Computational Thinking and Problem Solving (COMP1002) and Problem Solving Methodology in Information Technology (COMP1001)

Assignment 2 Sample Solutions

1. [25 marks] In cryptography, a polyalphabetic substitution cipher is an encryption algorithm that converts an English letter to another letter based on a key. For example, if the English text (*p*), is “dennisliu” and the key (*k*) is “comp”, the encrypted text (*c*) will be “fszckgxxw”. The following table shows the conversion:

English text (*p*)	d	e	n	n	i	s	l	i	u
Key (*k*)	c	o	m	p	c	o	m	p	c
Encrypted text (*c*)	f	s	z	c	k	g	x	x	w

In the above example, the first letter is ‘d’, and the corresponding key letter is ‘c’. The encrypted text is ‘f’. This is because if the English letter, p_i, is ‘a’, the encrypted letter, c_i, is ‘c’. If p_i is ‘b’, c_i becomes ‘d’. If p_i is ‘c’, c_i becomes ‘e’. So, ‘d’ becomes ‘f’, if k_i is ‘c’. $i = 0 \ldots \text{len}(p) - 1$. So, the relation is:

$$c_i = \text{str}((\text{ord}(p_i) + \text{ord}(k_j) - 2 * \text{ord}(\text{‘a’})) \bmod 26 + \text{ord}(\text{‘a’}))$$

Note that if *p* is longer than *k*, *k* will be reused. Therefore, $j = i \bmod \text{len}(k)$. In the above example, *k* is thus “compcompc”.

Answer the following questions:

a) Why does the above relation require “mod 26”? Illustrate with an example.

b) Write down the *pseudo-code* to illustrate the above encryption process with the following input-output specifications:

Input: *p* and *k*
Output: *c*

c) Write down the *pseudo-code* to illustrate the decryption process, i.e., given *c* and *k*, convert *c* to *p*, with the following input-output specifications:

Input: *c* and *k*
Output: *p*

a)
This is because (ord(p_i) + ord(k_j) – 2 * ord(‘a’) + ord(‘a’)) may result in a value that is not a representation of English letter. For example, if p_i is ‘z’, k_i is ‘b’. c_i = ord(‘*z*’) + 1, which is not valid letter. ((ord(‘z’) + ord(‘b’) – 2 * ord(‘a’)) mod 26 + ord(‘a’)) will give ord(‘a’).

b)

```
c = empty string
For each letter in p and each keyletter in k
        c += str(ord(letter) + ord(keyletter) – 2 * ord(‘a’) mod 26 + ord(‘a’))
return c

p = empty string
```

```
For each letter in c and each keyletter in k
        p += str((ord(letter) – ord(keyletter) + 26) % 26 + ord('a'))
return p
```

2. [25 marks] Suppose there is a “coin-moving game” and here is the description:

- M square tiles are placed consecutively in a straight line. The distance between every two adjacent tiles is 1.
- N coins are placed in N different tiles which may not be consecutive. You are asked to move all the piles to the same tile, which M >= N.
- Find the section which will require the smallest total moving distance.

a) Write down the pseudo-code of how to move all the coins to one single tile and count the total number of moves. Write down the *input* and *output* specifications.

b) Use your solution in 2a) as a function and write down the pseudo-code to find the smallest total moving distance. Write down the *input* and *output* specifications.

a)
Input: (1) A list, L, of size M, representing the tiles. If there is a coin, the position will have a value 1; otherwise, 0. (2) i, which represents the position of the tile, to where the coins will be moved.
Output: The total number of moves

```
pos = 0         #represent the first tile
count = 0       #represent the total move
while pos < M
        if pos < i and L[pos] == 1
                count = count + (i – pos)
        else if pos > i and L[pos] == 1
                count = count + (pos – 1)
        pos = pos + 1   #move to the next position
return count
```

b)
Input: A list, L, of size M, representing the tiles. If there is a coin, the position will have a value 1; otherwise, 0.
Output: The smallest total moving distance, x

Let the function of a) called, countMove(L, i)

```
pos = 0         #represent the first tile
count = 0       #represent the total move to a particular tile
x = (M-1)M/2    #represent the minimum
while pos < M
        count = countMove(L, pos)
        if count < x
                x = count
        pos = pos + 1
return x
```

3. [30 marks] Complete the following tasks:

a) **Create** your own *min* function in Python, which finds the minimum number and its location (in zero-based index) in a set of different numbers.

