C Final Test

Imperial College London

Department of Computing

Laser

Monday 13 June 2022
09h00 to 12h00
THREE HOURS
(including 10 minutes planning time)

- The maximum total is **100 marks**: 50 for each of the two parts.
- **Important:** TEN MARKS will be deducted from solutions that do not compile. Comment out any code that does not compile before you submit.
- Your code needs to compile and work correctly in the test environment which will be the same as the lab machines.
- The files can be found directly under the `laser` directory inside your gitlab repository.
- You can edit, compile and run your code using the terminal or an IDE like **CLion**.
- Push the final version of your code to `gitlab` before the deadline, and then go to `LabTS`, find the final/correct commit and submit it to `CATe`.
- **Important:** You should only modify the file `laser.c`. All other files are read-only, and are going to be overwritten by our autotester.

Problem Description

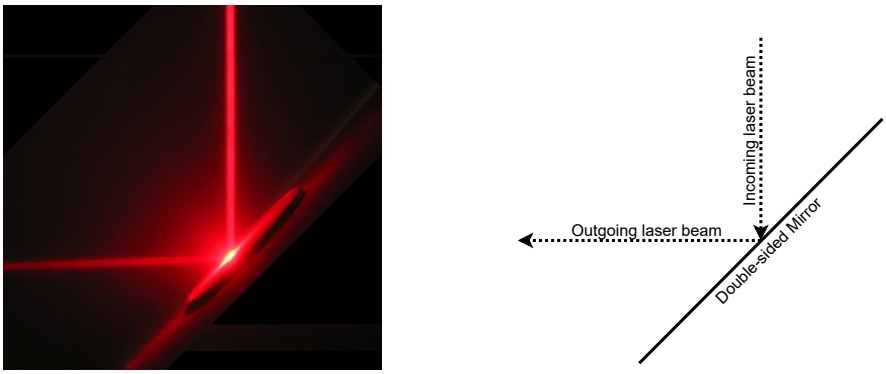

Figure 1: Laser beam striking one side of a double-sided mirror (left) and corresponding diagram (right).

Prof Biscuit, the famous physicist cat, is experimenting with a Helium–Neon laser that emits a bright laser beam, and several double-sided highly-polished mirrors. Your task is to write a computer program to help him design experiments based on the principles of laser light reflection (as illustrated in Figure 1).

```
          1         2         3
  01234567890123456789012345678901
 0 ################@########@#######
 1 #              B               #
 2 >               \              @
 3 #        C                     #
 4 #       /                \     #
 5 #                              #
 6 #     U       I                #
 7 #      \       \        /      @
 8 #                     S        #
 9 @               \              @
10 #                T             #
11 ################################
```

```
          1         2         3
  01234567890123456789012345678901
 0 ################@########@#######
 1 #              B               #
 2 >--------------\              @
 3 #        C     |               #
 4 #       /------+--------\      #
 5 #       |      |        |      #
 6 #     U|      I|        |      #
 7 #      \-------\--------/      @
 8 #              |       S       #
 9 @              \---------------@
10 #                T             #
11 ################################
```

Figure 2: Two-sided mirror layout (left) and path of laser beam which spells out message "BISCUIT" (right)

For each of his experiments, Prof Biscuit labels some of the mirrors with letters then places them strategically on a (flat rectangular) board so that the path of the laser beam spells out a message. Figure 2 shows the idea.

Here '`>`' denotes a laser emitter, '`/`' and '`\`' are mirrors (placed only at 45° angles relative to the laser light), '`#`' are edges of the board, and '`@`' are laser light absorbers. The characters '`-`', '`|`' and '`+`' indicate the path of the laser beam.

In planning an experiment, Prof Biscuit can sometimes get puzzled about whether to use a forward-leaning ('`/`'), a backward-leaning mirror ('`\`') or an empty space in a given board position. In this case he uses a question mark character ('`?`') to indicate his confusion; hopefully you can help to resolve this in due course.

Implementation

In `laser.h` you are provided with data structures for the board and the laser state as shown below:

```
typedef struct {
    int height;
    int width;
    char **cells;
} board_t;
```

```
typedef struct {
    int row;
    int col;
    direction_t direction;
    char msg[MAX_MSG_LEN];
} laser_state_t;
```

A board is a data structure that contains two integers that represent the dimensions of the board and a two dimensional array of cells. A laser has a state, which is given by its current position on the board, its direction of travel, and a char array which stores the message accumulated by the path of the laser from the starting point to its current position. You may assume the messages in this test will not exceed `MAX_MSG_LEN`.

Furthermore, the following files contain structs, enums, macros and functions that should be used in your code. Please familiarize yourself with them:

1. `laser.h` contains:
    - an enum `direction_t` describing the four possible directions of the laser beam
    - a series of ASCII art macros denoting important characters on the board, which will be used for tracing the laser through the board
    - four macros (`MIRRORS, CHOICES, PASSTHROUGHS, BLOCKERS`) represent logical groupings of ASCII art characters
    - `#define print_board(board)` prints the current state of the board to standard output
2. `biscuit.txt` contains an example of a board (see Figure 3 below).
    - The first line of the file consists of two numbers: the first is the number of rows that the board has, the second is the number of columns. Note that the indexing of the rows and columns start from 0.
    - The rest of the lines make up the board to be read in.

```
12 32
################@########@#######
#             B              #
>              ?             @
#       C                    #
#      /               ?     #
#                            #
#     U       I              #
#      ?       ?        ?    @
#                      S     #
@              \             @
#               T            #
################################
```

Figure 3: The contents of `biscuit.txt`. This board has 12 rows and 32 columns.

