Introduction to Software Engineering (ISAD1000/5004)
Semester 2, 2024

Due: Friday 18 October, 23:59 GMT+8
Weight: 50% of the unit mark

Note

This document is subject to clarifications and minor changes that remove ambiguity or address frequently asked questions. It is strongly recommended that you monitor announcements for any additional information.

1 Introduction

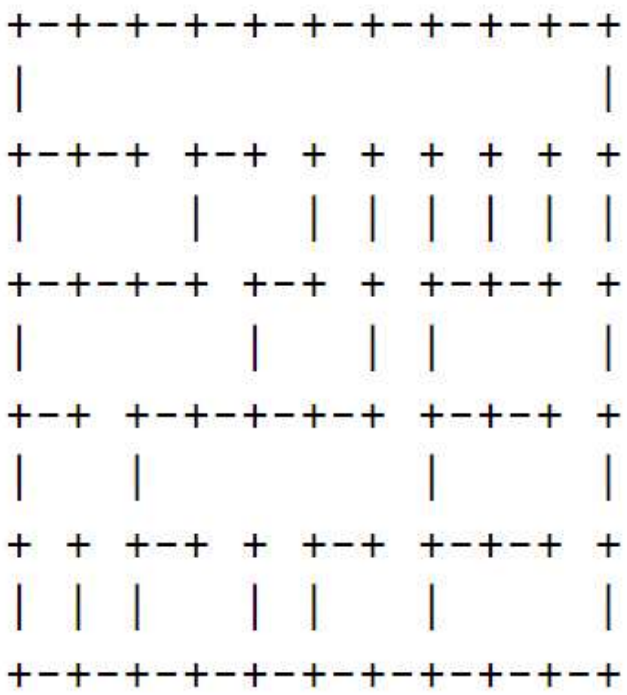
This is the final assessment for Introduction to Software Engineering (ISAD1000/5004). It will assess your competency in designing modular software, using version control to develop your software, design test cases, and implement those test cases for your software. As part of each of these tasks, you will document your work clearly enough that others would be able to follow along and be able to maintain your work.

You will have three weeks to complete this assessment. It is strongly recommended that you plan your time accordingly and begin work as soon as possible.

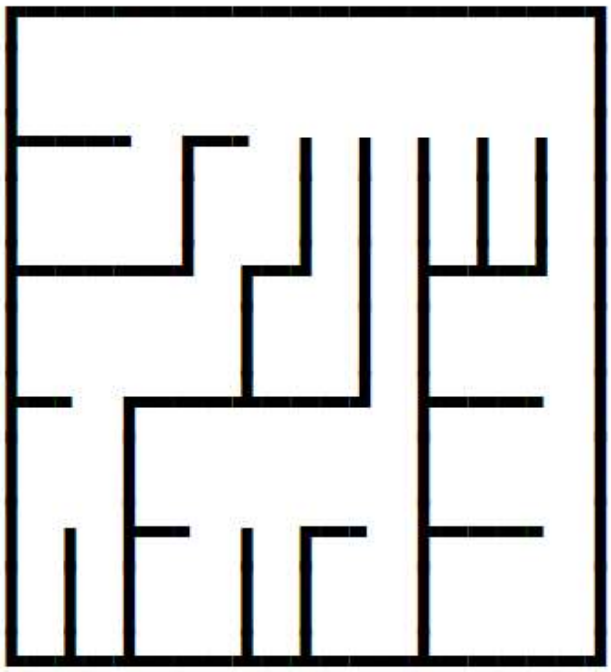
2 The Scenario

Your task will be to process output from a maze-generating program in order to create a smoother, more appealing output for terminal display. The maze-generating program has already been created for you; your task is solely to modify the output. Currently, the program generates a simple maze and displays the maze in a format made up only of '+', '-' and '|' characters. While this is suitable for simple display, it is not the most intuitive to read for a human and your task is to improve on this using Box-Drawing Characters (which are discussed in more detail in section 2.1).

As an example, here is the kind of input you can expect (on the left) and how your program should output the maze (on the right):



Unprocessed Maze



Expected Output

Note that the primary display for this will be on the terminal. You may find that due to differences in font spacing, word processors may display the output incorrectly. Editors dedicated to coding, such as vim or VS Code, will likely display the characters correctly.

