Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Spring 2021

Homework 1 – Word Placement in Matrix

Due: 11/03/2021, Wednesday, 21:00

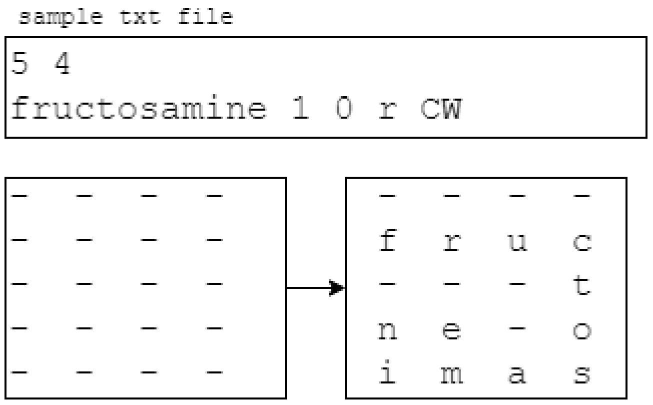
PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; 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 and homework trading will not be tolerated!

Introduction

In this homework, the aim is to put given words into a matrix letter by letter. Your program should read a .txt file and, first, create a matrix (vector of vector) using the dimensions given in the first line of the file. Then, your program will fill the matrix with some words and using their associated attributes. All words in the txt file have 4 attributes: starting point (row, column), direction ('`r`', '`l`', '`u`' or '`d`'), and orientation (clockwise or counter-clockwise). An example matrix created and filled by given sample input is shown in Figure 1. The word "fructosamine" starts at (1,0), goes right and tries clockwise when blocked (details of the rules will be given later).



sample txt file

```
5 4
fructosamine 1 0 r CW
```

```
-   -   -   -
-   -   -   -
-   -   -   -
-   -   -   -
-   -   -   -
```

```
-   -   -   -
f   r   u   c
-   -   -   t
n   e   -   o
i   m   a   s
```

Figure 1: A sample txt and corresponding matrix

Inputs, Outputs and Program Flow

First, user will be prompted for the name of the txt file. If the file with the given name cannot be opened, user will be prompted again until a file can successfully be opened. Your program should read this txt file. First line contains two integers, for the number of rows and columns, respectively. You **must** create a matrix using a **2D vector** (i.e. a `vector` of `vector` of `char`). All of the elements of the matrix are to be initialized to dash character, '`-`'. Once the matrix is created, your program should start reading the rest of the txt file line by line. Each line consists of 5 different elements, first one is the word to be placed into the matrix, second and third are the row and column number of the starting point, fourth one is the direction of placement, and finally the fifth one is orientation. After reading a line, your program tries to fill the matrix with given word according to words' attributes.

Starting point indicates where we must put the first letter of the given word. Starting point consists of two different integers, first one is for the row index and the second one is for the column index (indices start from 0). Direction indicates which direction your program should check first while putting the letters of the word. Direction will be given as single character as '`r`', '`l`', '`u`' or '`d`', which stand for right, left, up and down. This attribute is related with orientation. Orientation indicates which direction should be tried if the cell at the current direction is not suitable (i.e. not empty or out of range). Orientation can be '`CW`' or '`CCW`', which stand for clockwise and counter-clockwise.

Placement algorithm should work as follows. First you put the first letter to starting point, if not occupied. If the starting point is occupied by another word's letter, then placement fails. If you succeed to put the first letter, then you try the cell facing *direction* to put the next letter. If that cell is available, you put the next character there. If not (i.e. occupied or out of range), you turn clockwise or counter-clockwise depending on the *orientation* and try to put the next character there. If still not available, continue to turn and try. If all four directions are unavailable, then the placement of that word fails. If you successfully put a letter, continue with the other letters of the word. For the other letters, you apply the same algorithm, first trying the cell facing the given original *direction*.

For instance, in Figure 2, our matrix is of size 3x5 and our word is '`fructosamine`' with attributes `1`, `2`, '`r`', and '`CCW`'. Therefore, the program puts '`f`' at the point (`1,2`) and continues with the point (`1,3`) because our direction is right. After putting '`fru`', program tries to put '`c`' at the point (`1,5`) but since this cell is invalid for a 3x5 matrix (out of range), program looks for a new direction. This new direction would be "up" since the orientation is counter-clockwise and current direction is right. Thus (`0,4`) is tried and the next character, '`c`' is put there. Actually, for every letter, program first checks right (given direction in the txt file), then up, then left and finally down (due to clockwise orientation). For the letters '`t`', '`o`', '`s`' and '`a`', since right and up cells are not available, they are put to left cells. After putting '`fructosa`', we are at (`0,0`). The only available cell is the one facing down and the next letter '`m`' is put at (`1,0`). Then program tries the one on the right since right is the given input direction and it is available. Thus, the next letter '`i`' is put at (`1,1`). Then, '`n`' is put at (`2,1`) and '`e`' at (`2,2`). In this example, if the orientation was clockwise instead of counterclockwise, program would check right first, then down, then left and finally up.

```
a     s     o     t     c

m     i     f     r     u

-     n     e     -     -
```

