CSP2151 Assignment 2: A mini-programming project

General Information:

This assignment is an extension of the Workshops you have been doing during the semester. Similar to all workshop assessments, this assessment component should be completed individually (i.e., it is not a group assignment).

Where appropriate, the standards set out in previous workshops should be followed and the code must make good use of functions, structures and logical control flow.

Tasks:

Design and implement the following tasks to manage the simulation of an Automated Crash Test System.

Your program should first load the names of up to 10 vehicles from a file named "Vehicle.txt" and load the names of up to 10 obstacles from a file named "Obstacle.txt" and store them in two arrays of structures, one for vehicles and the other for obstacles. The minimum requirement for the data structures representing a vehicle and an obstacle can be found from Section (1).

Next, the program should display a Menu that allows the user to navigate through and execute the following options:

- Search a vehicle
- Search an obstacle
- Input/update vehicle information
- Input/update obstacle information
- Simulate a vehicle crash against an obstacle
- Display simulation results
- Display all results
- Save all results
- Save vehicle/obstacle information
- Exit

The "*Search a vehicle*" feature should allow the user to search for information of a vehicle by its Vehicle_Name. Information stored in the related entry/entries in the vehicle array will be displayed. If the vehicle is not found, an error message should be displayed. The "*Search an obstacle*" feature is similar, but it searches information of an obstacle instead.

The "*Input/update vehicle information*" feature allows the user to input information of a vehicle. If the input Vehicle_Number matches an existing vehicle, the information entered is used to

update the information of the vehicle stored in the array of vehicles. Otherwise a new entry is created in the array of vehicles and information is stored there. The "*Input/update obstacle information"* feature is similar, but it inputs/updates information of an obstacle instead.

The "*Simulate a vehicle crash against an obstacle*" feature should allow the user to select a vehicle and an obstacle (from the respective arrays), and then simulate a vehicle crash scenario. Detailed simulation task is described in "Vehicle crash test simulation" section.

The "*Display simulation results*" feature should allow the user to choose a vehicle (e.g., by its vehicle name) and display all test simulation results against some or all obstacles. If the vehicle has not yet been tested (i.e., no simulation was recorded), the program should inform the user that they must test that vehicle before displaying any results of the vehicle/s. If multiple vehicles have the same name, all vehicles' test results with that name should be displayed.

The "*Display all results*" feature should display an error message if no vehicle has been tested. Otherwise it displays the results of all vehicles that have been tested.

The "*Save vehicle/obstacle information*" feature should extract all vehicle names from the vehicle array and save them back to "Vehicle.txt" file. It also extracts all obstacle names from the obstacle array and saves them back to "Obstacle.txt" file for possible future use/test. For simplicity, only up to 10 (vehicle/obstacles) names are required to save.

The "*Save all results*" feature should display an error message if no vehicle has been tested. Otherwise it creates a file, named "results.txt", and save the names and test results for all vehicles that have been tested, to the file.

Menu design/implementation requirement:

> Create one function for each menu item (i.e., to implement its feature/s). If any of these functions was not completed as required in the final submission, use a *dummy* function that prints the following statement only:
>
> "<menu item name> feature has not been completed yet."
>
> For example, if the "Display all results" feature was not completed at submission, the function should print the following statement only:
>
> "Display all results feature has not been completed yet."

(1) Data structures for Vehicles and Obstacles

Referencing the Vehicle structure defined in your Workshop 3, define a new structure to represent a vehicle for the crash simulation purpose. Your Vehicle structure should include at least 6 fields, including Vehicle_number, Vehicle_name, Manufactur_date, Top_speed, Mass, Number_of_seats. You may add any extra field/s as you need.

The Obstacle structure should include at least 3 fields, Obstacle_number, Obstacle_*Name*, Stopping_distance. You may add any extra field/s as you need.

(2) *Vehicle crash test simulation*

This task simulates a vehicle crash scenario, completing calculations on various data items, and displaying test simulation result.

