CS 161 Assignment #3 – Flight Animator

Design Document due: Sunday, 2/2/2020, 11:59 p.m. (Canvas)
Design Peer Review due: Weds., 2/5/2020, 11:59 p.m. (Canvas)
Assignment due: Sunday, 2/9/2020, 11:59 p.m. (TEACH)
Demo/Grade Your Assignment by: 2/23/2020, 11:59 p.m. (In person)

Goals:

- Practice good software engineering design principles:
  - Design your solution before writing the program
  - Develop test cases before writing the program
  - Run tests and assess results to iteratively improve the program
- Use loops to perform repeated activities
- Use functions to modularize code to increase readability and reduce repeated code

Part 0. Sign Up for Your Demonstration

You are required to **meet with a TA within the two weeks following the due date** to demo. Please sign up for your slot now (yes, before you begin the assignment!) to ensure your work will be graded. You can schedule a demo with a TA on the course website (http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020) in the far right column of the bottom table labeled "Grading Hours". You can select one of your lab TAs, or a different TA.

Your goal in the demonstration is to show your TA how to use your program, explain how it works, and answer questions. Be proud of your work and ready to show it off!

Part 1. (10 pts) Design a Flight Animator

Rather than a full flight simulator, in this assignment you will design a flight *animator*. Given information about a planned airplane flight (distance, airspeed, and wind conditions in addition to the airplane's fuel burn rate, the price of fuel per gallon, and the number of passengers), your program will calculate **how long the flight will take** and **flight cost per person due to fuel burned**. The program will display an animation of the outbound flight in which (for example) 1 second of animation = 1 hour of flying time. Then, the program will do the same computation and animation for the return flight, in which the wind has the opposite effect (a headwind becomes a tailwind and vice-versa).

Here are the attributes your Flight Animator must have:

- Use **loops** (for, while, or do-while) to avoid code duplication
- Use **functions** to define re-usable modules of code (and avoid code duplication). You must have **at least 3 functions** (in addition to `main()`) to calculate flight duration, calculate flight cost per passenger, and output flight animation.
- At least one function should specify a **default value** for at least one parameter.
- Check all user inputs and **handle invalid inputs gracefully** (i.e., do something sensible, prompt the user to try again, and definitely do not crash!)
- No use of global variables
- Employ good programming style and follow the course style guidelines
- **See section 3 for additional specific requirements on the implementation (after you do the design document)**

Have fun and be creative!

Here is an example run:

```
Please provide the following flight information:
1) Flight distance (miles): 100
2) Airplane airspeed (miles per hour): 50
3) Wind speed (miles per hour): 3
4) Fuel burn rate (gallons per hour): 7
5) Fuel price (dollars per gallon): 4
6) Number of people onboard: 3
7) Seconds of animation per flight hour: 1

Your flight will take X hours and Y minutes and cost $ZZ.ZZ per person.
<animation follows>

Your return flight will take P hours and Q minutes and cost $RR.RR per
person.
<animation follows>
```

Tips:

- Speed = distance / time
- Interpret a positive windspeed as a headwind (thus slowing your plane down), while a negative windspeed would indicate a tailwind (speeding your plane up).
- Ignore groundspeed effects due to changes in altitude (assume the flight is done at a single altitude).
- "Flight cost per person" is the total amount of money spent as a function of fuel price, flight duration, and fuel burn rate, divided by the number of passengers.
- A typical 4-seat Cessna cruises at about 120 mph and burns 8 gallons per hour.
- A typical Boeing 737 cruises at about 550 mph and burns 750 gallons per hour.
- Animation:
  - Your animation should show progress from **left to right** with one unit (of your choice) printed to represent each hour of flying (**rounded up**), printed at the rate specified by the user. For example, if you choose a unit to be a single character, a 4-hour flight could show as 4 characters (e.g., **** or ==== or ….) with a character printed once per second (if the user entered "1" for "seconds of animation per flight hour") or a character once every 0.5 seconds (if the user entered "0.5"). You could also define a unit as a string such as "=>" and then show "=>=>=>=>" for 4 units.
  - The **return** flight animation should show progress from **right to left**.
  - You can change the color of your animation and add in a lot of fancy effects (bold, blink, etc.) using ANSI escape sequences in strings. For example, this will turn the cursor red:
    `cout << "\033[31m";`
    and this will reset the cursor to normal:
    `cout << "\033[0m";`
    Here is a good reference: https://stackoverflow.com/a/33206814

