Sabancı University

Faculty of Engineering and Natural Sciences

CS204 Advanced Programming
Spring 2023

Homework1 - Shape Placement in Matrix

Due: March 15, 2023, Wednesday, 21:00

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases (with the exceptions of assumptions given in the homework text); you are expected to take actions accordingly!

You can NOT collaborate with your friends and discuss solutions. You have to write down the code on your own. Plagiarism will not be tolerated!

Introduction

The aim of this homework is to practice on the topic of two dimensional (2D) matrices implemented using vector of vector. In this homework, you will read character data from text file and from the standard input (keyboard) as well. Then you will use a **`vector`** of **`vector`** of **`char`** (i.e., **`char`** matrix) as the data structure to store and manipulate the input data to achieve our goal of placing an irregular shape. The representation of this irregular shape to be placed has its own string-based encoding format. Additional information is given in the rest of this document.

Input

We have both file and keyboard input in the homework. File input is to read the source matrix content. Keyboard inputs are for the shapes to be placed.

Input from file

You will read a text file that contains a 2D matrix of characters structured in rows and columns. In each row, there are same number of characters and there could be any number of rows. In the file, only '`-`' and '`#`' characters can exist. Figure 1 shows an example of such a sample input file content with 3 rows and 5 columns (indices are added to the figure as extra information; they are not part of the file content). Your program should work with all files having structures of any number of rows and any number of columns.

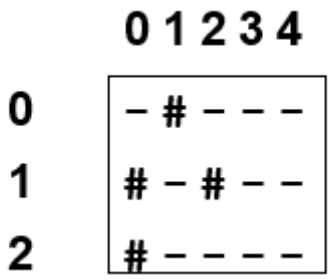

Figure 1. Sample input file (indices are added for informative reasons; they are not part of the file content)

You should read the content of this file into a **`vector`** of **`vector`** of **`char`** data structure. You are not allowed to use another structure for this purpose.

We strongly recommend you to read the file line-by-line using `getline` function, since it reads a line entirely into a string by automatically stripping off the invisible end-of-line character(s). After that, you can process the line string as you want. However, if you read by some other means, you take the risk of processing the invisible end-of-line characters; this might cause inconsistencies in different operating systems.

There are some input checks that you should perform for the file content.

- Your program should check for consistency in the number of characters in each row. In other words, all rows must be of the same length.
- Your program should check that the file only contains dashes '-' and hashes '#'. Any other symbols in a file will make that file invalid.

In case of a problem with the file content, your program must display a generic error message and quit. See the sample runs for the error message content.

Before reading the file content, you will need to read the file name and open it. There are some checks to be done with file opening/existence as well, but this is going to be explained in the "Program Flow" section below.

Input from keyboard

After reading the input file, your program will read multiple queries, each as a string of a specific structure and encoding to represent an irregular shape. Below are the rules and assumptions of this encoding scheme.

A valid encoding consists of a sequence of character pairs, optionally separated by one or more forward slash '/' symbol(s).

- Each character *pair* is a digit followed by a letter.
    - The digit is a single digit in the range 1 to 9 (inclusive).
    - The letter is either lowercase *s* or lowercase *b*.

<u>Interpretation of the encoding:</u>

- Letter *s* corresponds to ***** character (star-shaped symbol; a.k.a. asterisk).
- Letter *b* corresponds to blank character.
- The digit preceding the letter represents the number of consecutive occurrences of that letter.
- A forward slash represents a line break.

