In your midterm, you are going to write a puzzle solver. Please follow the steps given below in order to prevent any point reduction from your grade.

First of all, check the .txt files shared with you. In each file, the first row includes the number of rows, number of columns, and the number of words to be solved. Starting from the second row of the file, information about each word is given in the order of: its direction, starting row and column number, its name, and finally the hint about the word given to the user. Let’s examine puzzle2010.txt:

5 5 7

H 1 1 MILK White liquid produced by the mammals

H 2 1 IN Used to indicate inclusion within space, a place, or limits

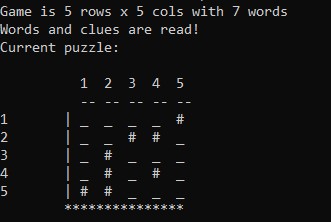
H 3 3 BUS A road vehicle designed to carry many passengers

H 5 3 DAN The name of a famous author whose surname is Brown

V 1 1 MIND A set of cognitive faculties, e.g. consciousness, perception, etc.

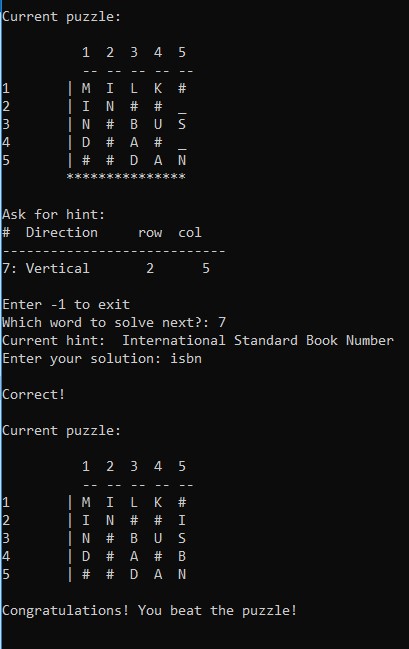
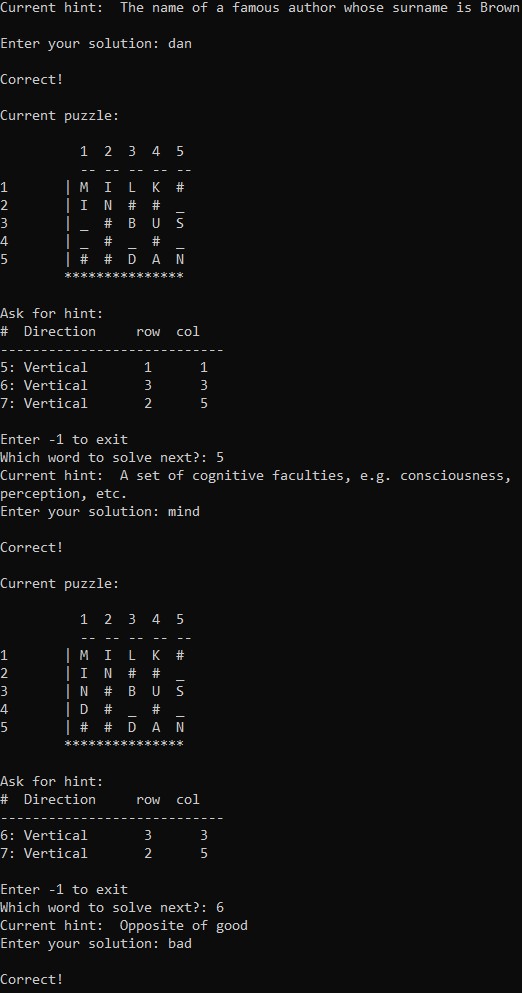
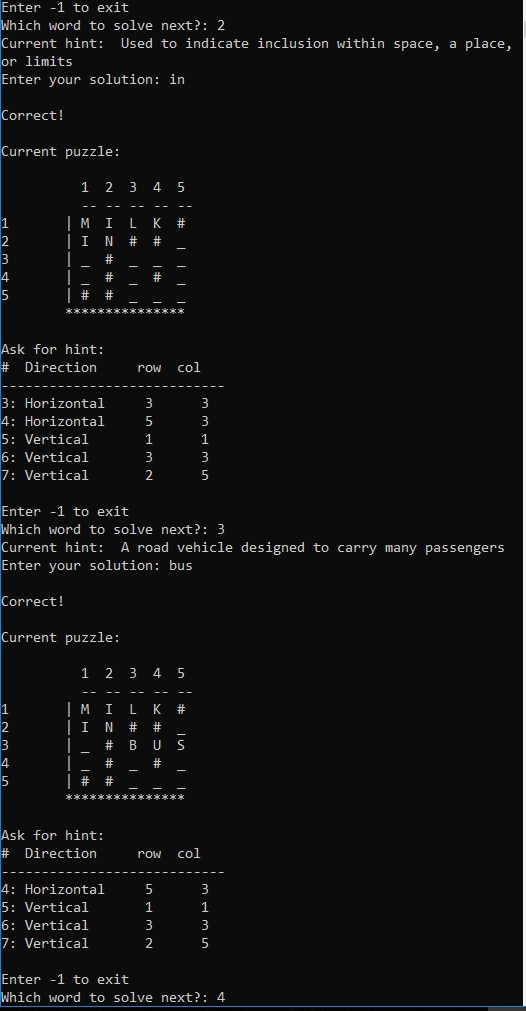
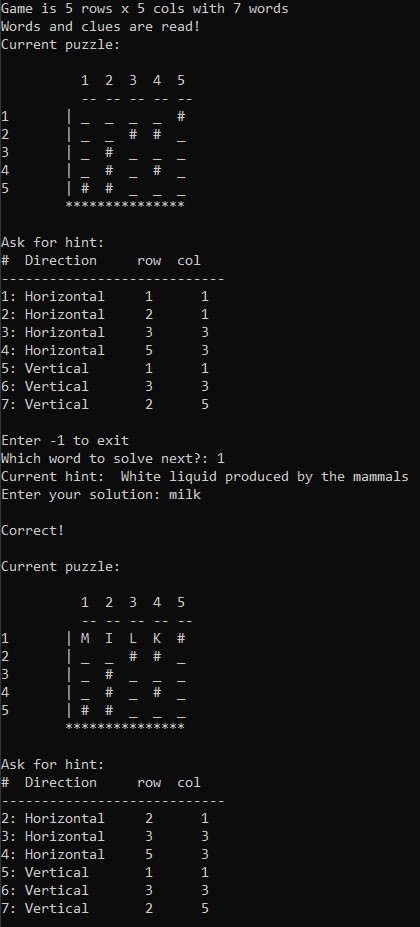
V 3 3 BAD Opposite of good