Your first step is to **write a Design Document** to plan your game. You can start on this right away. It does not require any programming; instead, it is your chance to plan out your program. Consult the Design and Peer Review Guidelines:
http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/cs161-design-review-guidelines.pdf
and the sample Design Document:
http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/Example_Design_Document.pdf

Your Design Document must include three sections:

1. **Describe the problem** in your own words (what does your program need to do?) and any assumptions you are making to help make the problem more concrete.
2. **Devise a plan**: design your solution and show how it will work with either (1) pseudocode (not C++) and/or (2) a flowchart diagram. You can create your flowchart using software or sketch it on paper/whiteboard, take a picture, and insert it into your Design Document. Break the program into individual pieces that each perform one part of the task and will correspond to a function in C++.
   - **You should have a separate section of pseudocode, or a flowchart, for each function**.
3. **Identify at least 8 test cases**. What kinds of input might a user try? (Think of typical inputs as well as all of the crazy and "wrong" things they might do! **The more tests you think of now, the better your program will be later.**) What do you want your program to do or output as a result? Since your program will have several places a user can provide input, you can (1) describe a setting with respect to previous prompts, (2) specify a set of user inputs as a batch, or (3) focus on a specific prompt.
   Examples of each:

Test case setting	User input	Expected result
(1) User entered an airspeed of 100 mph and is now prompted to enter the windspeed.	`-3`	Increase airplane speed by 3 mph (to 103 mph) when calculating flight duration.
(2) All inputs in second column	Flight distance = 100, airspeed = 50, windspeed = -5, fuel burn rate = 10, fuel price = $4.00	Flight duration: 1 hr 49 minutes Flight cost: $72.73
(3) Prompt: `Seconds of animation per flight hour:`	`-1`	Output `"That is not a valid choice."` and prompt the user again.

Submit your Design Document on **Canvas** (not TEACH):

https://oregonstate.instructure.com/courses/1771939/assignments/7815091

Part 2. (10 pts) Provide a Peer Review of 2 Flight Animator Designs

You can work on this part in parallel with your work on Part 3. In this part, you will provide feedback on your classmates' designs (and receive feedback on your own design to help improve your final submission). **Your peer reviews do not affect the other student's grade.**

Canvas will randomly assign you two other students' design documents to review. Please enter your review as **comments** on the Canvas page where their submission is.

Peer reviews must be written **constructively and positively**. Your goal is to identify good aspects of the design, plus ways it can be improved. Negative, harsh, or mean comments are not appropriate and will cause you to lose points.

1. (3 pts) Describe the problem:
   - Does the document list assumptions? Do you have suggestions to clarify the assumptions?
   - If any parts are unclear, write down questions that will help the student clarify their description.

2. (3 pts) Devise a plan:
   - Can you understand the pseudocode or flowchart diagram? If hand-drawn, is the handwriting legible? Does it match the planned design? Are there parts that seem incomplete or confusing?
   - Write down at least one suggestion for how to improve the design. **There is always a way to improve any design!**
3. (4 pts) Test cases:
   - Do you understand the test cases, and do they match with the design as described? If so, write "All test cases look good." If not, suggest changes or improvements to clarify the test cases.
   - Propose one new test case not included by the student (e.g., use one of their "Test case settings" but change what the player types). You do not need to write the "Expected result" for this test case because that is up to the student to decide.

When you receive your peer reviews, **read them carefully and decide if you would like to modify your design (and implementation).** Often other people will spot issues you did not anticipate!

Part 3. (105 pts) Implement Your Flight Animator

In this part, you will implement your Flight Animator in C++ following the design you have developed. **Note: It is normal (in fact, expected) that your design will evolve as you write the program and figure out new details.** The design provides a starting point, but you can deviate from it.

There are some specific requirements for your implementation **(read this carefully)**:

- Name your file `assign3_flight.cpp`.
- (10 pts) Choose appropriate data types for the information you need to store. **Just as in Assignment 1, include a justification and min/max for the chosen data type in a comment above the variable's declaration.**
- (5 pts) Calculate and output flight duration correctly.
  - Output duration in hours and minutes (no decimal places).
- (10 pts) Calculate and output flight cost per passenger correctly.
  - Output flight cost per passenger with a dollar sign ($) and two decimal places.
- (20 pts) Use **loops** correctly.
  - Animations must be output using a loop (for, while, or do-while).
- (30 pts) Use **functions** correctly.
  - Each function must have a comment header (see style guide).
  - Each function must use appropriate parameter and return data types.
  - There must be at least 3 functions (in addition to `main()`):
    - Calculate flight duration
    - Calculate flight cost per passenger
    - Output flight animation
  - No function (except `main()`) should be more than 25 lines long.
  - At least one function must specify a **default value** for at least one parameter, and this function must be called elsewhere without that parameter specified (so the default is used).
- (15 pts) Check **every** user input for validity. If a user enters an invalid input, tell them it was invalid and let them try again.