2.1 Maze Structure

The final output for the maze will be drawn using entirely box-drawing characters. The full list of box-drawing characters may be found in the Wikipedia article, but you will only need a small subset of characters for this assessment. A maze may be made up of the following characters (and their associated codes):

- Full lines:
  - ┃ (U+2503)
  - ━ (U+2501)
- Half lines (i.e. for open ends of walls):
  - ╺ (U+2578)

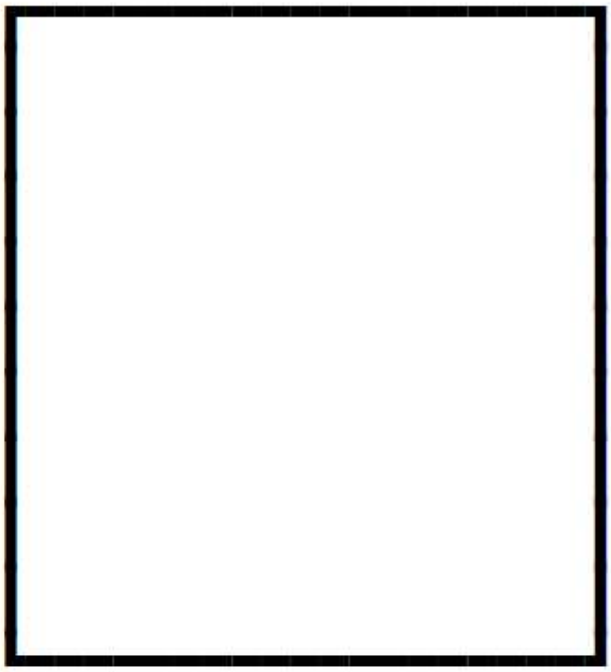
- ╷ (U+257B)

- Corners:
  - ┌ (U+250F)
  - ┘ (U+251B)
  - ┐ (U+2513 - Only used for the outside, top-right border)
  - └ (U+2517 - Only used for the outside, bottom-left border)
- Intersections:
  - ┴ (U+253B)
  - ├ (U+2523)

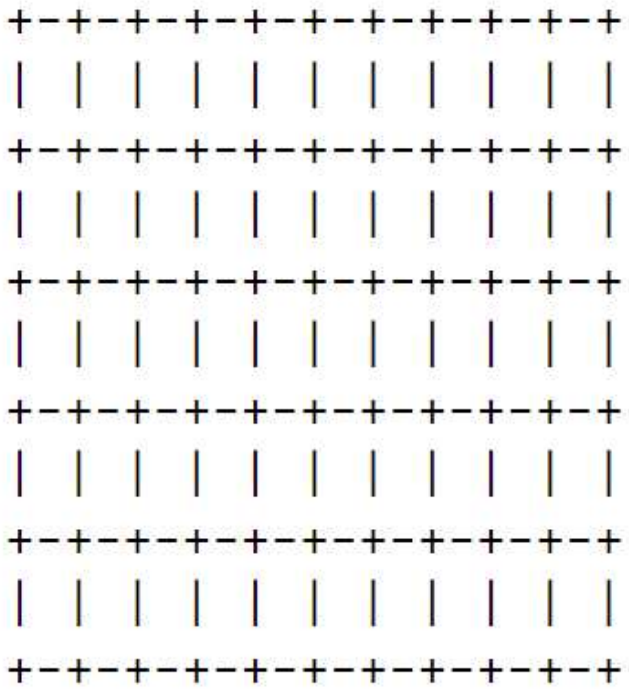
If you happen to be familiar with mazes, this might not seem like a complete set, as the lines are ever only being drawn up and right. However, the algorithm for creating the maze will *never* generate a case where other characters (such as ┼ or ┬) would be used. Only the characters listed above will be used.

