Curtin University — Discipline of Computing

Unix & C Programming (COMP1000)

Semester 1, 2024

Assignment One

Due: Sunday, 21 April 2024, 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:

- Able to create dynamically-allocated 2D char array to make a simple ASCII-based game.
- Receive user input to control the game.
- Utilize pre-written random number generator and terminal behaviour configuration for the game.
- Write a proper 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, instead #include .h files only.

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 at all, you will lose marks.

Please be aware of our coding standard (can be found on Blackboard and the marking criterias) 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 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 **"eat"**. Your executable should accept 2 command-line parameters/arguments:

./eat <map_row> <map_col>

<map_row> and <map_col> define the size of the playable map area for the player (the size does not include the border of the map). You can assume the user will provide proper datatype for the arguments (integer), but you still need to check for negative numbers and reject them accordingly. The minimum number for <map_row> and <map_col> are **5**.

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 argument is **NOT** in the correct range, please end the program in the same manner as well.

Since the player can enter arbitrary number for the map size, it means you need to use **malloc()** function to properly allocate sufficient memory for the 2D array for the map. Keep in mind that in C89, you cannot allocate the size of stack-based array with a variable, that is why you need malloc() function.

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

3.3 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 along with the player, the food, and the snake:

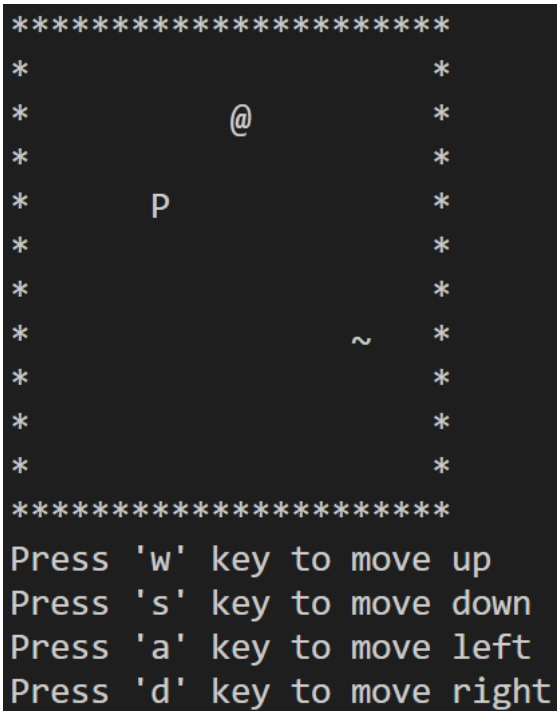


Figure 1: Main visual interface of the game with the player, the food and the snake. In this example, there are 10 rows and 20 columns of the playable area. The food location is randomized.

This is the main interface of the game. When you start the program, the player is always located at the top left corner (i.e row 0, column 0), while the snake is always located at bottom right corner of the map. Keep in mind that the initial location of the food is randomized everytime you execute the program (See Section 3.5.3). The user can type the command to move the player. The snake will move one grid randomly on 8 directions for every player movement (See Section 3.5.4). 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 3.5.1 for more detail on the user input.

3.4 Character on the Map

You have to use the correct characters on the game interface:

- Use char '*' for the map border.
- Use char 'P' for the Player.
- Use char '@' for the Food.
- Use char '~' for the Snake.

3.5 Gameplay

Please watch the supplementary videos for visual demonstration. The main gameplay is to move the player to eat the food while avoiding being eaten by the snake.

3.5.1 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.
- **'s'** moves the player one block below.
- **'a'** moves the player one block left.
- **'d'** moves the player one block right.
- Any other character should be ignored, and nothing changes on the map. Feel free to print a warning message such as "invalid key".

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 valid action should update the 2D map array, clear the terminal screen and reprint the map.

3.5.2 Winning/Losing the Game

The game will continue playing until either the player manages to **eat the food** (**winning**) OR the snake manages to **eat the player** (**losing**). You can print message such as "You win!" or "You lose!" when the game ended. Please do **NOT** use **exit**() or **break** to end the program. You must free the malloc'ed memories and ensure the program reaches the end of the main function properly.