V 2 5 ISBN International Standard Book Number

In the above example (puzzle2010.txt), we have 5 rows, 5 columns and 7 words. For instance, MILK starts at position 1-1, goes horizontal, and its hint is: White liquid produced by the mammals.

Initially, our puzzle should look like the right image where words to be solved are empty and shown with underscores (‘\_’)while blank areas are shown with #. Row and column numbers together with framing |, \* and dashes are also given.

Below, you can find step-by-step gameplay of puzzle2010.txt. Gameplay with a longer text file and also with an erroneous text file is also given at the end of this document.



First of all, include below struct and function declarations in your code.

**typedef struct**

{

**char** \*word; //word and corresponding hint **char** \*clue;

**int** x; //Starting x and y positions **int** y;

**char** direction; //H for horizontal, V for vertical **int** f; //solved or not

} Word\_t;

Word\_t\* loadTextFile( FILE\* myFile, **int** nrWords ); **void** displayBoard(**int** rows, **int** cols, **char**\*\* myBoard); **int** isBoardFilled(**int** rows, **int** cols, **char**\*\* myBoard); **char**\*\* createArray(**int** rows, **int** cols); **int** findIndex(**int** arr[], **int** size, **int** val);

**void** playGame(**char**\*\* myBoard, **int** words, Word\_t \*words, **int** x, **int** y, **int** countToFinish);

**char**\*\* updateBoard(**char**\*\* myBoard, Word\_t \*words, **int** solve);

Function explanations (you can add more variables to each function and add more functions to your code if you need):

* main:
  + declarations you may need:

 a dynamic array of board (e.g. **char**\*\* board)  a file pointer to the file

 number of rows, columns and words  a dynamic array of Word\_t type

* + functionality  get the name of the file from the user and open the file

 read number of rows, columns and words

 dynamically allocate your Word\_t type array in the size of words

 call loadTextFile function and assign the returned array to your Word\_t type array  call createArray and assign the result to your board  call playGame

