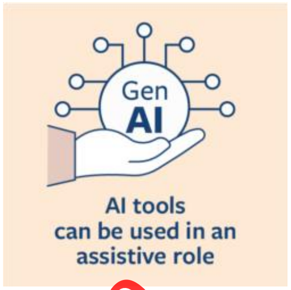


School of Computing: Assessment brief

Module title	Programming Project
Module code	COMP1921
Assignment title	Resit Assessment
Assignment type and description	You will produce code to meet a given specification.
Rationale	This assessment gives you an opportunity to develop a small working game using professional techniques such a modularizing code and defensive design. You will work to create code which meets a client brief, whilst also ensuring that the code meets basic standards of structure, documentation and memory management.
Word limit and guidance	You should spend 20-25 hours working on this assessment.
Weighting	100%
Submission deadline	9th August 2024 @ 23:59 **Late submission is not permitted.**
Submission method	Gradescope
Feedback provision	Marked rubric, Autograder output & in-code comments via Gradescope.
Learning outcomes assessed	- apply professional programming practices to programming projects. - 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	Amy Brereton (@scsabr)

1. Assignment guidance

You are tasked with creating a basic treasure hunt game in **C**, where players navigate a map to find hidden treasures using command-line inputs.

You should design your code to be **defensive** and handle a range of errors in a graceful way, without crashing unexpectedly. Consider the full range of mistakes which a user could make when trying to run the program.

Treasure Island Game

The game loads a map file, which is provided on the command line in the format:

./island <mapfilepath> <dimension>

A map file can contain:

Symbol	Meaning
' ' (space)	Land which the player can move across.
'w' (lower case w)	Water – this surrounds the island and the player cannot move through it. There must be only water in the first and last row and column.
'T' (upper case t)	Palm trees which block the player from moving on land.
3 x 'H' (upper case h)	Hidden treasures which the player is searching for.
1 x 'S' (upper case s)	The starting point, where your player will be placed when you start the game.

And is always a square with the width and height ***dimension*** which is also provided on the command line.

The game involves the player moving around the island looking for hidden treasure. The player can move using the WASD keys (w/W = up, a/A = left, s/S = down, d/D = right) or display a map using the m/M key.

The locations of the hidden treasure and the starting point should **not** be shown by the map, these should be showed by blank spaces ' '.

When the player finds the hidden treasure, they should receive some message telling them how many they have found such as 'You have found 1 out of 3 hidden treasures'.

When the player has found all 3 hidden treasures, they have won and the game ends successfully.

There is no exit or quit option, so the only way to complete the game is to find all treasures.

Return codes and Outputs to the User

Any outputs such as error messages can be any text you like, as the grader does not read them. However, there are certain return codes which you have to use:

0 = success (the game was able to run correctly)

1 = argument error (bad number of arguments, or bad dimension)

2 = file error (the file cannot be read – doesn't exist or no read permissions)

3 = data error (the file is not a valid map)

Where an error could fall into multiple categories, the autograder will accept either – or you can ask me via Teams.

Map Files

Map files are text files containing a 'map' for the game. They have some rules:

- The map should always have a border of water ('w's) around the edges (i.e. every first and last character of a row and column should be a 'w').
- A map is always a square (width and height equal), and should match the dimension given on the command line.
- There is exactly one start point marked by 'S'.
- There are exactly 3 treasures marked by 'H'.
- The map only contains characters 'w', 'T', ' ', 'H' and 'S'.
- The size is a minimum of 5x5 and a maximum of 100x100.
- They may end with a trailing newline character (a '\n' as the final character).

A selection of map files have been provided to help you test your code – note that these will **not** be the final files used to test your code, so it's important for you to ensure that your code works on a variety of different files.

These example files also **do <u>not</u> contain every possible error** – try and think of other ways in which a map file could be wrong, and make some of your own to test your code.

You do not need to check whether there is a valid route between the start and the 3 treasures – you can assume there always is.

Additional Task – Map Generator – 30 marks

This task is optional and should not be attempted if you are sitting a capped resit – this is only for those with uncapped marks who are aiming for higher marks and may take significantly longer than the suggested time for this assignment.

The developer wants to **procedurally generate** a range of different maps to build up a website of maps which people can download and use with the game. They would like you to create a script which is able to generate these maps.

You may use C, Python, or Java for this extension.

You will produce a program which can generate maps with a given filename and size. For example:

```
./islandGenerator new_island.txt 40
```

Would create a random, solvable 40x40 map and save it into new_island.txt.

Note: as other languages are permitted, please provide running instructions in a readme.md file.

Your islands need to be valid by the rules given in the 'Map Files' section above, and it is recommended that you try and ensure that around 60% of your map is covered by island (rather than having a lot of water around small islands) to make your islands more interesting and varied. Your islands must also be **solvable** – it must be possible to start at the starting point and reach all 3 treasures.

