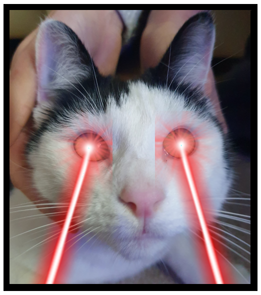
`MSc (Computing Science) 2021-2022`
`C/C++ Laboratory Examination (Timed Remote Assessment)`

Imperial College London

Tuesday 11 January 2022, 10h00 – 12h00



- ☞ You are advised to use the first 10 minutes to read through the questions.
- ☞ You may work locally on your own computer or connect to DoC lab computers and work on them.
- ☞ You are required to add to the header file **laser.h** and the implementation file **laser.cpp** according to the specifications overleaf.
- ☞ Source files **laser.cpp**, **laser.h** and **main.cpp**, and data files **biscuit.txt**, **biscuit-plan.txt**, **happynewyear.txt**, and **happynewyear-plan.txt** are in the **skeleton.zip** file which you can download from `https://www.doc.ic.ac.uk/~wjk/skeleton.zip`.
- ☞ If you are missing files or if you have queries about the specification please email the examiners (`wjk@imperial.ac.uk` and `fp910@imperial.ac.uk`).
- ☞ Submit your **laser.cpp** and **laser.h** files into the AnswerBook system **before** the end of the exam.
  LATE SUBMISSIONS WILL NOT BE ACCEPTED.
- ☞ You are advised to **save your work regularly** and to **make regular submissions of your work into AnswerBook**.
- ☞ No communication with any other student or persons other than the examiners is permitted.
- ☞ **This question paper consists of 6 pages.**

Problem Description

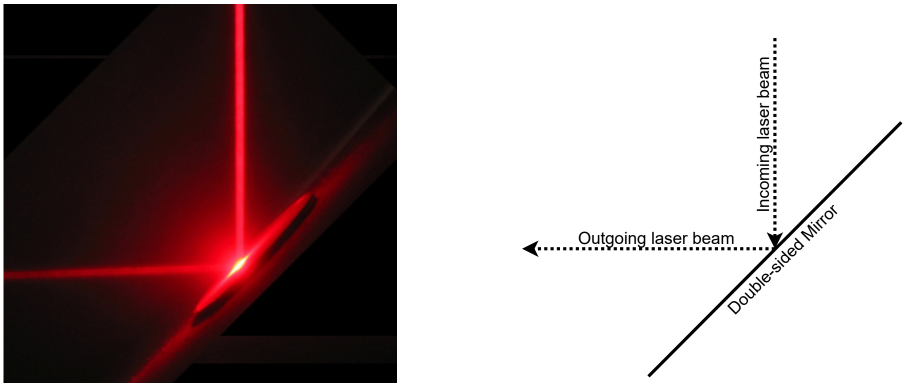


Figure 1: Laser beam striking one side of a double-sided mirror (left) and corresponding diagram (right).

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

Figure 2: Two-sided mirror layout (left) and path of laser beam which spells out the 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 the laser, ’/’ and ’\’ the mirrors (placed only at 45° angles relative to the laser light), ’#’ the edges of the board, and ’@’ a laser light absorber. 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 (’/’) or backwards-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.

Pre-supplied functions and files

To get you started, you are initially supplied with some functions (with prototypes in **laser.h** and implementations in the file **laser.cpp**):

1. `char **allocate_2D_array(int rows, int columns)` is a helper function that dynamically allocates a two-dimensional (`rows` × `columns`) array of characters, returning the 2D array.
2. `deallocate_2D_array(char **array, int rows)` is a helper function that frees up the dynamically allocated 2D array `array`.
3. `char **load_board(const char *filename, int &height, int &width)` is a function which reads in a board from the file with name `filename`, sets the output parameters `height` and `width` according to the dimensions of the board, and returns a 2D (`height` × `width`) array of characters representing the board.
4. `void print_board(char **board, int height, int width)` is a function which prints out the board stored in the 2D (`height` × `width`) array of characters `board`. Row and column numbers are also shown.

You are supplied with a main program in **main.cpp** and boards in **biscuit.txt**, **biscuit-plan.txt**, **happynewyear.txt** and **happynewyear-plan.txt**. As illustrated in Figure 3, **biscuit.txt** (resp. **happynewyear.txt**) contains the fully-specified board for the message "BISCUIT" (resp. "HAPPYNEWYEAR") while **biscuit-plan.txt** (resp. **happynewyear-plan.txt** contains a partially-specified board for the message "BISCUIT" (resp. "HAPPYNEWYEAR").

