Curtin University — Discipline of Computing

Unix & C Programming (COMP1000)

Semester 1, 2022

Assignment Two

Due: Saturday, 21 May 2022, 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 snake information from a text file provided in command line argument.
- Able to let the snake eat multiple food to win the game.
- Utilize a generic linkedlist and struct to store snake information and able to expand the length after eating food.

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 snake program you did for assignment 1 (The "unbeatable" feature is NOT needed to complete assignment 2. At the very least the snake should be able to move around and consume the food). Further details on the specification will be explained on the oncoming sections.

3.2 Command Line Arguments

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

./snake <snake_file_name> <food_amount_to_win>

<snake_file_name> is the textfile name that contains the metadata about the snake. You have to use the proper File I/O functionalities to retrieve the information. For more detail, please refer to Section 3.3. <food_amount_to_win> is the amount of food the snake need to eat to win the game. The minimum for the <food_amount_to_win> is 2.

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

Please watch the supplementary video about the File I/O. The text file will look like this:

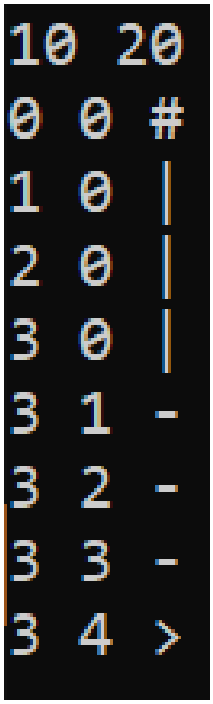


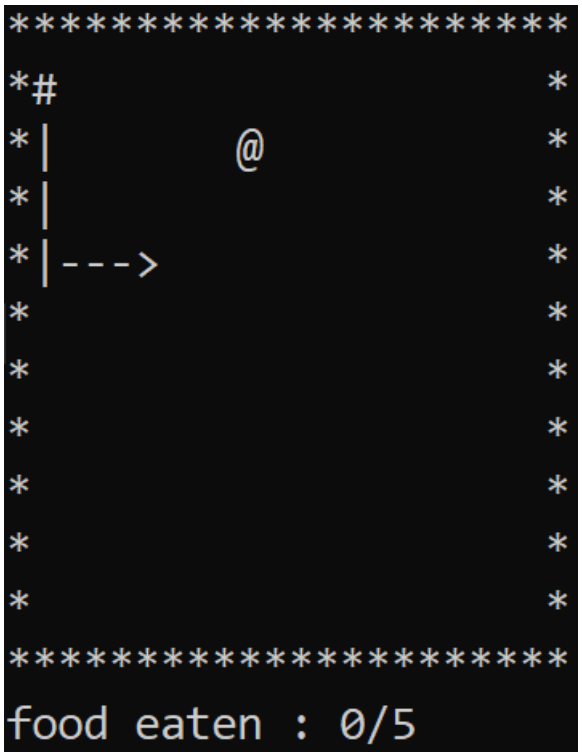
Figure 1: Example of a text file containing the map size and snake information.

The two integers on the first line are the playable row and column map size respectively. Afterwards, each line represents the location (starting from index zero) for each grid of the snake. Each line will have the row position, column position, and the char for that location. This information will start from the tail position, followed by the body components, and ends with the snake head. You can assume the topology structure of the snake is always valid. (it will be a solid and complete snake).

Note: Keep in mind that we will use our own snake 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:



Then the user can enter the command via key 'w', 'a', 's', and 'd' to control the snake. 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 Struct and Generic Linked List to store Snake information

Since the snake consists of multiple parts (i.e tail, body, and head), we want you to store the location for each snake component in a struct. One possible struct template you can use is to have two integers to represent row and column locations, and a char to store the character of the snake component in that location. (You can create different struct if you want, as long as it does the task.)

Once you have the snake component struct, you need to use the generic linkedlist code that you have written. Please use the linkedlist to store each component in the right order when you read the information from the text file. It is up to you whether you want to store it with the "insert First" or "insert Last" function, as long as it allows your program to access the linkedlist in the correct way to update and play the game. Please make sure you free the memory properly before the program ends.

Note: Please do not copy the whole linkedlist code from someone else because it can lead to collusion academic misconduct.

3.6 Winning Condition

Unlike the first assignment, the snake need to eat sufficient amount of food to win the game. The last command line argument represents the food amount. Every time the food is eaten by the snake, you should spawn a new one on random location. Please keep track of the amount of eaten food and print it below the map (see supplementary videos).

Note: The losing condition is not relevant to this assignment 2. If for some reason, your losing condition does not work on the assignment 1, you do not have to fix it here.

3.7 Snake Growth

In this assignment 2, the snake will grow by one block for every food eaten. In order to achieve this, you need to add a new node to your linkedlist and update it on the map. It is up to you whether you want to grow it on the tail or the head component. The example shown on the supplementary video is growing on the tail side.

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 snake parts provided in the text file are all valid. (NOT out of bound)
- There will be exactly one tail and one snake head.
- 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.
- You only need to handle lowercase inputs ('w', 's', 'a', 'd').

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 (5 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. (5 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 (correct amount of command line arguments AND <food_amount_to_win> has to be at least 2) with proper response (10 marks)
  - Correctly handles the file I/O to open and read the provided text file to retrieve the map size and snake information/location. (15 marks)
  - Implement the generic linked list to store the snake's location with appropriate struct. (15 marks) (If not generic at all, only 7.5 marks max)
  - Winning the game when the snake consumes the required amount of food. (5 marks)
  - Keep track of the amount of food eaten below the map. The food should be spawned randomly every time it is consumed. (10 marks)
  - The snake grows properly when the food is consumed by modifying the linkedlist. (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 32.5 out of 65 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