Department of Computing and Mathematics

ASSESSMENT COVER SHEET 2023/24

Unit Code and Title:	6G6Z0036: Programming Languages and Paradigms
Assessment Set By:	Matthew Shardlow
Assessment ID:	1CWK100
Assessment Weighting:	100%
Assessment Title:	Multiparadigmatic Solutions to a Self Proposed Task
Type:	Individual (100%)
Hand-In Deadline:	9pm, 19th Jan 2024
Hand-In Format and Mechanism:	Submission is online, via Moodle

Learning outcomes being assessed:

LO1: Implement solutions to common algorithmic tasks using a range of programming languages and paradigms.

LO2: Compare and contrast design and implementation aspects of core programming concepts using multiple programming languages.

Note: it is your responsibility to make sure that your work is complete and available for marking by the deadline. Make sure that you have followed the submission instructions carefully, and your work is submitted in the correct format, using the correct hand-in mechanism (e.g., Moodle upload). If submitting via Moodle, you are advised to check your work after upload, to make sure it has uploaded properly. If submitting via OneDrive, ensure that your tutors have access to the work. Do not alter your work after the deadline. You should make at least one full backup copy of your work.

Penalties for late submission

The timeliness of submissions is strictly monitored and enforced.

All coursework has a late submission window of 7 calendar days, but any work submitted within the late window will be capped at 40%, unless you have an agreed extension. Work submitted after the 7-day late window will be capped at zero unless you have an agreed extension. See 'Assessment Mitigation' below for further information on extensions.

Please note that individual tutors are unable to grant extensions to assessments.

Assessment Mitigation

If there is a valid reason why you are unable to submit your assessment by the deadline you may apply for assessment mitigation. There are two types of mitigation you can apply for via the unit area on Moodle (in the 'Assessments' block on the right-hand side of the page):

- **Self-certification**: does **not** require you to submit evidence. It allows you to add a short extension (usually, but not always, seven days) to a deadline. This is not available for event-based assessments such as in-class tests, presentations, interviews, etc. You can apply for this extension during the assessment weeks, and the request must be made **before** the submission deadline.

- **Evidenced extensions:** requires you to provide independent evidence of a situation which has impacted you. Allows you to apply for a longer extension and is available for event-based assessment such as in-class test, presentations, interviews, etc. For event-based assessments, the normal outcome is that the assessment will be deferred to the Summer resit period.

Further information about Assessment Mitigation is available on the dedicated Assessments page: https://www.mmu.ac.uk/student-life/course/assessments#ai-69991-0

Plagiarism

Plagiarism is the unacknowledged representation of another person's work, or use of their ideas, as one's own. Manchester Metropolitan University takes care to detect plagiarism, employs plagiarism detection software, and imposes severe penalties, as outlined in the Student Code of Conduct and Regulations for Undergraduate Programmes. Poor referencing or submitting the wrong assignment may still be treated as plagiarism. If in doubt, seek advice from your tutor.

If you are unable to upload your work to Moodle

If you have problems submitting your work through Moodle, you can email it to the Assessment Team's Contingency Submission Inbox using the email address submit@mmu.ac.uk. You should say in your email which unit the work is for, and provide the name of the Unit Leader. The Assessment team will then forward your work to the appropriate person. If you use this submission method, your work must be emailed **before the published deadline**, or it will be logged as a late submission. Alternatively, you can save your work into a single zip folder then upload the zip folder to your university OneDrive and submit a Word document to Moodle which includes a link to the folder. **It is your responsibility to make sure you share the OneDrive folder with the Unit Leader, or it will not be possible to mark your work.**

Assessment Regulations

For further information see Assessment Regulations for Undergraduate/Postgraduate Programmes of Study on the Student Life web pages.

Formative Feedback:	You are encouraged to share your work with your tutor and your peers for discussion and feedback.
Summative Feedback:	You will receive written feedback on your work within 20 working days of submission, in the form of a feedback sheet as shown in Appendix B. There will also be general feedback offered to all students studying the unit.

