CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

C PROGRAMMING

Assignment 1: Escape from the Labyrinth.
Part 1: Pointers and Arrays. Stack Memory Area.

1. Background:

In this assignment we are going to create our own version of the classic game ‘Escape from the Labyrinth’. On it, we are in charge of a knight (from now on ‘K’), which is placed on the entrance of a 6x6 board. ‘K’ can be moved across the board by using the keys ‘a’, ‘d’, ‘w’ and ‘s’ for left, right, up and down, respectively. The board also contains a barrel (from now on ‘B’), some wall positions (from now on ‘W’) and some treasures to be collected (from now on ‘T’). The goal of the game is to use 'K' to push 'B' towards the escape of the labyrinth, indicated by ‘→’. On its way, 'K' can collect any 'T' so as to make extra points on its journey. However, each ‘T’ is susceptible of being buried by pushing 'B' into its position.

2. Game Logic:

The information we want to represent in our game, together with the data structures we are going to use to represent such this information are as follows:

1. **Board:** We represent it via the 6x6 two-dimensional char array **char board[6][6]**.
    - Empty positions are represented by ‘ ’.
    - Wall positions are represented by ‘W’.
    - Treasure positions are represented by ‘T’.
    - Barrel position is represented by ‘B’.
    - Knight position is represented by ‘K’.

    Figure 1 presents the content of board when the game starts.

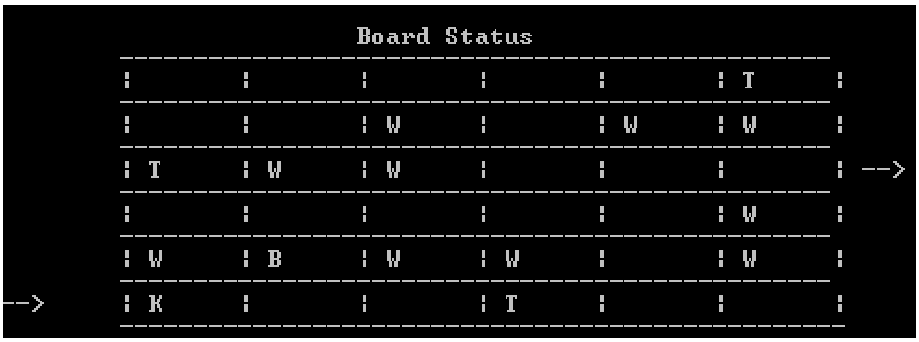


Fig. 1: Initial status of the board

Note: Whereas ‘W’ and ‘T’ positions are static, ‘B’ and ‘K’ positions are dynamic, as they are susceptible to be changed via user interaction.

2. **Knight:** As ‘K’ position is dynamic, we use a char pointer **char* knight** for direct access to the position of **board** it is placed at.
3. **Barrel:** Likewise, as ‘B’ position is dynamic, we use a char pointer **char* barrel** for direct access to the position of **board** it is placed at.
4. **Steps:** We use an integer variable **int steps** to count the movements done by the knight during the game.
5. **Treasures:** We use an integer variable **int treasures** to count the treasures collected by the knight during the game.

3. Interactive Session:

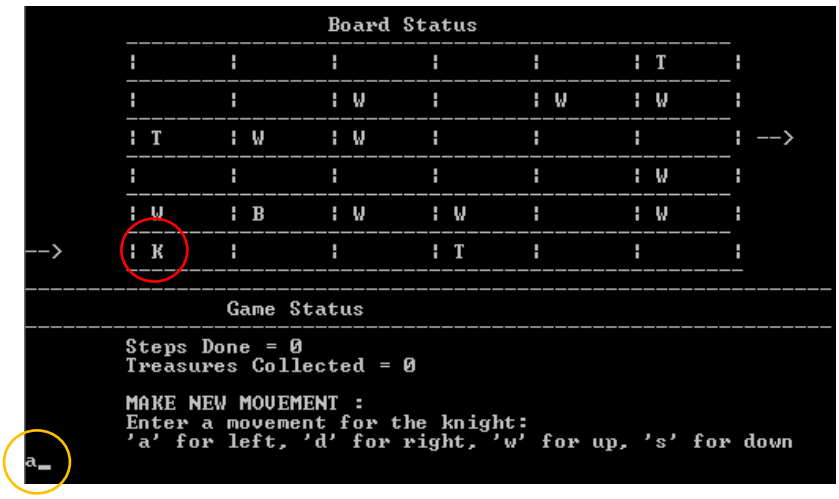
The game starts with a board status as the one presented in Figure 1. Then, an interactive session is triggered, in which the user is asked to move ‘K’ with the aim of collecting ‘T’ and push ‘B’ towards the exit of the labyrinth (marked by ‘→’). The user can move ‘K’ left, right, up and down via the commands ‘a’, ‘d’, ‘w’ and ‘s’, respectively.