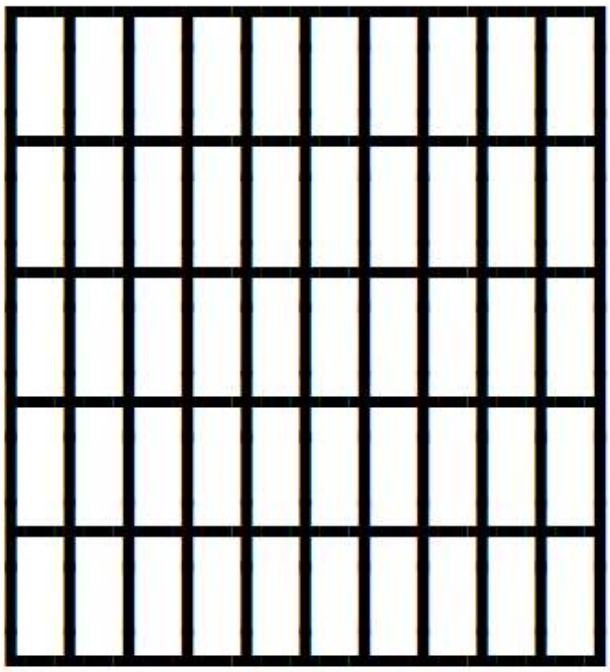
The maze is structured as a 2D array, where each 'cell' is either linked or not linked to one of its neighbours. As an example, if the program were to generate a "maze" where each cell was linked to every neighbour it would generate (and your expected output would be):

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



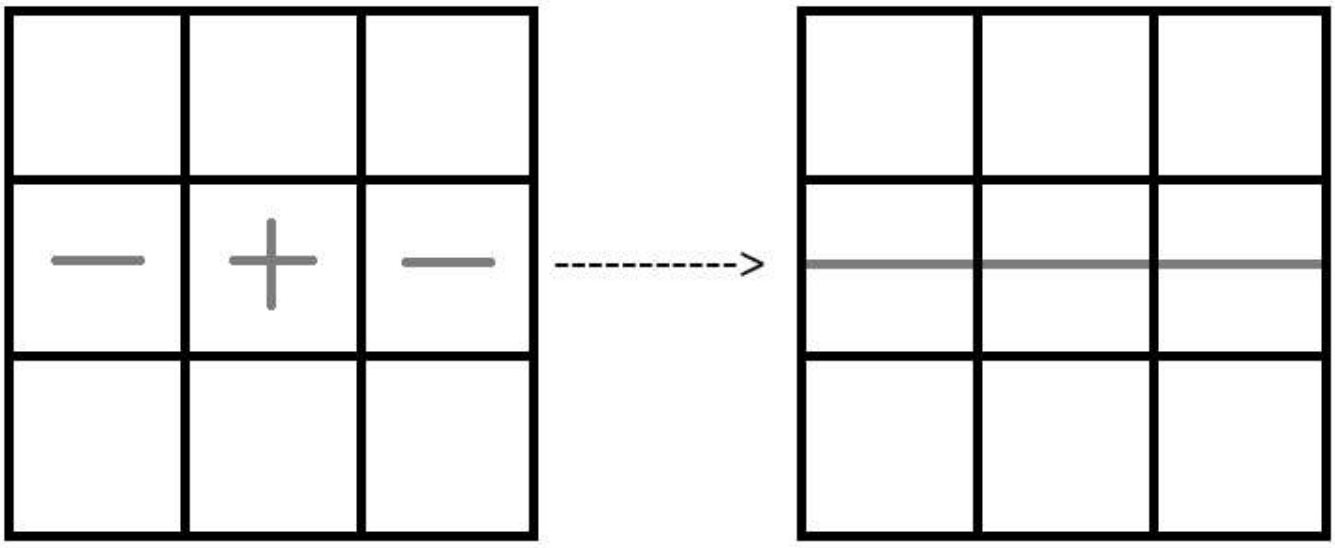
... if *no* cell were linked, the following maze would be generated (and your expected output would be):





In order to determine which is the appropriate character, you will need to inspect a cell in this 2D array and look at each of its neighbours.

A full horizontal line can be determined in the following case, because the corner character ('+') has walls ('-') on the left and right, but not up or down:



Note

HINT: It is suggested that you read from the source maze and write to an output maze: do not overwrite the source maze, as you will lose/overwrite information that may be valuable for both earlier and later cells.

