THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

2024 SCHOLARSHIP EXAMINATION

DEPARTMENT	Computer Science
COURSE TITLE	Year 13 Scholarship
TIME ALLOWED	FIVE hours with a break for lunch at the discretion of the supervisor
QUESTIONS	There are TWO questions in the paper. Candidates are to answer BOTH questions. Answer as much of each question as you can. Note that Question 2 is significantly more difficult than Question 1. Plan your time to allow a good attempt at each.
INSTRUCTIONS	Candidates may use any text or manual or online programming language documentation for reference during the examination. Candidates may not copy code from the internet or consult anyone other than the examiners during the examination
DETAILS	Both questions pose problems which you are asked to solve by writing computer programs. They may also ask for written answers for some problem parts. In programming you may work in the programming language of your choice. However, the examiners need to be able to read your program text and if at all possible, test run it. If problems arise from your choice of programming language, we may contact you after the examination, for clarification. Written answers to parts of questions can be submitted in text files; included as comments in your program text; or as photographed or scanned images of hand written documents. Remember also that partial marks may be awarded for programming ideas written down, but not yet implemented.
CALCULATORS PERMITTED	Yes

1. **Animal Wordle** (Careful and Accurate Programming)

Your programming work in this question will be assessed on three criteria:

(a) Completeness and accuracy of the program. It may be that this problem statement does not state exactly what the program should do under all circumstances. If you find a situation of that nature, choose a solution and write down, either on paper or in the comments of your program what the difficulty was and how you chose to resolve it.

(b) Good presentation. That is, it should make good use of programming language facilities, be well organised, neatly laid out, and lightly commented.

(c) Careful checking. Wherever possible check input from the program user in case they have made errors.

Wordle is a word game that was created and developed by Welsh software engineer Josh Wardle. In the original version of Wordle, players have six attempts to guess a five-letter word, with feedback given for each guess in the form of coloured tiles indicating when letters match or occupy the correct position.

In this question, you are asked to write a program that simulates a game of Animal Wordle. In Animal Wordle, players have six attempts to guess a four-letter animal, with feedback given for each guess indicating when letters match or occupy the correct position. If a character is in the correct position, it is indicated with a hash (#). If a character is correct, but in the wrong position, it is indicated with a hyphen (-). If a character is incorrect, it is represented with a full stop (.).

Your program should randomly select a four-letter animal from a pre-defined list (included below). It should then allow the user to guess the animal, one letter at a time. Each letter must be one character long, must contain alphabet characters only (i.e. a-z, A-Z), and should not be case sensitive. *Please note, your program is not expected to validate whether the user's guess is a valid word. For example, both "duck" and "abcd" would be valid guesses.* Your program should store the guess as a list and then compare each letter with the animal. Finally, your program should provide feedback to the user in the form of matching characters (#, -, and .) as outlined in the Animal Wordle description above. The program should do as follows.

- ⇨ Ask the user their name
- ⇨ Ask the user if they would like to play Animal Wordle
- ⇨ Randomly select a four-letter animal from a pre-defined list
- ⇨ Allow the user to guess the word, one letter at a time
- ⇨ Provide feedback to the user in the form of matching characters (#, -, and .)
- ⇨ Repeat until either (a) the user has run out of guesses or (b) the user has guessed the correct the word

The transcript of a sample interaction with such a program is given on the next page. In the transcript, information entered by the user is shown in **bold** type. You don't have to follow this style of data entry or format results in the same way. The sample is just here to show the kind of interaction expected of your program.

The pre-defined list of animals is as follows.
["duck", "goat", "bear", "lion", "frog", "deer", "swan", "wolf", "crab"]