Vehicle crash test simulation can be done *under random conditions* (i.e., conditions are randomly generated automatically within a reasonable value range) or *by manually entering a set of test*

conditions. - Due to time and workload constraints, we only perform the latter case in this assignment (that is, all test conditions are to be entered manually under prompt).

Please note:

Vehicle crash test simulation can be much more complicated in a real world scenario, as there would be many more variables to take into account when designing such a system. We will only design and implement a *simplified version* of such a system in this assignment.

At the start, the function/program prompts the user to choose one vehicle from the available vehicles and one obstacle from the available obstacles. It then prompts user to enter a variety of test conditions (including the velocity, or alternatively named *impact_velocity*, of the vehicle) for the crash test simulation.

The test conditions are set as follows and must comply with the listed restrictions:

- Vehicle_age
  - Measured in years
  - Must not be greater than the current date minus the Vehicle manufacture date or less than zero
- Impact_velocity
  - Display/entered in kmph
  - Use m/s in calculations
  - Must not be greater than the Vehicle's top speed or less than zero
- Number_of_passengers
  - Must not be greater than the number of seats of the Vehicle or less than zero
- Number_of_seatbelt_wearing_passengers
  - Must not be greater than the number of passengers or less than zero

If the user enters an invalid value for any test condition, the program should print an error message, together with the data entered, and continue to prompt user again until a valid value is entered.

Based on the above information, the simulation calculates various data items, including Safety_rate and Damage to the vehicle participating the simulation, and Survival_rate of passengers involved in the crash. It then displays (and optionally saves) these data items as the simulation results.

Calculation of Safety-rate

Each test condition confers a bonus or penalty to the overall percentage Safety_rate of the test.

The Safety_rate is calculated using the following steps/algorithm:

a) Safety_rate = 1; // initialized as a percentage value of 100%

b) Decrease Safety_rate by a percentage factor of the vehicle's age using:

$$\frac{1}{4} * (\text{Vehicle\_rge} - 2)^2$$

c) Decrease Safety_rate by F /10000.0, which is the approximate impact force F on the vehicle, where

$$F = \frac{\frac{1}{2}mv^2}{s}$$

m is the vehicle's mass, *v* the impact_velocity (in m/s) of the vehicle, and *s* the stopping distance of the obstacle.

d) If Safety_rate is below zero, set Safety_rate to 0; if Safety_rate is above 100% (or Safety_rate = 1), set Safety_rate to 100%.

Once the details of the test have been finalized, the user should be shown all condition information and the Safety_rate. The user will have an opportunity to accept the information and start the test or discard (and regenerate) all conditions.

Calculation of Damage-rate

In a crash, the damage to a vehicle depends on its Safety_rate of the vehicle in that crash. This is set as a random value within 25% * (1- Safety_rate).

Calculation of Passenger's Survival-rate

In a crash, the Survival-rate of a passenger involved in the crash depends on two factors, the Safety_rate of the vehicle, and whether or not he/she is wearing a seatbelt.

Each passenger will have, initially, a Safety_rate (percent) chance of survival. However, if any passengers are not wearing a seatbelt, all passengers are 4 times less likely to survive. Furthermore, those who are not wearing a seatbelt are 9 times less likely to survive.

Finally, display simulation result (together with test conditions) in a properly designed table. As an example, the table may look like:

Test Simulation Result	
Test ID (optional):	(?)
Vehicle information:	
Vehicle_number	?
Vehicle_name	?
Manufactur_date	?
Top_speed, Mass	?
Number_of_seats	?
Obstacle information:	
Obstacle_number	?
Obstacle_*Name*	?
Stopping_distance	?
Test conditions:	
Vehicle_age	?
mpact_velocity	?
Number_of_passengers	?
Number_of_seatbelt_wearing_passengers	?
Simulation result:	
Vehicle's Safety_Rate	?
Vehicle's Damage-rate	?
Damage-rate	?
Passenger's Survival-rate	?
Survival-rate (wearing seatbelts)	?
Survival-rate (not wearing seatbelts)	?

Offer the option of saving this output to a file (and actually save it if requested) before starting a new simulation or quitting.

