



Document tabs

- Tab 1
- Introduction
 - Input to Your Program
 - Task of Searching
 - Building the Algorithms
 - Calculating the Score
 - Output of Your Program
 - Sample Runs
 - Some Important Rules
 - What and where to subm...
 - How to get help?
 - Plagiarism
 - Important Notes on Pla...

- Tab 1
- Introduction
 - Input to Your Program
 - Task of Searching
 - Building the Algorithms
 - Calculating the Score
 - Output of Your Program
 - Sample Runs
 - Some Important Rules
 - What and where to subm...
 - How to get help?
 - Plagiarism
 - Important Notes on Pla...

- Tab 1
- Introduction
 - Input to Your Program
 - Task of Searching
 - Building the Algorithms
 - Calculating the Score
 - Output of Your Program
 - Sample Runs
 - Some Important Rules
 - What and where to subm...
 - How to get help?
 - Plagiarism
 - Important Notes on Pla...

Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Summer 2025-2026

Take-Home Exam 1 – Searching for Words in a Fancy Fashion
Due: Monday, 28 July 2025, 23:55 (SHARP)

DISCLAIMER:

Only checking the sample run cases might not be sufficient as your solution will be checked against a variety of samples different from the provided samples; however checking these cases is highly encouraged and recommended.

You can NOT collaborate with your friends and discuss your solutions with each other. You have to write down the code on your own. Plagiarism will not be tolerated AND cooperation is not an excuse!

Introduction

The aim of this take-home exam (THE) is to recall CS201 material and practice on matrices (i.e., two-dimensional (2D) vectors). You are asked to search for a list of words in a character matrix, but in a way that is a bit different than what you usually do in normal Sunday morning puzzles. When searching for a word, you will calculate a score based on the existence/absence of that word in the matrix (please refer to the "Calculating the Score" section for more details).

Input to Your Program

All of the inputs in this THE are given to the program through a file, except for the name of the file itself. The filename is provided via the standard input (cin). After reading the filename from the console, your program will open this file and read it.

Format of the Input Filename Content

The first value within the file is a positive integer, *numRows*, that indicates the number of rows in the matrix of characters; followed by *numWords* number of lines, representing the actual matrix of characters. Here, each line/row is guaranteed to have the same number of characters, i.e., all matrix rows will have the same length, and thus, you don't need to carry out any input checks on that end.

The next line of the file contains another positive integer, *numWords*, which indicates the number of words that will be read and checked for their existence in the matrix; followed by *numWords* number of lines (only if specified greater than zero) where each line contains a single word. If the number of words is set to be zero (0), then no other inputs should be written afterwards.

All the character inputs in this THE (both the matrix elements, and the words to be searched) are uppercase English letters with no spaces, tabs, or special characters. You can assume that this will be true for all the test cases that will be used. Therefore, you do not need to perform any extra checks.

Regarding the integer inputs (*numRows* and *numWords*), *numRows* is a positive integer (at least one (1)), with rows and the number of columns (inferred from the first row's length) up to 100 to ensure reasonable matrix sizes for processing, and *numWords* is greater than or equal zero (0). You do not need to enforce these constraints explicitly. Regarding the words, you can also assume that all the words that will be used in all the test cases will be of length greater than two (2).

Please note that, while reading the matrix, you **must** store it in a vector of vectors. Otherwise, the final grade of your submission will be zero (0).

Below is a sample input to your THE code program:

```
10
NKENWOLDS
ICTESVOROE
DWPYBHLRS
HIOBMLKPN
DNRALDQIS
PROGGENFX
QIHASYMVC
QITINTNSOV
NYREINGLIJ
QILIKAYPPY
3
NIKE
LAMBORGHINI
RESPONSIBILITY
```

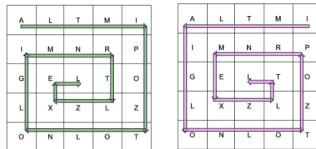
Note: As stated, there are no spaces between the characters in each row, and rows are separated by a new line character.

Task of Searching

This section clarifies the algorithm you need to develop.

After reading the matrix and the words to search for, your program will search for each word, obtained from the standard input, within the 2D-vector that your program had constructed from the matrix. This search can be done in two (2) special, fancy ways: **Spirally** (clockwise), **Counter-Spirally** (counter-clockwise).

The following 2 illustrations show possible directions of search in the matrix:



More explicitly, your program should do a word search in both of these 2 manners: **Spiral** (Green arrows) and **Counter Spiral** (Pink arrows) as shown in the above figures.