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

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

Figure 3: The contents of **biscuit.txt** (left) and **biscuit-plan.txt** (right)

Finally, **laser.h** declares an enumerated type `Direction` which can take on values `NORTH`, `EAST`, `SOUTH` and `WEST` (being the directions in which laser light can travel).

Specific Tasks

1. Write a function `find_laser(board, height, width, row)` which scans the leftmost column of the `height` × `width` 2D array of characters `board` for the character '`>`' (representing the laser). Where the leftmost column of `board` contains the character, output parameter `row` should contain the corresponding row index, and the function should return `true`; otherwise `row` should be set to -1 and the function should return `false`.

   For example, the code:

   ```
   int height, width, row;
   char **board = load_board("biscuit.txt", height, width);
   bool success = find_laser(board, height, width, row)
   ```

   results in `success` set to `true` and row set to `2`.

2. Write a function `mirror_label(board, height, width, row, column)` which returns the alphabetical character label (if any) attached to the mirror found at coordinates (`row`, `column`) within the `height` × `width` 2D array of characters `board`. If there is no mirror at the coordinates, or if there is no alphabetical character attached then the function should return the character '`\0`'.

   For example, the code:

   ```
   char **board = load_board("biscuit.txt", height, width);
   char label = mirror_label(board, height, width, 2, 15);
   ```

   results in `label` set to '`B`'. Whereas the code:

   ```
   char **board = load_board("biscuit.txt", height, width);
   char label = mirror_label(board, height, width, 4, 24);
   ```

   results in `label` set to '`\0`'.

3. Write a Boolean function `shoot(board, height, width, message, last_row, last_col)` which plots the path of the laser beam across the given `board`, updating `board` and output string `message` appropriately. The output parameters `last_row` and `last_col` should be set to the final coordinates of the laser beam, which should terminate at an edge, the laser light absorber or a '`?`'. The function should return `true` if and only if the final coordinates of the laser beam correspond to the laser light absorber.

For example, the code:

```
board = load_board("biscuit.txt", height, width);
int last_row, last_col;
success = shoot(board, height, width, message,
  last_row, last_col);
print_board(board);
```

results in the output shown on the right in Figure 2. Further, `success` is set to `true`, `message` is set to “BISCUIT”, `last_row` is 9 and `last_col` is 31.

If the laser beam encounters a ’`?`’ or a ’`#`’ (edge) then the function should terminate with a return value of `false`, `message` should reflect the letters accumulated by the laser beam so far, and `last_row` and `last_col` should be set to the coordinates of the encountered ’`?`’ or ’`#`’. As such, the code:

```
board = load_board("biscuit-plan.txt", height, width);
int last_row, last_col;
success = shoot(board, height, width, message,
  last_row, last_col);
print_board(board);
```

results in the output:

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

Now, `success` is set to `false`, `message` is set to “BIS”, `last_row` is 4 and `last_col` is 24.

4. Write a **recursive** Boolean function solve(board, height, width, target) which takes in a board containing ’?’ characters and replaces each of these by a ’/’, a ’\’ or a ’ ’ (space) in such a way that the laser beam (a) accumulates the message given by input string target and (b) terminates at the laser light absorber. If these objectives can be met, the function should return true; otherwise the function should return false.
   For full credit for this part, your function should be recursive.

   For example, the code:

   ```
   int height, width;
   board = load_board("biscuit-plan.txt", height, width);
   success = solve(board, height, width, "BISCUIT");
   print_board(board, height, width);
   ```

   results in the output shown on the left in Figure 2, and success is true.

Place your function implementations in the file **laser.cpp** and corresponding function declarations in the file **laser.h**. Use the file **main.cpp** to test your functions. You may find it convenient to create a **makefile** which compiles your submission into an executable file called **laser**. Submit your **laser.cpp** and **laser.h** files into AnswerBook regularly. **You should not hand in any other files**.

(The four parts carry, respectively, 15%, 20%, 40% and 25% of the marks)

Hints

1. The character for the backward-facing mirror ’\’ is represented in C++ as ’\\’ (necessary because ’\’ is used for escaping characters).
2. Questions 1 and 2 can be tackled independently. Questions 3 and 4 will be much easier if you exploit the answers to previous questions.
3. Feel free to define any auxiliary functions which would help to make your code more elegant. For example, in Question 3, an auxiliary function which moves the laser beam forward by one step is extremely useful.
4. You are explicitly required to use recursion in your answer to Question 4. You are not obliged to use recursion in answering any other question.
5. A highly efficient approach to Question 4 exploits how Question 3 is designed to behave when it encounters a ’?’.