Figure 2: 3x5 matrix with given input `fructosamine 1 2 r CCW`

If a particular word cannot be placed, then an appropriate error message is displayed (see sample runs) and all letters of the word placed so far must be undone (see sample runs for example cases).

Your program should take the same actions for every word in txt file. Only condition for early termination is invalid matrix size. More details are given in Input Checks section.

Input Checks

Before the input checks, let us give the assumptions first. You may assume that there will not be any empty lines in the file. Moreover, all integer inputs are assumed to be entered as integer. You do not need to make a check for those.

There are a couple things you need to pay attention when dealing with inputs. First, the number of rows and columns, which are in the first of the file, must be positive integers (i.e. cannot be negative or zero). If any of them is zero or negative, then your program should terminate and give an appropriate message (see sample runs).

The elements for other lines of the txt file should also be checked before processing them. First of all, number of inputs in a line must exactly be 5: word itself, starting position's row number, column number, direction, and orientation. If a line consists of less or more than 5 values, your program should give an appropriate message (see sample runs) on the screen and skip that line of the file.

Also, the values should be checked one by one, other than the word itself (since word is a string there is nothing to check). Once you find a problematic value, give an appropriate message specifying the first problematic input encountered (see sample runs) on the screen and skip that line of the file.

Starting points must be in the range of our matrix. That means, the starting row number must be between 0 and number of rows -1; and the starting column number must be between 0 and number of columns -1. If not, an error is displayed and that line is skipped. Directions must be exactly a single character and they can be only 'r', 'l', 'u' or 'd'. If your program encounters other than these characters as a direction input, it should give an appropriate message and continue with the next line of the txt file. Similar rule applies for orientation as well. Orientations can be only 'CW' or 'CCW'; anything other than these inputs means error and your program should give an error message and continue with the next line.

Sample Runs

Some sample runs are given below, but these are not comprehensive; therefore, you must consider **all possible cases** to get full mark.

You do not need to give the output exactly as in the sample runs, provided that your output is understandable and clear.

Sample Run 1:

```
Please enter the name of the file: test
File name is incorrect, please enter again: test1
File name is incorrect, please enter again: test.txt
File name is incorrect, please enter again: test1.txt

"sample" was put into the matrix with given starting point: 0,0
direction: r   orientation: CW
```

```
s       a       m       p       l
-       -       -       -       e
-       -       -       -       -
-       -       -       -       -
-       -       -       -       -

"puzzle" could not be put into the matrix with given starting point:
0,1   direction: d   orientation: CW
s       a       m       p       l
-       -       -       -       e
-       -       -       -       -
-       -       -       -       -
-       -       -       -       -

"puzzle" was put into the matrix with given starting point: 1,1
direction: l   orientation: CW
s       a       m       p       l
u       p       e       -       e
z       z       l       -       -
-       -       -       -       -
-       -       -       -       -

"apple" was put into the matrix with given starting point: 3,2
direction: d   orientation: CCW
s       a       m       p       l
u       p       e       -       e
z       z       l       -       -
-       -       a       -       e
-       -       p       p       l

"four" was put into the matrix with given starting point: 3,0
direction: u   orientation: CCW
s       a       m       p       l
u       p       e       -       e
z       z       l       -       -
f       r       a       -       e
o       u       p       p       l

"one" could not be put into the matrix with given starting point: 2,3
direction: r   orientation: CW
s       a       m       p       l
u       p       e       -       e
z       z       l       -       -
f       r       a       -       e
o       u       p       p       l

"one" was put into the matrix with given starting point: 1,3
direction: r   orientation: CW
```

```
s       a       m       p       l
u       p       e       o       e
z       z       l       n       e
f       r       a       -       e
o       u       p       p       l

"a" was put into the matrix with given starting point: 3,3
direction: l   orientation: CW
s       a       m       p       l
u       p       e       o       e
z       z       l       n       e
f       r       a       a       e
o       u       p       p       l
```