You will develop an algorithm to produce these more complex maps. You may use existing research to help you to do this, but you should also experiment with how changing existing algorithms affects the maps you produce. You should be writing all code **yourself,** and citing any research you use.

Aim to create an algorithm which produces visually interesting and challenging maps, and which produces an interesting range of shapes and styles of island.

You will produce a short report which explains how you developed your island-generating algorithm, focusing on how you iteratively developed and improved your code, justifying changes you made and explaining the impact of these. You can include screenshots, code snippets and images to demonstrate this

You should also include a **reflective conclusion** discussing:

- The limitations of your solution
- What you found challenging in designing the algorithm
- Future improvements you would like to make

I recommend writing no more than 10 pages (including images and code snippets) but there is no page or word limit.

You should ensure that you cite any sources using Leeds Harvard referencing.

Please upload your report as a PDF.

2. Assessment tasks

You should develop a C program to fulfil the brief given above. You will submit your source code, and if you attempt the map generator challenge task you will also submit a short report.

Your code should be:

- Defensively designed
- Sensibly structured
- Modular
- Memory efficient

And you should ensure that you test your code throughout development. On submission, you will receive feedback for some tests which should help you to ensure that you are meeting the requirements of the specification such as correct exit codes.

If tests are failing and you are not sure why, you can contact me via Teams/email for additional feedback.

3. General guidance and study support

You should refer to the previous lab exercises and lecture notes to support you. Procedural Programming covered the basic C code needed so you should refer back to this module's notes.

4. Assessment criteria and marking process

50 marks will be calculated by an autograder which runs your code through a number of scenarios testing invalid inputs, files, and some integration tests ensuring your code can navigate a full game. You will see the result of a small number of these tests on upload, but the majority are hidden.

20 marks for code quality will be manually assessed by code inspection.

30 marks for the extension task will be manually assessed from your report and running your code.

A full breakdown is available in section 8.

5. Presentation and referencing

In your report, you should use Leeds Harvard referencing which you can learn more about:
https://library.leeds.ac.uk/info/1402/referencing

The quality of written English will be assessed in this work. As a minimum, you must ensure:

- Paragraphs are used
- There are links between and within paragraphs although these may be ineffective at times
- There are (at least) attempts at referencing
- Word choice and grammar do not seriously undermine the meaning and comprehensibility of the argument
- Word choice and grammar are generally appropriate to an academic text

Referencing of Code

Two simple rules:

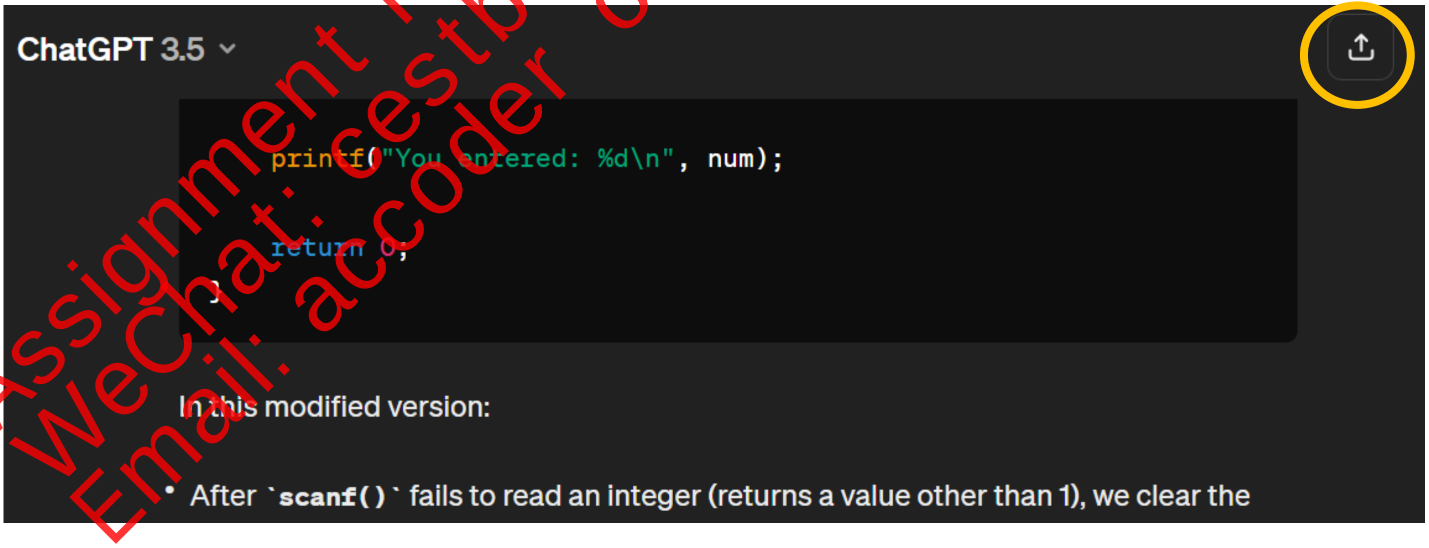
1. **You should not be directly copying any code from external resources, even with a reference.**
2. **Use of generative AI needs to be referenced with a link/copy of the conversation.**