A movement is non-valid if one of the following conditions apply:

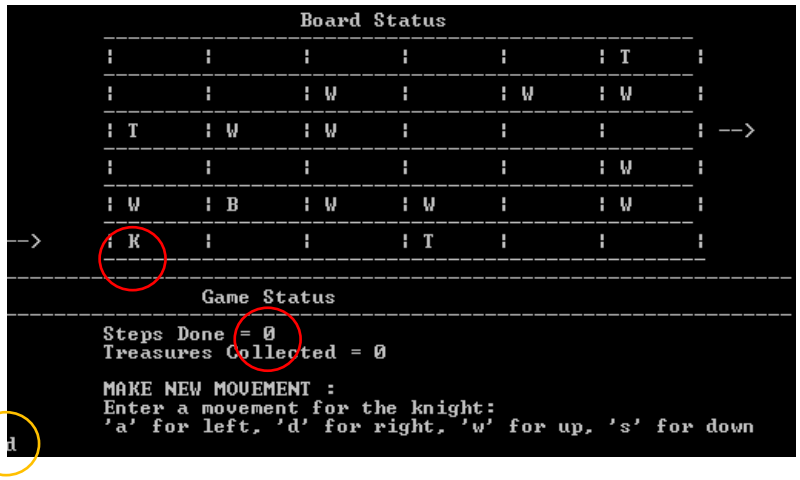
1. ‘K’ is asked to move out of the bounds of board.
2. ‘K’ is asked to move towards a wall.
3. ‘K’ is asked to push ‘B’, but then:
   3.1. ‘B’ is pushed out of the bounds of the board (the labyrinth exit is the only exception to this).
   3.2. ‘B’ is pushed towards a wall.

If a movement is non-valid, then nothing happens, the game status of **(board, knight, barrel, steps, treasures)** remains the same. Otherwise, if the movement is a valid one, the status of **(board, knight, barrel, steps, treasures)** is updated accordingly.

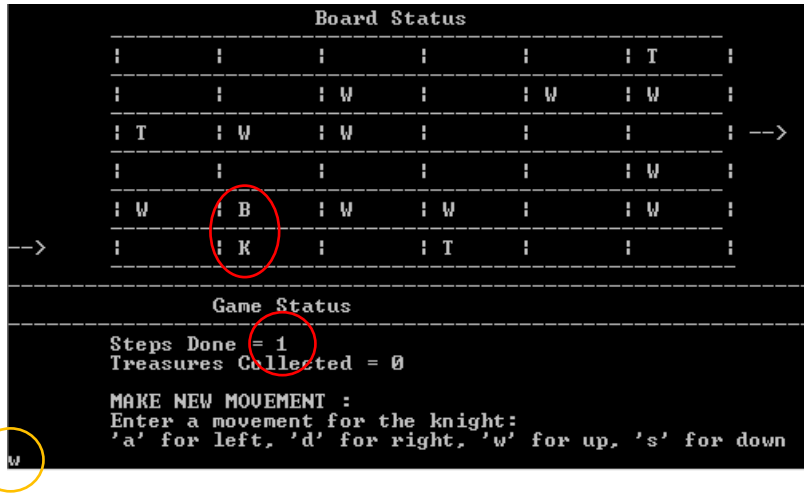
The following presents a possible game interactive session. The game starts with the current status displayed in Figure 1.



