CSDS 297/397 Special Topics: C/C++ Programming Spring 2021

Assignment 1: C++ Basics, Loops, File I/O, String Manipulation

Instructor: Orhan Özgüner Due: Sunday, January 17 before 11:59 PM

Question 1

Write a program that asks the user to input one string (let's call it A), and then press `Enter`. Then it will ask the user to input a second string (we'll call it B), and press `Enter` again. Then ask the user to enter an integer number (we'll call it n), and press `Enter` one more time. Finally, the program will output "`TRUE`" if the first n characters of A and B are equal, or "`FALSE`" if they are not. The program should then exit. The program should exit immediately if n is not a valid positive integer.

Here are a couple of examples:

A	B	n	result
`this is a test`	`this is a trial`	11	`TRUE`
`this is a test`	`this is a trial`	12	`FALSE`
`this is a test`	`This is a trial`	4	`FALSE`
`*/-+`	`*/-+`	2	`TRUE`

Question 2

Your program should read in a file that is placed alongside it called `question 2.txt`, and output the number of *words* in that file to the console. A word is defined as any collection of non-space characters with a space either before or after it. For example,

```
"Pray tell, Dr.  Dijkstra, when do you think computers will be able to think?"
```

contains 14 words, and

```
 "Pray tell, Dr.  Dijkstra, when do you think computers will be able to think?"
"I don't know, Mr.  Reporter, when do you think submarines will be able to swim?"
```

contains 29 words.
Note that there may be more than one space between words. Be aware that we will be putting new material in `question 2.txt` during grading- your program should work with any content that adheres to the rules above, not just what is already in `question 2.txt`.

Question 3

Create a simple arithmetic expression parser that can handle the operations $+$, $-$, $*$, and $/$.

You can assume that all of your inputs will be positive integers, and should use truncating integer division. The parser does not need to handle parentheses, but does need to respect the order of operations

$$/ \rightarrow * \rightarrow - \rightarrow +$$

That is, $10 + 10/9 * 4 - 1$ should evaluate to 13, and so on. The program does not need to be able to evaluate expressions that produce a negative number. You can assume that your input does not contain any spaces.

When it runs, the program will wait for the user to type in an expression and press `Enter`, at which point it will evaluate the expression, print the result to the command line, and then wait for another expression to be entered, over and over again. Giving the program an empty expression, an invalid expression (like "1 + 2 + + 3") or an expression containing letters or any other symbols that are not $+$, $-$, $*$, $/$, or the digits 0-9, should cause it to quit.

Hint: You will want to make four passes, gradually simplifying the expression in order from the most important operator to the least and substituting the results. For instance, the expression above would transform as follows:

Input:
$S =$ "$10 + 10/9 * 4 - 1$"
$S =$ "$10 + \mathbf{1} * 4 - 1$"
$S =$ "$10 + \mathbf{4} - 1$"
$S =$ "$10 + \mathbf{3}$"
$S =$ "$\mathbf{13}$"
Output.

Input:
$S_2 =$ "$123 - 4 * 3 + 3/2 * 6 + 24/2 - 8$"
$S_2 =$ "$123 - 4 * 3 + \mathbf{1} * 6 + 24/2 - 8$"
$S_2 =$ "$123 - 4 * 3 + 1 * 6 + \mathbf{12} - 8$"
$S_2 =$ "$123 - \mathbf{12} + 1 * 6 + 12 - 8$"
$S_2 =$ "$123 - 12 + \mathbf{6} + 12 - 8$"
$S_2 =$ "$\mathbf{111} + 6 + 12 - 8$"
$S_2 =$ "$111 + 6 + \mathbf{4}$"
$S_2 =$ "$\mathbf{117} + 4$"
$S_2 =$ "$\mathbf{121}$"
Output.

(You just need to print the final value, you don't need to show the intermediate lines. They are for example only.)

Question 4

Create a program that identifies mathematical expressions within other text, and evaluates them in place while preserving the rest of the text. Characters +, −, ∗, and / that are *not* part of a mathematical expression should not be altered. For instance, the phrase

```
4+20 blackbirds baked in a 2200/7.
When no-one was looking, Lex Luthor stole 50-10 pies!  20+20!  That's as many as
4*10!
```

should transform into

```
24 blackbirds baked in a 314.
When no-one was looking, Lex Luthor stole 40 pies!  40!  That's as many as 40!
```

Here, we will define a "valid expression" as an operator (+, −, ∗, or /) where the nearest word on either side consists only of the digits 0-9. Expressions do not have any spaces inside of them. Each expression will contain only one operation.

The template program has a file called `question_4.txt` in the same directory as the executable. You should read this file, transform the text in it, and write it into another file called `question_4_transformed.txt`. The program should then exit.

Be aware that we will be putting new material in `question_4.txt` during grading- your program should work with any content that adheres to the rules above, not just what is already in `question_4.txt`.