CS1010E Practical Examination 2

Grading Guidelines:

- No mark will be given if your code cannot run, namely any syntax errors or crashing.
  - The graders will just grade what you have submitted and they will **not** fix your code.
  - Hint: Comment out any part that you do not want.
- Workable code (that can produce correct answers) will only give you partial marks. Only good and efficient code will give you full marks
- Marks will be deducted if your code is unnecessarily long or hard-coded.
- You must use the same function names as those in the skeleton given
- You ***cannot import*** any package or functions, but you can use built-in functions of Python. Your solution ***cannot use any data structures not covered*** in this course.

Important Note on Coursemology

For each task, you will only be allowed to run your codes for a total of **5 (five) times** on Coursemology. Public test cases have been included in the Practical Exam template for local testing (in IDLE environment). Nevertheless, you should **spread out testing of your codes in Coursemology** (and NOT only at the end of the Practical Exam) to prevent slowdowns when submitting at last moment.

Note that it is NOT a must for you to run your code for it to be submitted. You can always delay running the code by clicking "Save Draft" to save without running.

Lastly, you **MUST press the submit button** to complete your submission in Coursemology. Failing to press the submit button will result in NO SUBMISSION of your code, and you will receive ZERO mark for the unsubmitted part.

Problem Dependency

There are two independent problems in this Practical Exam. Each problem has its sub tasks, and the latter sub tasks **may** depend on the former sub tasks. **But**, *unless otherwise mentioned*, when submitting your solution to a sub task, you should submit a **full** solution; specifically, you cannot assume that the code for earlier sub tasks will be provided when you submit solutions to latter sub tasks.

Problem 1: Translation [35 marks]

Given a string of text, your task is to translate the text by replacing some character strings by another, as dictated by a dictionary.

For instance, given the following string of text:

`Without` That Breath!

And the following dictionary:

{ `'th'` : 'zh', 'a' : 'ai', 't' : 'se' }

The translated string will be:

`Wizhouse Thaise Breaizh!`

Note the following rules about the translation:

1. The translation is **case sensitive**. In the above, we translate the text 'th' but not the text 'Th', because 'th' is the key in the dictionary, but not 'Th'.
2. If there are more than one possible key in the dictionary that can be applied at the same time, the longer key will be used. In the above, we found that both 'th' and 't' are applicable when we encountered the string 'th' in the word 'Without'. Here, we choose to apply the rule for 'th' and not the rule for 't', because 'th' is a longer string than 't'.

Task 1: [15 marks]

Write a Python function *prefix*(*aString*, *keys*) that takes in a text string and a list of keys, and returns the **longest** key in the list that **matches the beginning** of *aString*. It returns the Boolean value `False` otherwise.

Sample runs:

```
>>> prefix('thout',
['t','a','th']) 'th'
>>> prefix('Thas',
['t','a','th']) False
>>> prefix('ttas',
['t','a','th']) 't'
```

Task 2: [15 marks]

Write a Python function ***trans***(***aString***, ***dictionary***) that uses *dictionary* to translate *aString*. The function returns the translated string.

Sample runs:

```
>>> tmap = { 'th' : 'zh',  'a' : 'ai', 't' : 'se' }
>>> trans('Without That Breath!',tmap)
'Wizhouse Thaise Breaizh!'
>>> trans("that's not the sameth",tmap)
"zhaise's nose zhe saimezh"
```

Task 3: [5 marks]

Repeat the same task as Task 2, but **write the function in recursion form**. If you have already written the function in recursive form in Task 2, simply copy and paste your code here. You will receive this 5 marks only when your code is fully correct and recursion is used as your main translation algorithm.