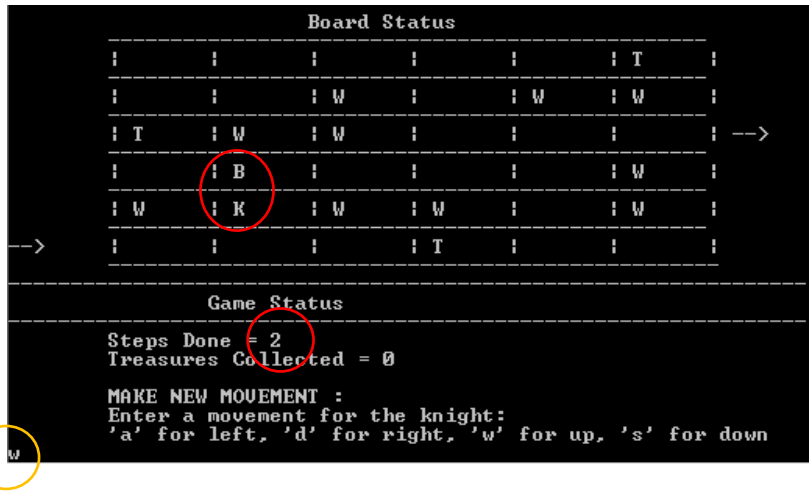
A first command ‘a’ is introduced. It is non-valid, as it incurs in condition 1 listed above. Thus, the status of **(board, knight, barrel, steps, treasures)** remains the same.



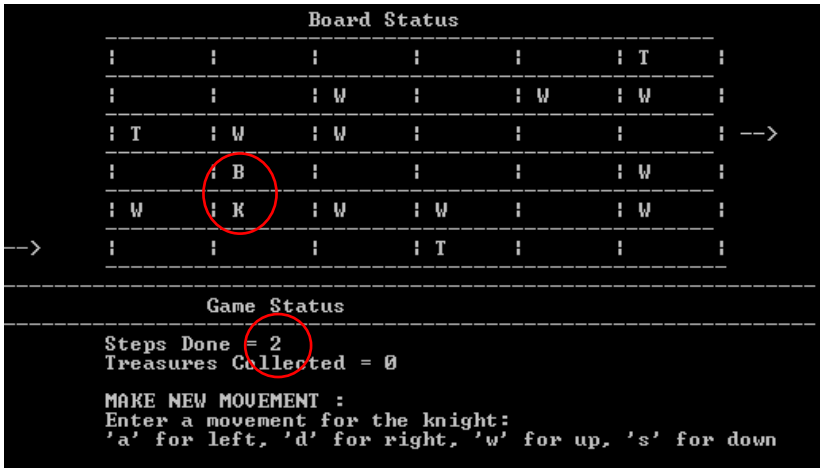
A second command ‘d’ is introduced. It is valid, as none of the conditions listed before apply. Thus, the status of **(board, knight, barrel, steps, treasures)** is updated accordingly.



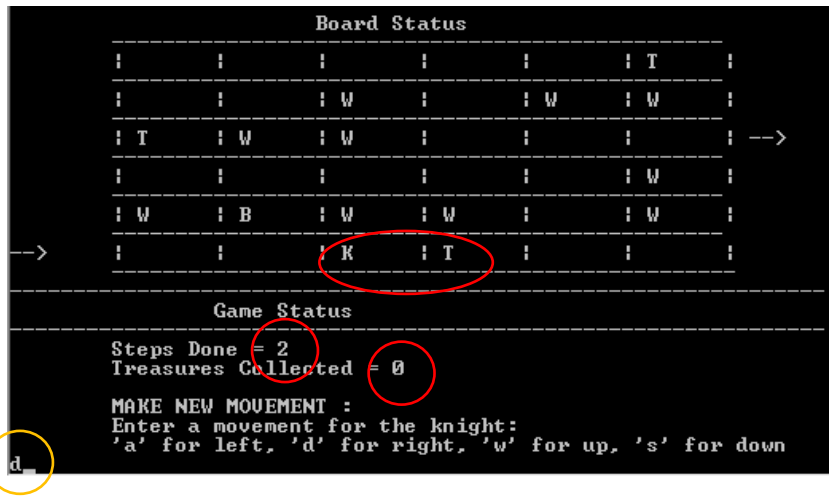
A third command ‘w’ is introduced. It is valid, as none of the conditions listed before apply. Thus, the status of **(board, knight, barrel, steps, treasures)** is updated accordingly.



