BLG 233E Data Structures HW#1

Due Date: 18.10.2018

In this homework, you are asked to place a couple of mathematical operators in a grid and perform specified operations on the placed opeartors. The size of the grid as well as size and position of the mathematical operators must be read from a text file (e.g., "**grid.txt**") and the operations that will be performed should be read from another text file (e.g., "**instructions.txt**") (**IMPORTANT: these file names are just used as examples in the discussion below. They are NOT FIXED and will be supplied as command line arguments. Don't use static file names in your program**). The details are as follows:

Initializing The Grid (grid.txt)

(1) The grid consists of **N rows** and **M columns** which are specified in the first line of the **grid.txt** file, as examplified in Figure 1. Note that **row/column numbers start from 1 and not from 0**.

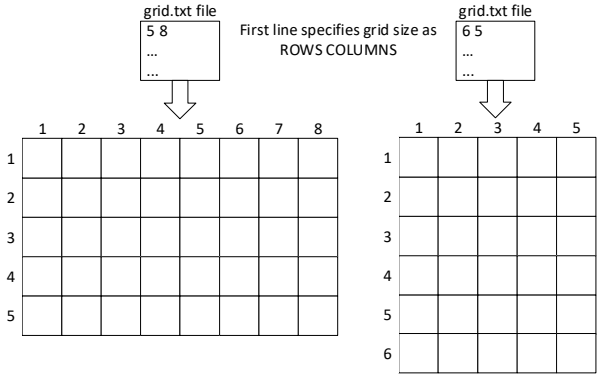

Figure 1: Two grids with sizes specified in the first line of grid.txt file

(2) The operators are added to the grid by specifying its **type** (+, -, x, /), its **center point** (row and column in the grid), and its **size** (a number in the interval [1..9]). Figure 2 shows placement of each arithmetic operator with size 1. Please note that, each operator is placed on the grid so that the complete shape reflects the type of the operator. For example,

a. Leftmost **grid.txt** file places a **+** (**type=+**) character on the center location **[2,2]** and adds one (**size=1**) **+** character to right, left, upper, and bottom of the center.

b. Rightmost **grid.txt** file places a **/** (**type=/**) character on the center location **[4,2]** and adds one (**size=1**) **/** character to upper-right and bottom-left of the center.

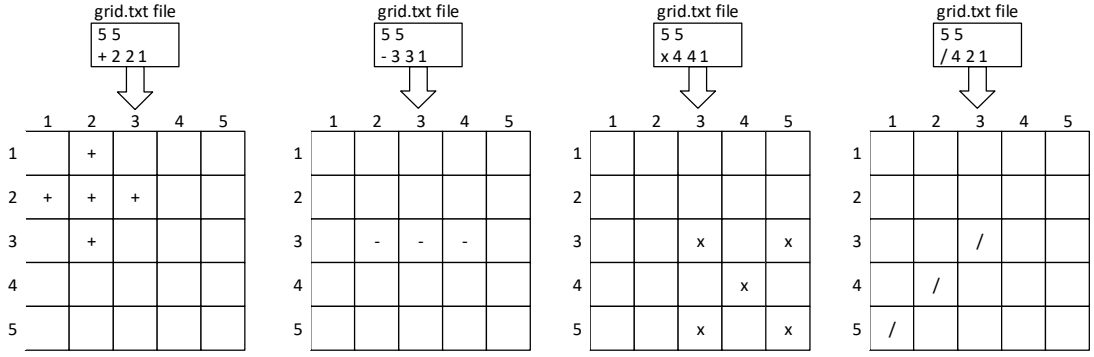

Figure 2: Placement of unit operators (i.e., operators with size 1) on 5x5 grids, centered on different notations.

Similarly, Figure 3 shows **grid.txt** files in which operator sizes are 2 and they are placed on the center of 5x5 grids.

