CS153/453 Fall 2017

HW 7

Due: Monday Oct 23, 2017 11:59pm (before midnight)

Task 1

Complete the codes for the template program given for Participation Activity 7.2.2 (Setting minimum field width of conversion specifiers).

Task 2

We want to re-do the program given for Participation Activity 7.3.2 (String searching example: Hangman).

The problem specification is the same as described in the activity. Our goal is to rewrite the program in a different style.

We combine all characters guessed so far as a string. Consider `word = 'onomatopoeia'` as in the given example. After we have entered the guesses `'y'`, `'m'`, `'n'` and `'a'`, we maintain the four guessed letters as a string called `guesses = 'ymna'`. The `hidden_word` is computed to be `'-n-ma------a'`.

Write a function `hidden_word( word, guesses )` which returns the hidden word given `word` and `guesses`. You are supposed to use a different logic than the one given in the sample program. You should initialize `hidden_word` to an empty string. Then for each character in `word`, decide (by checking with `guesses`) to see if the character should be hidden, or displayed.

Using the function `hidden_word`, you are asked to rewrite the program. You are also asked to improve the dialog between the program and the user. When the length of `user_input` is not 1, you should print a meaningful message before asking the user to enter another character.

Task 3

Consider the Challenge Activity 7.3.2 (Replace abbreviation). We generalize the problem to the function `decoded( user_tweet, acronym, full_word )` that returns the string obtained by replacing every occurrence of `acronym` in `user_tweet` by `full_word`. For the example given in the example, the decided tweet is obtained by a call to `decoded( 'Gotta go.  I will TTYL.', 'TTYL', 'talk to you later' )`.

For the general problem, one attempt is to repeatedly expand `user_tweet` by replacing each occurrence of `acronym` by `full_word`. But there is a danger to this approach. Consider `decoded('abcdd', 'cd', 'bc')`. After replacing the first occurrence of `'cd'`, the user tweet becomes `'abbcd'`, which again has another occurrence of `'cd'`. Replacing one more time, we end up with the string `'abbbc'`. But, the resulting string obtained is not the intended answer. It makes more sense if we just replace the single occurrence of `'cd'` in `'abcdd'` by `'bc'` to give `'abbcd'`, and returns that as the answer without further replacement. On the other hand, `decoded('abcdcd', 'cd', 'bc')` should return `'abbcbc'` after two rounds of replacement.

Implement the function `decoded` according to the logic explained. Also, provide a main program to test the correctness of `decoded`.

Task 4 (for CS453 only)

We consider a way to encrypt a message using a substitution method.

Example:

```
alphabet = "abcdefghijklmnopqrstuvwxyz "
key      = "sxzaijhbwpekfcqrgdtluv noym"
```

Note that the `key` is a permutation of the 26 letters from `a` to `z`, and the space characters.

The coding scheme above stated that an `'a'` is coded as `'s'`, `'b'` as `'x'`, `'c'` as `'z'`,... etc.

Example, given a plain text `'today is tuesday'`, the encrypted text is `'lqasomwtmluitaso'`.

You are asked to write two functions

```
substitutionEncrypt(plainText, key)
```

which returns the ciphered text given a `plainText` string, and the 27-character string `key`, and

```
substitutionDecrypt(cipherText, key)
```

which returns the plain text given a `cipherText` string, and the 27-character string `key`.

You should provide a main program that test the correctness of your two functions.

For the function `substitutionDecrypt(cipherText, key)`, you are required to compute an inverse of the key first, before applying the inverse key mapping to decipher the coded text.