CS19001: Programming and Data Structures Laboratory

Lab Test – 2 (EVEN-PC)

Date: 01-November-2019

Problem Statement A: [*Reverse-Homomorphism*] **Marks: 35**

Mathematical functions are not only useful for number-theoretic computations, but it can also be applied to strings as well. This is usually done by substitution of characters within a string and is called *homomorphism*. For example, we can define the homomorphism for functions, $f()$ and $g()$, as follows:

$$f(a) = ab, \quad f(b) = c, \quad f(c) = b \quad \textit{and} \quad g(a) = ab, \quad g(b) = ba$$

Moreover, inverse homomorphism for functions, $f()$ and $g()$, may be defined in similar lines as follows:

$$f^{-1}(ab) = a, \quad f^{-1}(c) = b, \quad f^{-1}(b) = c \quad \textit{and} \quad g^{-1}(ab) = a, \quad g^{-1}(ba) = b$$

Now applying function $f^{-1}()$ to the string $abcb$ yields, $f^{-1}(abcb) = f^{-1}(ab)f^{-1}(c)f^{-1}(b) = abc$
You can apply $f^{-1}()$ again to the resulting string producing, $f^{-2}(abcb) = f^{-1}(f^{-1}(abcb)) = f^{-1}(abc) = ab$
Moreover, you can generate *composite* functions as, $g^{-1}(f^{-2}(abcb)) = g^{-1}(ab) = a$
Your task is to write a C-program that performs the following activities:

- It takes from the user a set of homomorphism for various functions,
- It, then, takes from user a set of composite functions with the number of times (negative) to apply inverse homomorphisms and the start string for each of these composite functions,
- It outputs the resulting string for each composite function after the application of inverse homomorphisms over each of the composite functions as given.

The user input will be presented in the following way (each line of user input are also explained below):

```
2                           || <Number-of-Functions> for which homomorphisms will be defined next
f 3  a->ab  b->c  c->b      || <Function-Name> <Number-of-Homomorphisms> <Set-of-Homomorphism-Rules>
g 2  a->ab  b->ba           || <Function-Name> <Number-of-Homomorphisms> <Set-of-Homomorphisms-Rules>
3                           || <Number-of-Composite-Functions> which will be mentioned next
1                           || <Number-of-Different-Functions> to be used
g -2  abba                     <...<Function-Name> <Number-of-Repeated-Homomorphisms>...> <Input-String>
2                           || <Number-of-Different-Functions> to be used
f -1 g -1  abbaabba            <...<Function-Name> <Number-of-Repeated-Homomorphisms>...> <Input-String>
1                           || <Number-of-Different-Functions> to be used
f -3  abcabcc                  <...<Function-Name> <Number-of-Repeated-Homomorphisms>... > <Input-String>
```

The first line provides the number of functions, N, for which the user shall input the homomorphism rules in next N lines. For each of the next N lines, the user inputs the name of the function, number of homomorphism rules and each such rule which is in `character` $\rightarrow$ `string` format. The next line captures the number of example composite functions, say M, for which you will produce the output results. For each of the next M pair of lines, the first line mentions the different function names present within the composite function and the second line gives the composite function details (the function name and its repeated inverse homomorphisms) and the input start string. For example, "`f -1 g -1  abbaabba`" means $f^{-1}(g^{-1}(abbaabba))$. Now, for the above input, the output will be:

```
g^-2 (abba) = a                  || indicating the result for g^{-2}(abba)
f^-1 g^-1 (abbaabba) = aa        || indicating the result for f^{-1}(g^{-1}(abbaabba))
f^-3 (abcbabcbc) = aab           || indicating the result for f^{-3}(abcbabcbc)
```

That is, after the application of homomorphism rules over the composition of functions, the results will be printed on a single line for each asked composite functions as:

```
<...<Function-Name>^<Number-of-Inverse-Repeatitions>...> (Start-String) = Result-String
```

You may assume that, for each character appearing in a string, a homomorphism is defined. Also, the inverse homomorphism derived is unique and unambiguous. All characters will be lowercase characters.

Example Inputs/Outputs A:

Sample-1:

```
2
f 3 a->ab b->c c->b
g 2 a->ab b->ba
3
1
g -2 abba
2
f -1 g -1 abbaabba
1
f -3 abcbabcbc

++ Applied Inverse-Homomorphism Results ++
g^-2 (abba) = a
f^-1 g^-1 (abbaabba) = aa
f^-3 (abcbabcbc) = aab
```

Sample-2:

```
2
f 3 a->ab b->c c->b
g 2 a->ab b->ba
1
2
g -1 f -1 abcabb

++ Applied Inverse-Homomorphism Results ++
g^-1 f^-1 (abcabb) = Inverse-Homomorphism for ‘ac’ in Function ‘g’ is Undefined! (ERROR)
```

Sample-3:

```
2
f 3 a->ab b->c c->b
g 2 a->ab b->ba
1
2
h -1 f -1 abcb

++ Applied Inverse-Homomorphism Results ++
h^-1 f^-1 (abcb) = Function ‘h’ Does Not Exist! (ERROR)
```

Problem Statement B: [*Range-One-Query*] **Marks: 15**

You start by writing a 0 on the first row. Now in every subsequent row, you look at the previous row and replace each occurrence of 0 with 10, and each occurrence of 1 with 01.

Write a *recursive* C-function such that, given row n and index k, it returns the k^{th} indexed symbol in row n (Assume that, the indexes of n and k starts from 1): `int kthSymbol(unsigned int n, unsigned long long int k)`
Note: *In* `kthSymbol` *function, you are NOT allowed to expand till n^{th} row everything and then return the k^{th} indexed symbol from such expanded string. Please do it smartly and recursively!*

Now, write a C-program with the `main()` function, where you read n, two indexes k_1, k_2 ($k_1 \le k_2$), and first print all the expanded strings for n rows. Then, the program determines/prints the number of ones (1s) from ${k_1}^{th}$ indexed symbol to ${k_2}^{th}$ indexed symbol in row n, *using the function* `kthSymbol` *only.*

(Assume that, n will be an integer in the range $[1, 30]$ and k_1, k_2 will be integers in the range $[1, 2^{n-1}]$.)

Example Inputs/Outputs B:

```
Enter n: 5
Enter k1: 4
Enter k2: 10
++ The 1-th Row: 0
++ The 2-th Row: 10
++ The 3-th Row: 0110
++ The 4-th Row: 10010110
++ The 5-th Row: 0110100110010110
++ The Number of 1s from 4-th index to 10-th Index in Row 5 is: 3
```

Submit a single C source file for each problem. Do not use global/static variables.