Instructions: Follow instructions carefully, failure to do so may result in points being deducted. Hand in all your source code files through webhandin and make sure your programs compile and run by using the webgrader interface. You can grade yourself and re-handin as many times as you wish up until the due date. Print a hardcopy of the rubric for this assignment and hand it in by the due date.

Partner Policy: You may work in pairs for this assignment if you chose. If you do work in any groups or pairs, you must follow these guidelines:

1. You must work on all problems together. You may not simply partition the work between you.
2. You should not discuss problem details with other groups or individuals beyond general questions.
3. Hand in only one hard copy (and one soft copy) under the first author’s name/cse login. Be sure to include both names.

Naming Instructions

- For problems 1 and 3, implement the programs in Java and place your source code in classes/files named Refraction.java and Bearing.java respectively. Place all classes in the default package.
- For problems 2 and 4 implement the programs in C and place your source code in files named rateOfReturn.c and cellPlan.c respectively.

For all of your programs, do some rudimentary input validation and exit the program on any erroneous input.

Programs

1. When light travels from one medium to another it may get slowed down. If the light is traveling at an angle, it may get refracted. That is, the angle at which the light enters the medium may get bent to a more extreme angle. This is depicted in Figure 1.

In this figure, light is depicted as moving from one medium with a refractive index of \( n_1 \) into a different medium at an angle of incidence \( \theta_1 \). The new medium has a (potentially different) refractive index of \( n_2 \) and thus the new angle of refraction is \( \theta_2 \).

This is all ruled by Snell’s Law which states that

\[
  n_1 \sin \theta_1 = n_2 \sin \theta_2
\]
Refractive indexes of materials are computed as a ratio of the speed of light in a vacuum and the phaser velocity of light in the medium \((n = \frac{c}{v})\). For example, the index of refraction of water is typically 1.333 which means that light travels 1.333 times faster in a vacuum than in water. Air (at one atmosphere) essentially has an index refraction of 1 (essentially the same as a vacuum).

Write a program that computes the index of refraction, \(\theta_2\) given the angle of incidence, \(\theta_1\) and the index of refraction of the medium, \(n_2\). We’ll assume that the first medium is always air and so \(n_1 = 1\).

Your program will prompt the user for \(\theta_1\) (in degrees) and \(n_2\) and do basic error checking, rejecting any invalid values. It will then compute the index of refraction using Snell’s Law. Your output may look something like the following.

```
Angle of incidence: 45 degrees
Index of refraction: 1.333

Angle of refraction is thus: 32.04 degrees
```

2. The rates of return on an investment are usually expressed in terms of an annualized return rate (or Compound Annual Growth Rate). Suppose that an investment has an initial value of \(v_0\). After a certain number of years, \(t\), suppose that the investment has a value of \(v_1\). To compute the annualized return rate, you can use the formula

\[
\left( \frac{v_1}{v_0} \right)^{\frac{1}{t}} - 1
\]

For example, suppose that an investment is made with an initial value of $200. Then 5 years, 7 weeks and 3 days later (that is, \(\approx 5.14256\) years later, ignore leap year days for this exercise), it has a value of $275. That is, the investment has grown by \(\frac{275-200}{200} = 37.5\%\), but that was over more than 5 years. Annually, the rate of growth would be:

\[
\left( \frac{275}{200} \right)^{\frac{1}{5.14256}} - 1 = 6.388\%
\]
Write a program that prompts the user for the initial and final value of the investment as well as the number of years, weeks, and days between the initial and final values and then computes and outputs the annualized rate of return. Your output may look something like the following.

<table>
<thead>
<tr>
<th>Initial Value: $200.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final Value: $275.00</td>
</tr>
<tr>
<td>Years: 5.14256</td>
</tr>
<tr>
<td>Annualized Rate of Return: 6.388%</td>
</tr>
</tbody>
</table>

3. A *bearing* can be measured in degrees on the scale of $[0, 360)$ with $0^\circ$ being due north, $90^\circ$ due east, etc. The (initial) directional bearing from location $A$ to location $B$ can be computed using the following formula.

$$\theta = \text{atan2}\left( \sin(\Delta) \cdot \cos(\varphi_2), \cos(\varphi_1) \cdot \sin(\varphi_2) - \sin(\varphi_1) \cdot \cos(\varphi_2) \cos(\Delta) \right)$$

Where

- $\varphi_1$ is the latitude of location $A$
- $\varphi_2$ is the latitude of location $B$
- $\Delta$ is the difference between location $B$'s longitude and location $A$'s longitude
- $\text{atan2}$ is the two-argument arctangent function

Note: the formula above assumes that latitude and longitude are measured in radians $r$, $-\pi < r < \pi$. To convert from degrees $d$ ($-180 < d < 180$) to radians $r$, you can use the simple formula:

$$r = \frac{d}{180} \pi$$

Write a program to prompt a user for a latitude/longitude of two locations (an origin and a destination) and computes the directional bearing (in degrees) from the origin to the destination. For example, if the user enters: $40.8206$, $-96.7056$ ($40.8206^\circ$ N, $96.7056^\circ$ W) and $41.9483$, $-87.6556$ ($41.9483^\circ$ N, $87.6556^\circ$ W), your program should output something like the following.

| From $(40.8206, -96.7056)$ to $(41.9483, -87.6556)$: | bearing 77.594671 degrees |

4. Write an app to help people track their cell phone usage. Cell phone plans for this particular company give you a certain number of minutes every 30 days which must be used or they are lost (no rollover). We want to track the average number of minutes used per day and inform the user if they are using too many minutes or can afford to use more.