Write a function, called `partA()`, to test your *min* function. When `partA()` is called, the input/output of your program will look like below:

```
Please enter a list of different numbers separated by ',': 4, -5,6,2,0, 1,-7,10,3
The minimum number is -7.
Its location is 6.
```

b) Using your *min* function in 3a), implement a *sorting* function based on the method discussed in Lecture 4 to sort a set of different numbers. The function will return a list of sorted values in ascending order.

Write a function, called `partB()`, to test your *sort* function. When `partB()` is called, the input/output of your program will look like below:

```
Please enter a list of different numbers separated by ',': 4,-5,6,2,0, 1,-7,10,3
A list of sorting values in ascending order: [-7, -5, 0, 1, 2, 3, 4, 6, 10].
```

You need to include *docstring* to describe the *min* and *sort* functions. Also, zero mark will be awarded if the built-in/external functions, *min* and *sort*, are used.

```
# function
def myMin(data):
    '''
    myMin(data) is used to find the minimal number and its location (in zero-based index)
    in a set of different numbers.
    Parameter:
      data: a list of number.
    return:
      minNum: the minimal number.
      location: the location of the minimal number in the data.
    '''
    minNum = data[0]
    location = 0
    counter = -1
    for i in data:
        counter = counter + 1
        if i < minNum:
            minNum = i
            location = counter
    return minNum,location

def mySort(data):
    '''
    mySort(data) is used to to sort a set of different numbers in ascending order.
    Parameter:
      data: a list of number.
    return:
      result: a list of sorting values in ascending order
    '''
    result = []
    index = []
    while len(data) > 0:
        minNum,location = myMin(data)
        result.append(minNum)
        data.pop(location)
    return result,index

def partA():
    data  = eval(input("Please enter a list of different numbers separated by ',': "))
    minNum,location = myMin(data)
    print("The minimum number is ", minNum, ".", sep="")
    print("Its location is ", location, ".", sep="")

def partB():
    data  = eval(input("Please enter a list of different numbers separated by ',': "))
    data  = list(data) # convert tuple to list
    result, index = mySort(data)
    print("A list of sorting values in ascending order: ", result, ".", sep="")

#Demo
partA()
partB()
```

4. [20 marks] Develop a Python function, named `changeString()`, to update the character in a string in a particular location. It should accept a string, a character, and the index, as parameters and return the converted string.

   Write a function, called `main()`, to test your `changeString()` function. When `main()` is called, the input/output of your program will look like below:

```
Input a string: Hello World
Input a zero-based location in a string to be changed: 5
Input a character to be updated in that location: @
The string updated: Hello@World
```

```
def changeString(stringInput, index, newChar):
    '''
    changeString(stringInput, index, newChar) is used to to
    change the character in a string in a particular location
    Parameters:
      stringInput:
      index: a zero-based location in a string to be changed
      newChar: a character to be updated in that location
    return:
      result: The string updated
    '''
    stringUpdated = ""
    counter = 0
    for a in stringInput:
        if counter != index:
            stringUpdated = stringUpdated + a
        else:
            stringUpdated = stringUpdated + newChar
        counter += 1
    return stringUpdated

def main():
    stringInput  = input("Input a string: ")
    index = int(input("Input a zero-based location in a string to be changed: "))
    newChar = input("Input a character to be updated in that location: ")
    print(f"The string updated: {changeString(stringInput, index, newChar)}")

# Demo
main()
```