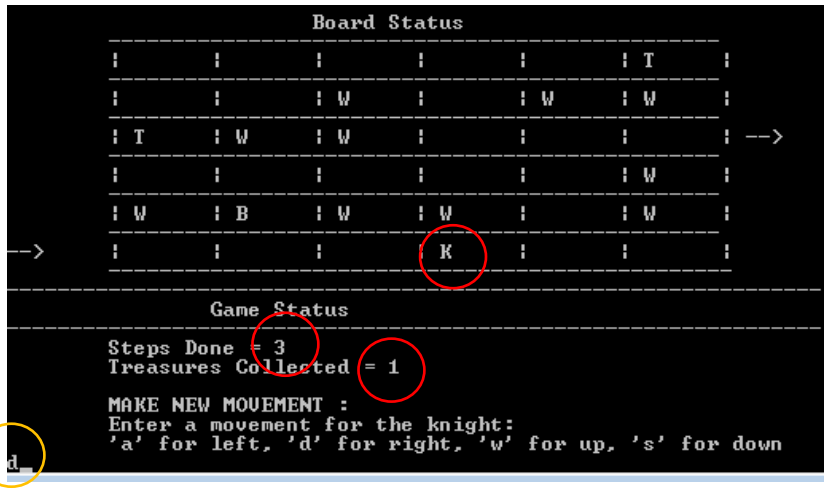
A fourth command ‘w’ is introduced. It is non-valid, as it incurs in condition 3.2 listed above. Thus, the status of **(board, knight, barrel, steps, treasures)** remains the same.



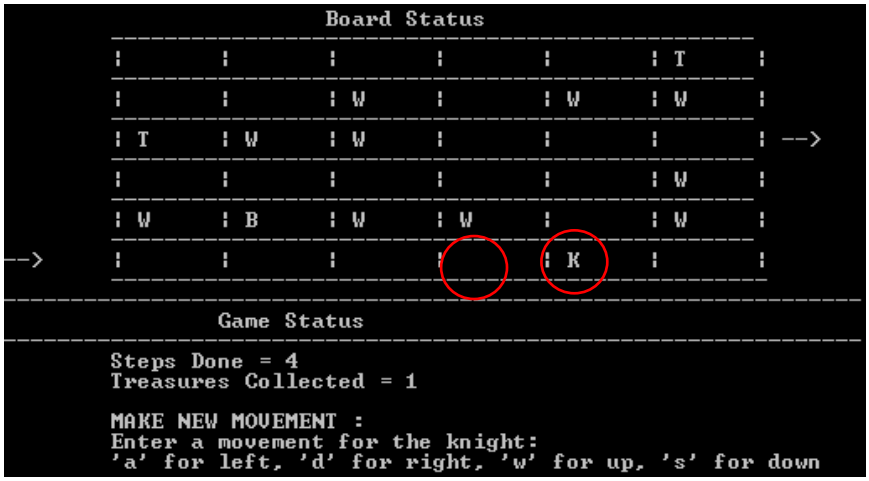
Another possible situation to be taken into account is the consumption of the treasures.



A command ‘d’ is introduced. It is valid, as none of the conditions listed before apply. Thus, the status of **(board, knight, barrel, steps, treasures)** is updated accordingly.