Hint: Your first goal is to fully traverse the matrix in spiral (or counter-spiral) order. Focus only on printing the characters in correct order. Do **not** attempt to solve the entire problem at once. Do **not** worry about searching for the word yet, just get the traversal logic right.

Combinations of different directions are not allowed within a word, meaning that you should be searching a word in one direction only. A word that your program is searching for may not be in the matrix at all and this is also a case that your program should handle (kindly refer to the sample runs and scoring criteria).

You can assume that a particular word will **not** occur in the matrix more than once, so you can safely stop the search regarding that word when the program finds it once.

Don't bother: The words may intersect in the matrix (i.e., two words may use a common letter) and this is totally okay for the game. You do not need to check against this. An example for this case exists in the [fourth sample test case](#) example matrix given in the [Sample Runs](#) section. Besides, even one word can contain another (for example **APART** and **PART** can both be accepted in the game).

Building Hint: Start with implementing the printing functionality for each traversal path. This will help you verify the order of characters visited. Once the traversal and printed output look correct, you can layer the search logic on top, using the same traversal but with an added check for substring matching. This step-by-step approach helps isolate and debug the traversal logic before introducing the complexity of word searching.

Tab 1

Introduction

Input to Your Program

Task of Searching

Building the Algorithms

Calculating the Score

Output of Your Program

Sample Runs

Some Important Rules

What and where to subm...

How to get help?

Plagiarism

Important Notes on Pla...

Building the Algorithms

For the two requested search methods, we have provided some hints to better understand the behavior of the two main traversal algorithms.

Think of the matrix as a set of concentric layers, starting from the outermost layer and moving inward.

For the *spiral traversal*, begin at the top-left corner and move right along the top edge of the current layer. Then turn down and traverse the right edge. Next, move left across the bottom edge, and finally move up along the left edge, stopping before you overlap the starting row.

After completing one full loop around the layer, shrink the boundaries; i.e. move the top boundary down, the bottom boundary up, the left boundary right, and the right boundary left, to focus on the next inner layer.

Repeat this process until all elements have been visited. This forms a continuous spiral path moving inward in a clockwise direction.

For the *counter-spiral traversal*, you should start at the top-right corner and move down the right edge of the current layer. Then turn left across the bottom edge, move up along the left edge, and finally traverse right across the top edge, stopping before you overlap the starting column. After each loop, adjust the boundaries inward just as before, and continue the process until all elements are visited in a counter-clockwise inward spiral.

Calculating the Score

Each word that your program can or cannot find in the matrix affects the score. The rules of the game to calculate the score are as follows:

- Words in Spiral Form:** If a word is found in the clockwise spiral path, the score increases by twice the number of letters in the word ($2 \times \text{Length}$). If the word is longer than 5 letters, an additional 2 points are added.
 - Example: "MONEY" (5 letters) contributes $2 \times 5 = 10$ points; "SCHOLAR" (7 letters) contributes $2 \times 7 = 14$ points.
- Words in Counter-Spiral Form:** If a word is found in the counter-clockwise spiral path, the score increases by twice the number of letters ($2 \times \text{Length}$) plus an extra 1 point. If the word is longer than 5 letters, it gets an additional 3 points instead.
 - Example: "MONEY" (5 letters) contributes $2 \times 5 + 1 = 11$ points; "SCHOLAR" (7 letters) contributes $2 \times 7 + 3 = 17$ points.
- Words Not Found:** For each word that cannot be found in any of the traversal paths, the total score is reduced by 5 points.

In other words, let s denote the length of the word. The score is calculated based on the traversal path where the word is found:

- Spiral:**
$$\text{Score} = \begin{cases} 2s + 2 & \text{if } s > 5 \\ 2s & \text{otherwise} \end{cases}$$
- Counter-Spiral:**
$$\text{Score} = \begin{cases} 2s + 3 & \text{if } s > 5 \\ 2s + 1 & \text{otherwise} \end{cases}$$
- Not Found:** Subtract 5 from the total score ($\text{Score} - = 5$).

Output of Your Program

The first output of your program should be the traversal strings for the spiral and the counter-spiral traversals. Each should be printed on consecutive lines of the output, with characters concatenated without spaces, but dashes ("-") inserted at points where the traversal direction changes (no trailing dash at the end).

Then, for each word searched within the matrix, aka the *searchWord*, your program should print either one of the following, depending on whether the word is found in spiral or in counter-spiral, or not found at all.

```
searchWord found with spiral search!
searchWord found with counter-spiral search!
searchWord not found!
```

This portion of the output should be surrounded by the following sequence of characters:

```
-----
```

Finally, your program should display the final score of the game, aka the *theScore*, if and only if there are words to be searched, in the following format.

```
Your score is: theScore
```

You can see the format of the output in the "[Sample Runs](#)" section. Please note that your program's output should be exactly the same as given in the sample runs, i.e., no extra spaces, empty lines, text output, etc.