- (15 pts) Use good programming style: file header, comment blocks, **function headers**, informative variable names, appropriate indentation, lines no more than 80 characters long, blank lines between code sections, correct spelling (in output and comments), etc. http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/cs161-style-guidelines.pdf

Implementation tips:

- Break the problem down. You can start top-down by writing your main() method with "stubs" for each function call and get that working, then fill in each stub, getting each one working before moving on to the next one. Or you could start bottom-up by implementing one function at a time (based on your design) and then adding calls to your functions in `main()` when they are ready.
- You can use `sleep(int seconds)` or `usleep(int microseconds)` to sleep (pause) for a specified amount of time. You will need to `#include <unistd.h>` .
- To **round up** when converting a floating point number `f` to an integer, use `ceil(f)` and `#include <math.h>` . http://www.cplusplus.com/reference/cmath/ceil/
- You can use `flush` instead of `endl` to "flush" the output buffer and show each character the moment it is output, without going to the next line. Example:
  ```
  cout << "*" << flush;
  ```
- To control the number of decimal places when printing a floating point number, specify that you want "fixed-place" numbers with a particular precision. **This does not change the value, just how it is printed.** Example with 3 decimal places:
  ```
  float g = 2.37653439;
  cout << fixed << setprecision(3) << g << endl;
  ```
  This prints out `2.377` . You will need to `#include <iomanip>` .
  Note that this changes the behavior of `cout` for all subsequent floats as well. To reset the output stream to its original behavior, do:
  ```
  cout.setf(0, ios::fixed);
  cout.precision(6);
  ```
- Printing "`\r`" gives a carriage return (moves to the beginning of the line without going to the next line) and "`\b`" is a backspace. More ideas here: https://en.cppreference.com/w/cpp/language/escape
- Here's the link to color and other effects again: https://stackoverflow.com/a/33206814
- Use **indentation** and **good comments** to keep track of the flow of execution in your program. Use vim's auto-indent capability.
- Remember the maximum line length of 80 characters.
- Before you submit, read the peer reviews you received and ensure that you have addressed any issues they identified. This will help you improve your score on the assignment.

Part 4. (15 pts) Analyze Your Work (in a Word/text file, section "Part 4")

(A) (2 pts) As you worked to implement your Flight Animator, you probably thought of new test cases after your Design Document was already submitted. List one here. (If you didn't think of any new test cases while implementing, create one now.)

(B) (9 pts) Try each of your 9 test cases (there are 9 now) and for each one, report whether or not the behavior is as expected. If not, state whether (1) your design has evolved and now the expected behavior is different (state what the new expected behavior is) or (2) this test helped you find (and fix) a bug in your program.

- If you didn't have 8 test cases in your Design Document, make them up now to allow you to get full credit here.

(C) (2 pts) Why do you think we advise keeping functions no more than 25 lines long?

(D) (2 pts) When is it useful to create a function with a `void` return type?

Part 5. Extra Credit (write in the Word/text file, section labeled "Part 5")

- (**up to 2 pts**) Ask another person (friend, family member, roommate, coffee shop stranger) to try out your program. In "Part 5" of your file, copy/paste their full interaction (program outputs and what they typed). Did the program crash or behave in a way you did not expect? Describe anything you found surprising.
- (**up to 2 pts**) Describe one improvement you would recommend to make your program better (more robust, more entertaining, more attractive, anything).
- (**up to 2 pts**) You may have noticed that the flight times and costs are not symmetric on the outbound and return flights, if there is any wind. For example, a 5-mph headwind increases the outbound time and decreases the return time – but not by the same amount. Why is this so?

Submit Your Assignment Electronically

(A) Submit your C++ program (.cpp file, not your executable) and your Word/text file (**must be converted to a PDF file**) with written answers (Part 4 and optionally Part 5) before the assignment due date, using **TEACH**.

(B) Remember to sign up with a TA on the course website to demo your assignment. **The deadline to demo this assignment is 02/09/2020.**