Sample Run 2:

```
Please enter the name of the file: test2.txt

"fructosamine" was put into the matrix with given starting point: 1,2
direction: r   orientation: CCW
a       s       o       t       c
m       i       f       r       u
-       n       e       -       -

"cryptoxanthin" could not be put into the matrix with given starting
point: 0,0   direction: l   orientation: CW
a       s       o       t       c
m       i       f       r       u
-       n       e       -       -
```

Sample Run 3:

```
Please enter the name of the file: test3.txt

"fructosamine" could not be put into the matrix with given starting
point: 1,0   direction: r   orientation: CCW
-       -       -       -       -
-       -       -       -       -
-       -       -       -       -
-       -       -       -       -
-       -       -       -       -

"fructosamine" was put into the matrix with given starting point: 1,0
direction: r   orientation: CW
-       -       -       -       -
f       r       u       c       t
-       -       -       -       o
-       -       -       -       s
e       n       i       m       a
```

```
"trouble" could not be put into the matrix with given starting point:
0,0   direction: r   orientation: CW
-       -       -       -       -
f       r       u       c       t
-       -       -       -       o
-       -       -       -       s
e       n       i       m       a

"trouble" was put into the matrix with given starting point: 2,2
direction: r   orientation: CW
-       -       -       -       -
f       r       u       c       t
e       -       t       r       o
l       b       u       o       s
e       n       i       m       a
```

Sample Run 4:

```
Please enter the name of the file: test4.txt

"a" was put into the matrix with given starting point: 0,0
direction: l   orientation: CW
a
```

Sample Run 5:

```
Please enter the name of the file: test5.txt

Invalid input for direction! Direction:  qsd

Invalid input for direction! Direction:  q

Starting point is out of range! Point:  3,1

Starting point is out of range! Point:  -1,1

Starting point is out of range! Point:  1,-1

Starting point is out of range! Point:  1,3

Invalid input for orientation! Orientation: CCWW

Invalid input for orientation! Orientation: 123

Invalid line! Number of values is different than 5.

Invalid line! Number of values is different than 5.
Invalid line! Number of values is different than 5.

Starting point is out of range! Point:  -1,-2

"trueInput" was put into the matrix with given starting point: 0,0
direction: l   orientation: CW
t       r       u
n       I       e
p       u       t
```

Sample Run 6:

```
Please enter the name of the file: test6.txt

Invalid number for row and/or column!
```

Sample Run 7:

```
Please enter the name of the file: test7.txt

"therefore" could not be put into the matrix with given starting
point: 0,1   direction: r   orientation: CW
-       -       -       -       -       -       -       -       -

"therefore" was put into the matrix with given starting point: 0,8
direction: r   orientation: CW
e       r       o       f       e       r       e       h       t
```

Sample Run 8:

```
Please enter the name of the file: test8.txt


"therefore" was put into the matrix with given starting point: 8,0
direction: r   orientation: CCW
e
r
o
f
e
r
e
h
t
```

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.

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 last name in the program (as a comment line of course).

Submissions guidelines are below. Some parts of the grading process might be automatic. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.

Name your cpp file that contains your main program using the following convention:

“SUCourseUserName_YourLastname_YourNames_HWnumber.cpp”

Your SUCourse user name is your SUNet user name which is used for checking sabanciuniv e-mails (not the numeric one). Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:

Cago_Ozbugsizkodyazaroglu_Caglayan_hw1.cpp

In some homework assignments, you may need to have more than one .cpp or .h files to submit. In this case, add informative phrases after the hw number. However, do not add any other character or phrase to the file names. Sometimes, you may want to use some user defined libraries (such as strutils of Tapestry); in such cases, you have to provide the necessary .cpp and .h files of them as well. If you use standard C++ libraries, you do not need to provide extra files for them.

These source files are the ones that you are going to submit as your homework. However, even if you have a single file to submit, you have to compress it using ZIP format. To do so, first create a folder that follows the abovementioned naming convention (“SUCourseUserName_YourLastname_YourNames_HWnumber"). Then, copy your source file(s) there. And finally compress this folder using WINZIP or WINRAR programs (or another mechanism). Please use "zip" compression. "rar" or another compression mechanism is NOT allowed. Our homework processing system works only with zip files. Therefore, make sure that the resulting compressed file has a zip extension. Check that your compressed file opens up correctly and it contains all of the files that belong to the latest version of your homework.

You will receive zero if your compressed zip file does not expand or it does not contain the correct files. The naming convention of the zip file is the same. The name of the zip file should be as follows:

SUCourseUserName_YourLastname_YourNames_HWnumber.zip

For example, zubzipler_Zipleroglu_Zubeyir_hw1.zip is a valid name, but

Hw1_hoz_HasanOz.zip, HasanOzHoz.zip

are **NOT** valid names.

Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
Albert Levi, Vedat Peran