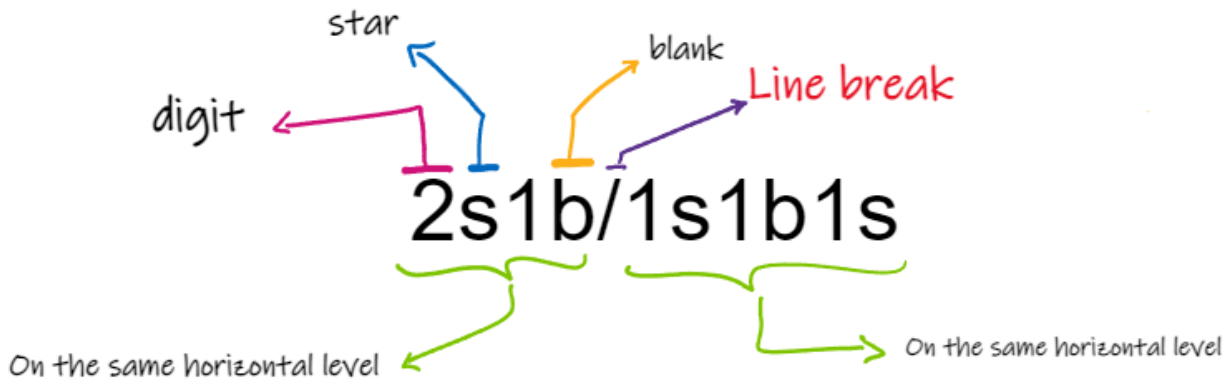

Figure 2. A sample of a string query

Figure 2 shows an example shape encoded as a string. The constructed shape out of this query will have two stars (**2s**) in the top row of the shape followed by a single empty cell (**1b**). Then, we advance the line (because of the line break **'/'**). On the line below, there will be a single star (**1s**), followed by a single empty cell (**1b**), followed by a single star (**1s**). Figure 3 shows the shape visually (Note: the "**1s1s1b/1s1b1s**" string also represents the same shape of Figure 3). Please refer to Figure 4 for more examples and their visuals.

