Lab 2: Prepare for a Moon Landing...

Due date: Friday, October 2, 11:59pm.

Assignment Preparation

Lab type. This is an **pair programming lab**. The pairs will be designated by the instructor at the beginning of the lab. Please ask the instructor if you are not clear on how to do this.

Collaboration. Students work in pairs, and it is considered cheating, if members of the team (pair) do not work together. Communication between pairs during lab time is allowed, but no direct sharing of code is allowed.

1 Purpose

To write a program requiring use of variable declarations, reading values into and writing the values stored in variables, assignment statements and conditional statements.

2 Program Description

In this project you will start implementing a version of one of the oldest known computer games: Moon Landing. In **Program 2** you will be expected to implement the entire game.

Imagine that you are piloting a one-man spaceship to the Moon. Your spaceship carries a certain amount of fuel and gives you certain *milage* for each gallon of fuel spent. Obviously you want to have some information at any moment of time about how far from the Moon you are and how much fuel you have left. The only information you have is

- Amount of fuel at the beginning of the flight
- Fuel Milage (miles/gallon)
- The velocity of your spaceship

- Distance you have to travel

Using this information you write a program that can compute how much fuel you have left and how far from the Moon you are right now (and can even give you an ETA).

There is a number of simplifying assumptions we are making here. One of them is that your spaceship travels with constant velocity. Another simplification is in the fact that we assume that the fuel milage does not depend on the velocity the ship is travelling with. We also do not take into account the loss of mass the ship is experiencing while burning fuel.

3 Program Specifications

Write a small C program which produces several lines of output, as follows (see the sample execution in Section **??**). Note that I suggest you develop this lab in two parts. The first part is mandatory and should include:

MLP1. The first output line shall contain the text `"Prototype Moon Landing Program."`

MLP2. The second output line shall contain your and your partner's full names.

MLP3. The next few lines will contain input prompts.

MLP3-1. The first prompt shall be

```
Enter the distance to the Moon (miles):
```

The program shall read in the distance to the Moon from the starting point of the space ship.

MLP3-2. The second prompt shall be

```
Enter current velocity (miles/sec):
```

The program shall read in the current velocity of the ship.

MLP3-3. The third and final prompt shall be

```
Enter the time in flight (mins):
```

The program shall read in the time in flight for which the computations shall be made.

MLP4. Distance to the Moon and current velocity are floating point numbers. Time in flight is an integer.

MLP5. Each value, as it is read from input shall be tested for conformance to the rules specified below. If the value falls outside the prescribed feasible range of values, the program shall produce an error message:

```
Input Error: Incorrect <Parameter>: <Value>
```

Here, `<Parameter>` is one of `"distance"`,, `"velocity"` and `"time in flight"`, corresponding to the parameter that was incorrectly entered. `<Value>` is the incorrectly entered value.

The feasible range of values for the parameters is defined as follows:

distance to the Moon (distance)	$0.0 \leq \text{distance} \leq 253560.00$
current velocity (velocity)	$7.0 \leq \text{velocity} \leq 385.9375$
time in flight (time)	$0 \leq \text{time} \leq \frac{\text{distance}}{\text{velocity}}$

Notes: 253560.00 miles is the distance from Earth to the Moon at the apogee, 7.0 miles/sec is the escape velocity of the Earth, 385.9375 miles/sec is the escape velocity of the Sun.

Note: Your program shall test each value as soon as it is read. If an incorrect value is given, then your program should prompt the user to input another value. If the second value is incorrect, your program should automatically assign a valid value (the maximum value in the range except for time in flight which is set to zero) and report this to the user. For example:

```
Input Error: Incorrect <Parameter>, Valid value <Value>, assigned!
```

Where `<Parameter>` and `<Value>` are as specified above.

MLP6. After all the values are read, and established to be in their respective feasible ranges, your program shall compute and output the following information:

1. The distance the spaceship has travelled in the time provided to it;
2. The remaining distance to the Moon;

For the exact format of the output, please consult Section 5

MLP7. The last line of your program's output should contain the phrase `''Good Luck''`. After this line is printed, your program shall stop.

MLP-optional Once you complete the above portion of the lab, you can then add the following prompts and reports. These additional prompts and reports will help you with Program 2. They are not required and are suggested here to assist in preparing you for Program 2. Add prompts which ask:

MLP3-4-opt. The fourth prompt shall be

```
Enter the initial amount of fuel (gallons):
```

The program shall read in the initial amount of fuel the ship has.

MLP3-5-opt. The fifth prompt shall be

```
Enter the fuel efficiency of the spaceship (miles/gallon):
```

The program shall read in the fuel efficiency of the ship.

MLP4-optional. Amount of fuel and fuel efficiency are floating point numbers.

MLP5-optional. Again test the validity of input and print the same error message but with `<Parameter>` values of `"amount of fuel"` and `"fuel efficiency"` where appropriate.

The feasible range of values for the parameters is defined as follows:

initial amount of fuel (fuel)	$0.0 \leq \text{fuel} \leq 40000.00$
fuel efficiency (efficiency)	$\frac{\text{distance}}{\text{fuel}} \leq \text{fuel} \leq 4 \times \frac{\text{distance}}{\text{fuel}}$

You should again only allow the user two chances to enter the correct values for the input and then assign the maximum range value to the variable.

MLP6-optional. After all the values are read, and established to be in their respective feasible ranges, your program shall now also compute and output the following information:

1. The amount of fuel used during the flight so far;
2. The amount of fuel left;
3. The estimated time of arrival (in hours and minutes)

General Notes

Math. You are responsible for the program design for this program. In particular, you are responsible for coming up with the correct math (physics) to compute the outputs of the program based on the inputs.

