Curtin University — Discipline of Computing

Unix & C Programming (COMP1000)

Semester 1, 2023

Assignment Two

Due: Wednesday, 31 May 2023, 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 information from a text file provided in command line argument.
- Add the wall object in the map.
- Utilize a generic linkedlist and struct to keep track of player movement and box location for ”undo feature”.

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 ”box” program you did for assignment 1 (The ”PULL” feature is NOT necessary to complete assignment 2. You can leave the feature as it is, or remove it if you prefer). Further details on the specification will be explained on the oncoming sections.

3.2 Command Line Arguments

Please ensure that the executable is called **“box”**. Your executable should accept one command-line parameters/arguments:

```
./box <map_file_name>
```

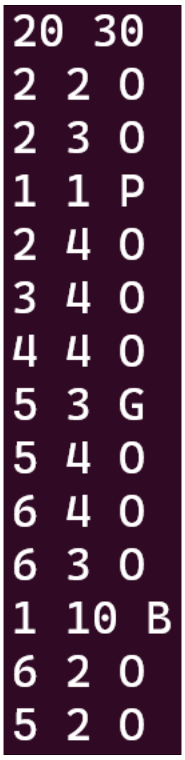
<map_file_name> is the textfile name that contains the map size and location of all the necessary characters of the game. You have to use the proper File I/O functionalities to retrieve the information. For more detail, please refer to Section 3.3.

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

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



```
20 30
2 2 O
2 3 O
1 1 P
2 4 O
3 4 O
4 4 O
5 3 G
5 4 O
6 4 O
6 3 O
1 10 B
6 2 O
5 2 O
```

Figure 1: Example of a text file containing the map size and location 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 the player, goal, box, and wall locations. We introduce wall as the new character in the game. The player and box are not able to pass through the wall. Each object is represented by:

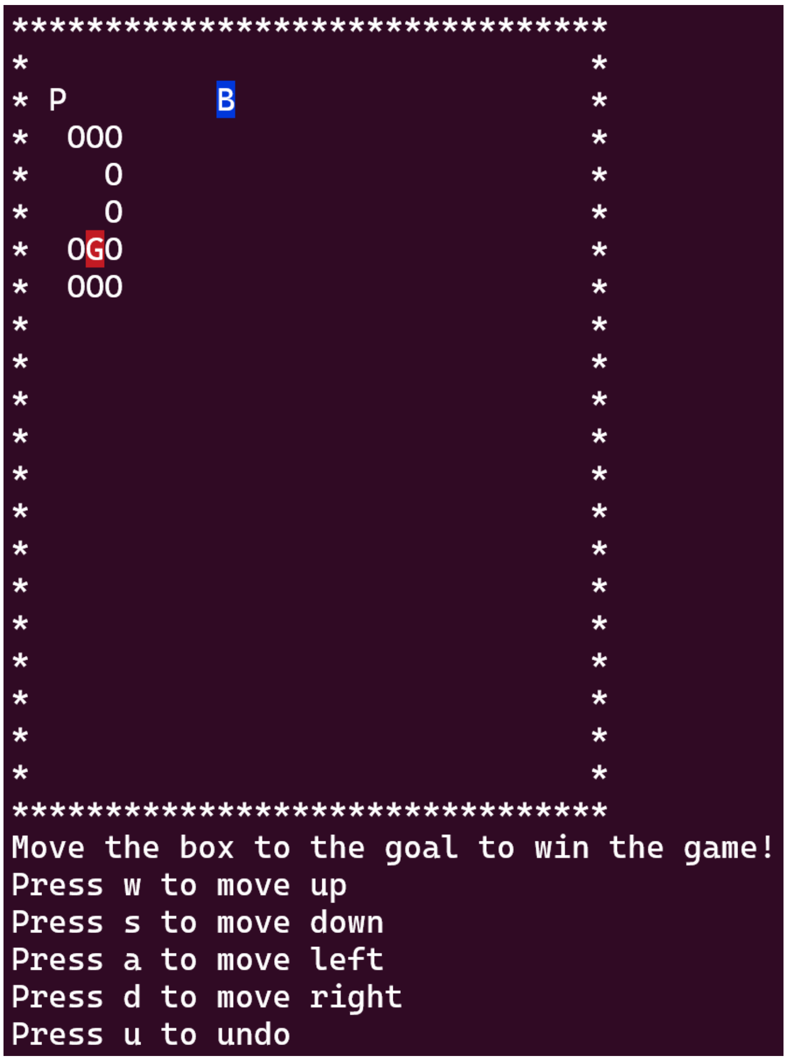
- 'P' represents the player.
- 'G' represents the goal.
- 'B' represents the box.
- 'O' represents the wall.

Each line will have the row position and column position of the associated character. You can assume there is one and only one player, one goal, and one box in the text file. You can also assume there is no overlapping location between different objects. The file might or might not contain any wall object(s) (zero or more occurences). Those characters can be listed in the file in any order. Your program should be able to read it correctly to initialize the game interface.

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 with various map files. Feel free to create your own map file.

