UNIVERSITY OF LEEDS

School of Computing: Assessment brief

Module title	Programming Project
Module code	XJCO1921
Assignment title	Assignment 1 – Project Plan
Assignment type and description	You will produce a plan for the project described below, which includes: - A test script - Test data - A basic skeleton for the program
Rationale	When approaching a programming project, it is important to start with a clear plan. One of the ways in which this is done in the industry is to use test-driven development (TDD), designing a suite of tests which provide the developer with a clear idea of what they need to create and the desired behaviour of a system whilst also giving you an opportunity to think about how to break down a large problem into many smaller ones! This task will give you an opportunity to design a set of tests for a problem and to produce a basic skeleton for your code, which will help you in the next assessment.
Word limit and guidance	You should spend 8-12 hours working on this assessment.
Weighting	40%
Submission deadline	11 April 2024 12 PM BST (19:00 Beijing Time)
Submission method	Gradescope
Feedback provision	Marked rubric
Learning outcomes assessed	- explain the importance of applying professional programming practices to programming projects. - design, implement, debug and test a modular programming solution to a real-world problem.
Module lead	Zheng Wang / Xiuying Yu

1. Assignment Guidance

In this assignment, you will be creating three assets:

- An automated test script
- Test data
- Skeleton code based around this module plan.

You are not expected to produce a working implementation of the program described below, and no marks or feedback will be given for implementing the program.

You should produce your plans and skeleton code based on the brief given below.

Maze Game

You are creating a basic game where players navigate through a maze.

The maze will be loaded by the program when it begins, and the filename and dimensions will be provided on the command line:

```
./maze filename width height
```

For example, `./maze maze.txt 20 43` specifies a maze with a grid of 20 rows (width) and 43 columns (height), where elements (characters) of the maze are given in maze.txt.

Mazes are made of 4 characters:

Character	Purpose
'#'	A wall which the player cannot move through
' ' (a space)	A path which the player can move across
'S'	The place where the player starts the maze
'E'	The place where the player exits the maze

A maze has a height and a width, with a maximum of 100 and a minimum of 5. Your program will ***dynamically allocate*** an appropriate data structure to store the maze.

The height and width do not have to be equal – as long as both are within the accepted range.

Within a maze, each 'row' and 'column' should be the same length – the maze should be a rectangle.

When the game loads, the player will start at the starting point ‘S’ and can move through the maze using WASD movement:

Key	Direction
W/w	Up
A/a	Left
S/s	Down
D/d	Right

Note: There is no expectation for these to be keypress triggered; you can expect the user to enter a newline between each direction.

The player can move freely through path spaces (‘ ‘) but cannot move through walls or off the edge of the map. If they try to do so, they should receive some message telling them not to do this but should be able to continue playing the game.

Players can also enter ‘M’/’m’ to see a map of the maze, with their current location shown with ‘X’.

When the user reaches the exit point ‘E’, the game is over and will close. The player should be given some message stating that they have won.

2. Assessment tasks

Test Script

You will produce a test script which can be used to test this code. You need to consider the range of different potential errors that could occur in the program, paying particular attention to places where data is being inputted, such as the command line or the stdin.

You should also think about potential mistakes you might make in programming and how you will ensure that your code is working successfully – for example, if a player walks into a wall, how will this be handled, and how can you test for this? You should test for success as well as failure.

You should have a minimum of **10 tests**, but more would be expected to fully encapsulate the range of potential errors.

Each test should have a comment above it stating what it is testing for.

You can assume that the executable will be named '`maze`'; you will need to consider what prints and outputs your code will produce and check these appropriately.

You can use either a bash/Python script or a unit testing library of your choice. Your code and test script must be able to run in Linux.

Test Data

You should create some test data that you can use with your script to test the cases you have written.

This should include:

- A variety of maze text files which you will use in your testing (valid and invalid)
- Some input files to navigate through your test mazes

Not only will these help you think about the kinds of errors you should be testing for in your script, but they will also make testing your code easier when you come to implement it.

Skeleton Code

Based on the provided brief, produce a skeleton for your code. This should include:

- At least one struct definition
- Function/procedure definitions
- A basic outline for your main

You do not need to write any of the functionality, but you should use comments to give a basic idea of how you think you will structure your code.

You can see several examples on Minerva.

3. General guidance and study support

You should refer to the lab exercises and lecture notes to support you.

4. Assessment criteria and marking process

Your test script will be manually checked in order to look at your handling of:

- system errors
- bad user inputs
- common logic errors

Your test data will be checked in order to make sure that it is used in your test script.

Your code skeleton will be checked for:

- Modular structure
- Struct design
- Basic structuring of your program

A full breakdown is available in section 8.

5. Presentation and referencing

You will submit your work to Gradescope. You should organise this appropriately, for example, by putting all test data inside a subdirectory in order to make it easier to find and mark your test plan and skeleton code.

If you do need to reference any resources, use a simple comment, for example:

```
// This test is adapted from an example provided on https://byby.dev/bash-exit-codes
```

You should not be directly copying any code from external resources, even with a reference.

6. Submission requirements

Submit via Gradescope.

7. Academic misconduct and plagiarism

Leeds students are part of an academic community that shares ideas and develops new ones.

You need to learn how to work with others, how to interpret and present other people's ideas, and how to produce your own independent academic work. It is essential that you can distinguish between other people's work and your own, and correctly acknowledge other people's work.

All students new to the University are expected to complete an online Academic Integrity tutorial and test, and all Leeds students should ensure that they are aware of the principles of Academic integrity.

When you submit work for assessment it is expected that it will meet the University's academic integrity standards.

If you do not understand what these standards are, or how they apply to your work, then please ask the module teaching staff for further guidance.

By submitting this assignment, you are confirming that the work is a true expression of your own work and ideas and that you have given credit to others where their work has contributed to yours.

8. Assessment/ marking criteria grid

Category	1st	2:1 / 2:2	Pass / 3rd	Below Pass
Test Script (50)				
Error Handling **15**	Comprehensive testing of potential errors in program execution.	Adequate testing of errors in program execution.	Partial testing of errors in program execution.	Limited or no testing of errors.
Handling Mistakes **15**	Thorough consideration of potential programming mistakes.	Consideration of potential programming mistakes.	Basic consideration of programming mistakes.	Lack of consideration of programming mistakes.
Success and Failure Testing **15**	Comprehensive testing for both successful and unsuccessful scenarios.	Adequate testing for both success and failure.	Basic testing for success and failure.	Limited or no testing for success and failure.
Commented Test Cases **5**	Clear comments describing each test case's purpose and expectations.	Adequate comments on test cases' purposes and expectations.	Basic comments on test cases' purposes and expectations.	Lack of comments or unclear descriptions.
Test Data (20)				
Variety of Maze Files **10**	Diverse maze files provided for testing.	Adequate variety of maze files.	Limited variety of maze files.	Insufficient variety of maze files.
Input Files for Navigation **10**	Comprehensive input files for navigating mazes.	Adequate input files for maze navigation.	Basic input files for maze navigation.	Lack of input files or insufficient for navigation.
Skeleton Code (30)				
Struct Definition(s) **10**	Correct and appropriate struct definitions.	Accurate struct definitions.	Basic struct definitions.	Incorrect or missing struct definitions.

Function/Procedure Definitions **10**	Clear and well-segmented function/procedure outlines.	Adequate function/procedure outlines.	Basic function/procedure outlines.	Unclear or missing function/procedure outlines.
Main Function Outline **10**	Clear logic and structure defined for the main function.	Adequate logic and structure defined for the main function.	Basic logic and structure defined for the main function.	Unclear or missing logic and structure for the main function.