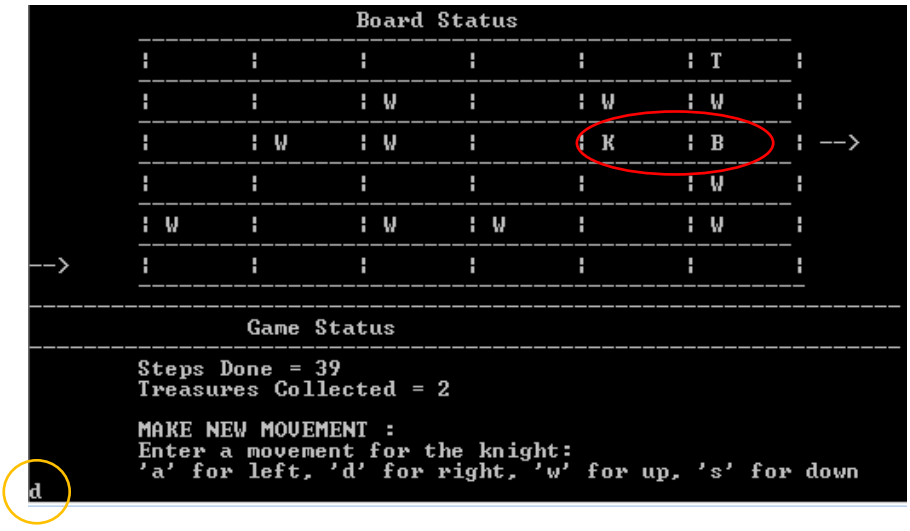


A command ‘d’ is introduced. It is valid, as none of the conditions listed before apply. Thus, the status of **(board, knight, barrel, steps, treasures)** is updated accordingly. In particular, we can see that the treasure has been found for good, as it does not appear anymore in the board.

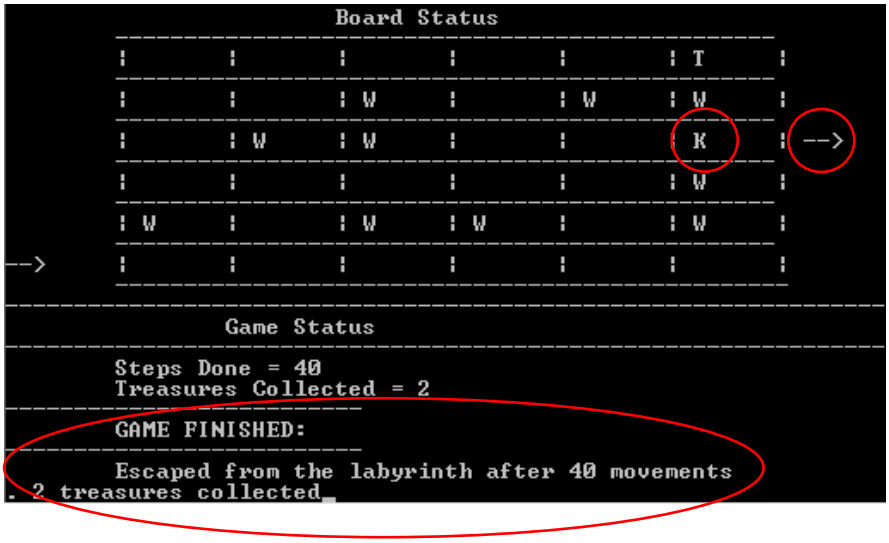


Interestingly, if it is the barrel the one being moved towards a treasure position, then the treasure is buried for good, so it cannot be found anymore.

Last situation to be taken into account is the one triggering the end of the game.



A command ‘d’ is introduced. It is valid, as none of the conditions listed before apply. Thus, the status of **(board, knight, barrel, steps, treasures)** is updated accordingly. Moreover, this movement is considered the winning one, as it makes the barrel to be pushed to the exit of the labyrinth. A message to show the end of the game is displayed.



4. Project:

The C project consists on 7 files:

1. auxiliary.h
2. auxiliary.c
3. gameSetup.h
4. gameSetup.c
5. gameMovement.h
6. gameMovement.c
7. main.c

auxiliary.h

It provides the following functionality:

1. Datatype: bool.
   It allows you to use Booleans within your project, under the values False and True (please note the capital letters).

