Technical Software Development Assignment 3
Bulk Analysis of Overhead Line Tripping Incident Data.

This assignment is worth 10% of the unit assessment.

Submit soft copies of the .h (if applicable) and .cpp files and report document (doc / pdf) through the assignment area of this unit on CANVAS http://www.swinburne.edu.my/canvas/

Due Date: 11:59pm, Monday 24 May 2021

Introduction:
This assignment requires knowledge of variables, file access, structure/class (optional), repetition, validation and function.

Background:
The decision tree that you have implemented in Assignment 1 enables you to predict the root cause of overhead line tripping based on five variables:

1. Evolving Fault (binary value: yes or no)
2. λ, Gradient or rate of change of the curve (real number)
3. Voltage Dip (real number)
4. Permanent Fault (binary value: yes or no)
5. Δt_f,Time interval between the last neutral current distortions and a flashover (real number)

In this assignment, you will re-use the same decision tree in Assignment 1 to predict the root cause of overhead line tripping based on the values of these five variables read from a text file for each overhead line tripping incident.

Software Development Task:

Develop and submit an original implementation of a C++ program that reads the values of five variables required by the decision tree from a text file and predict the root cause of overhead line tripping based on the decision tree in Assignment 1. The program will then write the values of five variables for each overhead line tripping incident into the corresponding root cause file based on the root cause predicted by the decision tree:

Current Transformer Explosion – output file name: ctexplod.txt
Tree Encroachment – output file name: treeencr.txt
Crane – output file name: crane.txt
Pollution – output file name: pollut.txt
Lightning Strike – output file name: lightnin.txt

A. Functional requirement:

Your program must be able to perform the following:

1. Get user's input for the name of **text** file that contains the values of the five variables for each overhead line tripping incident (e.g. "oht_data.txt" provided with this assignment. This file contains 113 rows of overhead line tripping incidents).
2. Open the text file for reading.
3. Read the values of the five variables required by the decision tree from the text file and store them. In the text file, assume that the values of the five variables for each overhead line tripping are given in a row (i.e. one line) in the sequence of Evolving Fault (yes/no), λ, Voltage Dip, Permanent Fault (yes/no), Δt_f, as illustrated in the example given in Figure 1. You do not need to validate the data read from the file (that is, assume that the data in the file are all valid).

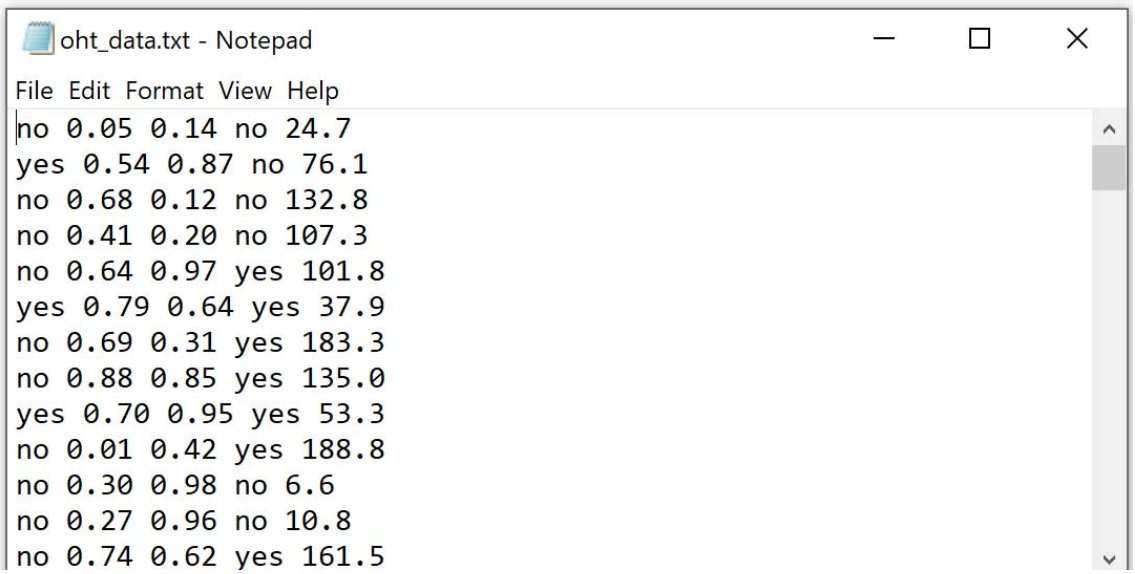

Figure 1: Data for each overhead line tripping incident are listed in one row within the text file.

