Sabanci University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Summer 2017-2018

Homework 1 – Lost for Words
Due: 4 July 2018 11.55pm (SHARP)

DISCLAIMER:

Your program should be a robust one such that you have to consider all relevant user mistakes and extreme cases; you are expected to take actions accordingly!

Only checking the sample run cases might not be sufficient as your solution will be checked against a variety of samples different than the provided samples; however checking these cases are 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 homework is to recall CS201 material and practice on matrices (i.e., two dimensional vectors). You are asked to search for a list of words in a character matrix, just like you would do with Sunday morning puzzles. A score will be calculated at the end of the program according to the existence of the words and their directions in the matrix.

Inputs to Your Program

Your program should prompt for an input file name and read the file name from the standard input (cin). This file will have the characters of a matrix. The file could look like this:

```
ACLOUYATSA
SHEEPTEETK
QTCLERTYUT
HJRLDGKUTR
AOYDOOOODA
TRYTRUOAAP
OEGEZCDNTA
MRYTABLEFG
```

matrix1.txt

If the input file cannot be successfully opened, your program should clear the error, ask the user for the file name again and try opening it repeatedly until the file is successfully opened.

While reading the file, **you must store the matrix in a vector of vectors**. Reading the matrix file more than once would clearly be a bad solution.

Your program will also read words, for which the existence will be checked within the provided matrix, from the standard input, which are further explained in the next section.

Format of the Inputs

Please notice that both input file and the words that your program will read will contain only upper-case letters. You can assume that this will be true for all of the test cases that we will be using. A second assumption that you can make is that there will be only alphabetic characters both in the file and in the word inputs, hence you do not need to do any extra check on this particular case as well. (The file will have newline and EOF characters for sure.)

In the data file containing the matrix of characters, the characters of the same row will not be separated by any character, meaning that they will appear consecutively one after another, and the rows will be separated by a newline (\n) character, as seen above in *matrix1.txt*. Also, there will not be any empty lines in the file. It is the safest approach to extract lines from the file with *getline()* function and feed the line strings to an *istringstream* object in order not to have any \n or \r characters at the end of the lines. You can then read characters from the *istringstream* object by forming a while loop on *get()* function defined in istringstream class.

The number of rows and the number of columns of the matrix can be any arbitrary numbers and you cannot make any assumptions on this (i.e., "It *probably* is a square matrix."). Besides, in this file, it cannot be assumed that there are the same number of characters in each row. For the proper execution of the program and for the proper utilization of the vectors, there must be equal number of characters in each row. Your program should check this; give an appropriate error message and halt the execution of the program in case of any such inequalities. Other than this situation, you can assume that there cannot be any other format issues with the file and continue right on.

The words to be searched will be asked to the user through the standard input (cin). This process will continue until the user enters a word of length less than 3. You can assume that there will be only uppercase letters in the input, but no spaces, tabs or anything else.

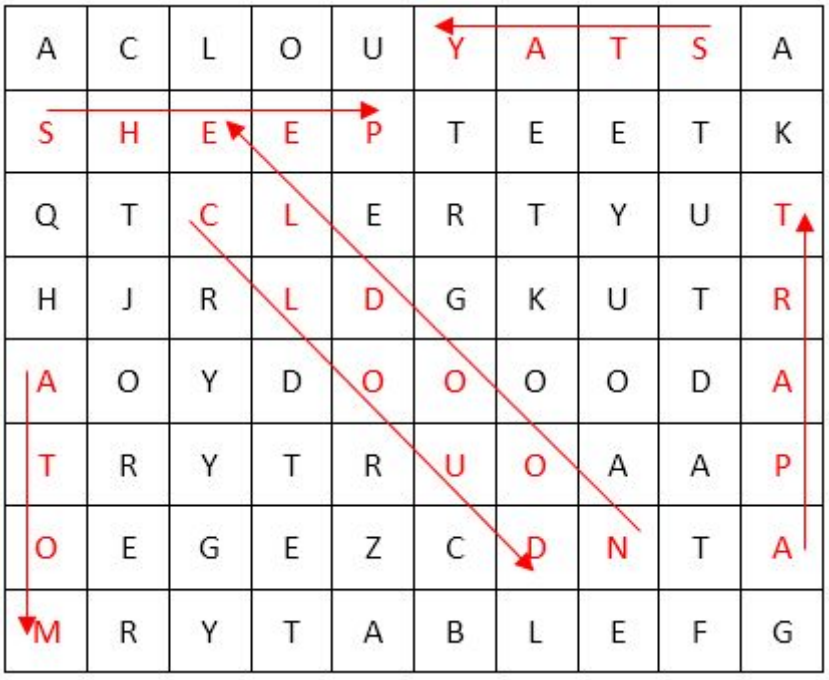
For any further details, you can see the sample runs.

