of each year between the start and end years, and computes the weekday of the first of each month. Since we're given `janFirstWeekday`, it is easy to determine the weekday of the first of each month:

```
weekDay = (weekDay + numDaysInMonth(month, year)) % 7;
```

Don't forget to switch the month back to January when you go past the end of the year!
We can compile and run the program as follows:

```
$ cc -o firstMondayCounter calendar.c firstMondayCounter.c
$ ./firstMondayCounter
Enter start year: 1945
Enter day of January 1st in that year (0 = sunday, 1 = Monday, ..., 6 = saturday): 1
Enter end year: 1990
Number of mondays: 78
```

2 Coins (20 points)

Here, we will implement a program that takes a positive integer x and computes the number of ways to make change using a set of k coin types $\{c_1, \ldots, c_k\}$. The central idea is this: Given x, there are two kinds of solutions: those that use coin c_k at least once, and those that do not. The recurrence can be formulated as follows:

$$count(x, \{c_1, \ldots, c_k\}) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x < 0 \\ count(x - c_k, \{c_1, \ldots, c_k\}) & \text{if } k = 1 \\ count(x - c_k, \{c_1, \ldots, c_k\}) + count(x, \{c_1, \ldots, c_{k-1}\}) & \text{else} \end{cases} \tag{1}$$

We'll use coins with values 1, 2, 4 and 8 in this assignment. Let's call these methanols, ethanols, butanols, and octanols. You need to fill in the `count` function in `coins.c`. Then you can compile and run as follows:

```
$ cc -o coins coins.c
$ ./coins
Enter amount: 120
Number of ways to make 120 cents: 5456
```

3 Pi (20 points)

We will implement the Leibniz formula for computing π, which is defined as follows:

$$\pi = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} \tag{2}$$

As there are infinite terms, we will compute this approximately, up to $k = n$:

$$\pi_n = \sum_{k=0}^{n} \frac{(-1)^k}{2k+1} \tag{3}$$

Start by opening `pi.c`. There are two functions you need to finish: `leibnizIterative` and `leibnizRecurrent`.

3.1 Iteration

Fill in the `leibnizIterative` function, to compute π_n using a `for` loop.

3.2 Recursion

The Leibniz formula can be turned into a recurrence:

$$\pi_0 = 1$$
$$\pi_n = \frac{(-1)^n}{2n+1} + \pi_{n-1}$$

Fill in the `leibnizRecurrent` function to compute π_n.

3.3 Compiling and Testing

Compile as follows:

```
$ cc -o pi pi.c
```

Note that you need to specify the *mode* and n on the commandline. The *mode* specifies whether to use the recurrent version (*mode = 1*) or the iterative version (*mode = 2*):

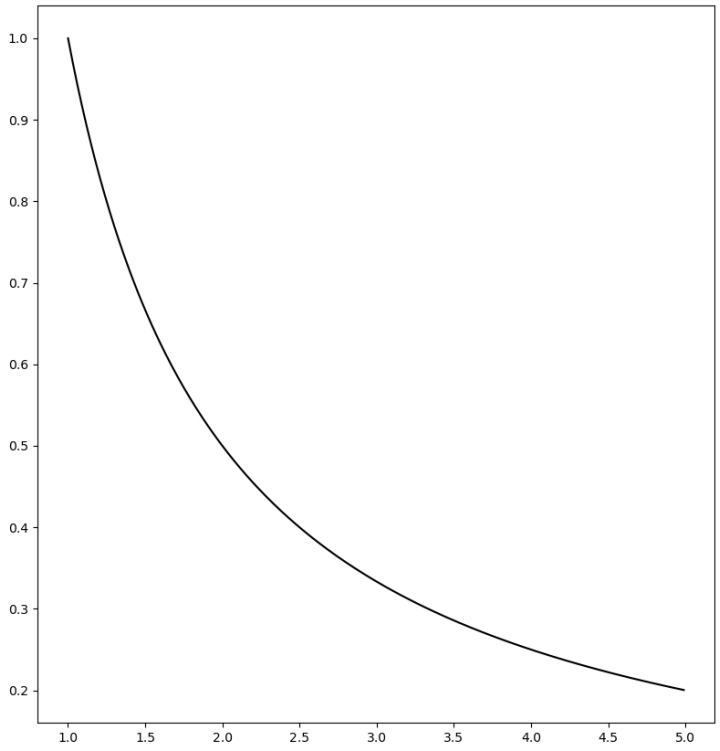
```
$ ./pi 1 200
PI: 3.146568
$ ./pi 2 200
PI: 3.146568
```

