THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

2023 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. **Musical Chairs** (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.

In this question, you are asked to write a program that simulates a game of Musical Chairs.

Your program should allow the user to specify the number of players, then enter the name of each player, one name at a time. Names must be one word long (i.e. they should not include spaces), must contain alphabet characters only (i.e. a-z, A-Z), and should start with an uppercase letter (A-Z), followed by lowercase letters (a-z). Your program should store the names in a list and then play the game as follows.

- ⇨ Ask the user to start the music
- ⇨ Ask the user to stop the music
- ⇨ Randomly select a player who doesn't get a seat
- ⇨ Inform the user which player didn't get a seat
- ⇨ Inform the user how many players are left, and the names of the players that are left
- ⇨ Repeat until only one player is left
- ⇨ Inform the user which player won

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.

QUESTION 1 CONTINUES ON THE NEXT PAGE

```
Welcome to Musical Chairs!

Please enter the number of players: 3
        Please enter the name of Player 1: Penny
        Please enter the name of Player 2: Sheldon
        Please enter the name of Player 3: Leonard

Thanks Penny, Sheldon, and Leonard. Are you ready to start playing? (y/n): y

        Press Enter to start the music ... [press enter]
        La la la la la
        Press Enter to stop the music ... [press enter]

        Oh no, Sheldon didn't get a seat!

        Number of players left: 2
        Names of players left: Penny, and Leonard

        Press Enter to start the music ... [press enter]
        La la la la la
        Press Enter to stop the music ... [press enter]

        Oh no, Leonard didn't get a seat!

        Number of players left: 1
        Names of players left: Penny

        Press Enter to start the music ... [press enter]

It looks like there is only one player left!
That means the game is over. Congratulations Penny, you WON!
```

2. **Treasure Map** (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: Four text files have been provided to you: treasure_map(1).txt, treasure_map (2).txt, treasure_map (3).txt, and treasure_map(bad).txt.

Embark on a thrilling adventure as you navigate a mysterious treasure map! In this game, you'll be faced with a grid filled with secrets, treasures, and obstacles. Your mission, should you choose to accept it, is to find the path to the hidden treasure while avoiding treacherous traps.

The treasure map is represented as a two-dimensional grid, where each cell can be empty, contain a treasure where X marks the spot ('X'), be blocked by a tree ('T'), or involve climbing a hill ('^'). Your task is to guide an intrepid explorer from the top-left corner of the grid to the coveted 'X' cell, marking the path with asterisk ('*') and hash ('#') characters.

```
+------------+
|        ^   |
|    T     T |
|        X   |
| T      ^   |
+------------+
```

```
+------------+
| *      ^   |
| *  T     T |
| *  *  X    |
| T      ^   |
+------------+
```

```
+------------+
| *  *  #    |
|    T  *  T |
|       X    |
| T     ^    |
+------------+
```

In this question, you are asked to write a program to display and explore a treasure map, using only text display. 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, J, and K are the final stages, in which you have the most freedom to explore algorithm 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 game, you can write different display procedures within the same program to make sure that all of your answers are still visible to the examiner.

QUESTION 2 CONTINUES ON THE NEXT PAGE

Setting up the Map

Stage A: Building a basic map

The treasure map is displayed in a grid. Write a program to draw a 4 x 4 grid using text characters. The result should be similar to the figure below (each cell in the grid is one line high and three characters wide).

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

Stage B: Adding features to the map

The next thing you need to do is add features to the treasure map. Given a 4 x 4 grid, add the following features at the following positions:

- Trees
  - X = 1, Y = 1, symbol = T
  - X = 3, Y = 1, symbol = T
  - X = 0, Y = 3, symbol = T
- Hills
  - X = 2, Y = 0, symbol = ^
  - X = 2, Y = 3, symbol = ^
- Treasure
  - X = 2, Y = 2, symbol = X

The results should be similar to the figure below.

```
+------------+
|        ^   |
|    T      T|
|       X    |
| T     ^    |
+------------+
```