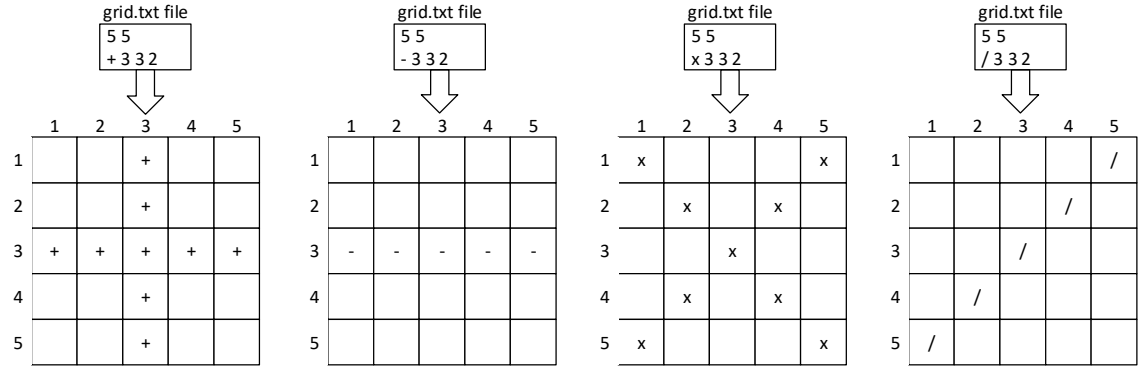

Figure 3: Placement of operators with size = 2 on 5x5 grids, operators are centered on the grid (i.e., center location = [3,3]).

(3) When you place an operator, you have to check whether the operator fits into the grid or not. There is a **BORDER ERROR** when **at least one cell** of the operator is placed outside of the grid. Figure 4 shows some example **BORDER ERROR**s (i.e., operator overflows outside of the grid).

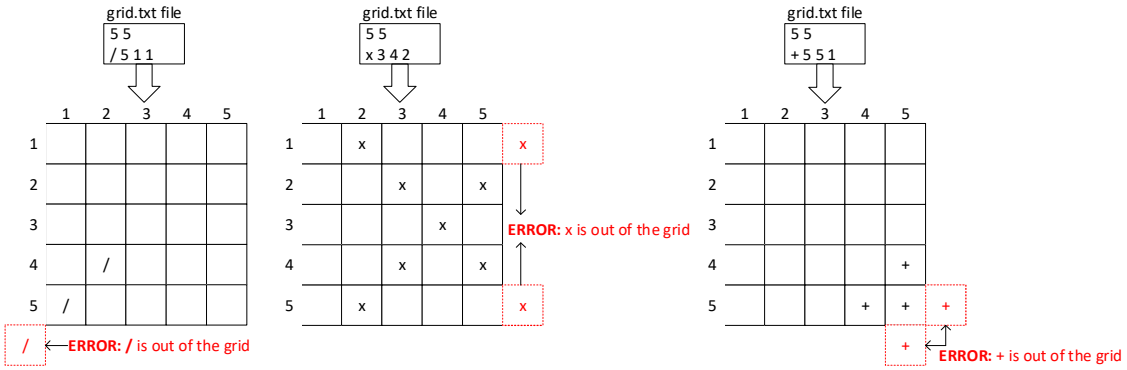

Figure 4: Some BORDER ERROR examples for invalid operator placements.

(4) When you place an operator, you have to check whether the cells were occupied by another previously placed operator or not. When **at least one cell** is occupied by a previously placed operator, there is a **CONFLICT ERROR**. Figure 5 examplifies confict errors.

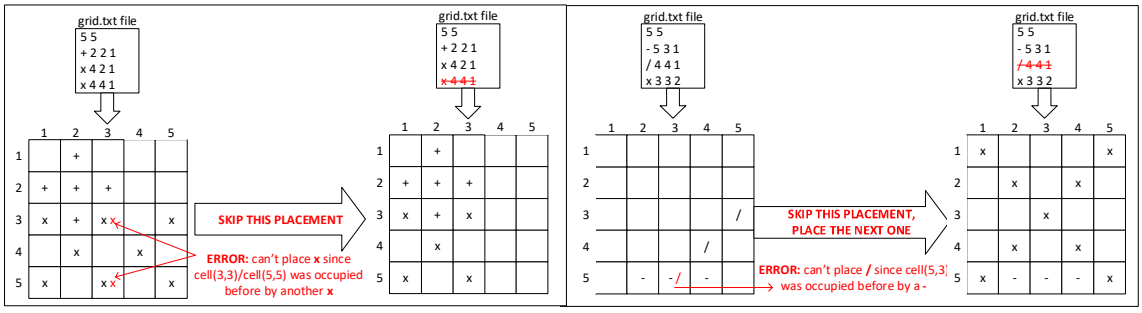

Figure 5: Two examples of CONFLICT ERRORs

(5) When an error (BORDER **and/or** CONFLICT error) occured in placing the operators, you skip the placement and continue with the next placement (if any) specified in the **grid.txt** file (e.g., see example on the right side of Figure 5).

