CS 240: Programming in C
Fall 2020
Homework 6

Prof. Turkstra

Due October 14, 2020 9:00 PM

1 Goals

The purpose of this homework is to give you experience using structures that contain fixed-size strings and using pointers to elements of arrays. You will use file manipulation operations to load values from files into structures. You will need to make sure that you do not copy values that are so long that they overflow their storage allocation.

Assume for a chess game, only white and black knights will be placed on a game board (the number of different color knights could be more than 2). To do this, we'll create an `knight_t` struct to hold data about the knight on the game board, and include eight more fields to the `knight_t` that will *point* to the next possible grids, if exists.

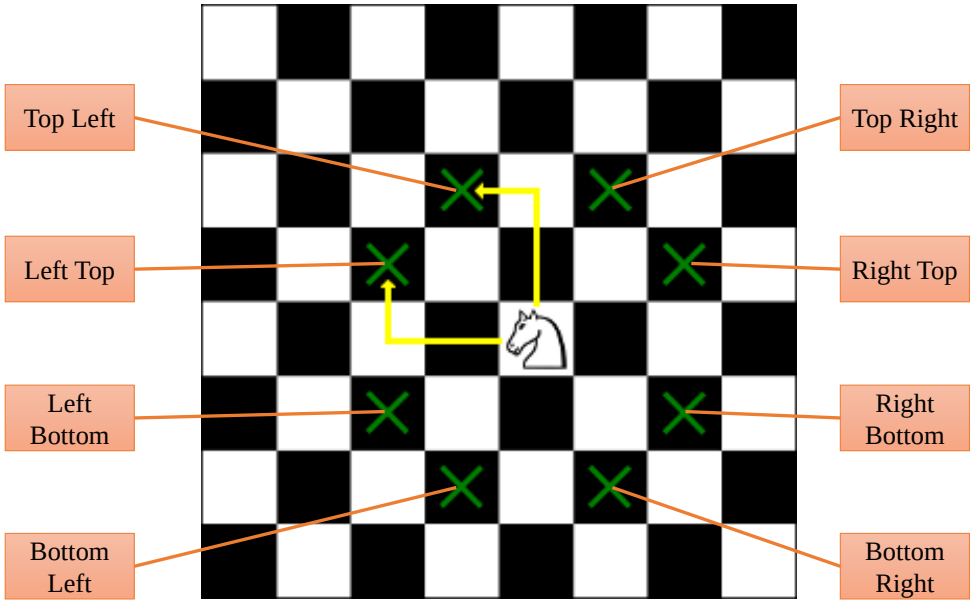
2 The Big Idea

At this point you understand how structures work and what they're used for. This time, we're going to concentrate more on the aspects of using strings. The primary complication this time is that you are going to have to scan in data that is often too large to be stored in the structures. I.e., all of the structure fields are 10 bytes in length. That is enough space to store 9-character strings safely. You must not attempt to copy more than 10 characters of a string into one of these fields. In order to do this, you must *truncate* strings that are any longer than this. It is also critical that the strings are properly terminated with a NULL character at the end. You'll also have to limit the size of the strings when searching.

To handle these nuances, you'll use the `strncpy()` function to copy only as many bytes as are possible to store and `strncmp()` to limit how many characters of a string participate in a comparison operation.

NOTE: `strncpy()` does not NULL-terminate long strings! You have to do that yourself.
The Knight is the most unique piece in chess, having a flexibility that makes it a powerful piece. It is the only piece on the board that may jump over other pieces. It moves two squares horizontally or vertically and then one more square at a right-angle.

In this homework, we are going to play with a new chess game which only contains white or black knights. And each knight can only jump to a grid which contains an **opposite color knight**. The 8 possible grids to jump for the knight are shown below:



Given an array of `knight_t` variables in a game board, we'd like to have a way of *directly* referring to the next possible grids. To do this, we can declare extra fields in the `knight_t` structure to be pointers to other structures.

