Curtin University — Discipline of Computing

Unix & C Programming (COMP1000)

Semester 2, 2021

Assignment One

Due: Monday, 13 September 2021, 11:59 PM (GMT+8)

Weight: 15% of the unit

Your task for this assignment is to design, code (in C89 version of C) and test a program. In summary, your program will:

- Read the metadata from the provided function.
- Able to create dynamically-allocated 2D char array to make a simple ASCII-based game.
- Receive user input to control the flow of the game.
- Write a suitable makefile. Without the makefile, we will assume it cannot be compiled and it will negatively affect your mark.

1 Code Design

You must comment your code sufficiently in a way that we understand the overall design of your program. Remember that you cannot use "//" to comment since it is C99-specific syntax. You can only use "/*.....*/" syntax.

Your code should be structured in a way that each file has a clear scope and goal. For example, "main.c" should only contain a main function, "game.c" should contain functions that handle and control the game features, and so on. Basically, functions should be located on reasonably appropriate files and you will link them when you write the makefile. DO NOT put everything together into one single source code file. Make sure you use the header files and header guard correctly. Never include .c files directly.

Make sure you free all the memories allocated by the malloc() function. Use valgrind to detect any memory leak and fix them. Memory leaks might not break your program, but penalty will still be applied if there is any. If you do not use malloc, you will lose marks.

Please be aware of our coding standard (can be found on Blackboard) Violation of the coding standard will result in penalty on the assignment. Keep in mind that it is possible to lose significant marks with a fully-working program that is written messily, has no clear structure, full of memory leaks, and violates many of the coding standards. The purpose of this unit is to teach you a good habit of programming.

2 Academic Integrity

This is an assessable task, and as such there are strict rules. You must not ask for or accept help from anyone else on completing the tasks. You must not show your work at any time to another student enrolled in this unit who might gain unfair advantage from it. These things are considered plagiarism or collusion. To be on the safe side, never ever release your code to the public.

Do NOT just copy some C source codes from internet resources. If you get caught, it will be considered plagiarism. If you put the reference, then that part of the code will not be marked since it is not your work.

Staff can provide assistance in helping you understand the unit material in general, but nobody is allowed to help you solve the specific problems posed in this specification. The purpose of the assignment is for you to solve them on your own, so that you learn from it. Please see Curtin's Academic Integrity website for information on academic misconduct (which includes plagiarism and collusion).

The unit coordinator may require you to attend quick interview 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. In addition, your assignment submission may be analysed by systems to detect plagiarism and/or collusion.

3 Task Details

3.1 Quick Preview

First of all, please watch the supplementary videos on the Assignment link on Blackboard. These videos demonstrates what we expect your program should do. You will implement a simple maze game that you can play on Linux terminal. Further details on the specification will be explained on the oncoming sections.

3.2 Command Line Arguments

Please ensure that the executable is called "maze". Your executable should accept one command-line parameters/arguments:

./maze <visibility_distance>

<visibility_distance> is an integer that defines the max horizontal/vertical distance on which the area is visible from the player's position on the map. If this value is zero, then the whole map is visible throughout the game. For more detail, please refer to Section 3.6.

If the amount of the arguments are too many or too few, your program should print a message on the terminal and end the program accordingly (do NOT use exit() function). If the integer is a negative number, please end the program in the same manner as well.

Note: Remember, the name of the executable is stored in variable argv[0].

3.3 Metadata and Map creation

Please watch the supplementary video on this topic for better intuition. Before you can print the map and start playing the game, you need to generate the 2D char map array that contains all the necessary objects in the map. In order to do generate the 2D map array, you need to retrieve the metadata information from the function getMetadata() in map.c.

```
C map.c > getMetadata(int ***, int *, int *, int *)
1   #include<stdlib.h>
2   #include"map.h"
3
4   /*  This file and the corresponding header file (map.c and map.h) contains a
5       single function you can use to get some metadata that you need to create
6       the 2D map array in the main function.
7
8       DO NOT add any new function here. We will test it with our own map.c
9       when we mark it. DO NOT change the name of the function and the filename
10      (getMetadata function,  map.c and map.h)
11
12      If you want to make your own metadata for the map (e.g adding wall),
13      you can modify the metadata table here. Once again, keep in mind that
14      we will test your assignment with our own metadata. */
15
16  void getMetadata(int *** metadataTable, int * metadataAmount, int * mapRow, int * mapCol)
17  {
```

You need to pass the address of a double pointer to int, and addresses of 3 integer variables to this function. The function will fill the value for you, and you can use the information to allocate memory for the 2D map array and fill the map. This is the list of the metadata you can retrieve from the function:

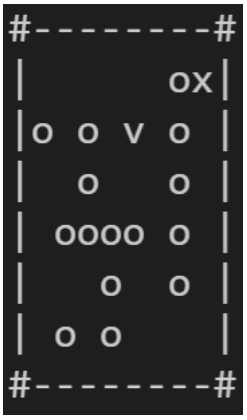
- metadataTable – This is the table containing the locations of all the objects in the playable area of the map. Please read the comment in map.c for more detail.
- metadataAmount – This is the amount of the objects (player, goal, walls) in the playable area of the map. This tells us the amount of the rows in metadataTable.
- mapRow – This is the row size of the 2D map array you will create later.
- mapCol – This is the column size of the 2D map array you will create later.