Part A Tasks

Testing frequently, complete the following functions in file `laser.c`:

1. `bool is_on_board(board_t board, int row, int col);`

   This function returns `true` only if the position (`row`, `col`) is within the bounds of the board (including the edges), indexing from 0 as shown in Figure 2.

   [5 Marks]

2. `int find_start(board_t board);`

   This function scans the leftmost column of the board for the character '`>`' (also defined as the macro `START` in `laser.h`).

   (a) When the leftmost column contains '`>`', the function should return the row index in which the character is found.

   (b) Otherwise, the function should return `-1`.

   [10 Marks]

3. `char get_mirror_label(board_t board, int row, int col);`

   This function should return the label of the mirror at location (`row`, `col`). The label is the only alphabetic character from the 3×3 square box around the mirror.

   If no mirror is present at the location or there is no character in the 3×3 square box, the function should return '`\0`'.

   Hint: You may use `isalpha` from the `ctype.h` library to identify a valid alphabetic character.

   [10 Marks]

4. `board_t load_board(const char *filename);`

   This function returns a `board_t` object by loading a board from a file such as `biscuit.txt`.

   The height and width dimensions should be read first before proceeding to read in the contents of the cells. You should be allocating memory for `board.cells` and for each row on heap.

   Note: For full marks, you need to manage the memory properly, exiting with an appropriate message in case of failure. For example, it is important to check any pointers that are returned from memory allocation methods.

   Hint: You may use `fscanf` to parse the board dimensions and `fgets` to read line by line.

   [15 Marks]

5. `void free_cells(board_t board);`

   This function frees the memory allocated to the board's cells.

   Hint: For each memory allocation made in `load_board`, there should be a corresponding de-allocation using the library function `free`.

   [10 Marks]

Part B Tasks

Testing frequently, complete the following functions in the file `laser.c`:

1. `direction_t change_direction(direction_t direction, char cell);`

   This function takes the direction of the head of the laser beam and the cell on the board that the beam is entering and returns the direction of the outgoing head of the laser beam.

   Hint: The direction of the laser beam changes only if the cell is a mirror.

   [7 Marks]

2. `char step_laser(board_t board, laser_state_t *laser);`

   This function advances the head of the laser beam by one step (or cell) and returns the character located at the new cell.

   Use `change_direction` to compute and update the laser direction before advancing the position. Remember to update the laser message as well.

   [10 Marks]

3. `bool shoot(board_t board, laser_state_t *laser, bool trace);`

   This function updates the laser state by advancing the laser beam until one of the `BLOCKERS` (`"?@#>"`) is hit and returns true only if the laser beam terminated on an `END` cell (`'@'`).

   When the parameter `trace` is `true`, any board cell of type `PASSTHROUGHS` that is on the laser path (including the initial position) should be updated with '−', '|', or '+' such that the board displays the path of the laser beam. On the other hand, when the parameter `trace` is `false`, the board cells should remain unchanged.

   Note: When `trace` is `true`, remember to replace only `PASSTHROUGHS` cells.

   Hint: Use `step_laser` to update the laser position until the cell returned is one of `BLOCKERS`.

   [15 Marks]

4. `bool solve(board_t board, laser_state_t laser, const char target[MAX_MSG_LEN]);`

   This function should attempt to replace certain `CHOICE` (`` `?` ``) cells with one of `CHOICES` (`"\/ "`) in order to find a solution to the board given the current laser state and target message.

   A board is considered to be solved only if there exists an assignment of mirrors such that the laser beam terminates at an `END` cell and the message accumulated by the path of the laser upon termination, is the same as the `target` message.

   If a solution can be found the function should return true, otherwise false.

   Note: This should be a recursive function for full marks.

   [15 Marks]

5. Use the supplied `main` function to find and print out the solution for the board given in file `biscuit.txt` using the target word: `BIUC`. Copy paste the output at the end of `laser.c` file.

   [3 Marks]

Building and Testing

The folder `laser` contains both a `Makefile` (for use with the command line and a text editor) and a `CMakeLists.txt` (for use with CLion, if you wish to use an IDE).

Targets: Both the `Makefile` and the `CMakeLists.txt` specify 2 targets:

1. `laser` runs the supplied `main` function in `laser.c`.
2. `test` runs all the tests for Parts A and B.

If using `make`, enter the `laser` folder and build a specific target (e.g. `make laser`, `make test`).

You can now run each target from the `laser` folder. You should at least run the following command, as this is what **the auto-tester will run**:

1. `./test`

If using CLion, import your project using: `File` → `New CMake Project from Sources` and open the `laser` folder as: `Open Existing Project`.

You can now build and run each target by selecting its configuration from: `Run` → `Edit Configurations` and then running it with `Run` → `Run`.

Important note to Windows users: As your code needs to compile in the test environment, we suggest you ssh on a lab machine and test your code there as well. A common issue is detecting the end of line using only '\r' as delimiter. You should use '\n' as well.

Important: In case you accidentally overwrite any file, you can revert it to a previous version using: `git checkout <git-commit-hash> -- <file-name>`

Good luck!