Problem 2: Tabulating A Recursive Function [35 marks]

Consider the following piece of recursive definition for $recurse$. (This code has been provided in the code template.) We would like to arrange the solutions of calling $recurse$ with different arguments in a table for easy reference.

```
def
    recurse(n,m)
    : if n == 0:
        return m
    if m == 0:
        return n
    if n - m % 2 == 0:
        return recurse(n-1,m-1) - recurse(n-1,m-2) +
    recurse(n,m-1) else:
        if n > m:
            return recurse(n-m,m)
        else:
            return recurse(n-1,m) + recurse(n,m-1)
```

Task 1: [10 marks]

Write a Python function $chkRecurse(k)$ that takes in a positive integer k and returns a list containing the solution of calling $recurse(j, j)$ for j from 0 to $k - 1$. You are allowed to call $recurse$ function directly in your solution. Your **must** produce your answers ***using list comprehension***.

Sample runs:

```
>>> chkRecurse(1)
[0]
>>> chkRecurse(6)
[0, 2, 6, 15, 40, 114]
```

Task 2: [25 marks]

Write a Python function *tabulate*(n, m) that takes in two positive integers n, m and returns two-dimensional array containing the solution of calling *recurse*(i, j) for i from 0 to $n-1$ and j from 0 to $m-1$.

The table created from evaluating *tabulate*(3,5) is shown below.

		j from 0 to 5					
i from 0 to 2		0	1	2	3	4	5
	0	**0**	**1**	**2**	**3**	**4**	**5**
	1	**1**	**2**	**4**	**5**	**9**	**10**
	2	**2**	**2**	**6**	**11**	**20**	**30**

Here, the entry indexed by `[2][2]` contains the value `4`, which is the result of evaluating *recurse*(2,2). Similarly, the entry indexed by `[2][5]` contains the value `30`, which is the result of evaluating *recurse*(2,5).

Sample runs:

```
>>> [ tabulate(5,5)[i]  for i in range(5 ]
    [0]                              )
[0, 1, 2, 3, 4]
>>> [ tabulate(5,5)[0]  for j in range(5 ]
    [j]                              )
[0, 1, 2, 3, 4]
>>> [ tabulate(5,5)[i]  for i in range(5 ]
    [i]                              )
[0, 2, 6, 15, 40]
```

(*Note: You are allowed to call function* *recurse* *in your code, but you have to ensure that values for all the entries of the table can be generated efficiently.*)

Problem 3: Word Search [30 marks]

Given a board of (lowercase) alphabets and a word, determine if the word appears in the board.

For instance, consider the following board, the word “imagination” does occur in the board, as highlighted in red:

```
m v l o
l i o n
g r t a
n r f y
n e p i
g i g a
m a k q
g v
```

The rules for determining if a word does appear in a board is as follows:

- We can trace the characters in the word from left to right in the board. Every two adjacent characters in the word are next to each other in the board. That means, one is to the left/right/up/down of the other. Eg: The following board **does not** contain the word “image” – the character “i” is not to the left/right/up/down of the character “m”:

```
y e p n n
i g g a m
k g t t m
l g t v i
n i z w
```

- A character in the word cannot be used twice in tracing the word in the board. Eg: The following board **does not** contain the word “imagine” because the character “i” is incorrectly used twice to form the word “imagine”.

```
n e p n n
i t g a m
k g t i m
l g t e n
n i z w
```

Write a function `search_word(board, word)` that takes a board and a word, and determines if the word occurs in the board. When the word is found in the board, your function will return the location (represented by a tuple containing the row and column indexes in the board) of the first character in the word; otherwise, it returns an empty tuple. There may be multiple occurrences of the word in the board, just return the position of the first character of any of the found words.

Here are some sample runs :

```
>>> prebd = ['mvlol','iongr','tanrf','ynepi', 'gigam','akqgv']
>>> board = list(map(list,prebd))
>>> word = 'imagination'
>>> word_search(board,word)
```

```
(3,4)

>>> prebd = ['yepnni','ggamkg','ttmlgt','vinizw']
>>> board = list(map(list,prebd))
>>> word = 'image'
>>>
word_search(board,word) ()

>>> prebd = ['nepnni','tgamkg','timlgt','ennizw']
>>> board = list(map(list,prebd))
>>> word = 'imagine'
>>>
word_search(board,word) ()
```