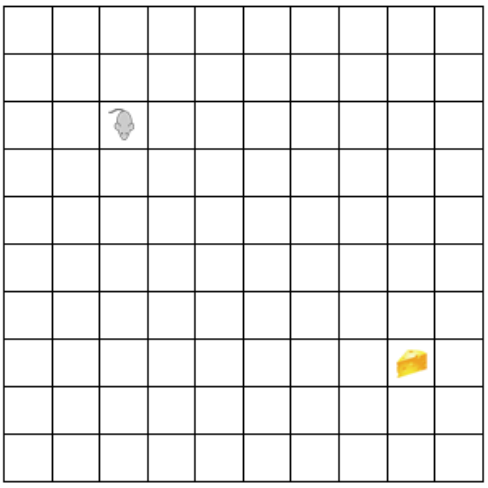
Computational Thinking and Problem Solving (COMP1002)

Assignment 2 Solutions

(Due on **29 October 2022 (Fri) at 12:00 noon**)

1. [20 marks] Suppose that is a mouse, which can freely move on a 10 x 10 2D plane. It initially occupies one of the slots of the plane. Its goal is to find out the cheese located in another slot. Every time, it can only move to its neighbor slot in either of the directions, N, E, S or W (its head in the diagram is facing S). It cannot move across the boundary of the plane. The mouse is shorted-sighted that it can only see objects that are up to 2 slots away horizontally and vertically and 1 slot away diagonally (not including its current slot). If it can see the cheese, it will move in a path that leads to the cheese. Otherwise, it will continue to move to next slot randomly.



Answer the following questions:

a) Write down the pseudocode of a procedure, called `move()`, that contains the logic of a single mouse move. [5 marks]

b) Write down the pseudocode of a function, called `findCheese()`, that contains the logic of checking the existence of the cheese in the visible area of the mouse. It should return true or false, and the location of the cheese, relative to the mouse. [5 marks]

c) Write down the pseudocode of a procedure, called main(), that contains the logic for the mouse to achieve its goal. You must use `move()` and `findCheese()` in `main()`. [10 marks]

a)

```
# Assume the position of the mouse is known by the system, or it can be an input of the function
move()
    Choice = nothing
    Repeat
        Choose randomly N, E, S or W and assign it to Choice
    Until Choice is not crossing the boundary

    The mouse moves Choice
```

[5 marks]

b)

findCheese()

```
        allVisibleNeighbours = [N, E, S, W, NN, EE, SS, WW, NE, SE, SW, NW]
        for each i in allVisibleNeighbours
                if i contains the cheese
                        return true, i
        return false, nothing
```
[5 marks]

c)
```
main()
        bingo = false
        repeat
                move()
                if current location contains the cheese
                        bingo = true
                found, location = findcheese()
                if found
                        if location has two characters
                                break down the characters to two, namely c1 and c2
                                move to c1
                                move to c2
                        else
                                move to location
                        bingo = true
        until bingo = true
```
[10 marks]

2. [25 marks] In this course, you have learnt the concepts of binary addition. Suppose you are given an integer, `A`, in 2s-complement binary representation with arbitrary length.

   Answer the following questions:

   a) Given `A` of length `m`. Write down the pseudocode, in terms of a function, to convert it to length `n`, where `n > m`. [10 marks]

```
function extend(A, m, n)
    bitsToExtend = n – m
    i = 0
    if A >= 0
        while i < bitsToExtend
            prepend 0 to A
    else
        while i < bitsToExtend
            prepend 1 to A

    return A
```
[10 marks]

   b) Use your answer in a). Given `A` of length `m` and `B`, another 2s-complement integer, of length `n`, write down the pseudocode, in terms of a function, to perform addition of `A` and `B`. You must demonstrate the addition in bit-by-bit level. Note that you may have to extend the resulting number to maintain a correct representation. [15 marks]

```
function add(A, B, m, n)
    # Bit Extension to make sure that length of A equals length of B [5 marks]
```

```
bitlength = 0
if m > n
        B = extend(B, n, m)
        bitlength = m
else if m < n
        A = extend(A, m, n)
        bitlength = n
else
        bitlength = m

#Assume the rightmost bit is having an index 0, its bit on the immediate left is 1 and so on
#Addition bit-by-bit [5 marks]
carry = 0
for i in [0 … bitlength - 1]
        carry, C[i] = A[i] + B[i] + carry

#Bit Extension for the Calculation Result [5 marks]
if A[bitlength – 1] = 0 and A[bitlength – 1] = 0
        if C[bitlength – 1] = 1
                prepend 0 to C
else if A[bitlength – 1] = 1 and A[bitlength – 1] = 1
        if C[bitlength – 1] = 0
                prepend 1 to C

return C
```

