

Cover page

Introduction to Software Engineering– ISAD1000 2024 Trimester 1 Final Assessment

Student Name: Zernish Shahid Nazir

Curtin Student ID: 21848838

Practical class (date/time): Friday 29th April (8:00 - 12:00)

Introduction

In this assignment, I've developed and tested software modules to determine seasons and analyze temperature data according to the given scenarios. The software comprises SeasonModule and TemperatureModule, with submodules for specific functionalities, split by the two given scenarios. SeasonModule validates user inputs for country and month, accurately determining the corresponding season. Meanwhile, TemperatureModule validates inputs for city and temperature readings, comparing them with city averages and generating appropriate messages. I implemented both black-box and white-box techniques for testing, ensuring accurate handling of various inputs and scenarios.

Module descriptions

SeasonModule:

Submodule validateInput

- **Imports:** country, month (strings) provided by user
- **Exports:** None
- **Description:** Validates the input provided by the user for determining the season of the year and checks if the provided country and month are valid inputs.

Submodule findSeason

- **Imports:** country, month (strings) provided by user
- **Exports:** season (string) determined by the module
- **Description:** Determines the season of the year based on the country name and month provided by the user. If the provided country and month are valid, the module returns the corresponding season.

TemperatureModule:

Submodule `validateInput`

- **Imports:** city (string), temperature (double) provided by user
- **Exports:** None
- **Description:** Validates the input provided by the user for analysing the temperature of a city and checks if the provided city name and temperature reading are valid inputs.

Submodule `compareTemperature`

- **Imports:** city (string), temperature (double) provided by user
- **Exports:** message (string) message generated by the module based on the temp analysis
- **Description:** Compares a given temperature reading with the average temperature of a city and generates an appropriate message. If the temperature reading is within the valid range for the city, the module compares it with the average temperature of the city and generates appropriate response.

Submodule `getDailyMeanTemperature`

- **Imports:** city (string) provided by user
- **Exports:** dailyMeanTemperature (double) determined by the module
- **Description:** Retrieves the daily mean temperature based on the city name provided by the user.

Design decisions and modularity I created two modules, one for each scenario and further divided each module into two submodules. #### Season-Module:

- Divided into two modules, `validateInput` and `findSeason`, to avoid problems and promote reusability.
- Ensures that each module has a single responsibility, making it easier to understand and maintain.

TemperatureModule:

- Also divided into two modules, `validateInput` and `compareTemperature`.
- Allows for better organization and clarity in the code.

Modularity Principles:

- **Divide and Conquer:** Dividing the functionality into smaller modules facilitates easier development, testing, and maintenance. Each module focuses on a specific task, promoting clarity and reducing complexity.
- **High Cohesion:** Each module performs a single well-defined task, enhancing clarity and reducing the likelihood of bugs. Promotes code reuse

and makes it easier to update or modify specific functionalities without affecting others.

- **Low Coupling:** Modules communicate through well-defined interfaces (Imports and Exports), reducing interdependencies. Changes in one module are less likely to impact others, promoting flexibility and maintainability. By having the functionality separated into different classes, I've reduced the coupling between them.
- **Input Handling and Output Availability:** When handling inputs, modules validate input data to ensure that only valid inputs are processed further. Error handling is implemented to handle invalid inputs and provide meaningful feedback to the user. As for the outputs, each module exports relevant data or messages, making them available for further processing or display.
- **The structure allows for easy extension or modification of functionalities throughout the project. New modules can be added or existing ones modified without affecting the overall system.**

Modularity

How to run code with correct commands

- Compile the code “javac WeatherTool.java”
- Execute, run the WeatherTool program “java WeatherTool”
- Follow the prompts; choosing between scenario 1 and 2
- Depending on the scenario chosen, the program will provide information about the season for a given country and month, or whether a temperature reading is above or below average for a city.

```
test@curtinlab:~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo$ java WeatherTool.java
Select scenario:
1. Finding the season of the year for a given country and month
2. Finding whether a given temperature reading is above or below the average temperature of a city
1
Enter country name:
Valid options: Australia, Noongar, UAE, Malaysia, Singapore
uae
Enter a month of the year:
february
Season: Winter
test@curtinlab:~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo$ java WeatherTool.java
Select scenario:
1. Finding the season of the year for a given country and month
2. Finding whether a given temperature reading is above or below the average temperature of a city
2
Enter city name:
Valid options: Perth, Dubai
perth
Enter temperature:
33
Temperature reading is above the average by more than 6 degrees
test@curtinlab:~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo$
```