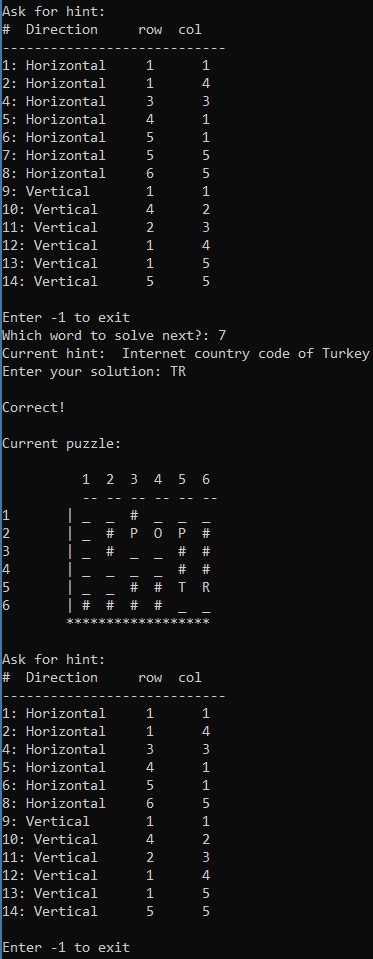
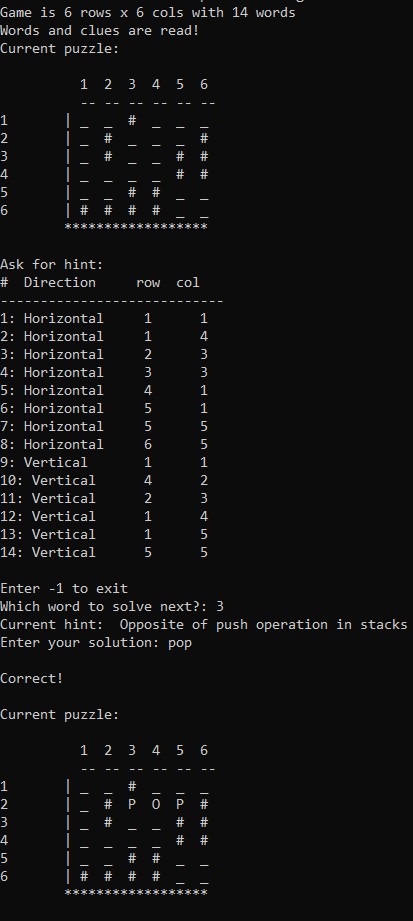
 free any dynamically allocated array

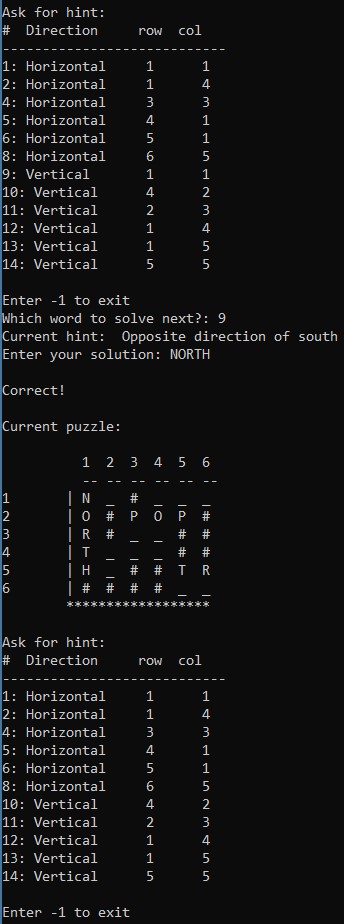
* loadTextFile:
  + Create a temporary Word\_t type array and dynamically allocate it using the number of words
  + For each element of this array, read the direction, row-column starting numbers, word to be solved and its hint from the file.
* createArray:
  + Dynamically allocate a temporary board array (remember the dynamic memory allocation of 2D arrays from the lecture. First allocate memory for number of rows, then for each row, allocate memory for columns)
  + Initialize all elements of the array to ‘#’
* displayBoard:
  + Traverse the column array and display the column number
  + Traverse the column array and print dashes as separators
  + Print row numbers followed by ‘|’ and print corresponding board element