If any code is adapted from examples found online, provide a basic comment with the URL on the line above the adapted line/section:

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

Generative AI

In ChatGPT, you can generate a link to the full conversation:



```
    printf("You entered: %d\n", num);

    return 0;
}
```

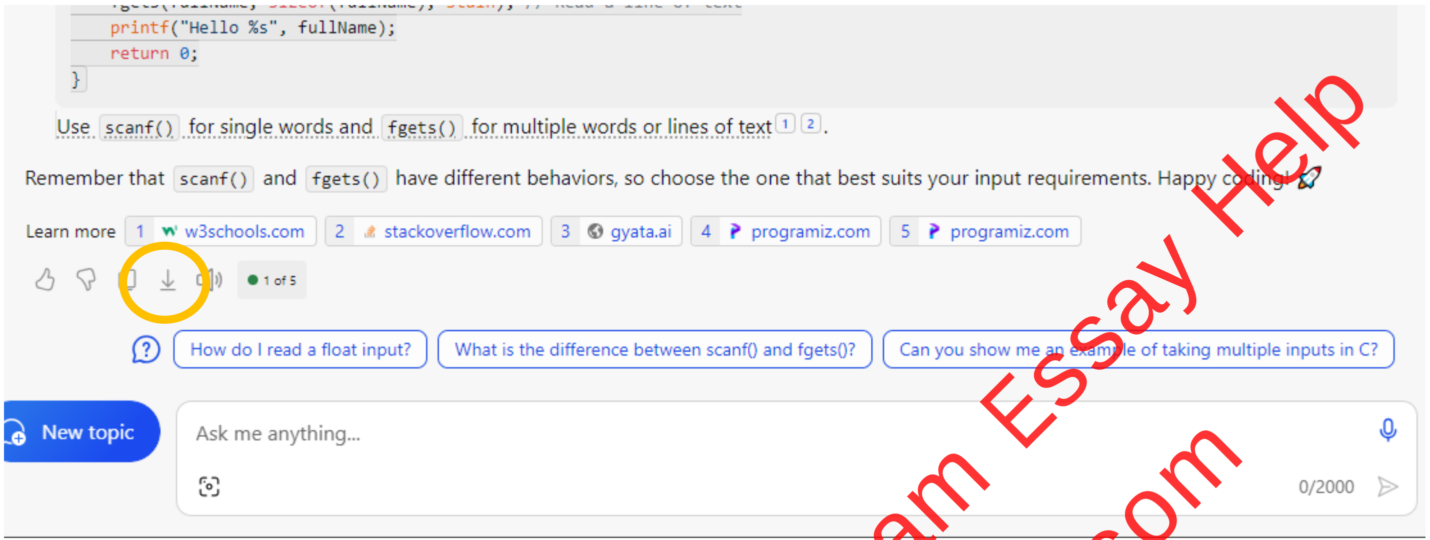
In this modified version:

- After `scanf()` fails to read an integer (returns a value other than 1), we clear the

And provide the reference as follows:

```
// Lines 1 – 7 were adapted from code provided by the following conversation
with chatGPT: https://chat.openai.com/share/c356221d-fb88-4970-b39e-d00c87ae1e0b
```

In Copilot, you will need to export the conversation as a text file:



```
    printf("Hello %s", fullName);
    return 0;
}
```

Use `scanf()` for single words and `fgets()` for multiple words or lines of text [1] [2].

Remember that `scanf()` and `fgets()` have different behaviors, so choose the one that best suits your input requirements. Happy coding! 🚀

Learn more 1 w3schools.com 2 stackoverflow.com 3 gyata.ai 4 programiz.com 5 programiz.com

1 of 5
How do I read a float input?
What is the difference between scanf() and fgets()?
Can you show me an example of taking multiple inputs in C?
New topic
Ask me anything...
0/2000

Save this with a filename including the date and 2-3 word summary of what the conversation was about ('11-03 inputs in C.txt') and ensure this is submitted with your work.

You can reference this in your code:

```
// Lines 1 – 7 were adapted from code provided by the CoPilot conversation
recorded in '11-03 inputs in C.txt'
```

If you are using a different Generative AI model, these instructions may differ – you must still provide a link to or copy of the full conversation and reference in the same manner above.

