ECE 2220: System Programming Concepts **Fall 2016**
Machine Problem 3 **Due: 11:59 pm, Thursday, September 22**

The goal of this machine problem is to learn to program with strings. There are two main tasks for this assignment. The first task is to process an input string to verify that the input format is valid and, if so, parse the input into a binary command operator and two strings representing the operands. The second task is to manipulate the operand strings using the arithmetic system defined for a finite field, and print the output as a string.

You are provided with a template program that prompts the user to enter an input string or to type ctrl-d to exit. Two function prototypes are provided that must be used as specified to perform the two tasks. The first function is called `process_input()` and the second one `calc_output()`. You are strongly encouraged to add additional functions.

Input Verification

Each command should have the structure **`[s1][op][s2]`** where **`[s1]`** and **`[s2]`** are strings. Each string must have at least one but no more than 12 valid *symbols.* The valid symbols are `'a'` through `'z'` and `'A'` through `'U'`; there are 47 valid symbols. The two strings do not need to have the same number of symbols. The **`[op]`** must be one of the following symbols: `'+'`, `'*'`, `'/'`, `'^'`. The program accepts input from the keyboard. The user is mischievous and can type anything on the keyboard in any order. However, our user has one principle: there will never be more than 78 characters per line (but the user will try 78!). You can assume that there will be no more than 78 characters per line. The input is **not** grammatically correct if:

- Either string is missing or consists of more than 12 characters.
- A string contains anything other than a lower-case letter.
- Any white spaces are found in any location include leading, trailing, or adjacent to the operator.

Output Generation

If a grammatically correct input is entered, the program should convert the two strings to two integer arrays and perform the given operation on each array location. For all operations, consider each corresponding array position independently (e.g., there are no carries). The resulting array should then be converted back to a string, and finally printed. If the two input strings are not the same length, then each output character beyond the length of the shorter string should be a copy of the character from the longer string with the opposite case.

Symbol	Value
a	0
b	1
...	...
z	25
A	26
B	27
...	...
T	45
U	46

The mathematical operations work as follows. Each symbol should be converted to an integer as shown in the table to the right (i.e., **`a`** is 0, **`b`** is 1, **`c`** is 2, …, **`U`** is 46). This mapping forms a finite field, and we can define addition and multiplication on this field.

Addition on a finite field: +

For this operation, for each array position find the sum of the two integers and if the result is larger than 46 find the remainder when the quotient is 47. That is, find $(X + Y)\ mod\ 47$. For example, if the user enters:

```
> abc+bbc
```

the result that should be printed is

```
'abc' + 'bbc' => 'bce'
```

```
> tuvwx+CBAzy
'tuvwx' + 'CBAzy' => 'aaaaa'
```

For this example the first string is longer than the second (the `'e'` in orange). The output corresponding to this extra symbol is `'E'`:

```
> orange+white
'orange' + 'white' => 'KyiGkE'
```

*Multiplication on a finite field: **

This operation is handled the same way as addition but uses modular multiplication. That is, find $(X \times Y)\ mod\ 47$.

```
> abcqH*AUydkzAB
```

the result that should be printed is

```
'abcqH' * 'AUydkzAB' => 'aUbbbZab'
```

```
> yyyyyyyyyyyy*abcdefghijkl
```

the result that should be printed is

```
'yyyyyyyyyyyy' * 'abcdefghijkl' => 'bybzcAdBeCfD'
```

Inversion on a finite field: /

For this operation, given two numbers X / Y, find a number Z such that $(X = Y \times Z)\ mod\ 47$. However, if Y is zero, then this operation is undefined, so set the output to zero.

```
> abcqHU/AUydk
```

the result that should be printed is

```
'abcqHU' / 'Aabdk' => 'aacviu'
```

```
> bbbbb/ABCDU
```

the result that should be printed is

```
'bbbbb' / 'ABCDU' => 'MhQnU'
```

Power on a finite field: ^

For this operation, given two numbers X ^ Y, calculate $X^Y \ mod\ 47$, or $\prod_1^Y X \ mod\ 47$. However, if Y is zero, then this operation is undefined, so set the output to one.

> yyyyyy^abcdef

the result that should be printed is

'yyyyyy' ^ 'abcdef' => 'bymgdz'

Notes

1. This lab must be done entirely without the string or character library functions! Any usage of `strlen()`, `strcpy()`, `strcmp()`, `islower()` or related functions will not be accepted. However, you are encouraged to write any functions to perform similar operations as needed.

2. You compile your code using:

```
gcc -Wall -g lab3.c -o lab3
```

The code you submit must compile using the `-Wall` flag and **no** compiler errors or warnings should be printed. To receive credit for this assignment your code must compile and at a minimum have each operator correctly calculate at least one input case. Code that does not compile or fails to pass the minimum tests will not be accepted or graded.

3. Submit your file `lab3.c` to the ECE assign server. You submit by email to ece_assign@clemson.edu. Use as subject header ECE222-1,#3. When you submit to the assign server, verify that you get an automatically generated confirmation email within a few minutes. If you don't get a confirmation email, your submission was not successful. You must include your file as an attachment. You can make more than one submission but we will only grade the final submission.

See the ECE 222 Programming Guide for additional requirements that apply to all programming assignments.

Work must be completed by each individual student. See the course syllabus for additional policies.