1. Introduction

The unit is 100% coursework based, and has a single component (1CWK100), weighted at 100% of the unit marks. In summary:

- You will propose a novel task.
    - Solutions for this task should not be easily available online.
    - You may take an existing task and add some sensible variation to it.
    - Your task should be personalized using some identifiable information (e.g., your full name, or your student ID number).
- You will write **Three** solutions to your task, using different programming languages and paradigms.
    - Your solutions must each exhibit a range of paradigmatic features
    - Your solutions must each use a unique programming language
- For each solution, you will provide:
    - (a) the name of the language and paradigm you have used.
    - (b) Screenshots of your design process, demonstrating how you created the code and the validation process you used to ensure that it was suitable for your task
    - (c) a short description of your programme, explaining how the code you have written completes the task and how your programme fits the named paradigm.
- The deliverables are:
    - Your uniquely proposed task description (unmarked)
    - A folder containing 3 subdirectories, one per language.
    - Each subfolder must contain:
        - A text file with your code
        - A word document or pdf documenting your design process (see Appendix A)
        - A word document or pdf with your code explanation
        - Assessment Overview (1CWK100)

a) Propose a novel task

You should write your own task description. This should be a new task, for which there are no solutions readily available via web search (or at worst, very few). You may base your task on an existing task, for example by starting with that task and then adding your own elements of specification to make the task unique.

For example, you may start with the task of developing a tic-tac-toe game. To make this unique you could implement some additional rule, or rules to the game, such as making the board larger (e.g., 7x7, or NxM) or implement some extra gameplay rule (e.g., you can choose to remove one of the opponents O's or X's every 3rd turn. You should be imaginative in creating your task to ensure it is unique to you and different to those in the rest of the class.

To further personalise your task you should incorporate either your student ID number or your name into the task description and code. This helps for future plagiarism detection. For example, you may seed a random number generator with your student ID, or if your task includes some text element

then you may use your name for this (e.g., a cryptograph). You should make it clear in your task description how the element of personalization will be done.

I am not making a limit to the difficulty or ease of the task. You should choose a difficulty level that you feel is appropriate to your coding ability and that will set you an appropriate challenge. Solutions to more difficult problems will likely expose more interesting features of the languages and paradigms, leading to the opportunity to score more highly on the assessed documents. As a rough guide, your task should be more difficult than a typical lab-exercise (e.g., fizzbuzz with the numbers changed is probably too simple), but less difficult than a typical end of year assignment (e.g., you should not propose to implement a full-fledged mobile app or web front end).

b) Choose three different languages

You should select three different languages to use to solve your task. These languages **must** be selected from those taught during the unit. You can refer to Moodle for a full list of languages that have been covered. The languages you choose should allow you to solve the task in a variety of programming styles. You must use a different programming paradigm for each solution and your choice of language should reflect this. The five programming paradigms we cover are as follows: Imperative, Procedural, Object-Oriented, Functional, Logic.

c) Create Solutions

You should write a bespoke solution in each language, conforming to a paradigm. You are allowed to use a large language model, or copilot to aid in your programming. Please ensure that your code is appropriately indented, well commented, and conforms to appropriate standards for the language you are coding in (e.g., variable naming conventions, etc.). You are welcome to use the same approach to solve the task you have designed across your three solutions, however you should design your solutions in such a way that the specific paradigmatic features of each language you have used may be properly showcased.

d) Document Solutions

You must provide two distinct documents for each language.

The first document should show your design and development process for your solution in that language. You should include any sketches, UML diagrams, class diagrams, pseudocode or wireframes that you create. There may be some crossover in your design work between languages, but you should still document this for each language. As part of your design document you should also capture the development process that you have undertook including testing and bug-fixing. You should include intermediary screenshots of your design process. If you use a large language model or copilot as part of your programming, you should include screenshots of all interactions, as well as some indication as to how you used this information and how you validated the results. This document may be presented in a 'scrapbook' format, and should be made mostly of images or figures (with short connecting texts) collected during your design and implementation process.

The second document is a short paragraph (max 300 words) describing the language features that you have used to solve the solution, and explaining how the solution conforms to the stated paradigm. A typical solution might spend 200 words on the former and 100 words on the latter, although this will vary from one language to another.