Figure 1: sample output

Sample output of running production code

brief explanation on application of modularity concepts

- My code applies modularity concepts, including low coupling and high cohesion. Low coupling is achieved by making sure that the modules are loosely connected, meaning changes in one module would minimally impact the others, which creates a more flexible and easily maintainable code.
- My code maintained high cohesion by grouping related functionalities with each module, separating them between the two given scenarios. Therefore my code has a modular design and is maintainable and scalable.

Checklist Review Results

Item	Checklist question related to modularity	Yes/No	Description of the issue if No the answer
1	Does code exhibit low coupling between modules?	Yes	
2	Are cross-module function dependencies minimised?	Yes	
3	Are global variables avoided?	Yes	
4	Control flags used appropriately?	Yes	
5	Are parameters received in correct order?	Yes	
6	Is a tolerance value used when comparing real numbers for equality?	Yes	
7	Does each method clearly person a single task?	Yes	
9	Is redundancy minimised?	No	There are similarities in the validation part for the methods in TemperatureModule

Refactoring Decisions Generally, my code shows good modularity practices. It demonstrates low coupling between modules, minimised dependencies between modules, and avoidance of inconvenient global variables. Each method performs a single task, enhancing clarity and maintainability. However, the review of the checklist highlighted the main issue related to code redundancy. Specifically there were similarities in the validation part for the methods in the TemperatureModule class, indicating potential redundancy.

To address the issue of redundancy, I refactored the 'TemperatureModule' class to reduce the duplication of code. Previously both the 'validateTemperature' and 'compareTemperature' methods had similar logic to assign the daily mean temperature based on the city. This duplication was removed by introducing a new private method 'getMeanTemperature'.

This method separated the logic for retrieving the daily mean temperature based on the city name, to a separate method. Both ‘validateTemperature’ and ‘compareTemperature’ now call this other method to get the daily mean temperature, reducing redundancy and making the code easier to maintain. By refactoring the code in this way, I ensured that if the logic for determining the daily mean temperature needs to be changed in the future, it only needs to be updated in one place (getDailyMeanTemperature method) rather than in multiple places, reducing confusion and errors. Finally, I revised module descriptions to add the new method.

Test design- Black-box test cases

Equivalence Partitioning

Submodule: validateInput

Category	Test Data	Expected result
Valid country and valid month combination	country = “Australia”, month = “January”	true
Valid country and invalid month combination	country = “Australia”, month = “InvalidMonth”	false
Invalid country and valid month combination	country = “InvalidCountry”, month = “January”	false
Invalid country and invalid month combination	country = “InvalidCountry”, month = “InvalidMonth”	false

Submodule: findSeason

Category	Test Data	Expected result
Valid country and valid month combination	Country = “Australia”, Month = “January”	Season = “Summer”
Valid country and invalid month combination	Country = “Noongar”, Month = “July”	Season = “Invalid input message”
Invalid country	Country = “Nazir”	Prompt for country again

Submodule: validateTemperature

Category	Test Data	Expected result
Valid City && valid temperature	City = "Perth", Temperature = 20.0	No output
Invalid Temperature	City = "Perth", Temperature = 838	"Invalid temperature reading. Valid range is 0.7°C to 46.0°C."
Invalid City	City = "Paris"	"Invalid city"

Submodule: compareTemperature

Category	Test Data	Expected result
Temperature above average by less than 6 degrees	City = "Perth", Temperature = 25.0	"Temperature reading is above the average"
Temperature below average by less than 6 degrees	City = "Perth", Temperature = 15.0	"Temperature reading is below the average"
Temperature above average by more than 6 degrees	City = "Perth", Temperature = 30.0	"Temperature reading is above the average by more than 6 degrees"
Temperature below average by more than 6 degrees	City = "Perth", Temperature = 5.0	"Temperature reading is below the average by more than 6 degrees"
Invalid City	City = "Paris", Temperature = 20.0	"Invalid city"

