CS 161 Assignment #3 – The Race is On

Design Document due: Monday, 4/27/2020, 8 p.m. (Canvas)
Program and Written Answers due: Monday, 5/4/2020, 8 p.m. (Canvas)
Demonstrate by: Wednesday, 5/13/2020, 8 p.m. (Zoom appointment)

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 arrays to store multiple related variables
- Use functions to modularize code to increase readability and reduce repeated code

Part 0. Sign Up for Your Demonstration

Please sign up for your demonstration now (yes, before you begin the assignment!) to ensure your work will be graded. You can schedule a demo with a TA on Canvas (https://canvas.oregonstate.edu/courses/1770357/pages/grading-slash-demo-signup). You can select one of your lab TAs, or a different TA. Use your OSU email address to sign up.

Tip: Pick an early slot before they are all taken! The earlier your demo, the sooner you get feedback, which will help you do even better on assignment 3. ☺

Brief reminders (see Syllabus and Assignment 1 for more details):

- **Demo After May 13:** 30-point deduction. (Also, we cannot guarantee a late grading slot.)
- **Demo Late Assignments:** Must be done by May 13, even if the submission was late.
- **Cancel/Reschedule Demo**: At least 24 hours in advance, or it will be a missed demo.
- **Missing a Demo:** 10-point deduction for each demo missed.
- **No demo**: If you do not demonstrate your assignment, you will receive a 0.
- **Submissions that do not compile on the ENGR servers will receive a 0 for the implementation part**. Please test your code before submitting.

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 Race

In this assignment, you will simulate a race. You get to decide what kind of a race it is: a race between cars? Olympic athletes? Characters from Dungeons and Dragons? Celebrities? Dogs? Avalanches? Anything with a measurable speed can work.

Your program will define the names for 10 contestants, then ask the user **what the minimum and maximum speeds are** and **how long the race will be (a distance)**. The program will generate a random speed for each contestant (within the specified range) and then report how long each one takes to complete the race. The program will **report the winner** and any prize you want to give as an award!

Here is example output from my program (user input is highlighted):

```
Welcome to the Alizarin Derby, where your favorite pilots come to compete!
What is the minimum airplane speed in mph (1-500)? 600
 Please try again; enter a number between 1 and 500: 100
What is the maximum airplane speed in mph (1-500)? 50
 Please try again; enter a number between 100 and 500: 150
How long is the race course, in miles (1-300)? 100

Plane 0 (Memphis Belle): speed 128, finishes in 46.875 minutes
Plane 1 (SpaceShipOne): speed 117, finishes in 51.2821 minutes
Plane 2 (Enola Gay): speed 129, finishes in 46.5116 minutes
Plane 3 (Spruce Goose): speed 120, finishes in 50 minutes
Plane 4 (Bleriot XI): speed 114, finishes in 52.6316 minutes
Plane 5 (Wright Flyer): speed 120, finishes in 50 minutes
Plane 6 (Spitfire): speed 112, finishes in 53.5714 minutes
Plane 7 (Spirit of St. Louis): speed 140, finishes in 42.8571 minutes
Plane 8 (Concorde): speed 111, finishes in 54.0541 minutes
Plane 9 (Fokker): speed 100, finishes in 60 minutes

And the winner is Spirit of St. Louis!
```

Colors are not required, but you can add them for fun. I used color codes embedded in each plane's name (a string), e.g. "\e[34mWright Flyer\e[0m" (\e[34m changes the text to blue and \e[0m changes it back to normal). More options here:
https://misc.flogisoft.com/bash/tip_colors_and_formatting

Here are the attributes your program must have:

- Use **loops** (for, while, or do-while) to avoid code duplication
- Use **functions** to define modules of code (and avoid code duplication). You must have **at least 3 functions** (in addition to `main()`). It is fine to have more.
- Use **arrays** to store multiple related variables, such as the contestant names, their speeds, and their finishing times. **Initialize** all array values to 0.
- Use a **constant integer** to define the number of contestants (i.e., 10) and use this to define **the size of every relevant array**.
- **Report the winner** at the end!
- Check all user inputs and let the user try again if the input is invalid. You can assume that the user will enter an integer, but you will need to check whether it is in the valid range.
- Employ good programming style and follow the course style guidelines
- **See part 2 for additional specific requirements on the implementation (applies after you do the design document)**

Have fun and be creative!