To encourage step-by-step development, test cases are constructed in a way that you can get partial credits even if the search functionality is not fully implemented. Some of the test cases are focused solely on verifying only the traversal-printing output.

Sample Runs

Below, we provide some sample runs (test cases) of the program that you will develop along with an explanation for them. Your program should read the inputs as in the Input column and should print the final score as an integer value as in the Output column.

Input	.txt File Content	Visualization	Output	Explanation
matrix1_no_search.txt	5 ALTKI 18009 GELTO LXLZL OHL0T 0		ALTKI-POZT-OLNO-LGI-MN 8-TL-ZX-C-L INTLA-ZOLO-NLO-T-ZOP-RN M-EX-ZL-T-L	This is an example test case focused solely on the printing functionality. The number of words to be searched is 0. The output consists only of the spiral traversal string on the first line, and the counter-spiral traversal string on the second line.
matrix1.txt	4 GOL 246 LTCR 2 GOLDEN LOVE		XGOL-DEN-QTU-ZL-OV-E-W LOOL-LZU-TQN-ED-VU-W-E ----- GOLDEN found with spiral search! LOVE found with spiral search! ----- Your score is: 22	4 represents the number of rows of the matrix, therefore it is followed by 4 lines of input. Then, 2 represents the number of words that will be searched for in the matrix, therefore it is followed by 2 lines of input. The first output line is the string of characters resulting from the traversal of the spiral form, and the second line from the counter-spiral form. The output score is 22, as follows: Golden: Spiral form: $2 \times 6 + 2 = 12$ Love: Spiral form: $2 \times 4 = 8$ In total: $12 + 8 = 22$
matrix2.txt	4 BDEP 8ICE NIZR 1800 1 INTEREST		BDEP-TSE-RET-NI-OI-Q-Z PEXK-INT-ERE-ST-IO-Z-Q ----- INTEREST found with counter-spiral search! ----- Your score is: 19	The first line is the string of characters resulting from the spiral traversal in the given matrix. The second line is the string resulting from the counter-spiral traversal. INTEREST (8 Letters) Counter-Spiral: $2 \times 8 + 3 = 19$

Input	.txt File Content	Visualization	Output	Explanation
5 ELKSA 8GHN NIZR 1800 1 SARLSNG		ELKSA-8GHN-GHTG-RAP-EN Q-EL-PP-A-Y ASKLE-PARG-THON-USM-QW E-AP-PL-E-Y ----- SARLSNG found with	<ul style="list-style-type: none">➤ The word SARLSNG can be found with the Spiral form, thus $2 \times 7 + 2 = 16$➤ The word LEAP is not present so we deduct by 5	

Introduction

Input to Your Program

Task of Searching

Building the Algorithms

Calculating the Score

Output of Your Program

Sample Runs

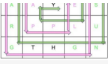
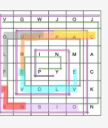
Some Important Rules

What and where to subm...

How to get help?

Plagiarism

Important Notes on Pla...

matrix3.txt	7 SAMSUNG LEAP GRAPE APPLE		spiral search: LEAP not found! GRAPE found with spiral search! APPLE found with counter-spiral search! ***** Your score is: 32	=> The word GRAPE is in Spiral form so: 2*5 = 10 => The word APPLE is in Counter Spiral form: 2*5 + 1 = 11 In total: 16 + (-5) + 10 + 11 = 32
matrix4.txt	6 VGMDOJ OUTPAC LEINMA FEPHIC TYOLVK ENNSON Z LOVELY TEN FLOUT CAP CAPTURE TENDON EVOLVE		VGMDOJ-CACKN-OTSNE-TFL O-UTPA-NEV-LOV-ER-IN-V -P ZONGV-OLTFE-NSION-HCA C-APU-REV-OLV-ER-NE-P -V ***** LOVELY not found! TEN found with counter-spiral search! FLOUT found with spiral search! CAP found with counter-spiral search! CAPTURE found with counter-spiral search! TENDON found with counter-spiral search! EVOLVE found with counter-spiral search! ***** Your score is: 68	=> The Letters C, A, P in capture are common in desired words; 1 CAPTURE and 2 CAPTURE => The letter E is common between CAPTURE and EVOLVE => Letters T, E, and N are common in TEN and TENDON => Letters U and T are common between FLOUT and CAPTURE In total: -5 + 7 + 10 + 7 + 17 + 17 + 15 = 68

Some Important Rules

In order to get full credit, your program must be efficient, *modular* (with the use of functions), *well commented* and *properly indented*. Besides, you also have to use understandable identifier names. Presence of any redundant computation, bad indentation, meaningless identifiers or missing/irrelevant comments *may decrease your grade* in case that we detect them. When we grade your THEs, we pay attention to these issues. Moreover, **we may test your programs with very large test cases**. Hence, take into consideration the efficiency of your algorithms other than correctness.