3.5.3 Randomized Food Location

Please watch the supplementary video to learn how to use the pre-written random number generator function. You will need this function to randomize the location of the **food '@'** everytime you execute the program. In the unlikely case the random food location collides with initial player location OR initial snake location, then please re-generate random location until there is no collision. (You can try to use while loop for this.)

3.5.4 Snake Movements

Whenever the player makes a valid movement, the snake will also move for one grid on the map. However, the snake movement is randomized. You have to use the random number generator function for the snake to decide 1 out of 8 possible directions. If the snake is at the edge of the map, then some movements will be restricted. (Please be careful here. Moving the snake outside the map might crash your program due to unauthorized out of bound memory access)

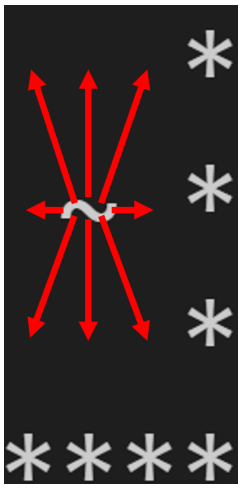


Figure 2: The snake can move to 1 out of 8 directions.

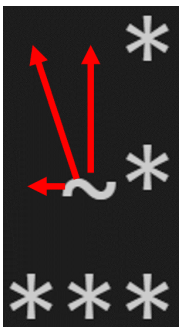
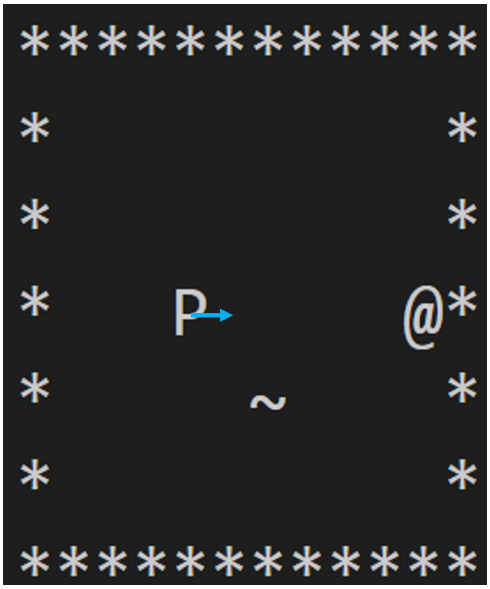


Figure 3: If the snake is at the edge of the map, then the choice of movements will be reduced. Please ensure that the snake does not move out of bound of the map.

However, if the player moves to a grid on which it is exactly **ONE grid away** from the snake, then the snake will just move to eat the player without any random movement.



(a) Player moves closer to the snake.

(b) Snake moves to eat the player because the player is only one grid away.

Figure 4: The snake will move in the direction of the player only if the player is one grid away from the snake. (after player movement)

3.6 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, 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 auto-generated by the VScode. Write it yourself.

3.7 Assumptions

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

- The datatype of all the command line arguments provided by the users are always integer. However, you still need to check if the numbers are valid (e.g not too small OR negative numbers). Keep in mind that the command line arguments will be stored as a string in argv array. Using the function atoi() can be useful.
- 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.
- You only need to handle lowercase inputs ('w', 's', 'a', 'd'). The other keys should be ignored (feel free to add warning message).

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 terminal.c and random.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''
  - Using goto
  - 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 command to check for any memory leak. 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. This includes checking the acceptable range of the arguments. (5 marks)
  - Proper memory allocation for the 2D map array. (5 marks)
  - Able to clear the screen and re-print the map on every action. (5 marks)
  - Able to move the player with the keyboard input from the user. (5 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)
  - Food initial location is randomized everytime the program is executed. (10 marks)
  - The snake moves randomly 1 out of 8 directions after every player movement without going out of bound. (15 marks)
  - When the player moves too close to the snake, the snake immediately eats the player. (10 marks)
  - Winning when the Player eats the Food AND losing when the Snake eats the Player. (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 short 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 into a **SINGLE zip file**. So, your submission should only has one zip file. If you do not compress the files, we reserve the right to refuse your submission.

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

<your-tutor-name>_<student-ID>_<your-full-name>_Assignment1.zip
Example: Antoni_12345678_Michael-Bay_Assignment1.zip

If you forget your tutor's name, please put the lecturer's name. 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