CSE 232 Spring 2018

Programming Project #1

Assignment Overview

This project focuses on some mathematical manipulation. It is worth 5 points (0.5% of your overall grade). It is due Monday, September 10th before midnight.

The Problem

Global warming discussions are a hot topic these days. A paper in early August 2018 called "Trajectories of the Earth System in the Anthropocene" predicted that if the global temperature increase by 7 degrees Fahrenheit, the change would be irreversible. We are going to explore the change in global temperature versus year based on a simple linear model and make some predictions as to when this might occur.

Some Background

Hopefully you remember that a linear model is created by specifying a slope and an intercept where you can use the equation y = slope * x + intercept (y is the temp, x is the year). I gathered (see spreadsheet) the average global temp in degrees Fahrenheit, 1880-2012, and fit a linear model to it. The slope is 0.01173 degreesF/year and the intercept is 34.3491 degreesF. Using this information, you will write a program that does as follows.

Program Specifications

Your program will do the following:

1. Take as input two values:
    a. a year, as a `long`
    b. a slope, as a `double`
2. Print three results:
    a. Print the **temperature** for the year read in above based on the slope and intercept provided in the Background section. Print it as a **double** of **precision 2**.
    b. Print the **year** when, given the temperature calculated in ***a.*** above, a temperature 7 degrees greater will occur again using the slope and intercept provided in the Background. Print it as a **rounded long**.
    c. Print the **year** when, given the temperature calculated in ***a.*** above, a temperature 7 degrees greater will occur using the second input, a new input **slope,** and the intercept provided in the Background. Print it as a **rounded long**.

Deliverables

proj01/proj01.cpp . The name of the directory is proj01, the name of the file you turn in should be exactly proj01.cpp in a directory named proj01. This is how all projects will be turned in: in a directory containing a file(s). To help with this, we will check that this is true with the first test case in Mimir. Just like the lab, you must click on proj01 within "Project 01 – model of global warming" to submit the file. **Not the file** proj01.cpp, **the directory** proj01 in which the file proj01.cpp is contained.

Assignment Notes:

1. We might as well try to make the output somewhat readable. You can look this up in the text or on the internet, but you can use the following modifiers that affect how numbers print.
    a. `cout << fixed;`
    Elements will be printed as floating point numbers (ex 123.456). **Use this for the project.**
    b. `cout << scientific;`
    Elements will be printed in scientific notation (ex 1.23456×10^2). This is an alternative but ***not what we want*** for this project.
    c. `cout << setprecision(2);` Floating point numbers will have 2 values after the decimal point and will be rounded, (123.46). To make this work we need another include, and that is
    `#include<iomanip>`
    Provide the `include` and use `setprecision(2)` for this project.
    d. Thus `cout << fixed << setprecision(2) << 123.4567 << endl;` will print 123.46
2. The following statement will read two variables off of the same, space separated line. It is an example of chaining input:

```
…
long lng;
double dbl;
cin >> lng >> dbl;
…
```

You can also do it on separate lines. Your choice

```
…
cin >> lng;
cin >> dbl;
…
```

3. If you `#include<cmath>` you get access to the `std::round` function. The `std::round` function takes as an argument the number to be rounded and returns the nearest integral value (see https://en.cppreference.com/w/cpp/numeric/math/round for full details)
4. There are 4 tests provided. The first is a file-exists test to make sure you got the directory issue correct, the remaining three are input-output tests.
5. You ***do not*** have to check for bad input values. In general, we will explicitly indicate the errors we are looking for, but for now we are not checking for input errors.
6. It gets irritating to type in 2 numbers every time you test. To get around this, you can use a handy trick off the command line. You can create a file with the 2 input values (on a single line, they need to be space separated, on 2 different lines is also fine). You can **redirect** the file to your main program. With that file in the same directory as the compiled `a.out`, you can do:

```
./a.out < fileOfInput.txt
```

This will automatically feed the input to the `cin` statement and produce the output. Makes it easier to test things. An example `input.txt` is provided in the project directory.

Getting Started

1. In Mimir, I create the directory `proj01` and place in it an empty file (no contents) with the correct name, `proj01`. You can update the file contents there.
2. If you are in a CSE lab (or whatever your environment), then bring up an editor and a terminal, create a project directory proj01 in your M: directory (or your local laptop) and create proj01.cpp within that directory.
3. Write your code and compile it.
4. Prompt for some of the values and print them back out, just to check yourself.
5. Check that your calculations are right (as indicated above).
6. In Mimir, create a directory proj01 and, within that directory, an empty proj01.cpp file.
7. However you choose to do your development, the easiest way at this point to check yourself on Mimir is to copy and paste from your editor to the proj01 file on Mimir and submit.
    a. You don't have to pass all the tests the first time! You can add more information and pass more of the tests as you progress. Do things incrementally.
8. Now you enter a cycle of edit-run to incrementally develop your program.
9. If a version you submit and test on Mimir passes all the tests, you are done. If time runs out and you didn't pass all the tests, then however many you passed indicates what grade you got for the project. Though it doesn't matter much now as this is relatively easy, at the end you do the best you can and pass as many tests as possible. However, ***you always know*** how you are doing and that is the advantage of Mimir.
10. ***Remember***: In the end, it only matters that it compiles and runs on Mimir. If it doesn't compile there, then it doesn't compile at all (no matter what else you did on whatever environment you used). Mimir is the last word.