4. For the five parameters for each overhead line tripping incident read from the text file
   a. use the decision tree implemented in Assignment 1 to predict the root cause of overhead line tripping.
   b. increment the corresponding root cause count by one (for example, increment current_transformer_explosion_count by 1 if the overhead line tripping incident is predicted to be caused by Current Transformer Explosion).
   c. Write the values of the five variables to the corresponding predicted root cause file listed in the Software Development Task section above.
5. Once all overhead line tripping incident data have been read from the text file and processed, display on the screen the number of overhead line tripping incidents that are predicted to have the root cause of:
   i. Current Transformer Explosion
   ii. Tree Encroachment

iii. Crane
iv. Pollution
v. Lightning Strike

Challenge functional requirement (Not mandatory - you may choose to implement some or all of the following requirements):

6. In step 2 to 5, assume that the number of rows in the input text file is not known in advance (that is, not fixed to 113 rows). Your program should be able to read text files of any number of rows.
7. Implement algorithm to determine the most common root cause (highest incident count) and the least common root cause (lowest incident count) for the overhead line tripping incident data read from the file.

B. Design Requirements:

You must demonstrate the use of modular decomposition in your program. Decompose your program into functions based on the principle that one function should be designed to perform only one task. Ideally, the main() function should only contain calls to other functions and minimal other necessary codes. At minimum, your program should contain five functions (excluding main function).

C. Good Programming Style:

1. Provide header comments that contain name, date, and program description.
2. Provide inline comments to explain any code which is not obvious to another programmer.
3. Use consistent naming conventions and indentation.

D. Testing Requirements

Use the *oht_data.txt* file to test your program. If you attempt the challenge functional requirement No. 6, you will need to test your program with another two input files provided (*oht_long.txt and oht_short.txt*). Screenshot(s) of the working program should be submitted as a report.

Submission:

Submission of the .h (if applicable) and .cpp files and report document must be made through CANVAS before the due date/time.

- The name of your report document (doc / pdf) files must be your student number.
- Do not include the Visual studio solution files or folders or exe files.

The report for program testing should contain:

- Screen shots of the working program
    - Prepare ONE screenshot for the program tested using *oht_data.txt* . If you attempt challenge functional requirement No. 6 (using *oht_long.txt and*

oht_short.txt), include two screenshot(s) for testing the challenge functional requirement.

You are REQUIRED to fill in the assignment cover sheet (Can be downloaded from CANVAS and submitted together with report).

E. Marking Scheme:

Criteria	Marks		
Correctly implemented Functional Requirements 1 to 5?	Yes: 3 marks	Partially Correct (1 to 3 mistakes): 1.5 marks	No (more than 3 mistakes): 0 mark
Correctly implemented Challenge Functional Requirements 6 and/or 7?	Yes: 1 mark for each Challenge Functional Requirement	Partially correct (1 to 2 mistakes): 0.5 mark for each Challenge Functional Requirement	No (more than 2 mistake): 0 mark
Correctly implemented Design Requirements?	Yes: 2 marks	Partially Correct (1 to 2 mistakes): 1 mark	No (more than 2 mistakes): 0 mark
Good programming styles: (1) Provide header comments that contain name, date, and program description. (2) Provide inline comments to explain your code. (3) Use consistent naming conventions and indentation.	1 mark for implementing all three good programming styles 0.5 mark for implementing two out of the three good programming styles. 0 mark for implementing less than two out of the three good programming styles		
Reporting Task	1 mark for screenshot of correct program testing using *oht_data.txt.* 1 mark for screenshot of correct program testing using *oht_short.txt and oht_long.txt (only if you attempt Challenge Requirement 6).*		