IC 150 P Computation for engineers lab

Lab assignment sheet no: 7, Odd semester, 2016

Strings

Prepared by: Maben Rabi

Objective for this lab session

- Working with strings by writing your own functions for manipulating them

Note:

- Every string must be terminated with the special character `'\0'`.
- Other than the two specific functions: `fgets`, `strlen`, you are not allowed to call any function defined in <string.h>.
- As before, you are expected to make your programs highly modular by neatly dividing computational tasks into modules that are written as functions.
- A suggestion: We recommend that you write one program for each task, and use the `switch` statement to handle the different parts of a task. You are of course allowed to reuse code from the other task.

Here is a simple program illustrating reading and outputting strings

```c
#include<stdio.h>
#include<string.h>

#define maxStringSize 100

int main() {
   char str[maxStringSize], shorterStr[2], temp;
   unsigned int i;

   printf("\nEnter the string :");
   fgets(str,maxStringSize,stdin); /* Reads string and stores it in
      str */

   i = strlen(str) ;
   shorterStr[0] = str[0];
   shorterStr[1] = str[i-1];

   printf("\nOriginal string is                          : %s\n",
      str);
   printf("String made up first and last characters is: %s\n",
      shorterStr);
   return (0);
}
```

getStringAndPrintFirstAndLastCharacter.c

Task one:

Get a string from the user and:

Part A: Print on screen the length of the string. You must compute the length without using the library function `strlen`. You may compute the length using the fact that every string is terminated with the character `'\0'`. Compare with what the function `strlen` computes.

Part B: Print on screen the original string and its reversed version.

Part C: Print on screen the original string and a substring whose starting and ending positions in the original string are specified by the user. (You must write a function to extract a substring from a given string)

Part D: Print on screen the (possibly shorter) string obtained by deleting all characters other than alphabetical characters, decimal digits, or the special characters: `'+'`, `'-'`, `'*'`, `'/'` (Write a recursive function to perform this).

Part E: Print on screen a string consisting only of the lower case alphabetical characters appearing in the order of first occurence in the original string. For example, if the original string is:

```
zaccaAbczz9/*+-lm1744AA
```

then the output should be:

```
zacblm
```

Part F **(optional)**: Print on screen a string consisting only of the lower case alphabetical characters appearing in decreasing order of number of occurrences. For example, if the original string is:

```
zaccaAbczz9/*+-lm1744AA
```

then the output should be:

`zcablm` or `czablm`

Task two:

Get a string from the user and delete all characters other than alphabetical characters or the special double quotes character `'"'`. Check if the character `'"'` occurs an even number of times. Print on screen the string of characters appearing between an odd numbered occurrence of the character `'"'` and its next occurrence. For example, if the input is

```
abzh"cjcchl"mo98767*+plma"zzz"hla
```

then the output should be:

```
cjcchl
zzz
```

Task three:

Get a string from the user and:

Part A: Delete all characters other than alphabetical characters, digits or the characters from the set:

$$\{ \texttt{'+'}, \texttt{'-'}, \texttt{'*'}, \texttt{'/'}, \texttt{'='}, \texttt{'('}, \texttt{')'} \}.$$

Part B: Check that character `'= '`, if it appears, appears only once, and never at the beginning or end of the input string.

Part C: Check that there are no consecutive occurrences of characters from the set:

`{ '+', '-', '*', '/'}.`

Part D: Check that the parantheses are matched. Precisely, check that there is a one-to-one correspondence between occurrences of the left paranthesis `'('`, and occurrences of the right paranthesis `')'`. Also make sure that there the character `'='` never occurs between matching occurrences of the left and right parantheses.

Part E **(optional)**: If the parantheses are all matched, check for redundant occurrences of parantheses, and remove them. Think carefully about which occurrences of parantheses are redundant.

Part F **(optional)**: Read about Lexical analysis from the PDF file at
`http://dragonbook.stanford.edu/lecture-notes/Stanford-CS143/03-Lexical-Analysis.pdf`