Stage C: Reading the map from file

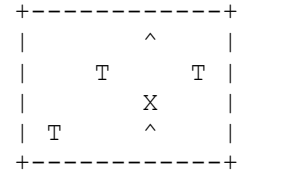
Extend your program to read map information from file. *Note: four text files have been provided to you for this section: treasure_map(1).txt, treasure_map (2).txt, treasure_map (3).txt, and treasure_map(bad).txt*

Each file includes one feature per line, where each line includes the feature type (Tree, Hill, or Treasure), and the x and y position of the feature, separated by spaces. The first three lines of *treasure_map(1).txt* are as follows.

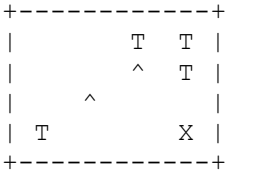
```
Tree 1 1
Tree 3 1
Tree 0 3
```

QUESTION 2 CONTINUES ON THE NEXT PAGE

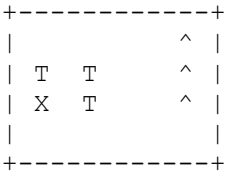
The images below show the expected output from *treasure_map(1).txt, treasure_map (2).txt, and treasure_map(3).txt. treasure_map(bad).txt* can be used as an example of a bad input file. Your program should handle bad input files gracefully.



treasure_map(1).txt



treasure_map(2).txt



treasure_map(3).txt

Finding the Treasure

Stage D: Game play setup

Now that you have set up the treasure map, you can set up a game to find the hidden treasure.

When the program starts, the following steps should be taken:

1. Welcome the player to your game and ask them their name
2. Randomly select one of the input files (*treasure_map(1).txt – treasure_map(3).txt*) and use it to set up your map
3. Display the map, including an asterisk in the 0,0 position to indicate the players starting point

The output below shows an example of gameplay.

```
Welcome to Treasure Map!

Embark on a thrilling adventure as you navigate a mysterious treasure map! In this game,
     you'll be faced with a grid filled with secrets, treasures, and obstacles. Your mission,
     should you choose to accept it, is to find the path to the hidden treasure while
     avoiding treacherous traps.

The treasure map is represented as a two-dimensional grid, where each cell can be empty,
     contain a treasure ('X') where X marks the spot, be blocked by a tree ('T'), or involve
     climbing a hill ('^'). Your task is to find the path from the top-left corner of the
     grid to the coveted 'X' cell. Your path will be marked with asterisk ('*') and hash
     ('#') characters as you go.

Greetings intrepid explorer! Please enter your name: Penny

Welcome Penny! We are finding your treasure map now. Your journey will begin in the top left
     corner of the map. From there, you must decide which path you would like to take.

Press enter to display the treasure map [press Enter]

+------------+
| *     T  T |
|       ^  T |
|    ^       |
| T        X |
+------------+
```

Stage E: Game play

Now that your game is set up, you can guide an intrepid explorer from the top-left corner of the grid to the coveted 'X' cell.

After game setup, ask the player to specify whether they want to move up, down, left, or right from their current position (starting from 0,0). This should continue, marking each new move with an asterisk, until the player reaches the treasure. Note, you cannot move off the grid and you cannot move through Trees ('T). You can move over Hills, but the path over a Hill should be marked with a hash ('#') rather than an asterisk ('*').