Instructions on The Placed Operators (instructions.txt)

There are four instructions defined for the operators placed on the grid. They are specified in the **instructions.txt** file as described below:

(1) **Move Right (MVR)** instruction moves the center of the operator by a specified amount to the right (i.e., it increments column part of the center location). Its format is as follows:

MVR row column move_by

where **row** and **column** specifies a cell on the grid on which **a part of the operator** resides. The cell (**row**, **column**) **MUST NOT** be the center of the operator. You must first find the center, and move the operator by the amount **move_by (e.g., see** Figure 6**)** .

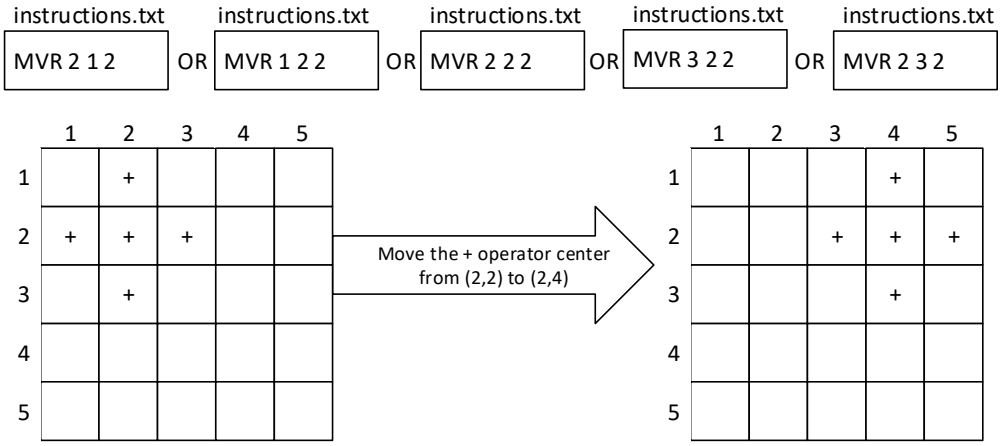

Figure 6: MVR Instruction Example

(2) **Move Left (MVL)** instruction moves the center of the operator by a specified amount to the left (i.e., it decrements column part of the center location). Its format is as follows:

MVL row column move_by

where **row** and **column** specifies a cell on the grid on which **a part of the operator** resides. The cell (**row**, **column**) **MUST NOT** be the center of the operator. You must first find the center, and move the operator by the amount **move_by (e.g., see** Figure 7**)** .