Submodule: getDailyMeanTemperature

Category	Test Data	Expected result
Valid city	City = "Perth"	dailyMeanTemperature = 18.2

Category	Test Data	Expected result
Valid city	City = "Dubai"	dailyMeanTemperature = 26.9
Invalid city	City = "Paris"	Throws IllegalArgumentException (Invalid city)

BVA for compareTemperature

Boundary category	Test Data	Expected Result
Between invalid (below min temp) and MORE THAN OR EQUAL TO 6 degrees less than mean	Temperature= 0.6, Temperature= 0.7	"Invalid temperature reading. Valid range is 0.7°C to 46.0°C.", "Temperature reading is below the average by more than 6 degrees"
Between MORE THAN OR EQUAL TO 6 degrees less than mean and valid temp range	Temperature= 12.2, Temperature= 12.3	"Temperature reading is below the average by more than 6 degrees", "Temperature reading is above/below the average"
Between valid temp range and MORE THAN OR EQUAL TO 6 degrees more than mean	Temperature= 24.1, Temperature= 24.2	"Temperature reading is above/below the average", "Temperature reading is above the average by more than 6 degrees"
Between MORE THAN OR EQUAL TO 6 degrees more than mean and invalid (above max temp)	Temperature= 46.0, Temperature= 46.1	"Temperature reading is above the average by more than 6 degrees", "Invalid temperature reading. Valid range is 0.7°C to 46.0°C."

Test design decisions

- **validateInput:** Equivalence partitioning was used to cover different combinations of valid and invalid inputs for country and month. Since this submodule deals with input validation, testing all possible combinations is necessary.
- **findSeason:** Equivalence partitioning was again used to test various combinations of valid and invalid country and month inputs, ensuring that the submodule returns the correct season based on the input. Invalid country inputs trigger a prompt for the country again, which is tested again.
- **validateTemperature:** Equivalence partitioning was applied to cover valid and invalid inputs for city and temperature. Each category was tested

to ensure the correct response: no output for valid city and temperature, appropriate error messages for invalid temperature and city inputs.

- **compareTemperature:** BVA was specifically applied to this submodule because it deals with numerical temperature values and involves comparing these values with the average temperature of a city. By focusing on the boundaries, BVA helps identify potential issues related to temperature comparisons, such as incorrect handling of extreme values. The chosen boundaries cover scenarios where the temperature is at or near the extremes of the valid range, therefore thoroughly testing the temperature.
- **getDailyMeanTemperature:** Equivalence partitioning was used to test the submodule’s behavior for valid and invalid city inputs. Since this submodule retrieves the daily mean temperature based on the city, testing different valid and invalid city inputs ensures correct functionality.

White-box test cases

validateTemperature (if block)

Path	Test Data	Expected Result
Enter the ‘Perth’ if part	City = “Perth”, Temperature = 20.0	No output (successful validation)
Enter the ‘Dubai’ if part	City = “Dubai”, Temperature = 30.0	No output (successful validation)
Enter the else part	City = “Paris”, Temperature = 20.0	Output: “Invalid city” (indicating invalid city entry)

validateTemperature (try-catch block)

Path	Test Data	Expected Result
Success	Valid city name, valid dailyMeanTemperature	No exception thrown, no output
Invalid City	Invalid city name (e.g., “Paris”)	Caught IllegalArgumentException, prints “Invalid city” message and returns

compareTemperature

Path	Test Data	Expected Result
Enter outer if part	temperatureDifference = 7.0	Temperature reading is above the average by more than 6 degrees”

Path	Test Data	Expected Result
Enter inner if part	temperatureDifference = 2.0	"Temperature reading is above the average"
Enter inner else if part (else-if 1)	temperatureDifference = -3.0	"Temperature reading is below the average"
Enter outer else part	temperatureDifference = 0.0	"Temperature reading is at the average"

Test design decisions

- These specific white-box test cases were chosen because they target critical paths within the methods that are essential for ensuring the correctness and robustness of the code.
- For the validateTemperature method, the test cases were structured to cover both the successful validation path and the exception handling path. On the other hand, for the exception handling path, a test case was created to simulate an invalid city name, triggering the catch block. This test aimed to validate the method's ability to catch and appropriately handle an IllegalArgumentException when an invalid city name is provided, confirming that the method responds as intended.
- As for the compareTemperature method, the white-box test cases were designed to thoroughly examine the nested if-else statement within the method. Each test case corresponds to a different branch of the nested structure, ensuring coverage of all possible code paths. By evaluating these separate parts, the test case verifies that the method produces the correct message based on the temperature difference. This approach ensures comprehensive testing of the method's logic and behavior under different conditions.