Task of Searching

This section clarifies the algorithm you need to develop.

After obtaining the required information from the file, your program will search for each word that it gets from the standard input, in the 2D vector that your program had constructed from the file. This search can be done vertically, horizontally or diagonally (anti-diagonal[1] direction is not included for the search). It is also possible to go in the backward directions of these three as well.

The following illustration shows possible directions of search in the matrix:



More explicitly, your program can do the search to the north, south, east, west, northwest or southeast of the first letter of the word to be found. You should base your search technique on this rule. 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. You can also 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.

Edge case (or a friendly hint): While searching the matrix, depending on the length of the word that your program considers in the current iteration, you may skip considering the cells that are close to the borders. This is a recommended approach as doing otherwise may result in going out of the range of the matrix and hence crashing your application.

[1] Anti-diagonal is the direction which is perpendicular to the main diagonal of the matrix.

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 example matrix given above, where SHEEP and NOODLE share the letter 'E' successfully. Besides, even one word can contain another (APART and PART can both be accepted in a game).

Last but not least, the user may use one **joker** to replace one of the letters in a word. The user will not explicitly take an action for the use of the joker feature. If the word can be found with one letter difference, your program will assume that the user uses a joker and take action accordingly. **You do not need to check whether that word can also be found without a joker**. Also, there is no limit on the total number of jokers in a game but only one joker can be used for each word. Moreover, there is no limitation on where the joker can be used in the word. The only limitation is: Joker can only be used in the diagonal direction (and of course in the reverse of it). Details on the scoring of this situation is given in the next section.

Calculating the Score

Each word that your program can or cannot find in the matrix effects the score:

- For each word that cannot be found in the matrix, the score will be deducted by 5,
- Words that lay horizontally or vertically increase the score by their number of letters (i.e., the word 'MONEY' would contribute 5 points),
- Words that lay diagonally increase the score by their number of letters multiplied by two (i.e., the word 'BIKE' would contribute 8 points),
- Words that are longer than 5 characters contribute 2 extra points to the overall score.
- In case of the use of a joker, 2 points will be deducted from the score.

Please note that the total score is allowed to go below zero.

Outputs of Your Program

For each word to be searched, your program will display how they lay on the matrix and how many points they contributed to the score. After all the words are searched and their respective points are accumulated, your program will display the final score as the last output and terminate.

You can see the format of the outputs in the sample runs.

Sample Runs

Below, we provide some sample runs of the program that you will develop. The *italic* and **bold** phrases are the standard inputs (cin) taken from the user (i.e., like ***this***). You have to display the required information in the same order and with the same words as here.

Sample Run 1

This program allows you to play a Sunday morning puzzle on your computer.

Enter the matrix file name: ***data3***
Invalid file name!

Enter the matrix file name: ***matrix3***
Invalid file name!

Enter the matrix file name: ***matrix3.tx***
Invalid file name!

Enter the matrix file name: ***matrix3.txt***

Enter a word: ***DOGS***
DOGS is not found in the matrix. 5 points are deducted.

Enter a word: ***ATOM***
ATOM is found in the column level. It contributes 4 points.

Enter a word: ***AS***

Game has ended.
Total score of the words: -1

Sample Run 2

This program allows you to play a Sunday morning puzzle on your computer.

Enter the matrix file name: ***matrix1.txt***

Enter a word: ***NOODLE***
NOODLE is found in the diagonal level. It contributes 14 points.

Enter a word: ***VEGETABLE***
VEGETABLE is not found in the matrix. 5 points are deducted.

Enter a word: ***SHEEP***

SHEEP is found in the row level. It contributes 5 points.

Enter a word: ***STAY***
STAY is found in the row level. It contributes 4 points.

Enter a word: ***ATOM***
ATOM is found in the column level. It contributes 4 points.

Enter a word: ***PRIDE***
PRIDE is not found in the matrix. 5 points are deducted.

Enter a word: ***APART***
APART is found in the column level. It contributes 5 points.

Enter a word: ***HEAP***
HEAP is not found in the matrix. 5 points are deducted.

