Computing Languages f. Math. (MA591K)

Project 1
Due: Friday Sep. 11, at midnight

Introduction

The purpose of the first project is to learn several language features of the American National Standards Institute (ANSI) C used for reading in information from *stdin* (standard input file, usually connected to the keyboard), manipulating pointers and accessing array elements. The project essentially deals with implementing a 2-dimensional character array that stores strings row-wise, and then doing some operations on that array.

$$\begin{bmatrix} A[0][0] & \cdots & A[0][\mathrm{MAXCOL}-1] \\ \vdots & \ddots & \vdots \\ A[\mathrm{MAXROW}-1][0] & \cdots & A[\mathrm{MAXROW}-1][\mathrm{MAXCOL}-1] \end{bmatrix}$$

Figure 1: The layout of the 2-D array A[MAXROW][MAXCOL]

Figure 1 shows what a 2-dimensional array A[MAXROW][MAXCOL] looks like. We intend to hold n character strings $\sigma_1, \ldots, \sigma_n$ in an array like that, such that the first character of σ_1, is in A[0][0], and so on. Thus we want character j of string σ_i to be in A[i-1][j-1].

Note that a static layout of the array forces us to include not more than MAXROW strings, and each of them can not be of more than MAXCOL-1 length (since ANSI C strings are NULL-terminated).

To begin with we want to be able to perform the following operations on an array of this type:

- Reading strings into the array.
- Writing out the array.
- Transposing the array.

After we have done that, we would like to modify the behavior of the system a bit, and add the following two capabilities:

- Copying the strings from the array into an one-dimensional vector of pointers to strings, using dynamic allocation.
- Writing out the strings in the one-dimensional vector.

Functional Descriptions, part 1

What follows is a description of the major functions your ANSI C program should contain along with their necessary prototypes, usage information, pre-conditions and post-conditions.

Reading the array

The first function you have to implement in this project is a function that reads strings into a 2-D array of characters. The function should read a few strings from the *stdin*, each string terminated by a carrige return and the entire list of strings terminated by an End-of-File character (Ctrl-D).

Proto: `void read_array( char A[MAXROW][MAXCOL], int* nptr );`

Usage: `read_array( A, &no_str );`

Pre: `A` is a MAXROW by MAXCOL array of char, `no_str` is an int.

Post: There have been some strings read from the *stdin* and inserted into `A` row-wise, such that the second character of the first string is in `A[0][1]`. The int `no_str` contains the number of strings that have been inserted into `A`. There have not been more than MAXROW strings read, and each of them is not longer than MAXCOL-1. If there there has been an attempt made to read more than MAXROW strings, or a string that's longer than MAXCOL-1, a proper error message has been written on *stderr*.

Hint: You can use the function `getchar` to get characters from *stdin*, and the function `fprintf` to write to *stderr*.

Writing the array

The second function you have to implement in this project is a function that writes out the strings in a 2-D array of characters. The function should write out n specified strings in the first n rows of the array to *stdout*, each string beginning on a new line.

Proto: `void write_array( char A[MAXROW][MAXCOL], int n );`

Usage: `write_array( A, no_str );`

Pre: `A` is a MAXROW by MAXCOL array of char, `no_str` is an int specifying the number of strings in the rows of `A`.

Post: The strings in the `no_str` first rows of `A` have been written out to *stdout*.

Hint: You can use the function `putchar` to write characters to *stdout*, and the functions `puts` and `printf` to write whole strings at a time to *stdout*.

Once you have written `read_array` and `write_array`, debug the 2 functions using a trivial main program to make sure they work. Only then, proceed to the next function.

Transposing the array

The third function you have to implement in this project is a function that transposes the 2-D array of characters. The formula for transposing an abstract two-dimensional array is:

$$A_{ij} := A_{ji} \qquad \text{or} \qquad A[i][j] := A[j][i]$$

This applies for *all* the elements of A so that you have to be careful not to overwrite something before you have used it when transposing the array.

Example: The transpose of the array $\begin{bmatrix} \texttt{Hello} \\ \texttt{World!} \\ \\ \\ \\ \\ \end{bmatrix}$ is the array $\begin{bmatrix} \texttt{HW} \\ \texttt{eo} \\ \texttt{lr} \\ \texttt{ll} \\ \texttt{od} \\ \texttt{ !} \end{bmatrix}$.