Test Implementation and test execution

```
test@curtinlab:~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo$ java ValidateInputTest
Testing validateInput method...
Invalid input: Month not recognized for the selected country.
Invalid input: Country not recognized.
Invalid input: Country not recognized.
All test cases passed successfully.
```

Figure 2: test1 result

ValidateInputTest

FindSeasonTest

ValidateTemperatureTest

```
test@curtinlab:~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo$ java FindSeasonTest
Testing findSeason method...
All test cases passed successfully.
```

Figure 3: test2 result

```
test@curtinlab:~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo$ java ValidateTemperatureTest
Testing validateTemperature method...
Invalid temperature reading. Valid range is 0.7°C to 46.0°C.
Invalid city
Invalid city
Invalid city
All test cases passed successfully.
```

Figure 4: test3 result

```
test@curtinlab:~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo$ java CompareTemperatureTest
Testing equivalence partitioning for compareTemperature method...
Exception in thread "main" java.lang.IllegalArgumentException: Invalid city
    at TemperatureModule.getDailyMeanTemperature(WeatherTool.java:153)
    at TemperatureModule.compareTemperature(WeatherTool.java:196)
    at CompareTemperatureTest.testEquivalencePartitioning(CompareTemperatureTest.java:33)
    at CompareTemperatureTest.main(CompareTemperatureTest.java:6)
```

Figure 5: test4 result

CompareTemperatureTest The `getDailyMeanTemperature` method is designed to throw an `IllegalArgumentException` when it receives a city name that it doesn't recognize or support. This behavior is intentional, and it's a way of handling invalid input in the code.

So, if the user inputs a city name other than "Perth" or "Dubai", the method will throw an exception with the message "Invalid city". This is a form of error handling to ensure that the program doesn't proceed with invalid input, and it provides feedback to the user that the input was not accepted.

•

GetDailyMeanTemperatureTest

```
test@curtinlab:~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo$ java GetDailyMeanTemperatureTest
Test passed for valid city Perth
Test passed for valid city Dubai
Test passed for invalid city Paris
```

How to run with correct commands- I worked on each test case individually, and compiled it with `WeatherTool.java`. I then ran the test files with the command 'java (name of test)'

Summary of my work

Version Control

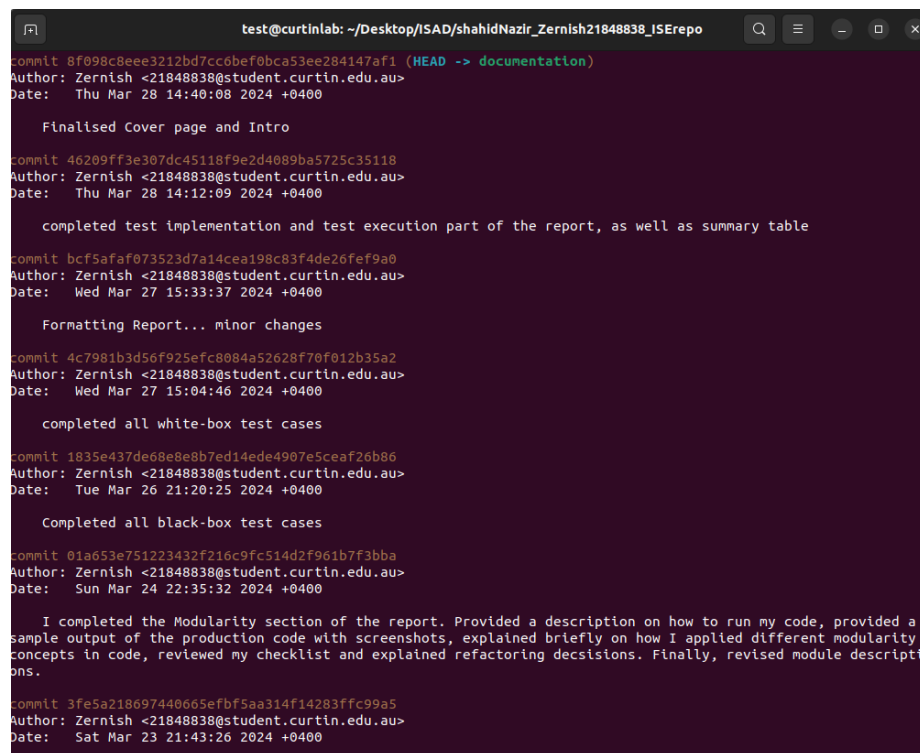
Explanation I set up my version control in the start two have two branches:
- **Code:** Where I wrote all my code including the main program **Weather-**