2.2 Maze Generating Program

As part of this assessment, you will be provided with a simple program that generates output as outlined above. You are encouraged to use this program to become familiar with the output and to use cases generated by this program to assist in testing (but they are NOT sufficient for your testing - you must construct test cases that are GOOD test cases, not just random tests). The program is in a language called Ruby. **You do not need to learn this language to use the program**. There are four source code files included with this:

- binary_tree_demo.rb
- binary_tree.rb
- grid.rb
- cell.rb

To run the program, you use the following command on either the lab machines or VMWare Horizon:

```
ruby -I. binary_tree_demo.rb
```

The program will output the maze to the terminal and exit. You may wish to save the output to a file, which can be done with I/O redirection with the following command:

```
ruby -I. binary_tree_demo.rb > maze_output.txt
```

(Any name can be specified instead of `maze_output.txt`)

3 The Tasks

For this assignment, you will create documentation for, and be assessed on, these tasks:

- Version Control
- Modularity Design
- Modularity Implementation
- Test Design
- Test Implementation
- Summary of Work

ALL these tasks will primarily be assessed through your documentation. All code that you create will, however, need to also be submitted to verify your work has been done.

3.1 Version Control

As part of this assignment, you are to apply version control to keep track of your work.

- Create a short plan, identify what branches you will need, why you need them, and when the branches will be merged.
- Create a Git **local repository** for use throughout this assessment. The repository name should be in the format of __ISE_Repo
  - e.g. Brower_Arlen12345678_ISE_Repo
- Commit all code and documents you create in the assessment
  - You are expected to use meaningful commits; while there is no hard rule about what each commit should contain, you are expected to show your ability to use version control meaningfully. Marks are not awarded for number of commits.
- Include evidence of your use of version control, such as an image of the log

Any other discussion or explanation on your use of version control should also be included in the documentation.

Warning

Do not use GitHub, BitBucket, or any other similar online repositories. Doing so runs the risk of potential collusion investigations. Similarly, do not upload your code to online repositories

even after the assignment due date; other students may have extensions or deferred assessments.

3.2 Modularity Design

With the given scenario, you are to identify the most suitable modules you will need for your software to achieve the required functionality, considering the good modularity principles discussed in lectures/worksheets. You are to:

- Write down module descriptions for each module you decide to implement. Descriptions should include:
  - A meaningful name
  - A clear and detailed explanation on the intended task of the module
  - Imports, if any
  - Export, if any
- Explain your design decisions and how they conform to good modularity principles

Note

It's rare that you will get the design 100% ideal from the very start. Design is often iterative. This is actually a very good, interesting thing to include in your documentation for this assessment. If you find it necessary to change or modify your design at some point in this assessment, make note and document those changes. Similarly, if you run into issues or find your design could be made better somehow, include that discussion as well – you might not have time to make the perfect assignment, but self-reflection about what could be improved is a vital skill.

3.3 Modularity Implementation

You are to implement the software using your designs, review the code you have written, and refactor it. As part of this task, you should:

- Implement the modules designed in the previous section

    - You may use Java or Python for your implementation
    - **Your code must run on the lab machines or VMWare Horizon**
- Create a short review checklist to determine if you have followed good modularity principles. You are expected to cover all basic guidelines covered in lecture 7.
- Review your code using the prepared checklist, identifying any issues. You must use the format suggested in worksheet 7 to record your results. Each module must be reviewed.
- If you have identified any issues, refactor your code to address such issues.
    - If you refactor your code, explain how it is improved;
    - If you do not refactor your code, justify your decision
- After refactoring, revise your preliminary descriptions of your modules.
- At every stage, you should be making changes and committing those changes to the repo whenever you have made a step you are satisfied with. Don't commit it all in one blob, but also don't commit just every little change.

3.4 Test Design

You are to design tests using both black box and white box methodologies.

- Black box test design:
    - For each module, according to module descriptions, design suitable test cases.
    - Describe how you decide upon your test values for each test case
    - Test cases should each also describe their expected output or behaviour
- White box test design:
    - Identify at least two modules where white-box testing will be beneficial
    - Design test cases to cover functionality of the selected modules using white box testing
    - You must test at least two different types of control flow constructs
    - Describe how you decide upon your test values for each test case
    - Again, be comitting these test cases to the repo as you write and are satisfied with them

3.5 Test Implementation

Implement your test designs in either Python or Java. You may use test fixtures to organise your test code. **Using a unit test framework is optional.**

Your documentation should:

- Identify the actual results for each test case
- Identify any failures, and attempt to improve your code
  - Be sure to document this process, as it again makes for useful discussion for this assessment

3.6 Summary of Work

You are to produce a table clearly showing your overall module design and the final status of each module. This is to primarily assist your marker in determining how complete your work is.