QUESTION 1 CONTINUES ON THE NEXT PAGE

```
Welcome to Animal Wordle!

Wordle is a word game that was created and developed by Welsh software engineer
 Josh Wardle. In the original version of Wordle, players have six attempts to guess
 a five-letter word, with feedback given for each guess in the form of coloured
 tiles indicating when letters match or occupy the correct position.

In Animal Wordle, players have six attempts to guess a four-letter animal, with
 feedback given for each guess indicating when letters match or occupy the correct
 position. If the character is in the correct position, it is indicated with a hash
 (#). If a character is correct, but in the wrong position, it is indicated with a
 hyphen (-). If a character is incorrect, it is represented with a full stop (.).

Please enter your name: Greta

Hi Greta! Would you like to play Wordle? [y/n]: y

You have 6 guesses left.

Please enter your four-letter guess:
        Letter 1: d
        Letter 2: u
        Letter 3: c
        Letter 4: k

Result: ....
Whoops, no letters are correct! Keep guessing!

You have 5 guesses left.

Please enter your four-letter guess:
        Letter 1: l
        Letter 2: i
        Letter 3: o
        Letter 4: n

Result: ..#.
Getting there! Keep guessing!

You have 4 guesses left.

Please enter your four-letter guess:
        Letter 1: b
        Letter 2: e
        Letter 3: a
        Letter 4: r

Result: ...-
Getting there! Keep guessing!

You have 3 guesses left.

Please enter your four-letter guess:
        Letter 1: w
        Letter 2: o
        Letter 3: l
        Letter 4: f

Result: .-.-
Getting there! Keep guessing!

You have 2 guesses left.
```

QUESTION 1 CONTINUES ON THE NEXT PAGE

```
Please enter your four-letter guess:
        Letter 1: g
        Letter 2: o
        Letter 3: a
        Letter 4: t

Result: --..
Getting there! Keep guessing!

You have 1 guesses left.

Please enter your four-letter guess:
        Letter 1: f
        Letter 2: r
        Letter 3: o
        Letter 4: g

Result: ####
Woohoo! You guessed the word correctly!
```

2. **Visualising Graphs** (Problem Solving and Programming)

Your programming work in this question will be assessed on three criteria:

(a) Your approach to the problem. We will be looking at your work for evidence that you found good ways of storing the necessary data, and devised algorithms for finding and displaying the requested results. ***Please hand in any notes and diagrams which describe what you are attempting to program, even if you don't have time to code or complete it. You may include comments in your program, or write a description of your program to hand in.***

(b) The extent to which your program works and correctly solves the problem.

(c) The extent to which you use results from your programming to explore the problem presented.

You may find that the programming language you use makes it difficult to produce output as shown in the example implementation steps below. If this is the case, feel free to build your program in a way that suits your circumstances.

Note: Five text files have been provided to you: simple_graph_1.txt, simple_graph_2.txt, simple_graph_3.txt, bad_graph.txt, and graph_surprise.txt.

You are a researcher at the University of Waikato. As a researcher diving deep into a sea of data, you're on a thrilling quest to uncover hidden patterns and insights. To make this adventure even more exciting, you're tasked with creating a program that crafts custom graphs to visualize your findings! Imagine transforming numbers into stunning visuals that tell the story behind your data. With each graph you build, you'll be one step closer to unlocking the mysteries and trends waiting to be discovered. Get ready to turn raw data into eye-catching, insightful visualizations that will make your research shine!

In this question, you are asked to write a program that takes a series of x and y co-ordinates and displays them on a graph visualisation. The question presents the problem in stages for you to program. We suggest that you build your program in the order given. This will make it likely that you have parts working at the end, even if you don't have time to complete the whole program. We also strongly suggest that you read through all the stages before starting to program. Stages I and J are the final stages, in which you have the most freedom to explore algorithmic ideas.

The stages of this problem involve building and changing a program. Instructions will be given in some detail for the first stages. Later stages require that you develop the code yourself. When you are making a major change, you should save a working version of your program. This will help us see what you have achieved, especially if you have difficulties with the altered version. Where stages ask you to try different ways of displaying the output, you can write different display procedures within the same program to make sure that all of your answers are still visible to the examiner.

(continued on the next page)

Setting up the Graph

Stage A: Building the graph axes

Write a program to draw the x-axis and y-axis of a graph using text characters. The y-axis should be 9 characters high, and the x axis should be 25 characters wide. The result should be similar to the figure below (each cell in the graph is one line high and one character wide).

```
|
|
|
|
|
|
|
|
|
+-------------------------
```

Stage B: Adding a scale

The next thing you need to do add a scale to the y-axis. The centre of the scale should be 0, with positive integers increasing and decreasing vertically. Given a y-axis that is 9 items high, your scale should look as below.

```
 4|
 3|
 2|
 1|
 0|
-1|
-2|
-3|
-4|
  +-------------------------
```

Stage C: Moving the x-axis

Your x-axis should align with the 0 position. Move your x-axis to match the graph shown below. *Hint, if you haven't already, it would be useful to make the height and width of your graph dynamic. E.g. use variables to store the height and width of your graph.*

```
 4|
 3|
 2|
 1|
 0+-------------------------
-1|
-2|
-3|
-4|
```

QUESTION 2 CONTINUES ON THE NEXT PAGE

Displaying data on your graph

Stage D: Adding data with y-values

Now that you have set up your graph, you can start using it to visualise data.