	<i>Design of test cases</i>					<i>Test code implementation and execution</i>		
<i>Module name</i>	<i>BB (EP)</i>	<i>BB (BVA)</i>	<i>WB</i>	<i>Data type/s</i>	<i>Form of Input/output</i>	<i>EP</i>	<i>BVA</i>	<i>White-Box</i>
validateInput	done	not done	not done	string	Keyboard input/ Console output	done	not done	not done
findSeason	done	not done	not done	string	Keyboard input/ Console output	done	not done	not done
validateTemperature	done	not done	done	String, double	Keyboard input/ Console output	done	not done	done
compareTemperature	done	done	done	String, double	Keyboard input/ Console output	done	done	done
getDailyMeanTemperature	done	not done	not done	string	Keyboard input/ Return value	done	not done	not done

Figure 6: summary table

Tool.java, as well as all the test codes - **Documentation:** Where I created my .md file for my report and gradually added to it as I worked through my assignment. I committed throughout this process to keep track of milestones within my assignment

Finally, I merged both the branches. I did so because merging consolidates the changes made in separate branches into the main branch (master), providing a comprehensive view of the assignment's progress, with all my work conveniently in one place.



```
test@curtinlab: ~/Desktop/ISAD/shahidNazlr_Zernish21848838_ISErepo
commit 8f098c8eee3212bd7cc6bef0bca53ee284147af1 (HEAD -> documentation)
Author: Zernish <21848838@student.curtin.edu.au>
Date: Thu Mar 28 14:40:08 2024 +0400

    Finalised Cover page and Intro

commit 46209ff3e307dc45118f9e2d4089ba5725c35118
Author: Zernish <21848838@student.curtin.edu.au>
Date: Thu Mar 28 14:12:09 2024 +0400

    completed test implementation and test execution part of the report, as well as summary table

commit bcf5afaf073523d7a14cea198c83f4de26fef9a0
Author: Zernish <21848838@student.curtin.edu.au>
Date: Wed Mar 27 15:33:37 2024 +0400

    Formatting Report... minor changes

commit 4c7981b3d56f925efc8084a52628f70f012b35a2
Author: Zernish <21848838@student.curtin.edu.au>
Date: Wed Mar 27 15:04:46 2024 +0400

    completed all white-box test cases

commit 1835e437de68e8eb7ed14ede4907e5ceaf26b86
Author: Zernish <21848838@student.curtin.edu.au>
Date: Tue Mar 26 21:20:25 2024 +0400

    Completed all black-box test cases

commit 01a653e751223432f216c9fc514d2f961b7f3bba
Author: Zernish <21848838@student.curtin.edu.au>
Date: Sun Mar 24 22:35:32 2024 +0400

    I completed the Modularity section of the report. Provided a description on how to run my code, provided a sample output of the production code with screenshots, explained briefly on how I applied different modularity concepts in code, reviewed my checklist and explained refactoring decisions. Finally, revised module descriptions.

commit 3fe5a218697440665efbf5aa314f14283ffc99a5
Author: Zernish <21848838@student.curtin.edu.au>
Date: Sat Mar 23 21:43:26 2024 +0400
```