3. [30 marks] Create a Python program with the following requirements:

   a) **Create** your own *max* function, called *myMax*, which finds the maximal number and its *i*-based location (*i* starting from 1) in a series of different numbers (no two or more numbers are of the same value and there are at least two numbers). The *while* loop is required to build the *myMax* function. Use *docstring* to describe your function. Marks will be deducted if *for* loop and/or the built-in function for *max* are used. In the demonstration of calling your *myMax* function, print the *docstring* about the function, **and** show how to use your function as below:

```
Please enter a list of different numbers separated by ',': 1,-3,4.5,5,18,-1,3,-4
The maximal number is 18 .
Its location index is 5 .
```

[15 marks]

```
# function
def myMax(data):
    '''
    myMAx(data) is used to find the maximum number and its location starting from 1
    in a set of different numbers.
    Parameter:
      data: a list of numbers
    return:
      maxNum: the maximal number
      location: the location of the maximum number in the data
    '''
    maxNum = data[0]
```

```
    n = len(data)
    location = 1
    i = 1  # index

    while i < n:
        if data[i] > maxNum:
            maxNum = data[i]
            location = i + 1  # starting from 1
        i = i + 1

    return maxNum,location

# demo
print(myMax.__doc__)
data  = eval(input("Please enter a list of different numbers separated by ',': "))
maxNum,location = myMax(data)
print("The maximal number is", maxNum, ".")
print("Its location index is", location, ".")
```

[Marking Scheme]
Create a max function
Max number: 5 marks
Location: 5 marks
(if for-loop is used, up to 5 marks)

Docstring: 2 marks
Demo: 3 marks
Accepting the other versions fulfilling the requirements.

b) Using *myMax* function created in b), **create** your sorting function, called *mySort*, to sort a set of different numbers. The function will return a list of sorting values in descending order. The *while* loop statement is required. Use *docstring* to describe your function. Marks will be deducted if the built-in functions for *sorting* are used. If you use the other sorting method without calling your *myMax* function, only a maximum of half of the marks of this question will be awarded. You may use build-in functions to add or remove an element from a list. In the demonstration of calling your sorting function, print the *docstring* about the function, **and** show how to use your function as below:

```
Please enter a list of different numbers separated by ',': 1,-3,4.5,5,18,-1,3,-4
A list of sorting values in descending  order: [18, 5, 4.5, 3, 1, -1, -3, -4] .
```

[15 marks]

```
def mySort(data):
    '''
    mySort(data) is used to sort a set of different numbers in descending order.
    Parameter:
      data: a list of number
    return:
      result: a list of values sorted in descending order
    '''

    result = []
```

```
    index = []
    while len(data) > 0:
        maxNum,location = myMax(data)
        result.append(maxNum)
        data.pop(location-1)  # Covert location starting from 1 to 0
    return result,index

#Demo
print(mySort.__doc__)
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 descending  order:",result, ".")
```

[Marking Scheme]
Create a mySort function: 10 marks
Docstring: 2 marks
Demo: 3 marks

Accepting the other versions fulfilling the requirements.

4. [20 marks] Develop a function, on input of a string, *s*, and a character, *c*, returns the number of occurrence of *c* in *s*, and a list of index(s) (one-based) of *s* that represent(s) the location(s) of *c*. Use *for* loop in your implementation. Use *docstring* to describe your function. Marks will be deducted if *while* loop and any built-in functions are used in the implementation of this function. In the demonstration of calling your function, print the *docstring* about the function, **and** show how to use your function as below:

```
Input text: Never Too late To Start. Please go ahead with us.
Input a character to be searched: t
The character t in the text occured 4 times at [13, 20, 23, 44].
```

```
def searchCharacter(stringInput, char):
    '''
    searchcharacte(stringInput, index, newChar) is used to to find the character \
    in a string with returning occurrence frequency and all locations.
      stringInput: input string text
      char: a character to be search in that location
    return: (fq,loc)
    fq: occurrence frequency
    loc: a list of all locations for the character in list
    '''

    i = 0 # index
    loc = [] #location
    fq = 0 #occurrence frequency
    for a in stringInput:
        if a == char:
            fq = fq + 1
            loc = loc + [i+1]
        i += 1

    return fq,loc
```

```
# demo
print(searchCharacter.__doc__)
stringInput  = input("Input text: ")
char = input("Input a character to be searched: ")
fq,loc = searchCharacter(stringInput,char)
print(f"The character {char} in the text occured {fq} times at {loc}.")
```

[Marking Scheme]
Create a *searchcharacter* function: 14 marks
Docstring: 2 marks
Demo: 5 marks

Accepting the other versions fulfilling the requirements.