3.4 Main Interface

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

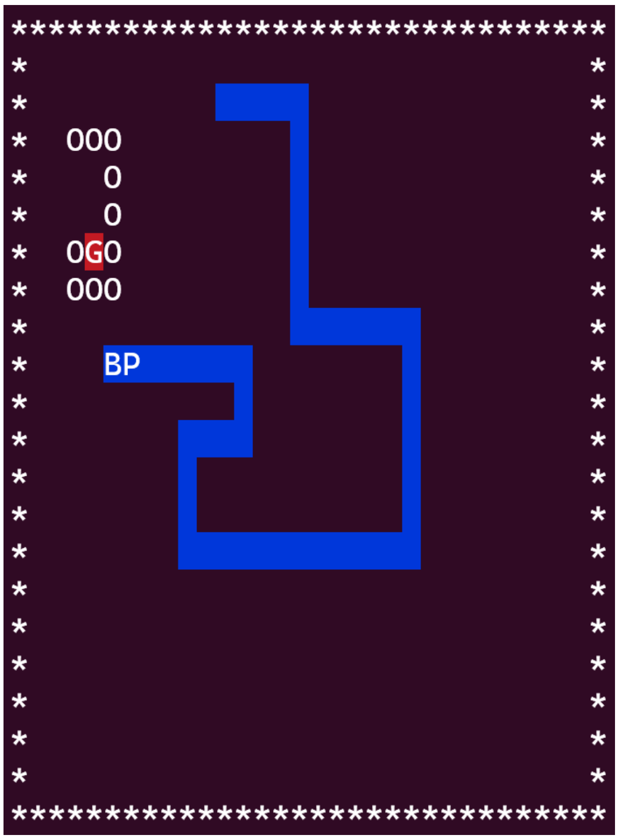


```
********************************
*                              *
* P        B                   *
*  000                         *
*    0                         *
*    0                         *
*  0G0                         *
*  000                         *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
********************************
Move the box to the goal to win the game!
Press w to move up
Press s to move down
Press a to move left
Press d to move right
Press u to undo
```

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). In addition, the player can enter the key 'u' to undo the movement of the player and the box. For more detail about UNDO feature, please refer to section 3.6.

3.5 Background Colors

In assignment 1, you have assigned red background color on the Goal character (and green when box reaches the goal). In this assignment 2, you need to add blue background color on the box character. However, it is applied not only on the current position of the box, but also on all the locations the box has traversed. Basically, the box leaves a blue-colored trail on the map.



```
********************************
*                              *
*                              *
*  OOO                         *
*    O                         *
*    O                         *
*  OGO                         *
*  OOO                         *
*                              *
*    BP                        *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
*                              *
********************************
```

Our recommendation to achieve this effect is by creating another 2D color array that stores the color information. By observing the box movement, you can update the color array accordingly. For example, you can store 0 for no color, 1 for red color, 2 for green color, and 3 for the blue color. When you print the map on the terminal, you can use information from both the 2D map array and color array to display it correctly. Feel free to use your creativity. If you have a completely different idea to solve this task, it is also fine to implement that as well.

Note: Keep in mind that when you use the undo feature from section 3.6, your program should also undo the blue trail as well.

3.6 Struct and Generic Linked List to implement UNDO feature

In this section, you will need to write a **generic linked list** with appropriate struct to implement **UNDO** feature. This feature is triggered when the user enters the key 'u'. Everytime the key is pressed, the player and the box will move back to the previous movement, and the blue trail need to be updated as well (if the box was moved). The player can use the undo feature at any point of the game as long as the game is not over. It should be possible to keep undo-ing until the initial state of the game when you just run the program.

Please refer to supplementary video for some advices regarding this implementation. The summary is that you can use the linkedlist node to store any information that you think are useful. For example, you can store the previous player 'P' location, and the box 'B' location (This is only one example, you can store more information if you need it). You will need to create a new linkedlist node every time the player moves and store it in the linkedlist. When the player presses 'u', your program should be able to retrieve the most recent node and update the map to the previous state. If game state is back to the initial configuration (when you just started the game), pressing 'u' should NOT do anything. One possible way to confirm this is by checking if the linkedlist is empty. (Once again, this depends on your implementation. Implement what you need to achieve this task.)

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

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 coordinates of all characters provided in the text file are all valid. (NOT out of bound)
- There will be exactly one player, one goal, and one box in the text file. However, they can appear in any order in the text file.
- The amount of the wall object is arbitrary. It ranges from zero to any positive integer. However, you can assume the wall will NOT block/isolate the goal which prevents winning.
- 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', '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 (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′′
  - 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 to check for any leaks. If your program is very sparse OR does not use any malloc()/calloc(), you will get zero mark on this category.
- **FUNCTIONALITIES**:
  - Correct command line arguments verification (correct amount of command line arguments) with proper response (5 marks)
  - Correctly handles the file I/O to open and read the provided text file to retrieve the map size and game characters to configure the initial game state. (15 marks)
  - Player and box should not be able to pass through the wall. (5 marks)
  - Correctly using appropriate blue background color for the box, which results in blue-colored trail based on the history of box movement. (20 marks)
  - Implement the generic linked list to store the relevant location information with appropriate struct. (10 marks) (If not generic at all, only 5 marks max)
  - Undo feature allows the user to undo the movement of the player and the box (5 marks)

  - Undo feature also undo the blue-colored trail correctly. (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 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:

<your-tutor-name>_<student-ID>_<your-full-name>_Assignment_2.zip
Example: Antoni_12345678_Tony-Stark_Assignment_2.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