Figure 7: docLog

Git log Documentation

Git log Code

Merging branches

```
test@curtinlab: ~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo
test@curtinlab:~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo$ git log
commit b8f221d457b0c59d538a7fdc6edd7fcc07e30a2d (HEAD -> code)
Author: Zernish <21848838@student.curtin.edu.au>
Date:   Wed Mar 27 22:04:38 2024 +0400

    minor changes to main code

commit 9f5192941131dd958a177f805d18157121b4a139
Author: Zernish <21848838@student.curtin.edu.au>
Date:   Wed Mar 27 22:03:37 2024 +0400

    Created all test cases for each method in separate files, compiled and ran test to check for errors

commit adb48dda48c25c99c99a5d2d9867c8613d408248
Author: Zernish <21848838@student.curtin.edu.au>
Date:   Mon Mar 25 22:14:34 2024 +0400

    Made a few changes to code

commit 8174f941da60bf7d7a62412f5d21316500c003dc
Author: Zernish <21848838@student.curtin.edu.au>
Date:   Mon Mar 25 17:50:20 2024 +0400

    Refactored my code

commit acadb3289e822d10864ae61b85f9ac56f4cd2b37
Author: Zernish <21848838@student.curtin.edu.au>
Date:   Sun Mar 24 20:00:52 2024 +0400

    Implemented my modules and code compiled and ensured there are no syntax errors.

commit 3493316bb7e020e700f7e66e88bb4bcf6cccd5a1b (master)
Author: Zernish <21848838@student.curtin.edu.au>
Date:   Fri Mar 22 21:50:09 2024 +0400

    testing commit
```

Figure 8: codeLog

```
test@curtinlab: ~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo
test@curtinlab:~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo$ git checkout master
Already on 'master'
test@curtinlab:~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo$ git merge code
Updating 3493316..b8f221d
Fast-forward
 CompareTemperatureTest.java      | 84 ++++++
 FindSeasonTest.java              | 25 ++++++
 GetDailyMeanTemperatureTest.java | 36 ++++++
 ValidateInputTest.java           | 27 ++++++
 ValidateTemperatureTest.java      | 44 ++++++
 WeatherTool.java                 | 235 ++++++
+++++++
6 files changed, 451 insertions(+)
create mode 100644 CompareTemperatureTest.java
create mode 100644 FindSeasonTest.java
create mode 100644 GetDailyMeanTemperatureTest.java
create mode 100644 ValidateInputTest.java
create mode 100644 ValidateTemperatureTest.java
create mode 100644 WeatherTool.java
test@curtinlab:~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo$ git merge documentation
Merge made by the 'ort' strategy.
 CompareTemperatureTest.png        | Bin 0 -> 130613 bytes
 FindSeasonTest.png                | Bin 0 -> 39780 bytes
 GetDailyMeanTemperatureTest.png   | Bin 0 -> 49183 bytes
 ValidateInputTest.png             | Bin 0 -> 72556 bytes
 ValidateTemperatureTest.png        | Bin 0 -> 66570 bytes
 ZernishShahidNazir_21848838_ISEReport.md | 246 ++++++
+++++++
 sampleCode.png                   | Bin 0 -> 71179 bytes
 table.png                        | Bin 0 -> 36109 bytes
8 files changed, 246 insertions(+)
create mode 100644 CompareTemperatureTest.png
create mode 100644 FindSeasonTest.png
create mode 100644 GetDailyMeanTemperatureTest.png
create mode 100644 ValidateInputTest.png
create mode 100644 ValidateTemperatureTest.png
create mode 100644 ZernishShahidNazir_21848838_ISEReport.md
create mode 100644 sampleCode.png
create mode 100644 table.png
test@curtinlab:~/Desktop/ISAD/shahidNazir_Zernish21848838_ISErepo$
```

Figure 9: merge

Discussion

- Through this assignment, I've gained proficiency in Git, facilitating organized collaboration and version control. Working with Git was enjoyable and efficient, aiding structured progress tracking.
- Designing test cases was insightful, ensuring code reliability. However, compiling code during testing posed challenges initially, but I overcame them through experimentation and online resources. Markdown documentation, while initially challenging, was quite straightforward, though working on a more familiar platform would have been easier.
- In coding, one thing I noticed was that writing test cases got a bit repetitive. Streamlining testing methodologies could enhance future projects. It made testing a bit confusing at times. And I realized my code could use more error handling to make it more user-friendly. For example implementing a loop so that the program only exits once the user wants to, instead of having to run the code once more if the user still wants to use it
- In summary, this assignment expanded my software engineering skills. Moving forward, I aim to refine testing methodologies, improve error handling, and explore interactive features for enhanced user experiences.