Your first step is to **write a Design Document**. 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 Guidelines:
https://canvas.oregonstate.edu/courses/1770357/pages/design-document-guidelines

Your Design Document must include three sections:

1. **Understanding the problem:** Describe the **goal** in your own words (what does your program need to do?) and any **assumptions** you are making to help make the problem more concrete. For this assignment, you should explain **the type of race and contestants, what names you will use, range of valid speeds you will allow, and**

maximum race length you will allow. You can use units that make sense for your race (does not have to be in miles per hour).

2. **Devise a plan**: Design your solution and show how it will work with a **flowchart** diagram. You can create your flowchart using software or sketch it on paper/whiteboard, scan it in (preferred) or 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 mini-flowchart for each function** (can be on the same diagram). See this example design document with functions: https://canvas.oregonstate.edu/courses/1770357/files/79399713

   Also include your **strategy** for how you will approach and complete the implementation, including your estimate of the time it will take.
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? (*E.g., what do you want to happen if the user specifies a max speed that is less than the min speed?)* Each test case must have the **setting, user input, and expected result**.

Submit your Design Document on **Canvas**:
https://canvas.oregonstate.edu/courses/1770357/assignments/7847905

Part 2. **(85 pts)** Implement Your Racing Program

In this part, you will implement your Racing Program 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. Full grading details are provided in the assignment rubric (on Canvas).

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

- Name your file `assign3_race.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) Use srand() and rand() to generate random speeds for each contestant.
- (10 pts) Use **one-dimensional arrays** to store the list of names to be used for the contestants, their randomly generated speeds, and the time they take to finish the race. These arrays must use the same **constant** variable to specify their size.
- (5 pts) Calculate the finish time for each contestant correctly. (speed = distance / time)
- (15 pts) Use **loops** correctly to iterate over the contestants (and elsewhere as needed).
- (15 pts) Use **functions** correctly to reduce code duplication and length.
  - 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()`)
  - No function (including `main()`) should be more than 20 lines long, excluding comments and blank lines.
- (10 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.

Refer to the style guidelines:
https://canvas.oregonstate.edu/courses/1770357/files/79125719/

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. A "stub" is a function that does nothing but return a default value. Once the steps are working (for loops, etc.), 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.
- Do not use global variables (-5 pts) or goto (-5 pts). They are not needed, and they tend to create bugs that will take extra time for you to correct.
- Use **indentation** (2-3 spaces) and **good comments** to keep track of the flow of execution in your program. Use vim's auto-indent capability (also, =G in command mode will indent from the current point to the end of the file).
- Remember the maximum line width of 80 characters.

Part 3. **(15 pts)** Analyze Your Work (in a separate file)

(A) (1 pt) Write down the demonstration timeslot you signed up for. Include the name of the TA, the date, the time, and the Zoom link.

(B) (8 pts) Try each of your 8 test cases 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) How much time did this program take you to finish? How does it compare to the time you estimated in your design document?

(D) (2 pts) Why do you think we advise keeping functions no more than 20 lines long?

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

Part 4. Extra Credit (in your separate file)

- (**up to 2 pts**) In addition to identifying the winner, calculate and output the average contestant speed. This should be implemented in an additional function you create.
- (**up to 2 pts**) Describe one improvement you would recommend to make your program better (more robust, more entertaining, more attractive, anything).

Submit Your Assignment Electronically

(A) Convert the file containing your written answers to Part 3 and (optionally) Part 4 into PDF format.
(B) Submit your C++ program (**.cpp** file) and your **PDF file with written answers** before the assignment due date, using **Canvas**.
https://canvas.oregonstate.edu/courses/1770357/assignments/7847907
(C) Remember to sign up with a TA to demo your assignment.
The last day to demo this assignment is 5/13/2020.