For this homework, you will be using `knight_t` which looks like this:

```
typedef struct knight {
  char position[MAX_POSITION_LEN];
  char color[MAX_COLOR_LEN];

  char top_right[MAX_POSITION_LEN];
  char top_left[MAX_POSITION_LEN];
  char bottom_right[MAX_POSITION_LEN];
  char bottom_left[MAX_POSITION_LEN];
  char right_top[MAX_POSITION_LEN];
  char right_bottom[MAX_POSITION_LEN];
  char left_top[MAX_POSITION_LEN];
  char left_bottom[MAX_POSITION_LEN];

  struct knight *top_right_ptr;
  struct knight *top_left_ptr;
  struct knight *bottom_right_ptr;
  struct knight *bottom_left_ptr;
  struct knight *right_top_ptr;
  struct knight *right_bottom_ptr;
  struct knight *left_top_ptr;
  struct knight *left_bottom_ptr;
} knight_t;
```

Examine hw6.h carefully. It contains declarations for the global variables that youll use. It also contains a declaration for the `knight_t` type and prototypes for all functions you will write.

Thought question:

Why did we have to declare the `top_right_ptr` as a `struct knight *` instead of `knight_t *`? Think about that. Try it and see what happens.

3 The Assignment

You are to write a series of functions that deal with arguments of the type `knight_t`. You will modify occurrences of `knight_t`s that exist in a global array of `MAX_KNIGHT_COUNT knight_t` structures called `g_knight_array`. You will keep track of how many structures you have initialized using a global variable named `g_knight_count`.

3.1 Functions You Will Write

You will write the following functions: `int read_knights(char *);`

This function will read a list of knights from the file specified by the first argument. The knights should be stored in the `g_knight_array`; the first one stored at offset zero and so on.

~~This function should assert that the argument is not NULL~~. If any other errors occur, it should return an appropriate error code from the list of error codes below. ~~Otherwise, the function should return the number of knights read from the input file.~~

Don't forget to update `g_knight_count`.

If `g_knight_count` is larger than `MAX_KNIGHT_COUNT`, this function returns `OUT_OF_BOUND` error.

Note:

A Knight's position is valid if the following conditions hold:

1. The first character in position entry is between 'A' and 'H'.
2. The second character in position entry is between '1' and '8'.

A Knight's color is valid if the following condition hold:

1. The color entry is "Black" or "White".

A Knight's 8 next possible grid's position is valid if the following conditions hold:

1. If next possible grid's position has no Knight on it, "??" will be used to denote.
2. If next possible grid's position has a Knight on it, the first character in this position entry is between 'A' and 'H'.
3. If next possible grid's position has a Knight on it, the second character in this position entry is between '1' and '8'.

Otherwise, this function returns `BAD_RECORD` error.

To implement this function, you need to set each character field from the input file and to set each of the pointer fields in each `knight_t` element of the `g_knight_array` to be NULL.

You can assume that the input file has at most one knight for each valid grid.

`void establish_game_board();`

This function accepts no values and returns no values. This function should traverse the values in the `g_knight_array` and find the knights to the Top Right, Top Left, Bottom Right, Bottom Left, Left Top, Left Bottom, Right Top and Right Bottom for each element.

We're going to leave it up to you to choose how to do this. There are many ways. One suggestion would be to use a loop to find the knight to the Top Right of each `knight_t` and set the `top_right_ptr` field to point to the Top Right's array entry. Then use another loop to set each entry's `top_left_ptr` field and so on.

Note that if you can not find any knights in a given direction for a specific `knight_t`, that direction pointer should be set to NULL.

`int count_next_possible_moves(int);`

This function will choose the Nth position (where N is the input argument) in `g_knight_array` and returns the number of next possible moves.

Recall the new rules for the game, each knight can only jump to a grid which contains an opposite color knight.

This function should assert that the input argument is greater or equal to 0 and less than `g_knight_count`.

`int check_destination(int, char *);`

This function will choose the Nth position (where N is the first input argument) in `g_knight_array` as the start Knight and returns the index of the knight pieces in `g_knight_array` that the knight can jump to if the start Knight could jump to the destination position which is the second input argument in one step. If the destination position is not inside the game board, `BAD_DESTINATION` error should be returned.

If no Knight is found, then this function should return `NO_KNIGHT` error.

This function should assert that the first input argument is greater or equal to 0 and less than `g_knight_count` and the second input argument is non-NULL.