4 Log (10 points)

In the lab, we showed you one way to compute the value of $\ln a$ (the natural logarithm). Recall that the natural logarithm can be expressed as an integral:

$$\ln a = \int_1^a \frac{1}{x} dx \tag{4}$$

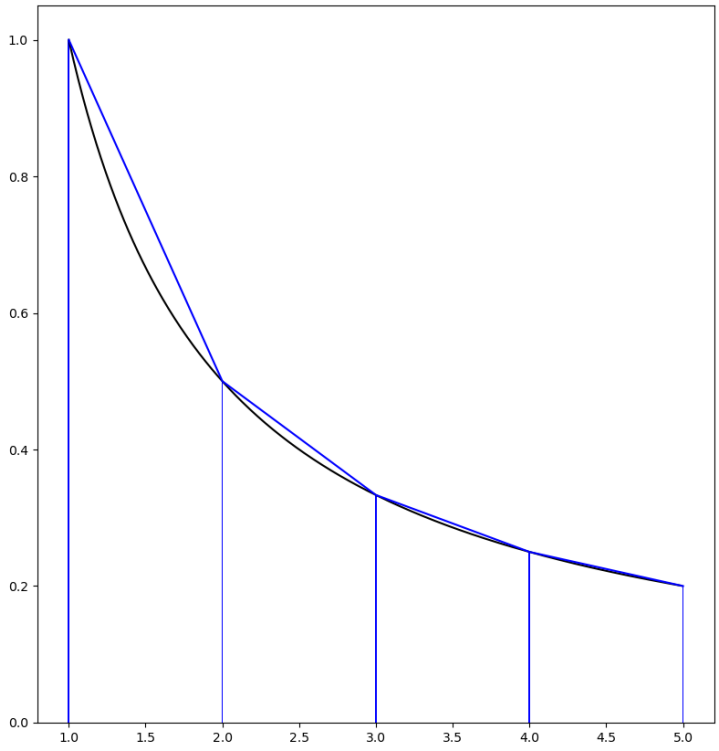
Let's start by plotting the function in the integral $\frac{1}{x}$, from $x = 1$ to $x = 5$:



One way to approximate the area under the curve is by dividing it up into a series of trapezoids. We divide the range $[1, a]$ into N partitions, each of width $\Delta x = \dfrac{a-1}{N}$:

$$[1, 1+\Delta x], [1+\Delta x, 1+2\Delta x], \cdots, [1+(N-1)\Delta x, a] \tag{5}$$

If we define $x_k = 1 + k\Delta x$, then trapezoid k has left side with x-coordinate x_{k-1}, and a right side with x-coordinate x_k. The height of these sides are $\dfrac{1}{x_{k-1}}$ and $\dfrac{1}{x_k}$. So, trapezoid k has area $\dfrac{\dfrac{1}{x_{k-1}} + \dfrac{1}{x_k}}{2}\Delta x$. Suppose $N = 4$, then we get 4 trapezoids each of width 1:



Now, we use the following formulas to compute $\ln a$ using N trapezoids:

$$\Delta x = \frac{a-1}{N}$$
$$x_k = 1 + k\Delta x$$
$$\int_1^a \frac{1}{x} dx \approx \Delta x \Big(\frac{\frac{1}{x_N} + \frac{1}{x_0}}{2} + \sum_{k=1}^{N-1} \frac{1}{x_k} \Big)$$

I've given you `log.c`, which prompts the user to enter a and N. However, you need to implement the `approxLog` function.

4.1 Compiling and Running

You can compile and run this program as follows:

```
$ cc -o log log.c
$ ./log
Enter a: 5
Enter n: 100
Natural logarithm: 1.609566
```

5 Lockers (10 points)

In a certain school, there are 1000 lockers, numbered $1, 2, \ldots, 1000$. There are also 1000 students. When the first student enters the building in the morning, she opens each locker as she walks by. When the second student enters the building, she *toggles* the state of every *second* locker. That is, if the locker's number is divisible by 2, then she changes the locker's state (open to closed, closed to open). Then the third student enters, she toggles the state of every third locker. This process continues, with the kth student toggling the state of all lockers whose number is evenly divisible by k.

A neat property of this process is that once l students have done their toggling, locker l is in its final state. You can see this from the fact that student $l+1$ will start by toggling locker $l+1$, and the same can be said for students $l+2$, $l+3$, and so on. Suppose we want a function $f(l,t)$ that computes the state of locker l after t students have done their toggling. If t is 1, then the locker is open (since the first student opened all of the lockers). Otherwise, suppose l is divisible by t. Then, student t toggles locker l. If l is not divisible by t, then student t does *not* toggle locker l. In either of these two cases, $f(l,t)$ is a function of the state of the locker after student $t-1$ did her toggling. This can be summarized as follows:

$$f(l,t) = \begin{cases} 1 & \text{if } t = 1 \\ !f(l,t-1) & \text{if } l \text{ is divisible by } t \\ f(l,t-1) & \text{else} \end{cases} \tag{6}$$

We've provided you with the file `lockers.c`. Your job is to implement the `lockerState` function, which computes the state of locker l after t students have done their toggling (i.e. is the C implementation of function f).

5.1 Compiling and Running

You can compile and run the program as follows:

```
$ cc -o lockers lockers.c
$ ./lockers
Enter locker number: 1
Open
Enter locker number: 2
Closed
Enter locker number: 3
Closed
Enter locker number: 4
Open
Enter locker number: 5
Closed
Enter locker number: -1
```