Notes: to save the simulation result, you may create an array of structure for the simulation, and save all simulations results to the array. The Simulation structure may have the following fields:

- Test ID
- Vehicle ID // or VehicleName?
- Obstacle ID // or Obstacle Name?
- Vehicle_age //Test conditions
- Impact_velocity
- Number_of_passengers
- Number_of_seatbelt_wearing_passengers
- Safety_rate //Test results
- Damage-rate
- Passenger's Survival-rate

You may add extra fields as needed.

Bonus Marks:

Additional work and effort to improve the program and make it more useful can be worth bonus marks.
Bonus marks are not required and will not improve your grade above 100%
Examples of features that may be worth bonus marks:

- Advanced search features (search for all simulation/test results above a threshold etc.)
- Use multiple files effectively with appropriate use of header files and good separation of concerns.
- Graphical user interfaces, animations or colours.
- Ask your tutor if you have other ideas!

NOTE: The Workshops have been working up to this point to provide you with a base program of your own work that should give you the best possible opportunity for completing this assignment. You should look to how these Workshops have developed your work for a guide on how to structure your final assessment.

(Go to the next page)

Assignment Submission Requirement:

You are required to submit, through Blackboard, one single compressed file (in either .zip or .rar format), which contains at least three documents including

(i) the design document (in Word or PDF format);
(ii) C source code/s (in .c or .txt format);
(iii) executable code (in .exe format only);

plus any other supportive documents, if any.

Please rename the .zip (or .rar) file as

CSP2151_A2_<Your Student ID>_< Your full name>.zip. (or …. .rar)

and submit this single file only.

Marking Guide:

Criteria	*Available Marks*	*You mark*
Design	/5	
System menu and functions (except for simulation part)	/4	
Simulation & calculation (e.g., Safety rate, Damage & Survival rate), etc.	/5	
File I/O	/4	
Code Legibility (formatting, comments, etc.)	/4	
Program Structure (e.g., functions, data storage, efficiency)	/4	
Compiling	/2	
Executable and outputs	/2	
Bonus	/5	
Global variables, Breaking scope, etc.	(-/10)	
Total:	**/30**	

Rubric

	Not proficient	Low Proficiency	Developing Proficiency	Moderate Proficiency	High Proficiency	Marks Allocated
Design	0 Design not present or does not meet requirements	1 Design has logical flaws or does not meet all requirements	2-3 Design meets some requirements but not all or has significant formatting issues	4 Design meets most requirements but has some issues	5 Design meets all requirements	5
Menu items & related functions	0 Menu/functions not present	1 Two or less Menu /functions present and are poorly implemented	2 Up to four Menu/functions implemented and work properly	3 Up to six Menu/functions implemented and work properly	4 All Menu/functions fully implemented and work	4
Simulation calculation	0 Simulation calculation not present	1 Few calculations are present, but do not work	2 Simulation calculation are all present but are poorly implemented, contain some errors.	3 Simulation calculation are all present but contain small errors.	4 Simulation calculation are present and work as intended	4
File I/O	0 File I/O not present	1 File I/O not present but not functional	2 File I/O present but has some errors that prevent the program from meeting the requirements	3 File I/O present but has some errors	4 Functions are present and work as intended	4
Code Legibility	0 Code is very difficult to read	1 Code has many readability issues such as poor naming conventions or lack of indenting	2 Code has some readability issues such as poor naming conventions or lack of indenting	3 Code has one or two small readability issues such as poor naming conventions or lack of indenting	4 Code is well written and highly readable	4
Program Structure	0 Functions and/or arrays not present or program is very poorly structured	1 Few functions and/or arrays are present or program is very poorly structured	2-3 Functions and/or arrays are present but program is poorly structured	4 Functions and arrays are present but with some small issues of program structure.	5 Program is well structured	5
Compiling	0 Code does not compile		1 Code compiles with many warnings		3 Code compiles with no warnings	2
Executable and Output	0 Output is incorrectly formatted, and no executable code seen		1 Output is correct but could be formatted better, executable code is seen, with running issue		2 executable is OK, Output is correct and formatted	2