The output below shows an example of gameplay.

```
Welcome to Treasure Map!

Embark on a thrilling adventure as you navigate a mysterious treasure map! In this game,
     you'll be faced with a grid filled with secrets, treasures, and obstacles. Your mission,
     should you choose to accept it, is to find the path to the hidden treasure while
     avoiding treacherous traps.

The treasure map is represented as a two-dimensional grid, where each cell can be empty,
     contain a treasure ('X') where X marks the spot, be blocked by a tree ('T'), or involve
     climbing a hill ('^'). Your task is to find the path from the top-left corner of the
     grid to the coveted 'X' cell. Your path will be marked with asterisk ('*') and hash
     ('#') characters as you go.

Greetings intrepid explorer! Please enter your name: Penny

Welcome Penny! We are finding your treasure map now. Your journey will begin in the top left
     corner of the map. From there, you must decide which path you would like to take.

Press enter to display the treasure map [press Enter]

+------------+
| *     T  T |
|       ^  T |
|    ^       |
| T        X |
+------------+

Would you like to go left, right, up, or down? (l/r/u/d): l
Whoops, that direction would take you off the map, you can't move there

Would you like to go left, right, up, or down? (l/r/u/d): u
Whoops, that direction would take you off the map, you can't move there

Would you like to go left, right, up, or down? (l/r/u/d): d
Great choice, let's keep going! Press enter to see an updated map [press Enter]

+------------+
| *     T  T |
| *     ^  T |
|    ^       |
| T        X |
+------------+

Would you like to go left, right, up, or down? (l/r/u/d): d
Great choice, let's keep going! Press enter to see an updated map [press Enter]

+------------+
| *     T  T |
| *     ^  T |
| *  ^       |
| T        X |
+------------+
```

```
Would you like to go left, right, up, or down? (l/r/u/d): d
Whoops, your path is blocked by a tree, you can't move there

Would you like to go left, right, up, or down? (l/r/u/d): r
Phew, that was a steep hill! But let's keep going. Press enter to see an updated map [press
     Enter]

+------------+
| *     T  T |
| *     ^  T |
| *  #       |
| T        X |
+------------+

Would you like to go left, right, up, or down? (l/r/u/d): r
Great choice, let's keep going! Press enter to see an updated map [press Enter]

+------------+
| *     T  T |
| *     ^  T |
| *  #  *    |
| T        X |
+------------+

Would you like to go left, right, up, or down? (l/r/u/d): d
Great choice, let's keep going! Press enter to see an updated map [press Enter]

+------------+
| *     T  T |
| *     ^  T |
| *  #  *    |
| T     *  X |
+------------+

Would you like to go left, right, up, or down? (l/r/u/d): r

Whoo-hoo! You found the treasure! Congratulations Penny!
```

Stage F: Measuring the Path

Right now, the player can take any path they like, long or short. That's okay, so long as they find the treasure in the end! But the player might like to know how long their path was. Extend your program to keep track of the path length, and display the full length to the player at the end. Hills are hard to climb, so they should count as double the length of a normal path.

Stage G: Getting lost

What happens if the player makes a wrong turn? Can they turn back? Make sure the player is able to retrace their own steps. Just make sure to let them know what they are doing. Use a '-' character to show their current location when they are following an already marked path. You can use '=' if the path goes over a hill.

Stage H: Saving the game

Once the player has found the treasure, they may like to keep a record of their journey. Extend your program so that the final map, including the player's path, is saved to a text file. Include the length of the player's path at the end of the file.

Advanced gaming

Stage I, J, and K are the final stages, in which you have the most freedom to explore algorithm ideas. These stages can be completed in any order, and can be completed on their own, or in conjunction. For example, you can find the shortest path for the base-version of the map (Stage G and earlier), or you can find the shortest path for a dynamic map (Stage I), or both. Include comments in your code that demonstrate your thought processes and your approach.

Stage I: Finding the shortest path

Is the path that the player followed the shortest one? Extend your program so that it calculates the shortest path, compares it with the player's final path, and lets them know whether they could have found the treasure faster. Your solution should include a written description of the method(s) that you try, and your reasoning for trying them.

Stage J: Creating a dynamic map

So far, the size of the map has been static (4 x 4) and the types and locations of the features have been determined by input files. Extend your program to include dynamic map size and feature placement, i.e. the size of the map, and the types and locations of the features are determined 'on the fly' by your program and may be different each time your program is run. Your solution should include a written description of the method(s) that you try, and your reasoning for trying them.

Stage K: Treasure map clues

Treasure maps quite often involve finding clues along the way. Extend your program to introduce a new Clue feature ('C'). This feature will require players to find all of the clues before they find the treasure.