You should not use a language model to produce either of these documents. They will typically produce hallucinated documentation or false reasoning for this type of task which will impede your marks. If you do choose to use a language model you must include all interactions as screenshots as an appendix to the specific document that they are relevant to.

e) Compare Solutions

Finally you should write a comparison of your three solutions. Your comparison should be no more than 1000 words and should highlight similarities and differences between your solutions in each pair of languages, especially considering the paradigms that you have conformed to. A typical solution might spend around 150 words introducing the three languages and paradigms, then 250 words per language pair highlighting similarities and differences in approach, with 100 words reserved for a summary conclusion. You may assume that the marker is aware of your task and has read your design and explanation documents.

Again, you should not use a language model to produce your comparison document as they are not suitable for this task and are likely to give incorrect answers, harming your chance to succeed. If you do use a language model, you must include screenshots of your interactions as an appendix to your comparison document.

2. The Submission

Your submission is via Moodle. You must submit a zip file containing a folder. The folder should have your ID number as its name. Inside the folder you should place:

(a) a text file containing the task description you have written.
(b) A word document or PDF containing a comparison of your solutions, highlighting similarities and differences in how you used specific paradigmatic features to solve each task.
(c) 3 sub-folders. Each sub-folder should have the name of the programming language that you have used for that solution. Inside each sub-folder you must have:

(c.1) a text file containing the code you used named **LANG_code.txt**, where LANG is replaced with the language you have used.

(c.2) A Word or PDF document containing your Design documentation, named **LANG_design**, where LANG is replaced with the name of the language you have used.

(c.3) A word or PDF document containing a statement of the paradigm that you have used and your explanation of how the code you have written meets your stated paradigm. This should be named **LANG_paradigm**, where LANG is replaced with the language you have used.

A sample file hierarchy is given below, Note, you are free to choose any 3 languages from the course:

- 99999999
  - Task.txt

- Comparison.docx
- Python
  - Python_code.txt
  - Python_design.docx
  - Python_paradigm.docx
- Prolog
  - Prolog_code.txt
  - Prolog _design.docx
  - Prolog _paradigm.docx
- GO
  - GO_code.txt
  - GO_design.docx
  - GO_paradigm.docx

3. Mark Scheme

Marks will be apportioned as follows:

For each language (25% of overall grade per language):

Design Document (10 marks)

Paradigm Document (15 marks)

Comparison (25% of overall grade)

Individual mark schemes for each section are given below:

Design

0 marks: No documentation.

1-3 marks: little design and implementation documentation work, or design and implementation documentation is incoherent and unrelated to submitted code.

4-6 marks: Adequate design and implementation documentation work. Design and implementation documentation is related to submitted code.

7-10 marks: Excellent and extensive design and implementation documentation work. Documentation goes beyond usual expectations for a final year undergraduate student.

Paradigm

0 marks: No documentation.

1-5 marks: Paradigm is incorrectly identified. Poor description of features, with little relationship to the code.

6-10 marks: A paradigm is stated with appropriate reasoning. Most features are correctly described, with few to no errors.

11-15 marks: Paradigm is correctly identified. Outstanding description of features, showing exceptional understanding of how the given paradigm is used.

Comparison

0 marks: No Comparison.

1-5 marks: An inadequate comparison, covering an incomplete set of paradigms. Little or no criticality in evaluation.

6-15 marks: An adequate level of comparison. At least two paradigms are correctly compared. Some appropriate features are identified and equivalencies are demonstrated in solutions with little or no errors.

16-20 marks: A good degree of comparison. All paradigms are compared appropriately. A complete set of features is identified with no errors made. High level of criticality and understanding of programming paradigms.

20-25 marks: An excellent degree of comparison, above and beyond the reasonable expectations for the final year of study. Each paradigm is compared to the other two paradigms. Highly coherent analysis of features used.

4. Feedback

Your feedback on the assessment will consist of an assigned marking boundary for each element as given above. You will also receive summary feedback indicating positive points of the assignment, as well as an indication of areas that you have lost marks. An example feedback sheet is given in Appendix B.

5. Support for the Assignment

a) Help! I don't know where to begin or what to do!

You should start by identifying the task and languages that you will use. Once you have decided on these, the rest of the assignment should fall into place. You may wish to discuss ideas with your peers in

order to get an understanding of whether the scope of your proposal is appropriate, but make sure you submit a different task to those you have discussed with.

b) Opportunities for Formative Feedback

You will be given an opportunity to submit your task description for formative feedback during the semester. See Moodle for the submission date. The feedback you receive will typically either be a confirmation that this task is acceptable, or a suggested modification to improve the task. If you miss the deadline, I will be unable to provide ad-hoc formative feedback for task descriptions.

c) Your Final Feedback

You will receive an overall mark, a breakdown of that mark according to the mark scheme (Section 4). You will also receive a short comment on what went well, allowing you to attain the given grade boundary and what could have been improved to attain the next boundary.

d) How do I contact the unit tutor?

