PROBLEM: TRANSFORM (contributed by F. R. Salvador)

There are several computer games[1] that are based on the idea of merging two objects/creatures to transform it to a higher level object/creature. For this hands-on exam, we will develop a similar simple game which we will simply call "TRANSFORM". The game uses the following data structures:

- **MasterList[]** - a 1D array of strings where each string is at most 8 characters excluding the null byte. **MasterList[]** contains a master list of ALL the creatures[2] used in the game. An example of **MasterList[]** contents is shown in Figure 1. Note that each creature has a **level** which is equal to the index where it is stored in the **MasterList[]**. For example, referring to Figure 1, the level of **-_-** is 0, while the level of **(^_^)** is 11. This means that the higher the index, the higher is the level of the creature.

- **Matrix[ROWSIZE][COLSIZE]** - is a 2D array of strings where each string is also at most 8 characters excluding the null byte. A matrix cell is either *unoccupied* or *occupied* by a creature. An unoccupied cell contains an empty string (or null string), i.e., **""** whose string length is 0. An occupied cell contains the string representing the creature. Assume that all creatures in the **Matrix[][]** are in the **MasterList[]**. Note also that all cells, i.e., ROWSIZExCOLSIZE number of cells, will be used. An example matrix configuration is shown in Figure 2.

-_-	(-_-)	(~_~)	(=_=)	(>_<)	(+_+)	(#_#)	('_')	("_")	(&_&)	(*_*)	(^_^)
0	1	2	3	4	5	6	7	8	9	10	11

Fig. 1: Example **MasterList[]** contents. Strings are in blue (double quotes are not shown), the numbers are indices.

	0	1	2	3	4
0	-_-	(^_^)	(~_~)		
1	("_")		-_-		
2		(#_#)		(#_#)	(-_-)
3	(>_<)	(*_*)	(^_^)		
4	-_-			(#_#)	

Fig. 2: Example **Matrix[][]** contents. The strings are in blue (double quotes are not shown), the numbers are indices. Note that a blank cell, such as the cell in row 0, column 3 actually contains a null string.

Your task is to edit the accompanying skeleton file **`LASTNAME-TRANSFORM.c`** and implement the 5 functions described below:

1. **int Search(String creature, String MasterList[], int n)**

Implement a linear search algorithm to search the **MasterList[]** if there is a matching element corresponding to the **creature** parameter. If there is a match, return the corresponding array index where it was found in the **MasterList[]**. Otherwise, return a value of -1 to indicate that it was not found.

Examples with reference to the **MasterList[]** contents in Figure 1:

- Example 1: **Search("(+_+)", MasterList, 12)** will return 5 which is the array index where the search key was found.
- Example 2: **Search("($_$)", MasterList, 12)** will return -1 since the search key was not found.

2. **int Spawn(String creature, int row, int col, String Matrix[][COLSIZE])**

If the cell corresponding to the matrix element with the specified **row**, and **col** indices is unoccupied, then *spawn* a creature in that location. *Spawn* means to store the value of the **creature** parameter in a cell ONLY if it is unoccupied. If the cell is already occupied, the spawn operation cannot proceed (i.e., it is an unsuccessful spawn). The function should return a 1 to indicate a successful spawn, otherwise it should return a 0.

Examples with reference to the **Matrix[][]** contents in Figure 2:

- Example 1 (successful spawn): **Spawn("(+_+)", 4, 4, Matrix);** will store **(+_+)** in row 4, column 4 cell, i.e., the last cell in Figure 2, and return 1 because it is a successful spawn.
- Example 2 (unsuccessful spawn): **Spawn("(#_#)", 0, 1, Matrix);** will not be a successful spawn because row 0, column 1 cell is already occupied by **(^_^)**. The function will return a 0.

[1] See for example https://poki.com/en/g/merge-tycoon

[2] Actually the strings were encoded similar to horizontal emoticons to convey the face of the creatures.

3. **int GetMatrixScore(String Matrix[][COLSIZE], String MasterList[], int n)**
Compute the Matrix Score as the sum of all the *cell scores*.

- The *cell score* of an unoccupied cell is 0.
  - For example, the cell at row 0, col 3 contributes a score of 0.

- The *cell score* of an occupied cell is equal to *2^level* where ^ means "raised to" and level is the creature's level. Recall that the level of a creature is the same as the index where the creature is found in the **MasterList[]** array.
  - For example, the cell at row 0, col 0 contributes a score of *2^0 = 1* since the creature **-_-** is at index 0 in the **MasterList[]**.
  - Another example, the cell at row 2, col 1 contributes a score of *2^6 = 64* since the creature **(#_#)** is at index 6 in the **MasterList[]**.

The **GetMatrix()** function must call and use the return value of the **Search()** function properly.