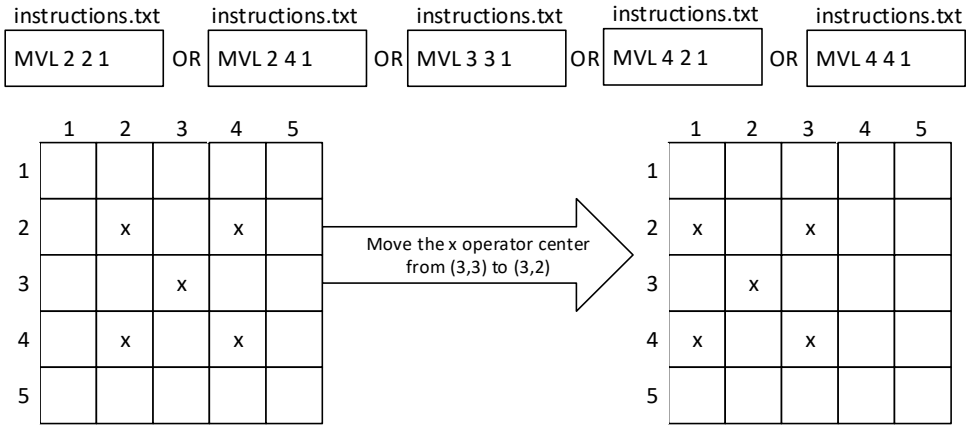

Figure 7: MVL Instruction Example

(3) Similar to **MVR** and **MVL** instructions, **Move Up (MVU)** and **Move Down (MVD)** instructions move the center of the operator by a specified amount to up (i.e., it decrements row part of the center location) and down (i.e., it increments row part of the center location), respectively. Their formats are also similar with **MVR** and **MVL** instructions:

MVU row column move_by

MVD row column move_by

where **row** and **column** specifies a cell on the grid on which **a part of the operator** resides. The cell (**row**, **column**) **MUST NOT** be the center of the operator. You must first find the center, and move the operator up/down by the amount **move_by**.

(4) When an error (BORDER **and/or** CONFLICT) occured in executing the instructions, you skip the instruction and continue with the next instruction (if any) specified in the **instructions.txt** file (e.g., see Figure 8).

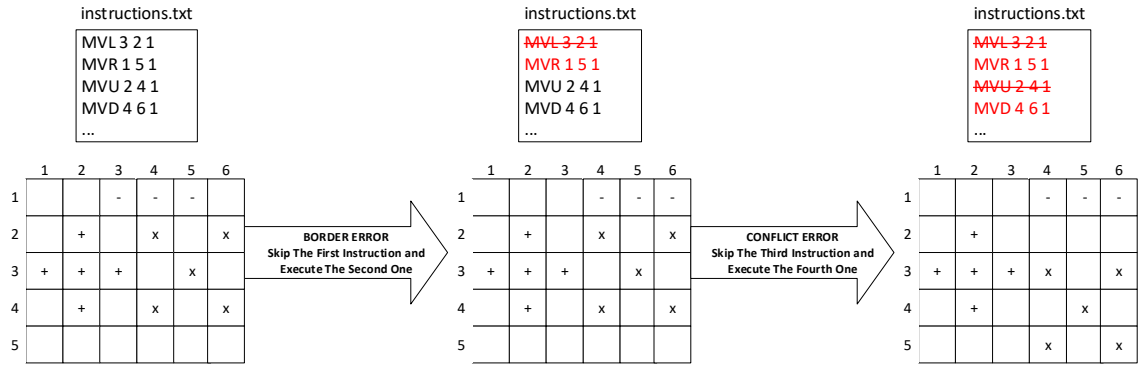

Figure 8: Example on Skipping ERRORs

HW#1 Program Flow

(1) Your program must take grid filename and instructions filename as command line arguments. If your executable is named as **assignment1**, then a sample run may be:

./assignment1 grid_file_name.txt instructions_file_name.txt

where **grid_file_name.txt** is the file that contains information about size of the grid and operators on the grid (we used **grid.txt** file name in the above discussion) and **instructions_file_name.txt** is the file that conatins instructions (we used **instructions.txt** file name in the above discussion).

(2) Open **grid_file_name.txt** file and read two numbers **ROWS** and **COLUMNS** from its first line which determines number of rows and columns in the grid, respectively (see Figure 1). Create a grid dynamically based on this grid size. Then, print out the following message:

A grid is created: **ROWS COLUMNS**

where **ROWS** and **COLUMNS** are the numbers read from the file

(3) Till end of the **grid_file_name.txt** file, read **type, row, column**, and **size** of the operators that will be placed on the grid. After reading an operator:

a) Print out following message if the operator successfully placed (see Figure 2 and Figure 3) on the grid:

SUCCESS: Operator **type** with size **size** is placed on (**row**,**column**).

where **type, row, column**, and **size** are values read from the file. Then, read the next operator (if any) from the **grid_file_name.txt** file.

b) If there is a **BORDER ERROR** (see Figure 4), then omit the placement, print out the following message, and read the next operator (if any) from the **grid_file_name.txt** file:

BORDER ERROR: Operator **type** with size **size** can not be placed on (**row**,**column**).