Sample runs give a good estimate of how correct your implementation is, however, *you will* test your programs with *different* test cases and **your final grade may conflict with what you have seen on CodeRunner**. We will also **manually** check your code, indentations and so on, hence do *not* object to your grade based on the **CodeRunner** results, but rather, consider every detail on this documentation. **So please make sure that you have read this documentation carefully and covered all possible cases, even some other cases you may not have seen on CodeRunner or the sample runs**. The cases that you do not need to consider are also given throughout this documentation.

Submit via SUCourse ONLY! Paper, e-mail or any other methods are not acceptable.

The internal clock of SUCourse might be a couple of minutes skewed, so make sure you do not leave the submission to the last minute. In the case of failing to submit your THE on time:

"No successful submission on SUCourse on time = A grade of zero (0) directly."

What and where to submit (PLEASE READ, IMPORTANT)

It'd be a good idea to write your name and last name in the program (as a comment line of course). *Do not use any Turkish characters anywhere in your code* (not even in comment parts). If your full name is "Duygu Karaoglan Altop", and if you want to write it as comment, then you must type it as follows:

```
// Duygu Karaoglan Altop
```

You should copy the full content of the .cpp file and paste it into the specified "Answer" area in the relevant assignment submission page on SUCourse. ***Please note that the warnings are also considered as errors on CodeRunner, which means that you should have a compiling and warning-free program.***

Since the grading process will be automatic, you are expected to strictly follow these guidelines. *If you do not follow these guidelines, your grade will be zero (0)*. Any tiny change in the output format **will** result in your grade being zero (0). Thus, please test your programs yourself, and against the sample runs that are available at the relevant assignment submission page on SUCourse.

In the CodeRunner, there are some visible and invisible (hidden) test cases. You will see your final grade (including hidden test cases) before submitting your code. There is no re-submission. You don't have to complete your task in one time, you can continue from where you left last time but you should not press submit before finalizing it. Therefore, you should make sure that it's your final solution version before you submit it. Also, we still do not suggest that you develop your solution on CodeRunner but rather on your IDE on your computer.

You may visit the office hours if you have any questions regarding submissions.

Introduction

Input to Your Program

Task of Searching

Building the Algorithms

Calculating the Score

Output of Your Program

Sample Runs

Some Important Rules

What and where to subm...

How to get help?

Plagiarism

Important Notes on Pla...

Introduction

Input to Your Program

Task of Searching

Building the Algorithms

Calculating the Score

Output of Your Program

Sample Runs

Some Important Rules

What and where to subm...

How to get help?

Plagiarism

Important Notes on Pla...

How to get help?

You may ask your questions to TAs or to the instructor. Information regarding the office hours of the TAs and the instructor are available at SUCourse.

Plagiarism

Plagiarism is checked by automated tools, and we are very capable of detecting such cases. Be careful with that. Exchange of abstract ideas are totally okay but once you start sharing the code with each other, it is very probable to get caught by plagiarism. So, do **NOT** send any part of your code to your friends by any means or you might be charged as well, although you have done your THE by yourself. THEs are to be done personally and you have to submit your own work. **Cooperation will NOT be counted as an excuse.**

Important Notes on Plagiarism and Use of GenAI Tools

- Plagiarism is **strictly** prohibited and checked using automated tools. We are very capable of detecting such cases.
- It is fine to discuss **abstract ideas or approaches**, but sharing code, even a single line, is not acceptable.
- **Do NOT send your code** to your friends or classmates, even if they are stuck or you wrote the code entirely by yourself. If your code appears in someone else's submission, you will both be held accountable.
- Each THE must be completed individually. Submissions must reflect your own work and understanding. Cooperation or claiming "I only helped" will **not** be accepted as an excuse.
- You may refer to GenAI tools for learning purposes. Such as understanding how a spiral traversal works. But, you are not allowed to copy-paste code from such tools or use them to generate your submission. **Your work must be your own.**

If the code appears AI-generated or identical to others, it will be treated as plagiarism.

In case of plagiarism, the rules on the [Syllabus](#) apply.

Good Luck!
Parsa Yousefinezhad