Program name. Name your program `travel.c`.

ANSI C. Your program shall be written in ANSI C. The instructor will compile your program using the following `gcc` flags:

```
gcc -ansi -Wall -pedantic -lm -o lab2 travel.c
```

Note that the new flag `-lm` makes sure to include the math library and `-o lab2` directs the compiler to name your executable **lab2**. This means that to run your program, you will need to type `lab2` instead of `a.out`. When in doubt, use the **ls** command to see which files were created from compilation.

Any program that does not compile in this fashion will be assigned a score of 0.

Style. Your code will checked for style. Your program should mostly conform to the style described at

http://users.csc.calpoly.edu/~cstaley/General/CStyle.htm

- **Header Comments.** The header comment **must be supplied** for each file submitted. The header comment must contain the following information:

- Course number.
- Instructor name.
- Your name.
- Name/Purpose of the program.
- Date.

Extra information in the header comment may be provided as well (version, extra dates, any runtime issues, etc. . .).

Any style violations are subject to an automatic 10% penalty.

Testing. Please see the comments about testing in Appendix A.

As usual the only dispensation is given to rounding errors due to floating point computations.

4 Submission Instructions

Submission.

Files to submit. You shall submit the `travel.c` file.

Submission procedure. You will be using `handin` program to submit your work. The procedure is as follows:

- `ssh` to `vogon` (`vogon.csc.calpoly.edu`).
- Students shall execute the following submission command:

```
> handin zwood csc101lab02  travel.c
```

Late submission. No late submissions are allowed

5 Sample Output

```
>lab2
Prototype Moon Landing Program
Zoe Wood and Alex Dekhtyar
Enter the distance to the Moon (miles):200000
Enter the current velocity (miles/sec):20
Enter the time in flight (mins): 120
The spaceship has travelled: 144000.00
56000.00 miles left to the Moon
Good Luck

>lab2
Prototype Moon Landing Program
Zoe Wood and Alex Dekhtyar
Enter the distance to the Moon (miles):-1
Input Error: Incorrect distance: -1.000000
```

```
Enter the distance to the Moon (miles):200000
Enter the current velocity (miles/sec):6
Input Error: Incorrect velocity: 6.000000
Enter the current velocity (miles/sec):390
Input Error: Incorrect velocity, Valid value of 385.9375 assigned!
Enter the time in flight (mins): 5
The spaceship has travelled: 115781.25
84218.75 miles left to the Moon
Good Luck
```

Appendix A. Testing

This appendix provides a brief description of testing procedures employed in this lab, as well as a general overview of testing.

General Notes. From now on testing is a mandatory activity for each of your assignments. Testing, i.e., *the process of running your program on a specific set of inputs* is done to ensure that your program is designed and implemented correctly.

Each set of inputs used in testing is called a test case.

Interactive testing. The simplest way to test your `travel.c` program is interactive mode. Simply start the program, and at each prompt pick a desired input, enter it into the program and, at the end, observe the output. Once the output produced it needs to be validated (for example against the sample run output shown above).

Batch testing. Testing a program in a batch mode involves two steps. On step 1, you create a *file that contains the test case*. Open any text editor and enter all inputs for the program (separate them with spaces). Save your file (give it an appropriate name). On step 2 you run your program using input redirection to read inputs from the file you have created, rather than from standard input (i.e., keyboard).

The < is the input redirection symbol. The syntax of an input-redirection command is

```
>  Command < file
```

Here `command` is the command/executable program to be run and `file` is the file from which the inputs for the command/program are read.

For example, the following commands run the Lab 2 programs in batch mode:

```
> lab2 < travel-test01
```

Test Files for Lab 2. The course web page contains a set of test cases for this program. Sample test cases can be downloaded directly from the **Class Examples** link as a gzipped tar file. The file name is `travel-tests.tar.gz`.

The archive should be downloaded to the directory where the respective C programs reside. A gzipped tar file is an archive that was constructed in two steps. First, a collection of files had been *merged together* into a single file. This is done using the tar command, and the resulting file is usually called a tar file and a `.tar` extension is used to designate it.

Second, the tar file was turned into a archive using Unix's (and now -Linux's) compression program `gzip`. Gzipped files gain a `.gz` extension.

To unpack the archive, follow the following two steps:

```
> gunzip travel-tests.tar.gz
> tar -xvf travel-tests.tar
```

After the first command (`gunzip`), the `travel-tests.tar.gz` will disappear from the current directory, and will be replaced by its extracted version, `travel-tests.tar`. The second command, if executed correctly will produce the following:

```
travel-tests/
travel-tests/travel-test01
travel-tests/travel-test02
travel-tests/travel-test03
travel-tests/travel-test04
travel-tests/travel-test05
```

Each file listed in the output of the `tar` command will now appear in your current directory.

Executables. To facilitate testing, and provide for you a "golden standard" by which your programs' output shall be measured, we provide an executable files, `travel-zoe` on the course web page. Download this file in your Lab 2 directory and use it with the test cases to find the correct output for each test case.

Interactive vs. batch mode. If you run your program in an interactive mode (i.e. by typing inputs from the keyboard) you will see a slightly different output than if you run the same program in the batch mode with the same data. This is because when input is redirected from the file, values read by the program are not automatically displayed (and no `<Enter>` key gets hit). Please note that **this is expected behavior**. To see what it is, run your program in interactive mode, and then in batch node with the same inputs. (note also, that **my** programs exhibit exactly the same behavior.

You are encouraged to create new test cases and verify that your program provides correct outputs for them.