Enter a word: ***TARGET***
TARGET is not found in the matrix. 5 points are deducted.

Enter a word: ***PART***
PART is found in the column level. It contributes 4 points.

Enter a word: ***CLOUD***
CLOUD is found in the diagonal level. It contributes 10 points.

Enter a word: ***TABLE***
TABLE is found in the row level. It contributes 5 points.

Enter a word: ***GEEK***
GEEK is found in the diagonal level. One letter was replaced by joker. It contributes 6 points.

Enter a word: ***LET***
LET is found in the diagonal level. One letter was replaced by joker. It contributes 4 points.

Enter a word: ***TRYTRUOAAP***
TRYTRUOAAP is found in the row level. It contributes 12 points.

Enter a word: ***A***

Game has ended.
Total score of the words: 53

Sample Run 3

This program allows you to play a Sunday morning puzzle on your computer.

Enter the matrix file name: ***matrix2.txt***

The matrix does not have row equality, terminating...

Some Important Rules

Although some of the information is given below, please also read the homework submission and grading policies from the lecture notes of the first week. In order to get a full credit, your program must be efficient, modular (with the use of functions), well commented and 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 homeworks, we pay attention to these issues. Moreover, in order to observe the real performance of your codes, we are going to run your programs in Release mode and **we may test your programs with very large test cases**. Hence, take into consideration the efficiency of your algorithms other than correctness.

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.

YOU SHOULD USE GRADE CHECKER FOR THIS HOMEWORK!

You should use Grade Checker (http://sky.sabanciuniv.edu:8080/GradeChecker/) to check your expected grade. Just a reminder, you will see a character ¶ which refers to a newline in your expected output.

Make sure you upload the *matrix1.txt*, *matrix2.txt* and *matrix3.txt* files, too.

Grade Checker and the automated grading system use a different compiler than MS Visual Studio does. Hence, you should check the "Common Errors" page to see some extra situations to consider while doing your homework. If you do not consider these situations, you may get a lower score (even zero) even your program works correctly with MS Visual Studio.

Common Errors Page: http://sky.sabanciuniv.edu:8080/GradeChecker/commonerrors.jsp

You will have a demonstration on how to use Grade Checker and how to avoid these common errors during the first week's lab. Please do attend.

Grade Checker can be pretty busy and unresponsive during the last day of the submission. Due to this fact, leaving the homework for the last day generally is not a good idea. You may wait for hours to test your homework or make an untested submission, sorrily..

Grade Checker and Sample Runs together give a good estimate of how correct your implementation is, however we may test your programs with different test cases and **your final grade may conflict with what you have seen on Grade Checker.** We will also **manually** check your code, indentations and so on, hence do not object to your grade based on the Grade Checker 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 Grade Checker or Sample Runs**. The cases that you *do not need* to consider are also given throughout this documentation.

Submit via SUCourse ONLY! **Grade Checker is not considered as a submission**. Paper, e-mail or any other methods are not acceptable, neither.

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 homework on time:

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

What and where to submit (PLEASE READ, IMPORTANT)

You should prepare (or at least test) your program using MS Visual Studio 2012 C++. We will use the standard C++ compiler and libraries of the abovementioned platform while testing your homework.

It'd be a good idea to write your name and lastname 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 Karaoğlan Altop", and if you want to write it as comment; then you must type it as follows:

```
// Duygu Karaoglan Altop
```

Submission guidelines are below. 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. The lack of even one space character in the output will result in your grade being zero, so please test your programs yourself and with the Grade Checker tool explained above.

- Name your cpp file that contains your program as follows:

  "SUCourseUserName_hw1.cpp"

  Your SUCourse user name is actually your SUNet username which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the

file name. For example, if your SU e-mail address is **atam@sabanciuniv.edu**, then the file name must be: **"atam_hw1.cpp"**

- Please make sure that this file is the latest version of your homework program.
- You should upload *matrix1.txt*, *matrix2.txt* and *matrix3.txt* to SUCourse as well.
- Do not zip any of the documents but upload them as separate files only.
- Submit your work **through SUCourse only**! You can use the Grade Checker only to see if your program can produce the correct outputs both in the correct order and in the correct format. It will not be considered as the official submission. You must submit your work to SUCourse.

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

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 homework by yourself. Homeworks are to be done personally and you have to submit your own work. **Cooperation will NOT be counted as an excuse.**

In case of plagiarism, the rules on the Syllabus apply.

Good Luck!
Tolga Atam, Duygu Karaoğlan Altop