- Each module should have one row
- For each module, you should identify if the following work has been done:
  - Whether or not the module is complete;
  - Whether or not the module's tests have been designed;
  - Whether or not the module's tests have been implemented;
  - Whether or not the module's tests are successful

Additionally, your submission **must include a short video** roughly two minutes in length demonstrating that your code is working. More information on this video is included in section 5 of this document.

4 Documentation

As noted, your work will primarily be assessed through your documentation. All information outlined above must be included in your documentation and your documentation must be submitted to Turnitin as a .pdf file. In addition to the tasks outlined above, your documentation must also include:

- A cover page; include the assessment name, your name in Blackboard, your student ID, and your practical date/time.
- An introduction; briefly describe and overview of your work
- Discussion; reflect on your own work, including a summary of what you have achieved, challenges you have faced, limitations, and ways to improve your work with other features you have not considered.
    - You may include other information if you feel it would be useful to clarify anything about your submission

Additionally, marks are allocated for a neat and professional document.

5 Video Demonstration

Your submission must include a short, two minute video that demonstrates your code working. As part of this video **state your name and student number first** and then give an overview of your code. You do not need to go into the source code and explain any decisions, it is sufficient to show that your code is working. This is helpful in assisting your marker troubleshoot issues if any are encountered.

> **Note**
>
> You may be asked after the submission to attend a short demonstration via Microsoft Teams in order to ensure academic integrity. Please keep an eye on your email and announcements to this effect. If you do have a request to attend a demonstration in this way, you must attend. A request to demonstrate is not an allegation of misconduct, but is part of the process to ensure academic integrity in the unit.

6 Assessment Submission

You are to submit your assessment in three locations on Blackboard:

- Declaration of Originality submission point
- A Turnitin submission point

- A general submission point

6.1 Declaration of Originality submission point

As part of this submission, you will submit a Declaration of Originality stating that this work is your own, has not copied anyone else, and any sources are appropriately referenced. This may be a .pdf, .odt, or .docx file and **must** be signed and dated.

If you forget to submit your Declaration of Originality, do not resubmit to any other submission point; only submit your Declaration of Originality to this location. A declaration of originality can be submitted after the due date if a mistake is made, however, you cannot receive a mark for this submission until a declaration has been received.

6.2 Turnitin submission

Your Turnitin submission will *only* include the Documentation .pdf as outlined above. No source code files are to be included in this submission; no .zip files will be accepted. **You** must be the one to submit your .pdf through Turnitin, at which point it is registered.

6.3 General submission point

The general submission point should include all files created throughout the assessment (including the .pdf submitted to Turnitin). This should include:

- Your documentation .pdf file
- A .zip (**not .rar, not .7z, etc.**) file containing your Git repository (**including the hidden .git folder**) and all associated source files, test inputs, etc.
  - Note that your Git repository is NOT just your working directory (source files), it is inside the hidden .git folder. Do NOT miss including this in the .zip file.
- A single README file; a short text indicating how to use your program
- Your short 2 minute video demonstrating how to run the system and test cases

7 Marking

The allocation of marks for this assessment are as follows:

- Version Control **[16 marks]**
- Modularity Design **[7 marks]**
- Modularity Implementation **[20 marks]**
- Test Design
  - Black Box **[16 marks]**
  - White Box **[9 marks]**
- Test Implementation **[20 marks]**
- Summary of Work **[5 marks]**
- Documentation-specific marks **[7 marks]**

Academic Integrity

Please see the *Coding and Academic Integrity Guidelines* on Blackboard.

In summary, this is an assessable task. If you use someone else's work or assistance to help complete part of the assignment, where it's intended that you complete it yourself, you will have compromised the assessment. You will not receive marks for any parts of your submission that are not your own original work. Further, if you do not *reference* any external sources that you use, you are committing plagiarism and/or collusion, and penalties for academic misconduct may apply.

Curtin also provides general advice on academic integrity at academicintegrity.curtin.edu.au.

The unit coordinator may require you to provide an oral justification of, or to answer questions about, any piece of written work submitted in this unit. Your response(s) may be referred to as evidence in an academic misconduct inquiry.