Curtin University — Discipline of Computing

Unix & C Programming (COMP1000)

Semester 2, 2021

Assignment Two

Due: Friday, 15 October 2021, 11:59 PM (GMT+8)

Weight: 35% 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:

- Retrieve map metadata from a text file provided in command line argument.
- Write a simple algorithm for a snake character to chase the player.
- Utilize a generic linkedlist to store movement log and use it to backtrack.

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 expand the basic functionalities of the maze program you did for assignment 1 (The ”limited visibility” feature is not needed to complete assignment 2). 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 <map_file_name>
```

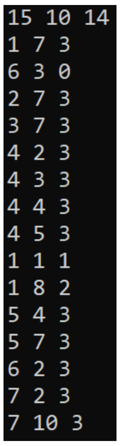
<map_file_name> is the textfile name that contains the metadata about the map. You have to use the proper File I/O functionalities to retrieve the information. For more detail, please refer to Section 3.3.

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 file does not exist, then you need to handle it with a proper error message.

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

3.3 File I/O and metadata

Please watch the supplementary video about the File I/O. Do you remember when you need to retrieve the metadata from the getMetadata() function in assignment 1? Now, you will retrieve the metadata from the textfile. This is one example:



```
15 10 14
1 7 3
6 3 0
2 7 3
3 7 3
4 2 3
4 3 3
4 4 3
4 5 3
1 1 1
1 8 2
5 4 3
5 7 3
6 2 3
7 2 3
7 10 3
```

Figure 1: Example of a text file containing the map metadata.

There are always 3 integers on every line. The first line always contains the <metadata_amount>, <map_row>, and <map_col> in that order. From the example of figure 1, the first line means that there are 15 objects on the map of size 10 rows and 14 columns. The following lines are the coordinates of the object with the pattern <object_row>, <object_col>, and <object_code>. The <object_row> and <object_col> are the zero-based index location of the objects (For example, <object_row> of 2 refers to the third row of the map). Please pay special attention on the <object_code> since it is slightly different from assignment 1:

- 0 – is the player that you control.
- 1 – is the snake that will try to bite the player.
- 2 – is the goal you want to reach to win the game.
- 3 – is the wall. Player and snake cannot go through it.

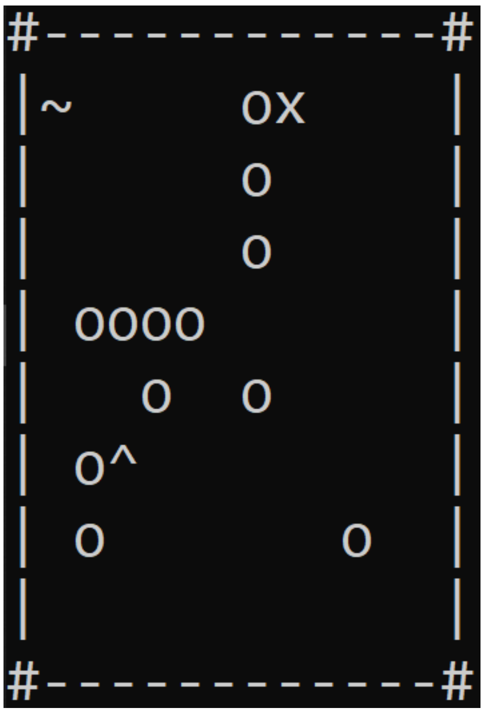
Unlike assignment 1, there is NO guarantee that the player and the goal would be the first two objects to be listed. It means that the coordinate of the player, snake, and the goal can be on any line. Therefore, your implementation should be able to handle that. However, you can assume that the amount of the objects matches the <metadata_amount> value, and there is exactly one player, one snake, and one goal.

Note: Since the edge of the map will contain the impassable border, the playable area size of the map is (<map_row>-2) x (<map_col>-2).

Note: Keep in mind that we will use our own map text file when we mark your assignment. So, please make sure your file I/O implementation works.

3.4 Main Interface

Similar to assignment one, your program should clear the terminal screen and print the 2D map:



```
#------------#
|~     0X    |
|      0     |
|      0     |
| 0000       |
|   0  0     |
| 0^         |
| 0       0  |
|            |
#------------#
```

Then the user can enter the command via key ’w’, ’a’, ’s’, and ’d’ to control 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).

3.5 Undo feature with Generic Linked List

In addition to the command to move the player, you will add another command with key ’u’ that allows the player to undo the steps taken when the player moves. Your program should utilise linked list to keep track all the steps taken since the beginning of the game. If the player attempts to walk to the wall, that is also recorded in the linked list. Please watch the supplementary video on this topic.

Please make sure you have a decent understanding on linked list practical exercises before you implement this feature. Implementing insertFirst() and removeFirst() function will be useful for the undo feature. In order to get full mark on this feature, you need to implement the generic linked list. If you implement the non-generic linked list, you still can receive decent partial marks.

3.6 Snake Movement and Losing condition

In this assignment, we introduce a new character '∼' (tilde character) to represent the snake on the map. This snake will gradually go closer to bite the player. Once the player and the snake is on the same location, the player loses the game. Please watch the supplementary video on this topic.

The snake will attempt to move one step closer every time the user enters one movement command key (not including undo key). The snake cannot move in the diagonal direction. If the player goes towards the wall, that also counts as a single movement and the snake will move one step as well. The snake also cannot go through the wall. You are free to design the algorithm that makes the snake move. The snake does not have to be very "smart" (can get stuck on the wall), but it should be reasonable so that it attempts to get closer to the player every time.

Note: When the snake bites the player, the player cannot use the undo feature anymore.

3.7 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.

3.8 Assumptions

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

- The metadata amount will always match the amount of the objects.
- The coordinates of all objects provided in the text file are all valid. (NOT out of bound)
- There will be exactly one player, one snake, and one goal from the metadata. (but the order in the text file is not guaranteed)
- 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 text file, you can assume the path from the player to the goal exist. So, it means the game is winnable. Similarly, you can assume the snake will have a path to reach the player. (not isolated)
- You only need to handle lowercase inputs ('w', 's', 'a', 'd', 'u').

4 Marking Criteria

The existing functionalities from assignment 1 will not be marked. Only the new functionalities are marked. This is the marking distribution for your guidance:

- Contains any NON-C89 syntax. Compile with -ansi and -pedantic flags to prevent losing marks on this. (-10 marks)
- 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, 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)
  - Correctly handles the file I/O to open and read the provided text file to retrieve the metadata. This includes being able to read the location of the player, snake, and the goal correctly despite being located on random lines. (15 marks)
  - Implement the generic linked list and store both the player and snake's location every time the player receives the command to move. (15 marks) (If not generic at all, only 7.5 marks max)
  - Able to "undo" the player and snake location based on the information stored in the linked list. By pressing 'u' multiple times, it is possible to go back up to the original map state. (10 marks).
  - The snake is able to move towards the player on every player movement. (15 marks)
  - Losing the game when the snake bites the player OR the player steps on the snake. (both objects are on the same coordinate) (10 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 could 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. If it is not zipped, you will get zero marks.

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

<student-ID>_<your-full-name>_<your-tutor-name>_Assignment2.zip
Example: 12345678_Antoni-Liang_Nadith_Assignment2.zip

Please make sure your latest 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. It means we will ignore the previous submission attempts.

End of Assignment