where **type, row, column**, and **size** are values read from the file.

c) If there is a **CONFLICT ERROR** (see Figure 5), then omit the placement, print out the following message, and read the next operator (if any) from the **grid_file_name.txt** file:

CONFLICT ERROR: Operator **type** with size **size** can not be placed on (**row**,**column**).

where **type, row, column**, and **size** are values read from the file.

d) If there are **both BORDER and CONFLICT ERRORS**, then omit the placement, print out the following messages **in order**, and read the next operator (if any) from the **grid_file_name.txt** file:

BORDER ERROR: Operator **type** with size **size** can not be placed on (**row**,**column**).
CONFLICT ERROR: Operator **type** with size **size** can not be placed on (**row**,**column**).

where **type, row, column**, and **size** are values read from the file.

(4) After placing the operators, you can close **grid_file_name.txt** file and open **instructions_file_name.txt** file to perform stated instructions on the placed operators. Till end of the **instructions_file_name.txt** file, read **instruction, row, column**, and **move_by** starting from the first line.

(5) Change the center location of the operator (**row,column**) by **move_by** based on the instruction (see Figure 6 and Figure 7):

a. Print out following message if the operator is successfully moved:

SUCCESS: **type** moved from (**row_c**,**column_c**) to (**row_n**,**column_n**).

where **type** is the type of the operator moved, (**row_c**,**column_c)** is position of the center before the move, and (**row_n**,**column_n)** is position of the center after the move. Then, read the next instruction (if any) from the **instructions_file_name.txt** file.

b. If there is a **BORDER ERROR** (see Figure 4), then omit the move instruction, print out the following message, and read the next instruction (if any) from the **instructions_file_name.txt** file:

BORDER ERROR: **type** can not be moved from (**row_c**,**column_c**) to (**row_n**,**column_n**).

where **type** is the type of the operator that can't be moved, (**row_c**,**column_c)** is position of the center before the move attempt, and (**row_n**,**column_n)** is position of the center for the target move.

c. If there is a **CONFLICT ERROR** (see Figure 5), then omit the move instruction, print out the following message, and read the next instruction (if any) from the **instructions_file_name.txt** file:

CONFLICT ERROR: **type** can not be moved from (**row_c**,**column_c**) to (**row_n**,**column_n**).

where **type** is the type of the operator that can't be moved, (**row_c**,**column_c)** is position of the center before the move attempt, and (**row_n**,**column_n)** is position of the center for the target move.

d. If there is both **BORDER and CONFLICT ERRORs,** then omit the move instruction, , print out the following messages **in order**, and read the next instruction (if any) from the **instructions_file_name.txt** file:

BORDER ERROR: **type** can not be moved from (**row_c**,**column_c**) to (**row_n**,**column_n**).
CONFLICT ERROR: **type** can not be moved from (**row_c**,**column_c**) to (**row_n**,**column_n**).

where **type** is the type of the operator that can't be moved, (**row_c**,**column_c)** is position of the center before the move attempt, and (**row_n**,**column_n)** is position of the center for the target move.

(6) Close the **instructions_file_name.txt** file and exit the program.

Submission and Rules

1. Your source code file must have the name **assignment1.cpp**.
2. Your program will be compiled using the following command on a Linux system. If it cannot be compiled and linked using this command, it will not be graded (failed submission).

   g++ -Wall -Werror assignment1.c -o assignment1

3. Your program will be checked using **Calico** (https://bitbucket.org/uyar/calico) automatic checker. Therefore, make sure you **print the messages exactly as given in the homework definition.**
4. Make sure your coding style is proper and consistent. Use the **clang-format tool** if necessary. Don't use any variable names in a language other than English.
5. Add comments to make your source code easy to understand.
6. This is an **individual** assignment. Collaboration in any form is NOT allowed. **No working together**, No sharing code in any form including showing code to your classmates to give them ideas.
7. All the code you submit must be your own. Don't copy/paste any piece of code from any resource including anything you've found on the Internet.
8. The assignments will be checked for plagiarism using both automated tools and manual inspection. Any assignment involving plagiarism and/or infringement of intellectual property will not be graded and is subject to further disciplinary actions.