`int find_knights_by_row(char * row, int * knight_array);`

It is expected that the caller of this routine will pass the address of an array of at least 8 integers. This function should traverse `g_knight_array` to find all instances of `knight_t`s where the row matches the first argument.

The function should add the knight pieces whose row matches the first argument to the `knight_array`. E.g., if three chess pieces are found, then `knight_array[0]`, `knight_array[1]`, and `knight_array[2]` should be set to the array indices of those knight pieces.

If no knight piece is found, the function should return 0. Otherwise, the function should return the number of knight pieces found.

~~The function should assert that none of the arguments are NULL. And if the first argument is not between 'A' and 'H', then this function should return `BAD_INPUT_ROW` error.~~

There is no way for you to check that `knight_array` points to a space of at least 8 integers, so don't waste your time trying.

3.2 Input Files

The input file will contain records of the following format:

`pos|color:top_right|top_left|bot_right|bot_left|right_top|right_bot|left_top|left_bot`

For example, consider the following game board:

	1	2	3	4	5	6	7	8
A	!!	!!	!!	!!	!!	!!	!!	!!
B	!!	!!	bk	!!	bk	!!	!!	!!
C	!!	wk	!!	!!	!!	wk	!!	!!
D	!!	!!	!!	wk	!!	!!	!!	!!
E	!!	bk	!!	!!	!!	bk	!!	!!
F	!!	!!	!!	!!	!!	!!	!!	!!
G	!!	!!	!!	!!	!!	!!	!!	!!
H	!!	!!	!!	!!	!!	!!	!!	!!

The input file for such a game board would be the following:

B3|Black:??|??|D4|D2|A5|C5|A1|C1
B5|Black:??|??|D6|D4|A7|C7|A3|C3
C2|White:A3|A1|E3|E1|B4|D4|??|??
C6|White:A7|A5|E7|E5|B8|D8|B4|D4
D4|White:B5|B3|F5|F3|C6|E6|C2|E2
E2|Black:C3|C1|G3|G1|D4|F4|??|??
E6|Black:C7|C5|G7|G5|D8|F8|D4|F4

Notice that we use `!!` to mean that for current position, there is no knight piece. And the input file only contains the non-empty position's information. And we use "??" to mean that the position is not on our game board. And its name should be set to '\0'.

Please see a sample input file named "game_board.txt" in ~`cs240/public/homework/hw6` for more information.

3.3 Header Files

We provide a header file, "hw6.h", for you. It contains prototypes for each of the functions that you will write as well as #definitions for the constants. You should not alter this file. We will replace it with the original when grading.

3.4 Error Codes

- `NON_READABLE_FILE`: the file cannot be opened for read access
- `BAD_RECORD`: a record was read from a file that didn't make sense or was incomplete
- `NO_KNIGHT`: the desired location does not hold a Knight
- `OUT_OF_BOUND`: indicates that the number of Knights in the input file is not within the expected range
- `BAD_DESTINATION`: indicates that the destination position is not within the expected range
- `BAD_INPUT_ROW`: indicates that the input row argument is not within the expected range

These Standard Rules Apply

- You may add any `#includes` you need to the top of your hw6.c file;
- You may not create any global variables other than those that are provided for you. Creation of additional global variables will impact your style grade;
- You should check for any failures and return an appropriate value.
- You should not assume any maximum size for the input files. The test program will generate files of arbitrary size.
- Do not look at anyone else's source code. Do not work with any other students.

Submission

To submit your program for grading, type:

```
$ make submit
```

In your hw6 directory. You can do this as often as you wish. We encourage you to submit your code as often as possible. Only your final submission will be graded.

4 Grading

The operation of your functions will be graded out of 100 points. The point breakdown will be determined by the test program.

The test program will be run many times when grading. It is your job to do the same. The lowest score will be your final grade.

This homework will also have a style grade based on 20 points, with 2 points deducted for each code standard violation found.

Your code must compile successfully using `-Wall -Werror -std=c99` to receive any credit. Code that does not compile will be assigned an automatic score of 0.

If your program crashes (e.g., segmentation fault) at any point during the testing process, your score will be reduced by 25 points.