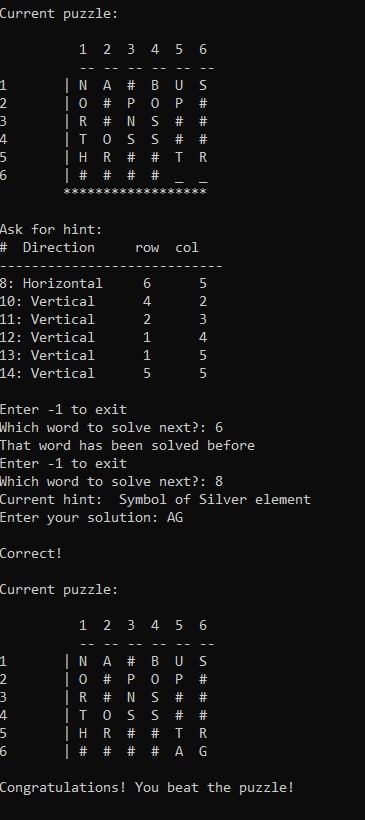
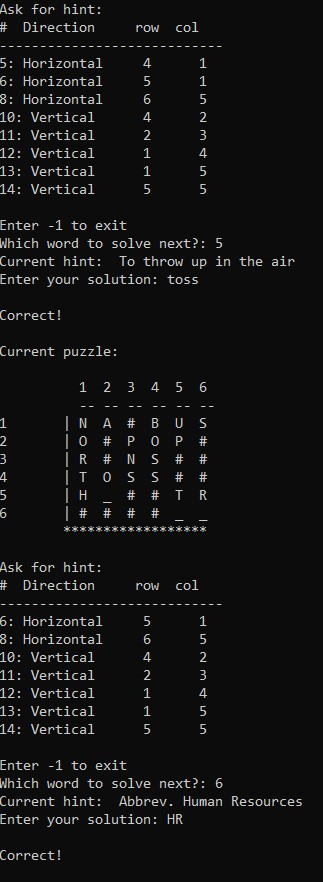
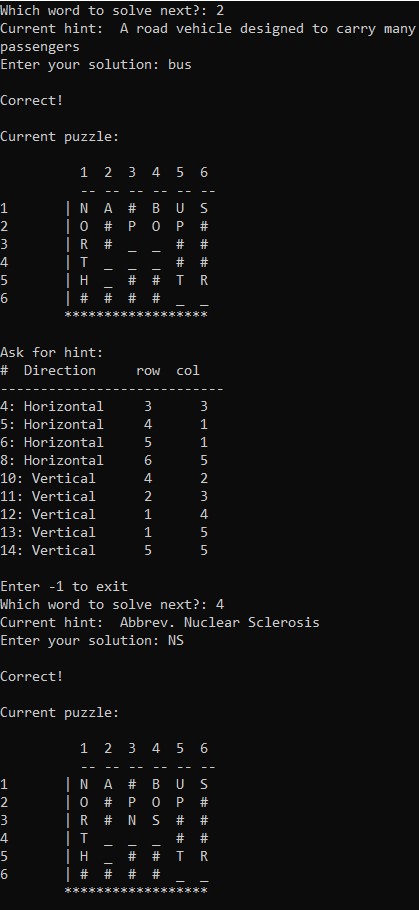
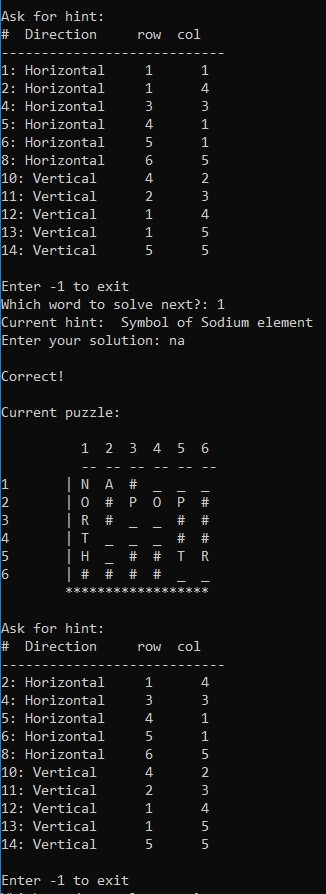
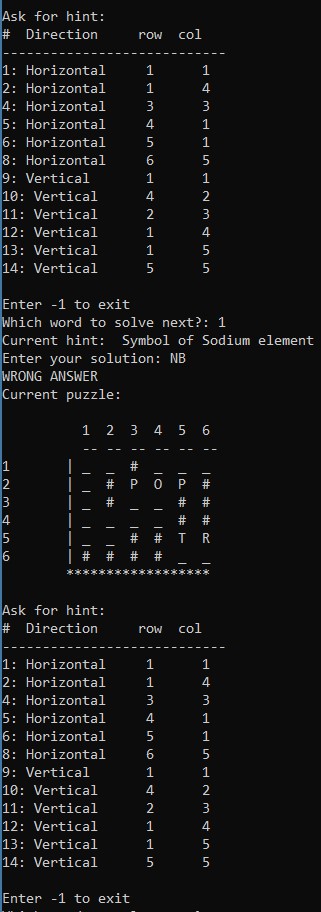
* isBoardFilled:
  + If there is any blank in the table (e.g. ‘\_’), return 0, else return 1
* findIndex:
  + Return 1 if the given value is found.
* updateBoard:
  + This function inserts the element with the index:solve into the board.
  + If the word is solved, insert the word. Else, put underscores (‘\_’) in the length of this word.
* playGame:
  + Initially, update and display the board.
  + Now, create a loop in which the game is played. Game finishes whenever gameOver reaches to the number of total words or whenever there is no more blanks in the board.
    - For each word, display the associated direction, row, column info if it is not solved before
    - Ask the user to enter a word number
    - Check the validity of the input (solved before, or valid in range, etc.)
    - Show hint
    - Get the answer
* If correct, increment gameOver counter, update the board and display the board.