Note: Since the edge of the map will contain the impassable border, the playable area size of the map is (mapRow-2) x (mapCol-2).

Note: Keep in mind that we will use our own map.c file when we mark your assignment. Therefore, please do not create any new function on map.c because we will overwrite it. Please do not change the name of the function as well.

3.4 Main Interface

Once you run the program, it should clear the terminal screen (watch the video about the method to clear the screen) and print the 2D map:



This is the interface where the user plays the maze game. The user can type the command to move the player. Every time the user inputs the command, the program should update the map accordingly and refresh the screen (clear the screen and reprint the map). Please refer to Section 3.5 for more detail on the user input.

3.5 User Input

The user only needs to type 1 character for each command (lower case). Here is the list of the possible commands:

- **'w'** moves the player one block above, ONLY if there is an empty space. It will change the player character to '^'.
- **'s'** moves the player one block below, ONLY if there is an empty space. It will change the player character to lowercase 'v'.
- **'a'** moves the player one block left, ONLY if there is an empty space. It will change the player character to '<'.
- **'d'** moves the player one block right, ONLY if there is an empty space. It will change the player character to '>'.
- Any other character should be ignored, and nothing changes on the map. You can add a warning message if you want.

Your program should be able to receive each input without pressing "enter" key everytime. Please refer to the supplementary video for a sample code to disable the Echo and Canonical feature on Linux terminal temporarily. If your terminal is stuck on this mode, you only need to re-open a new terminal.

Note: Every action should update the 2D map array, clear the terminal screen and reprint the map.

3.6 Visibility Distance (with Conditional Compilation)

Do you remember the integer you read as the command line argument? It is now the time to use it. Please watch the supplementary video that explains this section. This integer determines the visibility range of the player of the map. Value of zero means that the whole map is always visible at all time. Any value greater than 0 defines the visibility range of the player in the square shape around it. For example, the value of 2 means that you can only see 5x5 square area around the player (2 blocks to the left, right, up, and down). For simplicity, the player can see beyond the wall as long as it is within the visibility range. Part of the map that is outside the visibility range will NOT be printed on the terminal.

This visibility feature will only be included if you activate the conditional compilation through makefile (Please see the practical supplementary video for an example). The name to be defined is DARK. If this DARK is not defined, then the default behaviour is to always print the whole map regardless the value of the integer.

Note: You can imagine as if you are holding a lit torch inside a dark cave, but the illuminated area is a square shape.

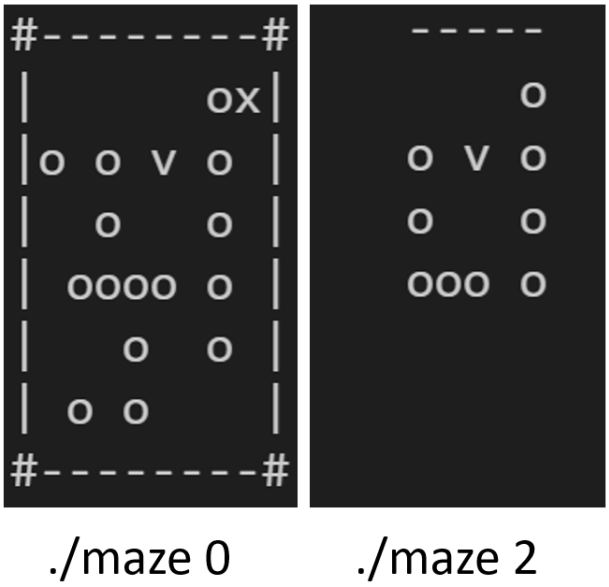


Figure 1: If you define the DARK through conditional compilation, the command line argument decides the visibility of the map. 0 means show everything. Anything greater than 0 define the visibility range of the player.

3.7 Winning Condition

The game will continue until the player reach the goal (represented by 'x'). You can assume that there is always a path to win the game. Once this occurs, you can print a celebratory message such as "You Win!" before ending the program. Please do not forget to **free** all the memories allocated via malloc() function before the program ends.