Write a program that prompts the user to enter the following pieces of data:
- Number of minutes in the plan per 30 day period, \( m \)
- The current day in the 30 day period, \( d \)
- The total number of minutes used so far \( u \)

The program should then compute whether the user is over, under, or right on the average daily usage under the plan. It should also inform them of how many minutes are left and how many, on average, they can use per day for the rest of the month. Of course, if they’ve run out of minutes, it should inform them of that too.

For example, if the user enters \( m = 250, \ d = 10, \) and \( u = 150 \), your program should print out something similar to the following.

```
10 days used, 20 days remaining
Average daily use: 15 min/day

You are EXCEEDING your average daily use (8.33 min/day),
continuing this high usage, you'll exceed your minute plan by
200 minutes.

To stay below your minute plan, use no more than 5 min/day.
```

Of course, if the user is under their average daily use, a different message should be presented. You are allowed/encouraged to compute any other stats for the user that you feel would be useful.

5. In this exercise you will get exposure to using git, in particular GitHub. Before you begin, you will need to create an account on GitHub and read the course tutorial on getting started with git: [http://cse.unl.edu/~cbourke/gitTutorial.pdf](http://cse.unl.edu/~cbourke/gitTutorial.pdf). Alternatively, you can find your own resources (and share them on Piazza). Tip: when you sign up on GitHub use a .edu email address and go through the process to get a free student account so that you can create private repositories. This is not necessary for this assignment, but you may find it useful later because you are highly encouraged to start using git/GitHub (or something similar) for all of your future assignments but be sure to commit code to a private repository so that you do not violate the department’s academic integrity policy. For these exercises, however, keep your repositories public so that we can grade you (you are being graded on the process, not the code itself).

Even if you choose to work with a partner on this assignment, you will both need to do the following exercises. You can do them together/side-by-side and help each other, but you must both have your own versions committed to your own GitHub account.

(a) For the first part, you will fork\(^1\) then clone\(^2\) an existing repository and make changes to it. Gomer has written a solution to his CS1 course’s first assignment which

---

\(^1\)This creates your own personal copy on the GitHub servers
\(^2\)This brings a copy to your local machine; be sure to clone your repository not mine.
involves computing whether or not a given integer is a Kaprekar Number (details can be found in his documentation). However, he has committed his code with a lot of errors, both syntax errors and logic errors. Fortunately Gomer is actually pretty good at writing unit tests to ensure his code actually works. He has written both Java and C versions of his program and has provided several ways of testing his code using both ad-hoc methods (there are demo programs, ad-hoc testing programs) as well as a more formal unit testing framework (JUnit for Java; cmocka for C). You should consider learning more about these and adopting them for future assignments. The project is available at:

https://github.com/cbourke/KaprekarProject

Fork this project so that you have your own copy. Then clone it to your local development environment so that you can work with it. Then:

1. Fix all the syntax errors so that you get a working executable, commit and push your results.
2. Run the test suites and fix all the logic errors so that all the test cases execute fully. For each change/fix, make an independent commit/push with specific commit comments so that a complete history will be available of all the bugs.

Be sure to commit each change/fix that you make independently so that you have your changes documented. The instructions for building and running the project are available in the project’s readme.md file.

(b) For the second part, you will create a basic repository from scratch, push it to GitHub and then collaborate with a partner to share changes (even if you choose to work alone on this assignment, you will need to team up with at least one other person for this exercise).

To keep things simple, your project will be a simple “Hello World” style program but instead of printing “Hello World”, it will print your name. Once you have created this simple project (Java and/or C, your choice), push it to GitHub. Then have your partner clone it and add their own name to the program (so that it prints both of your names). Then either a) add them as a collaborator so they can push their changes or have them make a pull request and pull their changes to your repository (or do it both ways to get more practice).

Hand-in artifact: Instead of handing your programs in, we’ll want to verify that you’ve followed the process by checking your repository and its commit history. Thus, hand in a text file named readme.md with the URLs of both of your repositories. If you work with a partner, your file should contain URLs for all four repositories.

Additional Resources

- JUnit Tutorial: https://www.tutorialspoint.com/junit/
• https://cmocka.org - Cmocka