Proto: `void transpose_array( char A[MAXROW][MAXCOL], int n, int* m );`

Usage: `transpose_array( A, old_no_str, &new_no_str );`

Pre: `A` is a MAXROW by MAXCOL array of char, `old_no_str` is an int specifying the number of strings in the rows of A, `new_no_str` is an int.

Post: The array `A` has been transposed, new_no_str contains the number of strings in `A`. An error message is produced if the transposed data does not fit.

Hint: Pad the ends of the strings starting at the null character with spaces, so they are all the same length, before transposing the array. Thus you will maintain the structure that the array contains strings that start in the first column after the transpose. To understand this, think about the space in front of the exclamation mark in the example. While padding with blanks, determine the length, call it ℓ, of the longest string. You can only transpose if $\ell \leq$ `MAXROW` and $n \leq$ `MAXCOL` $- 1$ (remember that the transposed strings must be terminated by a null character). If this condition is not met, print an error message on `stderr` and return. Otherwise, swap the character entries `A[`i`][`j`]` and `A[`j`][`i`]` for $0 \leq i < \ell$ and $0 \leq j <$ `n`. Finally, place null characters in the column of column index `n` to terminate the transposed strings.

Please note that you may **not** create another array to help you while transposing the array.

Functional Descriptions, part 2

Having thus completed the first half of the project, we now move on to the second half. Please do not move on to this second half before you have gotten the first part to work correctly.

Creating a string-pointer vector

The fourth function you have to implement in this project is a function that creates a one-dimensional array of pointers to copies of the strings in the 2-D array of characters. To create the copies you must use dynamic allocation of memory and the string functions of ANSI C.

Proto: `void get_vector( char A[MAXROW][MAXCOL], char* v[MAXROW], int n );`

Usage: `get_vector( A, v, no_str );`

Pre: `A` is a MAXROW by MAXCOL array of char, `v` is a MAXROW vector array of char pointers, `no_str` is an int specifying the number of strings in the rows of A.

Post: The vector `v` contains pointers to copies of the strings in the `no_str` first rows of `A`. The copies have been dynamically allocated.

Hint: You can use the functions `calloc`, `strlen` and `strcpy` to perform much of what this function does.

Writing the vector

The fifth function you have to implement in this project is very similar to the second one. It should write out the strings pointed to by a 1-D vector array of character pointers. The function should write out the first n specified strings to *stdout*, each string beginning on a new line.

Proto: `void write_vector( char* v[MAXROW], int n );`

Usage: `write_vector( v, no_str );`

Pre: The vector `v` is a MAXROW array of char pointers, `no_str` is an int specifying the number of strings pointed to by `v`.

Post: The strings pointed to by the `no_str` first rows of `v` have been written out to *stdout*.

Hint: You're on your own here!

The `main` function

Your `main` function should create the necessary variables and then call the functions in sequence. It should first read in the 2-D array, then create the 1-D vector array, then write out the 1-D vector, then transpose the 2-D array and finally write out the 2-D array.

Grading policies

The code must be written and compiled in a modular fashion. There should be a file for each function, including the `main` function.

The set of files and file names must be:

array.h Prototypes and preprocessor definitions of `MAXROW` and `MAXCOL`. This header file is `#include`d by the remaining files.

array.c The `main` function.

read.c The `read_array` function.

writeA.c The `write_array` function.

trans.c The `transpose_array` function.

vector.c The `get_vector` function.

writev.c The `write_vector` function.

The code should be clearly written. Each function should be well documented, both in the header file, and in the implementation file. The documentation should include pre- and post-conditions and appropriate comments on important aspects of the implementation.

The `read_array` and `write_array` functions are 30% of the grade, the function `transpose_array` is another 30% of the grade and the two `*_vector` functions are each 15% of the grade. This leaves 10% of the grade, which is given for the program structure.

The program is due Friday, September 11, at midnight. Penalities for unexcused late submissions are described in the syllabus.

Submission procedure

The document `submit.html` (linked from the syllabus under the section Grading) contains a description of the submission procedure, which you should follow carefully.