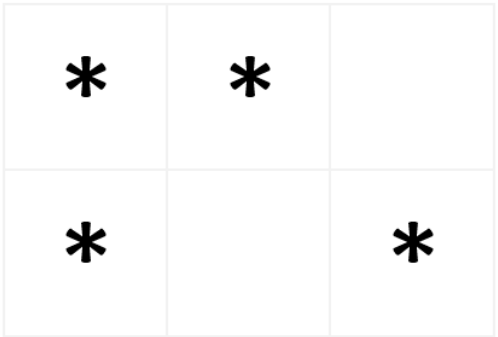

Figure 3. Visual representation of the string in Figure 2

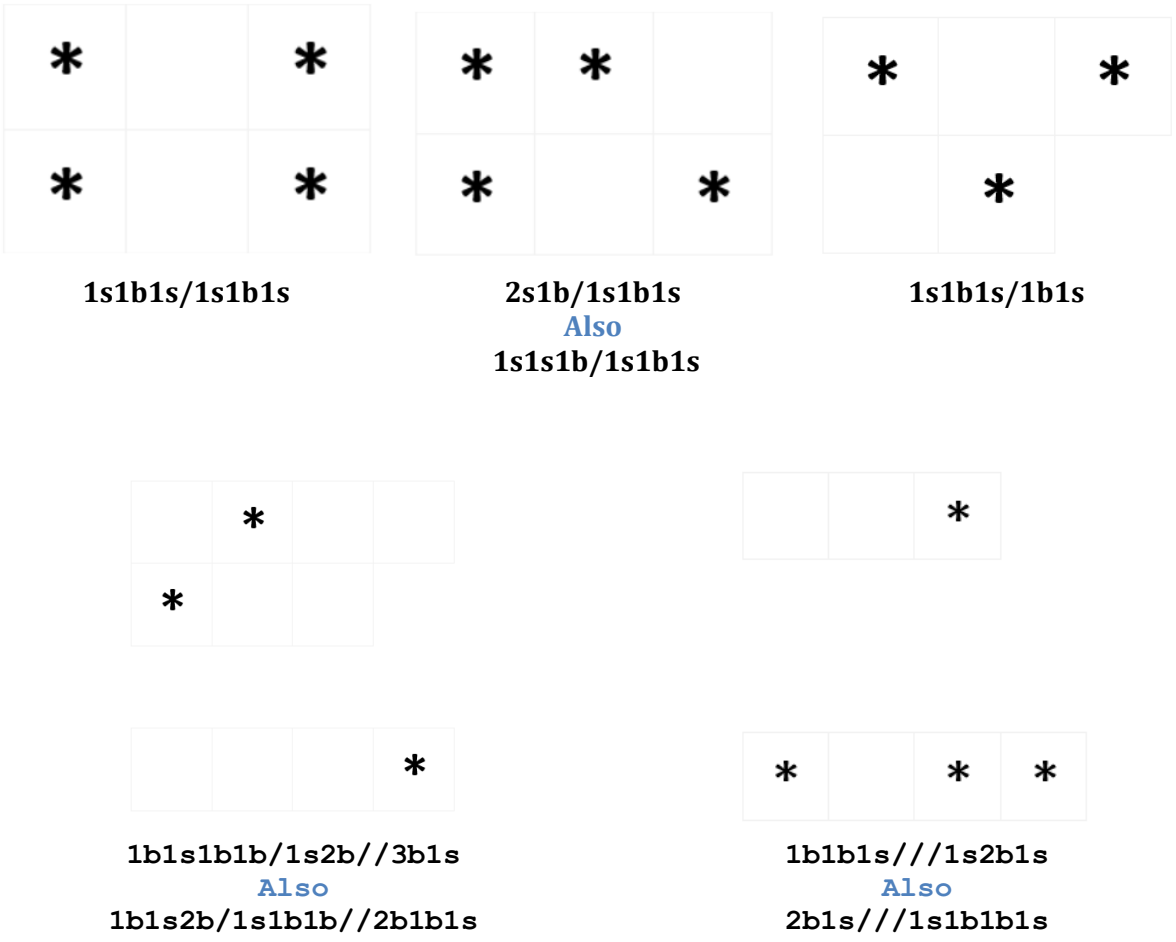

Figure 4. Extra examples for shapes and their corresponding string representations

Assumptions (you will **not** check these cases; we will **not** test your codes using the cases violating them)

- The abovementioned structure holds; no extra characters are used; no encoding rule issues (i.e., the input string is always valid).
- There will **not** be any slashes at the beginning and at the end of the string. This implies that the string will always start with a digit and end with a letter. It is also guaranteed that there will not be any empty lines at the beginning and at the end of the shape.
- There will be at least one star in each side of the outer boundary of the shape. That means each of the first row, first column, last row, and last column will not be all-blank.

Output

Your program's output will be all of the possible *full placements* of the shape, which is represented by the given string query, on the matrix that was read from the file.

Definition of *full placement:* We place only the asterisks of the shape. For a shape to be fully placed somewhere on the matrix, all the asterisks of the shape must relatively match with dash cells in the original matrix. The blanks of the shape might match with either dash or hash characters of the matrix. Figure 5 shows some example full placements.