Use of Generative AI in this Assessment

This assessment is rated 'amber' according to the university guidelines around generative AI. This means that you can use genAI models such as ChatGPT or CoPilot to explain concepts which may be useful in this assessment, but you must not **use any code it generates or give it any part of this specification.**

Here are some examples of **reasonable** things to ask a generative AI model:
- Explain how to use the fgets function to read a file in C
- How do I create a struct in C?
- How do I allocate a 2D array in C?

These are asking for help with concepts, and not with the assignment itself and are therefore acceptable – although you **must** reference your use of generative AI with a full transcript of the conversation, as shown in the section above.

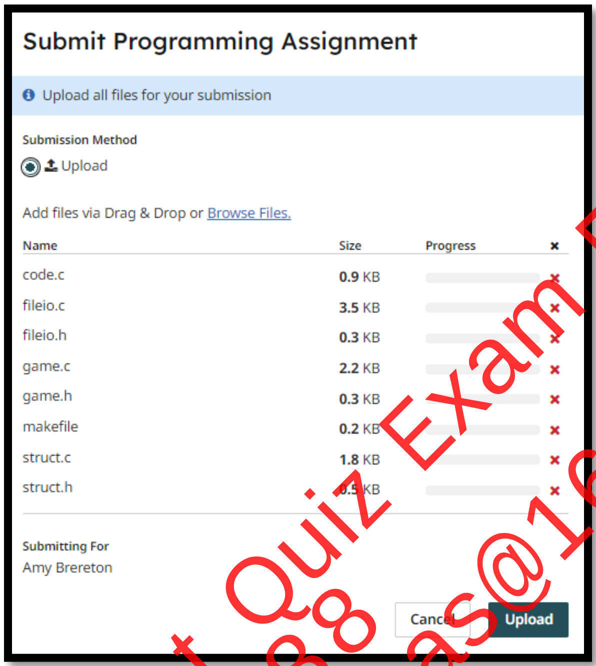
If it is suspected that you have used generative AI without reference, the standard academic integrity process for plagiarism will be followed.

6. Submission requirements

Submit via Gradescope.

Main task:

Submit your code and a makefile to Gradescope along with any referenced generative AI conversations. Your code should not be inside any subfolders, and must compile on Linux.



Submit Programming Assignment

Upload all files for your submission

Submission Method

Upload

Add files via Drag & Drop or Browse Files.

Name	Size	Progress
code.c	0.9 KB	
fileio.c	3.5 KB	
fileio.h	0.3 KB	
game.c	2.2 KB	
game.h	0.3 KB	
makefile	0.2 KB	
struct.c	1.8 KB	
struct.h	0.5 KB	

Submitting For
Amy Brereton

This is an example of a correct upload – you can see that my files do not have a folder name before them, and there is a makefile provided.

The autograder will show the result of **4 different tests,** one from each section (arg errors, file errors, map errors and success tests). Use these to ensure your code is returning the correct value.

Extension task:

Submit your code, report, and any instructions for running your program to the 'Resit – Extension' assignment on 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	3rd / Pass	Fail
Island Game (70)				
Functionality 50 AUTOGRADED	Game works with very few/no errors.	Game works relatively well with some issues.	Game semi-functional with significant errors.	Severe functionality issues; game unplayable or non-functional.
Code Structure 10	Code is readable and well structured. It has been documented clearly with doc comments and some in-code comments.	Code is generally easy to follow, with a good attempt at clearly documenting code with comments.	There has been some attempt to make code readable, with sensible comments used.	Code is poorly structured and lacks documentation.
Memory Management 5	Dynamic memory allocation used appropriately, with memory freed before all exits.	Dynamic allocation used, with frees before the majority of exits.	Dynamic allocation attempted although may be inconsistent.	No attempt at dynamic allocation.
Modularity 5	Code has been split into sensible modules (files) with good functional breakdown.	There is at least one additional module (file) and reasonable functional breakdown.	Code may all be in one file, but functional breakdown is acceptable.	Code all in one function, or functional breakdown is very poor.

Island Generator (30)				
Functionality 10	Generates interesting maps which meet the criteria and are varied and consistently playable.	Mostly generates good maps, but may have some limitations which do not prevent the game from being playable.	Generates maps with significant limitations which may impact playability.	Fails to generate valid or interesting maps.
Algorithm design 10	It is clear how the algorithm was developed and the changes made to improve it. There are sensible justifications for changes made.	It is clear how the algorithm was developed, although there may be limited iterations or little justification for changes.	Not fully clear how the algorithm was developed, but there is some narrative.	Unclear how algorithm was developed, or lack of explanation of changes.
Reflection 10	There is a clear understanding of the current solution and any limitations, with a good critical analysis provided.	Reflects on the iterative process but may lack critical analysis.	Limited reflection on the process, limitations, or improvements.	Minimal or absent reflection.