3.8 Makefile

You should manually write the makefile according to the format explained in the lecture and practical. It should include the Make variables, appropriate flags, appropriate targets, correct prerequisites, conditional compilation, and clean rule. We will compile your program through the makefile. If the makefile is missing, we will assume the program cannot be compiled, and you will lose marks. DO NOT use the makefile that is generated by the VScode. Write it yourself.

3.9 Assumptions

For simplification purpose, these are assumptions you can use for this assignments:

- The coordinates of all objects provided in the map.c are all valid. (NOT out of bound)
- The metadata provided in map.c will contain all the necessary components such as the player and the goal location.
- The size of the map will be reasonable to be displayed on terminal. For example, we will not test the map with gigantic size such as 300 x 500. It is the the same for the opposite scenario, we will not test your program with the map size less than 5 x 5.
- When we mark your assignment with our own map file, you can assume the path from the player to the goal exist. So, it means the game is winnable.
- You only need to handle lowercase inputs ('w', 's', 'a', 'd').
- The command line argument is always an integer. However, you need to check whether it is a positive or negative number. (Keep in mind that it will be stored as a string in argv.)

4 Marking Criteria

This is the marking distribution for your guidance:

- Properly structured makefile according to the lecture and practical content (5 marks)
- Program can be compiled with the makefile and executed successfully without immediate crashing and showing reasonable output (5 marks)
- Usage of header guards and sufficient in-code commenting (2 marks)
- The whole program is readable and has reasonable framework. Multiple files are utilized with reasonable category. If you only have one c file for the whole assignment (not counting map.c), you will get zero mark on this category. (3 marks)
- Avoiding bad coding standard. (5 marks) Please refer to the coding standard on Blackboard. Some of the most common mistakes are:
  - Using global variables
  - Calling exit() function to end the program abruptly
  - Using "break'' not on the switch case statement
  - Using "continue''
  - Having multiple returns on a single function
  - include the .c files directly, instead of the .h files
- No memory leaks (10 marks) Please use valgrind to check for any leaks. If your program is very sparse OR does not use any malloc(), you will get zero mark on this category.
- **FUNCTIONALITIES**:
  - Correct command line arguments verification with proper response (5 marks)
  - Proper memory allocation for the 2D map array based on the metadata from map.c file. All the objects are allocated correctly on the map. The program should work with general metadata (15 marks)
  - Able to clear the screen and re-print the map on every action (5 marks)
  - Able to move the player with the command from the user. The player should not be able to go through the wall. The player icon changes everytime the player changes direction (10 marks)
  - Successfully disable the Echo and Canonical temporarily so that the user does not need to press the "enter" key to issue command (5 marks)
  - Implement the visibility range feature based on the command line argument (20 marks)
  - Visibility range feature should be associated with the conditional compilation (5 marks)
  - Winning the game when the player reaches the goal and closes the program afterwards. (5 marks)

Note: Remember, if your program fails to demonstrate that the functionality works, then the mark for that functionality will be capped at 50%. If your program does not compile at all OR just crashed immediately upon running it, then the mark for all the functionalities will be capped at 50% (For this assignment, it means maximum of 35 out of 70 marks on functionalities). You then need describe your code clearly on the short report on the next section.

5 Short Report

If your program is incomplete/imperfect, please write a comprehensive report explaining what you have done on for each functionality. Inform us which function on which file that is related to that functionality. This report is not marked, but it will help us a lot to mark your assignment. Poorly-written report will not help us to understand your code. Please ensure your report reflects your work correctly (no exaggeration). Dishonest report will lead to academic misconduct.

6 Final Check and Submission

After you complete your assignment, please make sure it can be compiled and run on our Linux lab environment. If you do your assignment on other environments (e.g on Windows operating system), then it is your responsibility to ensure that it works on the lab environment. In the case of incompatibility, it will be considered "not working'' and some penalties will apply. You have to submit a single zip file containing ALL the files in this list:

- **Declaration of Originality Form** – Please fill this form digitally and submit it. You will get zero mark if you forget to submit this form.
- **Your essential assignment files** – Submit all the .c & .h files and your makefile. Please do not submit the executable and object (.o) files, as we will re-compile them anyway.
- **Brief Report** (if applicable) - If you want to write the report about what you have done on the assignment, please save it as a PDF or TXT file.

Note: Please compress all the necessary files in a SINGLE zip file. So, your submission should only has one zip file.

The name of the zipped file should be in the format of:

<student-ID>_<full-name>_Assignment1.zip
Example: 12345678_Antoni-Liang_Assignment1.zip

Please make sure your submission is complete and not corrupted. You can re-download the submission and check if you can compile and run the program again. Corrupted submission will receive instant zero. Late submission will receive zero mark. You can submit it multiple times, but only your latest submission will be marked.

End of Assignment