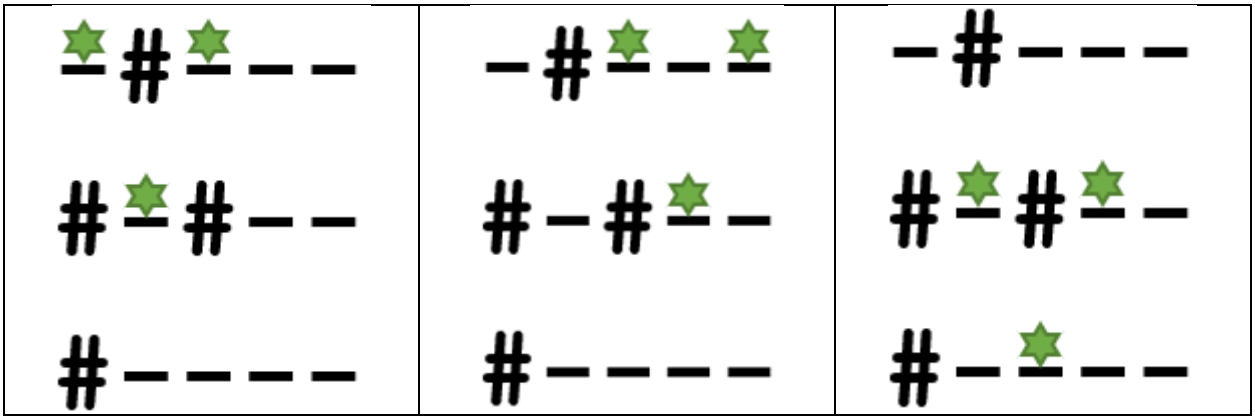

Figure 5. All possible full placements of shape **1s1b1s/1b1s** on the matrix mentioned in Figure 1. Sample run 1 demonstrates this case.

For each of the possible full placement of the shape on the matrix, your program should produce an output as a sequence of cell indices in (row, column) format for the matches of the asterisks.

- Each cell is represented as a tuple of the row and column indices of that cell on the original matrix.
- The output should be ascendingly sorted, primarily according to the row number, and for the cells on the same row, according to the column number.
- These tuples must be written on the same line using the format given in the sample runs.
- Remark that upper-left index of the matrix is (0,0).

The abovementioned rules are displaying a possible full placement. Your program should actually display all possible full placements that might have overlaps. Thus, there might be same cells in different full placements of a particular shape.

The printing order of possible full placements is the ascending order of the index of the first matched asterisk (primarily row indices; for the cells in the same row, secondarily column indices). Please check out sample runs for different cases.

It is also possible not to have any possible full placements. In such a case, you have to display a message (again please see the sample runs).

Program flow

Your program will start by asking the user to enter a file name for the input text file. Then, your program will open the file and check if it is opened successfully or not. If, for some reason, the program could not open the file, it should keep asking for a valid file name until a valid one is provided and opened. When the program opens the file and reads its content into the matrix, firstly it should check for the validity of the matrix by checking the rules mentioned in the "Input" section above. If the matrix is not valid, then the program should display an error message and terminate.

Then, if the matrix is a valid one, your program should print the matrix on the screen and then proceed with reading string queries (for the shapes to be placed) until the user enters the word **`Quit`** to stop the program execution and terminate (the word **`Quit`** is case-sensitive).

For every input string query, your program should search for the corresponding shape in the original matrix. If the shape can be *fully placed* in the matrix, your program should display all possible full placements to the screen following the rules mentioned above in the "Output" section.

For the output messages and the format, please check out the sample runs.

Some Important Rules

In order to get a full credit, your programs must be efficient and well presented, presence of any redundant computation or bad indentation, or missing, irrelevant comments are going to decrease your grades. You also have to use understandable identifier names, informative introduction and prompts. Modularity is also important; you have to use functions wherever needed and appropriate.

When we grade your homework, we pay attention to these issues. Moreover, in order to observe the real performance of your codes, we may run your programs in *Release* mode and **we may test your programs with very large test cases**. Of course, your program should work in *Debug* mode as well.

Please do not use any non-ASCII characters (Turkish or other) in your code (not even as comments).

You are not allowed to use codes found somewhere online or other than course material. You can use only the material that are covered in lectures (CS201 and CS204 so far). However, if you use a non-standard C++ library/function/class (such as strutils) covered in the courses, you have to submit your main code together with these extra source and header files; otherwise CodeRunner cannot run your code.