▪ Check whether the board is filled or not.

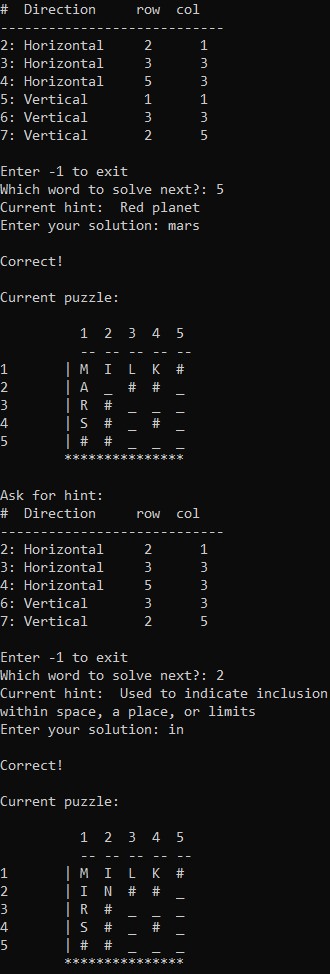
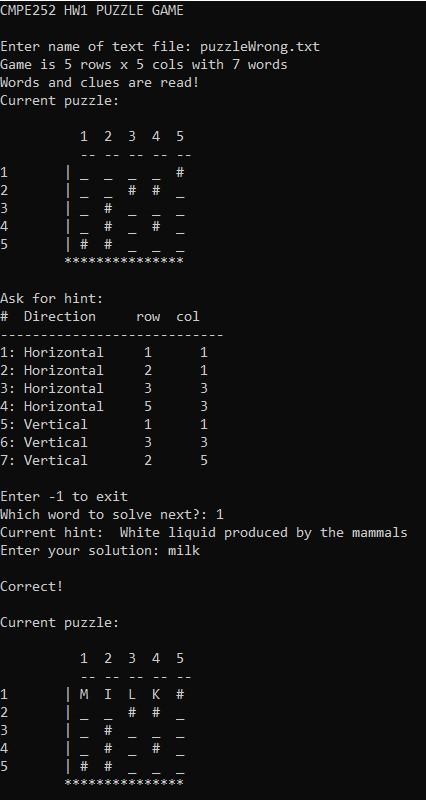
# Step-by-step gameplay of puzzle2010long.txt







Problem with the puzzlewrong.txt



The program does not have to check the final condition of the word entered before (e.g. MARS → MIRS). Therefore, assume that .txt files are prepared correctly.

Asst. Prof. Dr. Gizem Kayar

11.04.2021