With to reference to the example **MasterList[]** and **Matrix[][]** contents in Figures 1 and 2 above, the Matrix Score is computed as:

```
Matrix Score =   2^0  + 2^11 + 2^2 + 0 + 0      // sum of cell scores at row 0
               + 2^8  +  0     + 2^0 + 0 + 0    // sum of cell scores at row 1
               +   0  + 2^6   + 0 + 2^6 + 2^1   // sum of cell scores at row 2
               + 2^4  + 2^10 + 2^11 + 0 + 0     // sum of cell scores at row 3
               + 2^0  + 0 + 0 + 2^6 + 0         // sum of cell scores at row 4

               = 5593
```

4. **void Frequency(int FC[], String Matrix[][COLSIZE], String MasterList[], int n)**
Determine the Frequency Count (FC) of the creatures that are currently in the **Matrix[][]**. Set the value of **FC[i]** such that it contains the number of times that the creature in **MasterList[i]** appears in the **Matrix[][]**. For example, **MasterList[0]** which is the creature **-_-** appears 3 times in the **Matrix[][]**; thus **FC[0]** is 3. **MasterList[5]** which is the creature **(+_+)** does not appear in the **Matrix[][]**; thus **FC[5]** is 0.

Using the **Matrix[][]** and **Masterlist[]** in Figures 1 and 2, the corresponding **FC[]** contents will be

3	1	1	0	1	0	3	0	1	0	1	2
0	1	2	3	4	5	6	7	8	9	10	11

5. int Merge(int row1, int col1, int row2, int col2, String Matrix[][COLSIZE], String MasterList[], int n)
If the creatures occupying the two cells with row and column indices indicated by the 1st four parameters are the **same**, then *merge* them. *Merge* in this case means to combine the two creatures into a single level-up creature. The level-up creature is the next level creature as listed in the **MasterList[]**. The level-up creature will replace the cell content at **row1**, **col1** indices.. The cell in **row2**, **col2** indices should then become unoccupied, i.e., it should be reset to a null string. The **Merge()** function must call and use the return value of the **Search()** function properly.

Note that there are three cases when a **Merge()** operation will be unsuccessful.
Case 1: when one or both cells actually contain a null string, i.e, the cell is unoccupied.
Case 2: when the two creatures are **not** the same.
Case 3: when the two creatures are identical but they are already the highest level creature. Recall that the highest level creature is the last creature, i.e., **MasterList[n - 1]**.

Merge() should return a 1 if the creatures were merged successfully. For an unsuccessful merge, return a 0 for Cases 1 and 2, and return a 2 for Case 3.

Examples with reference to the **MasterList[]** and **Matrix[][]** contents in Figures 1 and 2:

- Example 1 (Successful merge): **Merge(0, 0, 1, 2, Matrix, MasterList, 12) ;** since the creature in row 0, col 0 is **-_-** and the creature is row 1, col 2 is also **-_-** are the same, they will be merged. The creature at row 0, col 0 will be replaced, i.e., upgraded to the next level creature which is **(-_-)**. The cell at row 1, col 2 will be replaced with a null string. Finally, the function will return a 1.
- Example 2 (Case 1: one of the cells is unoccupied): **Merge(0, 1, 4, 2, Matrix, MasterList, 12);** since row 4, col 2 contains a null string, the function will return a 0.
- Example 3 (Case 2: creatures are not the same): **Merge(0, 1, 4, 3, Matrix, MasterList, 12);** since the creature in row 0, col 1 is not the same with the creature in row 4, col 3, the function will return a 0.
- Example 4 (Case 3:highest level creatures): **Merge(0, 1, 3, 2, Matrix, MasterList, 12);** since the creatures in row 0, col 1 and row 3, col 2 are already the highest level, the function will return a 2.

You are given the following files for this problem:

- **`LASTNAME-TRANSFORM.c`** -- skeleton file which contains some initial code that you'll need to complete
- **`test_functions.c`** -- contains the functions that will call/test the functions that you defined; make sure to read this file
- **`main.c`** -- contains the **`main()`** function for testing purposes; make sure to read this file
- **`transform.h`** -- header file containing macro definitions, typedef and function prototypes; make sure to read this file
- **`INPUT.txt`** -- contains example input data; test your solution exhaustively by creating another text file with your own data
- **`EXPECTED.txt`** -- this contains the expected results when **`main.exe`** is executed using data stored in **`INPUT.txt`** file

DELIVERABLES: Submit/upload two files before the Canvas submission deadline

1. **`LASTNAME-TRANSFORM.c`** -- this file contains your solution (i.e., C source code)
2. **`LASTNAME-ACTUAL.txt`** -- this is the actual output redirected text file produced by your own exe 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-TRANSFORM.c and SANTOS-ACTUAL.c.

TESTING & SCORING:

- Your program will be tested using a different input text file (containing different creature names), and a different main() function.
- Each correct function definition will be given **10 points** each. Thus, the maximum total score will be 50/50.
- **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.**

--- THE END ---