You are allowed to use sample codes shared with the class by the instructor and TAs. However, you cannot start with an existing .cpp or .h file directly and update it; you have start with an empty file. Only the necessary parts of the shared code files can be used and these parts must be clearly marked in your homework by putting comments like the following. Even if you take a piece of

code and update it slightly, you have to put a similar marking (by adding "`and updated`" to the comments below.

```
/* Begin: code taken from lab1.cpp */
…
/* End: code taken from lab1.cpp */
```

Submission Rules (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). For example, If your full name is "Satılmış Özbugsızkodyazaroğlu", then you must type it as follows:

// Satilmis Ozbugsizkodyazaroglu

We use CodeRunner in SUCourse for submission. No other way of submission is possible. Since the functionality part of the grading process will be automatic, you have to strictly follow these guidelines; otherwise we cannot grade your homework.

The advantage CodeRunner it is that you will be able to test your code against sample test cases. However, the output should be exact, but the textual differences between the correct output and yours can be highlighted (by pressing "show differences" button) on the submission interface.

You should copy the full content of the main .cpp file and paste it into the specified “Answer” area in the relevant assignment submission page on SUCourse. Then you can test your code via CodeRunner against the sample runs (by pressing the "Check" button). Since you will not upload a file, your local cpp file name is not important.

Even any tiny change in the output format will result in your grade being zero (0) for that particular test case, so please test your programs yourself, and against the sample runs that are available at the relevant assignment submission page on SUCourse (CodeRunner).

In the CodeRunner, there are some visible test cases. However, none of them will be used in grading. The grading test cases will be different and it is always a possibility that your program works with these given test cases, but does not work with one or more grading test cases. Thus you have to test your program thoroughly.

You have to manually "Submit" after you test and finish with your code (there is no automatic submission). There is no re-submission. That means, after you submit, you cannot take it back. On the other hand, this does not mean that you have one shot to test CodeRunner. 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.

Last, even if you cannot completely finish your homework, you can still submit.

Sample Runs and Sample Input Files

Sample runs are given below, but these are not comprehensive, therefore you have to consider **all possible cases** to get full mark. Inputs are shown in **bold**.

The test input files are also provided in the homework package. In C-Lion, input text files must be placed in the folder that starts with cmake-build-. This is something different than Visual Studio.

We will configure CodeRunner to test these sample runs for you. However, grading test cases will be totally different.

Sample run 1

```
Please enter the file name:
m0.txt
The matrix file contains:
-#---
#-#--
#----
Enter the string query for a shape, or "Quit" to terminate the program:
1s1b1s/1b1s
Placement number 1:
(0,0) (0,2) (1,1)
Placement number 2:
(0,2) (0,4) (1,3)
Placement number 3:
(1,1) (1,3) (2,2)
Enter the string query for a shape, or "Quit" to terminate the program:
Quit
```

Sample run 2

```
Please enter the file name:
m0.txt
The matrix file contains:
-#---
#-#--
#----
Enter the string query for a shape, or "Quit" to terminate the program:
9s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
1s/1s/1s/1s/1s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
1s/3b1b1s/1s
```

```
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
1s/3b1b1s/1b1s
Placement number 1:
(0,0) (1,4) (2,1)
Enter the string query for a shape, or "Quit" to terminate the program:
Quit
```

Sample run 3

```
Please enter the file name:
textFile.txt
The matrix file contains:
-#--
----
--#-
#---
---#
Enter the string query for a shape, or "Quit" to terminate the program:
3s
Placement number 1:
(1,0) (1,1) (1,2)
Placement number 2:
(1,1) (1,2) (1,3)
Placement number 3:
(3,1) (3,2) (3,3)
Placement number 4:
(4,0) (4,1) (4,2)
Enter the string query for a shape, or "Quit" to terminate the program:
1s2s1b/1s2b1s
Placement number 1:
(1,0) (1,1) (1,2) (2,0) (2,3)
Enter the string query for a shape, or "Quit" to terminate the program:
1b1s2s////1s2b1s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
1b1s2s///1s2b1s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
1b1s2s//1s2b1s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
1b1b2s//1s2b1s
Placement number 1:
(0,2) (0,3) (2,0) (2,3)
```

```
Enter the string query for a shape, or "Quit" to terminate the program:
Quit
```