2. Function: int gen_num(int lb, int ub);
   It generates a random integer number in the interval [lb, ub). This functionality is not needed in this first part of the assignment.

3. Function: char my_get_char( );
   Waits for the user to enter some character by keyboard and returns it.

auxiliary.c

This file is provided complete.

gameSetup.h

It provides the following functionality:

1. Function: void game_instructions( );
   It shows the instructions of the game before it actually starts.

2. Function: void set_game_initial_status(char board[6][6], char** knight, char** barrel, int* steps, int* treasures);
   It sets the initial status of **(board, knight, barrel, steps, treasures)** before any user interaction takes place. In particular: board is filled as it is shown in Figure 1; knight and barrel are then set to point to the concrete board positions hosting 'K' and 'B', respectively; steps and treasures are set to 0.

3. Function: void display_board(char board[6][6]);
   It pretty prints the information hosted on the array board, so that we have a nice graphical representation of it (as the one shown in Figure 1).

4. Function: void display_game_status(char board[6][6], int steps, int treasures);
   It displays the status of **(board, knight, barrel, steps, treasures)**. In particular, it calls to display_board to display the status of board.

5. Function: void play_game( );
   It provides the main method of the game. As you can imagine it first prints the game instructions and then sets the game to its initial status. After that, it enters the game main loop, which keeps track of whether the game is over, and while this is not the case keeps asking the user for a new movement and performing it. Once the game is finished, it just displays a game solved message, including the number of steps and treasures collected.

gameSetup.c

Whereas function 1 is provided complete, functions 2, 3, 4, and 5 are to be completed.

gameMovement.h

It provides the following functionality:

1. Function: char ask_for_movement( );
   It asks the user for a valid command 'a', 'd', 'w' and 's', returning it.

2. Function: void get_position_coord_from_pointer(char board[6][6], char* object, int* x_pos, int* y_pos);
   It applies maths with pointers to, provided board and either knight or barrel, compute the concrete position (x,y) coordinates of either knight or barrel. For example, from

Figure 1, we know the coordinates of barrel are (4,1) and the coordinates of knight are (5,0). So, the idea of the function is, given the address of knight and the address of board, determine this knight (x,y) coordinates.

3. Function: void get_new_position_coord(int x_pos, int y_pos, char movement, int* new_x_pos, int* new_y_pos);
   Given the current (x,y) coordinates of an object, and the type of movement the user wants to perform (either 'a', 'd', 'w' or 's'), this function computes which will be the new coordinates (x', y') the object would be moved to.

4. Function: bool is_winning_movement(char board[6][6], char* barrel, char movement);
   It detects whether the next movement to be performed is the winning one, i.e., the one pushing barrel towards the exit of the labyrinth.

5. Function: bool is_movement_safe(char board[6][6], char* knight, char* barrel, char movement);
   It detects whether a command movement entered by the user is valid or non-valid (c.f., non-valid conditions 1, 2, 3.1 and 3.2 were listed in Section 3).

6. Function: bool perform_movement(char board[6][6], char** knight, char** barrel, int* steps, int* treasures);
   It asks the user for a command character. Then, it checks whether the movement associated to this command is valid. If it is, the function updates the status of **(board, knight, barrel, steps, treasures)** accordingly. In particular, it also checks whether the valid movement is the winning one or not, returning this information.

gameMovement.c

Functions 1, 2, 3, 4, 5 and 6 are to be completed.

main.c

It provides the following functionality:

1. Function: int main( );
   As there program do not receive any input parameters, the main method just calls to the main function play_game( ) and returns 0 afterwards.

5. Evaluation and Submission Date:

- The full assignment is worth 25 marks. I will wait until the second part of the assignment is released for providing the mark breakdown.
- The completed files gameMovement.c and gameSetup.c have to be submitted as a single zip file to Blackboard before Sunday 6th November 11.59pm.
- A short interview which each student will take place in week 9 to discuss the code.