Display data on your graph using the following y-values. Each data point should be represented by an "x". Your graph should be 9 items heigh, and should have a width that allows for all of the y-values (15) to be displayed.

[2,2,2,2,2,2,2,2,2,2,2,2,2,2,2]

The output below shows an example of the graph.

```
 4|
 3|
 2|xxxxxxxxxxxxxxx
 1|
 0+---------------
-1|
-2|
-3|
-4|
```

Stage E: Adding data with x-values

Extend Stage D to include an additional set of data points, based on the x-values below. Each data point should be represented by an "x". Your graph should have a height that allows for all the x-values (11) listed below to be displayed.

[4,4,4,4,4,4,4,4,4,4,4]

The output below shows an example of the graph.

```
 5|    x
 4|    x
 3|    x
 2|xxxxxxxxxxxxxxx
 1|    x
 0+----x----------
-1|    x
-2|    x
-3|    x
-4|    x
-5|    x
```

Stage F: Adding data with x and y values

Now that you can add data with x-values and y-values separately, expand your program so that it can display a data point based on both its x and y co-ordinates. A 2-D array of x and y values is included below.

[[0,0],[1,1],[2,2],[3,2],[4,1],[5,0],[6,-1],[7,-2],[8,-2],[9,-1],[10,0]]

QUESTION 2 CONTINUES ON THE NEXT PAGE

The output below shows the visualization that should be displayed when using the x and y values included above.

```
 2|  xx
 1| x  x
 0+x----x----x
-1|      x  x
-2|       xx
```

Stage G: Reading a graph from file

Extend your program to read x and y co-ordinates from file. *Note: five text files have been provided to you: simple_graph_1.txt, simple_graph_2.txt, simple_graph_3.txt, bad_graph.txt, and graph_surprise.txt.*

Each file includes one set of x and y co-ordinates per line, where each line includes the x-axis value followed by the y-axis value, separated by spaces. The first three lines of *simple_graph_1.txt* are as follows.

```
9 -1
6 -1
1 1
```

The images below show the expected output from *simple_graph_1.txt, simple_graph_2.txt,* and *simple_graph_3.txt. bad_graph.txt* can be used as an example of a bad input file. Your program should handle bad input files gracefully. *graph_surpise.txt* contains an additional graph – you won't know what is in it unless you can successfully read it into your program!

```
 2|  xx
 1| x  x
 0+x----x----x
-1|      x  x
-2|       xx
```
simple_graph_1.txt

```
 5|            x
 4|           x x
 3|          x   x
 2|         x     x
 1|  x     x       x
 0+xx-x---x--------xxx
-1|    x x
-2|     x
```
simple_graph_2.txt

```
 5|xxx
 4|  x
 3|  x
 2|  xxxx
 1|     x
 0+-----x------
-1|     xxxx
-2|        x
-3|        x
-4|        xxxx
```
simple_graph_3.txt

Stage H: Writing to file

Once you have visualised the graph, you may like to keep a copy of it. Extend your program so that the graph is saved to a text file.

Advanced features

Stage I and J are the final stages, in which you have the most freedom to explore algorithmic ideas. These stages can be completed in any order, and can be completed on their own, or in conjunction. For example, you can create your own graph (Stage I), or you can investigate techniques for filling in missing data points (Stage J), or both. Include comments in your code that demonstrate your thought processes and your approach.

QUESTION 2 CONTINUES ON THE NEXT PAGE

Stage I: Creating your own graphs

So far, your program can only visualise graphs that are hard-coded into your program (Stages D-F) or that have been provided in a text file (Stage G). Extend your program so that you can create your own custom graphs. This could involve allowing the user to input their co-ordinates, randomly generating a set of co-ordinates on the fly, both of these techniques, or something completely different.

Stage J: Connecting the dots

So far, all of our graph examples have included a complete set of co-ordinates, e.g. we could connect all of the points on the graph to create a continuous line. However, real data will often be missing some data points. For this task, investigate techniques for filling in missing data points. *Hint: to do this, you could manually create a new set of co-ordinates where some are missing, or you could alter one or more of the existing files, manually removing a subset of data points.* Once you have an incomplete set of co-ordinates, you can then start investigating techniques for filling in the missing values.