Sample run 4

```
Please enter the file name:
m.txt
Could not open the file. Please enter a valid file name:
mm.text
Could not open the file. Please enter a valid file name:
inputFile.txt
Erroneous file content. Program terminates.
```

Sample run 5

```
Please enter the file name:
m3
Could not open the file. Please enter a valid file name:
m3.txt
Erroneous file content. Program terminates.
```

Sample run 6

```
Please enter the file name:
m4.txt
The matrix file contains:
-
Enter the string query for a shape, or "Quit" to terminate the program:
1s
Placement number 1:
(0,0)
Enter the string query for a shape, or "Quit" to terminate the program:
1s1b1s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
Quit
```

Sample run 7

```
Please enter the file name:
m5.txt
The matrix file contains:
-#-#--#--#-#
--##--#-----
```

```
--#------###
-----###----
--##----###-
-#-#---#----
-----##--##-
--#-#---#---
--##--##-#--
-#--#--#--#-
-----##-----
--#------##-
```

Enter the string query for a shape, or "Quit" to terminate the program:
1s///5s///5s
Placement number 1:
(0,0) (3,0) (3,1) (3,2) (3,3) (3,4) (6,0) (6,1) (6,2) (6,3) (6,4)
Enter the string query for a shape, or "Quit" to terminate the program:
1s///5s///5s////5s/2s
Placement number 1:
(0,0) (3,0) (3,1) (3,2) (3,3) (3,4) (6,0) (6,1) (6,2) (6,3) (6,4) (10,0) (10,1) (10,2) (10,3) (10,4) (11,0) (11,1)
Enter the string query for a shape, or "Quit" to terminate the program:
4s8s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
Quit

Sample run 8

Please enter the file name:
m6.txt
Erroneous file content. Program terminates.

Sample run 9

Please enter the file name:
m7.txt
Erroneous file content. Program terminates.

Sample run 10

Please enter the file name:
m8.txt
Erroneous file content. Program terminates.

Sample run 11

Please enter the file name:
m9.txt
The matrix file contains:
#
Enter the string query for a shape, or "Quit" to terminate the program:
1s
No placements found.

```
Enter the string query for a shape, or "Quit" to terminate the program:
2s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
1b1s/1s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
Quit
```

Sample run 12

```
Please enter the file name:
matrix.txt
The matrix file contains:
----##-
Enter the string query for a shape, or "Quit" to terminate the program:
4s/2s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
2s//3s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
1s
Placement number 1:
(0,0)
Placement number 2:
(0,1)
Placement number 3:
(0,2)
Placement number 4:
(0,3)
Placement number 5:
(0,6)
Enter the string query for a shape, or "Quit" to terminate the program:
2s
Placement number 1:
(0,0) (0,1)
Placement number 2:
(0,1) (0,2)
Placement number 3:
(0,2) (0,3)
Enter the string query for a shape, or "Quit" to terminate the program:
3s
Placement number 1:
(0,0) (0,1) (0,2)
Placement number 2:
```

```
(0,1) (0,2) (0,3)
Enter the string query for a shape, or "Quit" to terminate the program:
4s
Placement number 1:
(0,0) (0,1) (0,2) (0,3)
Enter the string query for a shape, or "Quit" to terminate the program:
5s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
Quit
```

Sample run 13

```
Please enter the file name:
mat.txt
The matrix file contains:
-
#
-
-
-
#
Enter the string query for a shape, or "Quit" to terminate the program:
2s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
1s/1s/1s
Placement number 1:
(2,0) (3,0) (4,0)
Enter the string query for a shape, or "Quit" to terminate the program:
1s/1s/1s/1s
No placements found.
Enter the string query for a shape, or "Quit" to terminate the program:
1s/1b/1s
Placement number 1:
(0,0) (2,0)
Placement number 2:
(2,0) (4,0)
Enter the string query for a shape, or "Quit" to terminate the program:
Quit
```

Good Luck
Ahmed Salem and Albert Levi