[Practice Problem for Hands-on Exam #1 – Arrays and Strings]

String Manipulations [contributed by F. R. Salvador] *Note: portions of this problem were given in an actual past exam*.

Consider a String Matrix (i.e., a 2D array of strings) that has a maximum size of 10 rows by 10 columns. Note, though, that the actual rows and columns used can be less than the maximum. For example, the matrix below shows that only rows 0 to 3 and columns 0 to 2 are populated with strings (shown in blue color). The rest of the matrix elements are uninitialized (or garbage values) visually represented by the question marks. The colons represent the other rows and columns that are not shown explicitly for reasons of brevity. Each string can have a maximum length of 15 characters only EXCLUDING the null byte.

	col 0	col 1	col 2	:	:	:	col 9
row 0	DOG	NEKO	CAT	?	?	?	?
row 1	HELLO	GOODBYE	WELCOME	?	?	?	?
row 2	GREEN	RED	BLUE	?	?	?	?
row 3	CODE	PROGRAMMER	SYNTAX	?	?	?	?
:	?	?	?	?	?	?	?
:	?	?	?	?	?	?	?
:	?	?	?	?	?	?	?
row 9	?	?	?	?	?	?	?

YOUR TASKS – edit the accompanying LASTNAME-PRACTICE.c skeleton file and complete the definition for the following functions:

1. **void Populate(Matrix M, int *ptr_nrows, int *ptr_ncols)**
   Populate *(this means initialize)* a String Matrix with words (not necessarily just English words, it may be a word from another language like "NEKO" above). Assume for simplicity that the strings will be made up only of upper case letters. Use **scanf()** to input the following:
   a. number of rows to be populated -- store the value in a variable named **nRows** (in the example above, nRows = 4)
   b. number of columns to be populated – store the value in a variable named **nCols** (in the example above, nCols = 3)
   c. the words (note: there are nRows times nCols words, in the example above there are 4x3 = 12 words)

2. **void Search(Matrix M, int n_rows, int n_cols, String key, int *ptr_row_index, int *ptr_col_index)**
   Search for a key in String Matrix using linear search algorithm. Access the matrix element in row major order. Assume for simplicity that the search key is also made up of upper case letters only. The result of the search function should set the parameters related to row index and column index to the corresponding indices where the search key was found in the matrix. If the key is not found, these numbers should both be set to -1.

3. **void Transpose(Matrix T, Matrix M, int n_rows, int n_cols)**
   Produce a transpose matrix T corresponding to the original String Matrix M. For example, the transpose of the example matrix above is shown as follows. Note that the row elements of the original matrix M will become the column elements of the transpose matrix T, and vice-versa. Note also that the corresponding transpose matrix would have n_rows = 3 and n_cols = 4.

	col 0	col 1	col 2	col 3	:	:	col 9
row 0	DOG	HELLO	GREEN	CODE	?	?	?
row 1	NEKO	GOODBYE	RED	PROGRAMMER	?	?	?
row 2	CAT	WELCOME	BLUE	SYNTAX	?	?	?
row 3	?	?	?	?	?	?	?
:	?	?	?	?	?	?	?
:	?	?	?	?	?	?	?
:	?	?	?	?	?	?	?
row 9	?	?	?	?	?	?	?

4. **void Filter(char *filter, Matrix T, Matrix M, int n_rows, int n_cols)**
   Given matrix M, produce a matrix T such that T contains the same strings as in M with the strings marked accordingly based on a given filter string. More specifically, if a word has all the characters in a filter string, then it will be enclosed with a pair of "+" (plus sign). Otherwise, it will be enclosed with a pair of "-" (minus sign). For example, if M is the originally given matrix, and the filter string is "EO", then the resulting matrix T should contain:

	col 0	col 1	col 2	:	:	:	col 9
row 0	-DOG-	+NEKO+	-CAT-	?	?	?	?
row 1	+HELLO+	+GOODBYE+	+WELCOME+	?	?	?	?
row 2	-GREEN-	-RED-	-BLUE-	?	?	?	?
row 3	+CODE+	+PROGRAMMER+	-SYNTAX-	?	?	?	?
:	?	?	?	?	?	?	?
:	?	?	?	?	?	?	?
:	?	?	?	?	?	?	?
row 9	?	?	?	?	?	?	?

5. void RowSort (Matrix T, Matrix M, int n_rows, int n_cols)

Given matrix M, produce a matrix T such that T contains the same strings as in M arranged in sorted alphabetical order **per row**. For example, if M is the originally given matrix, then matrix T should contain:

	col 0	col 1	col 2	:	:	:	col 9
row 0	CAT	DOG	NEKO	?	?	?	?
row 1	GOODBYE	HELLO	WELCOME	?	?	?	?
row 2	BLUE	GREEN	RED	?	?	?	?
row 3	CODE	PROGRAMMER	SYNTAX	?	?	?	?
:	?	?	?	?	?	?	?
:	?	?	?	?	?	?	?
:	?	?	?	?	?	?	?
row 9	?	?	?	?	?	?	?

You are given the following accompanying files:

- **practice.h** is a header file that contains the macro definitions and function prototypes of the functions that you need to define
- **LASTNAME-PRACTICE.c** is the skeleton file where you will encode your function definitions
- **test_functions.c** contains the functions that will call/test the functions that you defined
- **main.c** contains the main() function that will call the test functions
- **INPUT.TXT** is the example file containing the input data. This will be used in input redirection.
- **EXPECTED.txt** is a text file containing the expected output to be produced by a correct solution when the main() function is called.

DELIVERABLES: Upload two files before the Canvas submission deadline

1. `LASTNAME-PRACTICE.c`
2. `LASTNAME-ACTUAL.txt` - this is the output redirected text file. Read **`main.c`** for details on how to test your solution and how to produce this file.

Make sure to rename these files using your own last name. For example, if your last name is SANTOS, you must upload your files as SANTOS-PRACTICE.c and SANTOS-ACTUAL.c.

TESTING & SCORING:

- Your program will be tested using a different input text file (containing different object names and coordinates), and a different main() function.
- Each correct function definition will be given **10 points** each.
- **A function that has a syntax/compilation error will result in a score of 0.**
- **A function that does not produce any of the required output (for example, due to runtime error) will result in a score of 0.**

DELIVERABLES:

Submit/upload two files via Canvas before the indicated deadline:

1. your PRACTICE-LASTNAME.c source file
2. your LASTNAME-RESULT.txt output text file

Don't forget to change the filename to your own last name.

--- THE END ---