MSC - INF101B
C Programming Language
Travaux Pratiques (TP) 2

Panagiotis PAPADAKIS

1 Introduction

The practical session TP 2 proposes coding exercises that will allow you to strengthen your understanding of the notions presented during Course 2. In the beginning of each TP, you should create a separate folder that will contain your work. For the second TP 2, create a folder named TP2 from a terminal and go into this directory.

2 Exercises

Write a function named `exchange` that will take 2 arguments of type `float` and will exchange (swap) their values. For example, if initially x1=3.5 and x2=4.2 then the function should perform x1=4.2 and x2=3.5. Then use this function as follows.

2.1

Write a program that asks from the user to give the initial values of the two numbers and then shows the result on the screen (hint, by using `scanf`). Allow your program to receive the values of the two numbers as arguments given from the command line (hint, by using `argc` and `argv` parameters of the `main` function). If these arguments are given from the command line then your program should use them, otherwise it should ask from the user to provide them, as before (hint, you will need the `atof` function).

2.2

Write another program that performs transposition of square matrices (equal number of rows and columns), namely, given a 2D matrix M with elements M_{ij}, it should produce a matrix M^T wherein $M^T_{ij} = M_{ji}$. You should use the function `exchange` that you coded earlier. For the input matrix, the user should specify the dimension of the matrix and your program will then fill the input

matrix with random numbers in the range [0, 10] (you will need the function `rand`). Here is an example output of your program:

Initial matrix:
8.40 3.94 7.83 7.98 9.12
1.98 3.35 7.68 2.78 5.54
4.77 6.29 3.65 5.13 9.52
9.16 6.36 7.17 1.42 6.07
0.16 2.43 1.37 8.04 1.57

Transposed matrix:
8.40 1.98 4.77 9.16 0.16
3.94 3.35 6.29 6.36 2.43
7.83 7.68 3.65 7.17 1.37
7.98 2.78 5.13 1.42 8.04
9.12 5.54 9.52 6.07 1.57

NOTE: take care to properly deallocate (`free`) the memory before the end of the program.

2.3

Write a program that asks from the user to provide a text from the keyboard and stores it into a dynamic character table for further processing (hint: you will need the functions `getchar, malloc` and `realloc`). The length of the character table should have the same length as the number of characters given, plus one extra at the end for the '\0' character. Your program should allow the following processing:

1. Create a function that changes all letters of the given string to upper-case or lower-case, depending on an argument (of type `enum`) given by the user, and prints the changed array to the screen (hint: you will need the functions `tolower` and `toupper`).

2. Use the string which contains all the text to create an array of strings (i.e. a `char **`). The length of the array should be equal to the number of words of the given text and each element of the array should correspond to a word (hint: you will need the function `strtok` and `strlen`). Show to the screen the total number of words given and the average number of letters per word.

Below is an example output of the program asked:
>./a.out ↩

```
Please give your text and press enter when you finish.
```
Hello there. This is my text, could you process it? Thanks.↩

```
You gave the following string:  Hello there.  This is my text,
could you process it?  Thanks.
Please choose if you want to change your string to lower-case
(type 0) or upper-case (type 1).
1↩
Here are the words that you gave after changing the case :
HELLO THERE. THIS IS MY TEXT, COULD YOU PROCESS IT? THANKS.
The total number of words is 11 and the average number of let-
ters per word is 4.090909
```

NOTE: take care to properly deallocate (`free`) the memory before the end of the program.

2.4

We can conveniently *redirect* the standard input that will be used by our program by adding the '<' operator after the executable file and then giving a text file as an argument. For example we can type:

```
./a.out < inputfile.txt
```

This means that whenever our executable requires input from the keyboard, it will instead process the file `inputfile.txt` as if it would be given by the keyboard.

Create a file named "inputfile.txt" and fill it with the text that you want to process, followed by the choice of uppercase or lowercase change. Remember that your program will treat this file in exactly the same way as if it would be given by the keyboard, so changing line in the text file would be considered as pressing enter from the keyboard. Verify that your program works as intended.

You may also redirect the standard output of your program by adding the '>' operator followed by the name of an output file in which you want to write the results. A command such as "`./a.out < inputfile.txt > output.txt`" will hence read from the inputfile.txt and write all results to output.txt. Verify this functionality with your program.