If you want to ask any questions on the requirements of the assessment then please do get in touch with me via Email, Teams, or face-to-face during my weekly office hours which will be held in the learning studio. See Moodle for my contact details.

Appendix A – Example Design and Implementation Document

I decided to implement the modified tic-tac-toe programme using Haskell. My high-level pseudocode for the overall algorithm is as follows:

1. Represent board as list of lists
2. Implement recursive function to go through a single list determining if there is a win (rows)
3. Implement recursive function to go through a list of lists determining if there is a win (cols)
4. Implement recursive function to go through a list of lists determining if there is a win (diagonal)
5. Implement function to add a 'M' or 'S' in a row-col position. Signature: char, [[Integer]] -> [[Integer]]
6. Implement function to remove a 'M' or 'S'. (reuse above function?)
7. Implement function to govern game logic
    a. M goes first, then S
    b. At each iteration get a number (1-49) indicating cell to play in
    c. Every 3rd turn players can remove a cell

I have provided screenshots of the pseudocode that I wrote on my whiteboard for each function below:

Recursive function: [SCREENSHOT 1]

Add/remove char to board: [SCREENSHOT 2]

Game Loop: [SCREENSHOT 3]

During my implementation process, I wrote the following code as a first iteration:

[SCREENSHOTS OF CODE]

This allowed me to identify the following errors in my approach, which led me to redesign my system as follows:

[SCREENSHOTS OF ERRORS AND UPDATED CODE]

Once I had a working system, I decided to test it. The tests that I ran are as follows:

1) Run to the end with M player winning
2) Run to the end with S player winning
3) Run to the end with a draw.

[SCREENSHOTS OF TESTING]

Appendix B – Feedback Sheet

Marker Name: Matthew Shardlow

Student Name: Matthew Shardlow

Student ID: 99999999

Solution 1		**Solution 2**		**Solution 3**			
Design (10)	**Explanation** (15)	**Design** (10)	**Explanation** (15)	**Design** (10)	**Explanation** (15)	**Comparison** (25)	**Total** (100)
7	11	3	8	10	15	17	71

This submission contains a tic-tac-toe game with a modified board design and an additional rule to allow players to remove their opponents tiles. Solutions 1 and 3 (Haskell and C++) were well implemented in the functional and OO paradigms. The OO structure in C++ was exceptionally well designed and led to efficient code. Solution 2 failed to use Rust correctly and did not state the paradigm that was being used.

I have used the criteria below to mark your work. You can see a further breakdown of your marks by matching your assigned grade to the given band for each category.

Design

0 marks: No documentation.

1-3 marks: little design and implementation documentation work, or design and implementation documentation is incoherent and unrelated to submitted code.

4-6 marks: Adequate design and implementation documentation work. Design and implementation documentation is related to submitted code.

7-10 marks: Excellent and extensive design and implementation documentation work. Documentation goes beyond usual expectations for a final year undergraduate student.

Explanation

0 marks: No documentation.

1-5 marks: Paradigm is incorrectly identified. Poor description of features, with little relationship to the code.

6-10 marks: A paradigm is stated with appropriate reasoning. Most features are correctly described, with few to no errors.

11-15 marks: Paradigm is correctly identified. Outstanding description of features, showing exceptional understanding of how the given paradigm is used.

Comparison

0 marks: No Comparison.

1-5 marks: An inadequate comparison, covering an incomplete set of paradigms. Little or no criticality in evaluation.

6-15 marks: An adequate level of comparison. At least two paradigms are correctly compared. Some appropriate features are identified and equivalencies are demonstrated in solutions with little or no errors.

16-20 marks: A good degree of comparison. All paradigms are compared appropriately. A complete set of features is identified with no errors made. High level of criticality and understanding of programming paradigms.

20-25 marks: An excellent degree of comparison, above and beyond the reasonable expectations for the final year of study. Each paradigm is compared to the other two paradigms. Highly coherent analysis of features used.