COMP1001 – Problem Solving in Information Technology
Final Examination

Please read the following instructions carefully.

1. Before the commencement of the examination
    a. Put away your phones, calculators, and other electronic devices.
    b. Put down your name, student ID and program name on the answer book.
    c. For the answer book:
        - Leave the first page empty.
        - Use the next two pages for Q1, the next two for Q2 and so on.
        - You can leave the two pages empty for the one you choose not to do.
2. About the questions and submissions
    a. Each question carries 10 marks.
    b. **Answer any 9 questions** and always attach succinct explanation. If you do all of them, the one with the highest mark will be discarded.
    c. Submit your code for Q6-Q10 to Blackboard.
    d. Write down your answers to Q1-Q5 in the answer book.
3. During the examination
    a. You can use everything available in your PC and access everything under Blackboard and nothing else.
    b. No communication with others in any form
    c. Bathroom policy: one at any time. Raise your hand and get the approval before going.
    d. Raise your hand if you have any question about the examination paper.
4. At the end of the examination period.
    a. Pass your answer book towards the middle aisle in your respective room.
    b. You can keep the examination paper.

1. (**Computation is the automation of our abstraction**) The table below defines the value of $x \oplus y$ where $x, y \in \{0,1,2,3,4,5,6,7,8,9\}$ and the result consists of two letters z_1z_0, where $z_1, z_0 \in \{0,1,2,3,4,5,6,7,8,9\}$.

$\oplus$	0	1	2	3	4	5	6	7	8	9
0	00	01	02	03	04	05	06	07	08	09
1	01	11	12	13	14	15	16	17	18	19
2	02	12	22	23	24	25	26	27	28	29
3	03	13	23	33	34	35	36	37	38	39
4	04	14	24	34	44	45	46	47	48	49
5	05	15	25	35	45	55	56	57	58	59
6	06	16	26	36	46	56	66	67	68	69
7	07	17	27	37	47	57	67	77	78	79
8	08	18	28	38	48	58	68	78	88	89
9	09	19	29	39	49	59	69	79	89	99

PROC0(x, y) $\rightarrow x \oplus y$
Inputs: $x, y \in \{0,1,2,3,4,5,6,7,8,9\}$
Output: $x \oplus y$ *(2 digits)*

return $x \oplus y$ *from the table above*

PROC1(x, y) $\rightarrow x\ (x_1x_0) \oplus y$
Inputs: $x_1, x_0, y \in \{0,1,2,3,4,5,6,7,8,9\}$
Output: $x_1x_0 \oplus y$ *which has three digits* $z_2z_1z_0$, *where* $z_2, z_1, z_0 \in \{0,1,2,3,4,5,6,7,8,9\}$

$u'v' \leftarrow$ *PROC0*(x_0, y)
$z_0 \leftarrow v'$

$u''v'' \leftarrow$ *PROC0*(x_1, u')
$z_1 \leftarrow v''$ *and* $z_2 \leftarrow u''$

return $z_2z_1z_0$

a. [2 marks] What is the output of $9 \oplus 8 \oplus 7$ (the left $\oplus$ will be performed first)? Show the steps.

b. [3 marks] We could extend $x_1x_0 \oplus y$ to $x_{n-1}x_{n-2} \ldots x_1x_0 \oplus y$ (which gives $z_nz_{n-1} \ldots z_1z_0$) by performing similar steps as above. What is the result of $1 \oplus 8 \oplus 0 \oplus 8 \oplus 7$?

c. [5 marks] Write in pseudocode for $x_{n-1}x_{n-2} \ldots x_1x_0 \oplus y \rightarrow z_n, z_{n-1}, \ldots z_0$. Your pseudocode must make use of *PROC0()* and *PROC1()*.

Solution:

a. [2 marks] The result is 789.

b. [3 marks] The result is 01788.

c. [5 marks] A solution using only *PROC0()* below.

```
c ← y
for i ← 0 to n-1 do
      cz_i ← PROC0(x_i, c)
z_n ← c
```

A solution using both *PROC0()* and *PROC1():*

```
if n is even
  c ← y
  for i ← 0 to n/2 - 1 do
    cz_{2i+1}z_{2i} ← PROC1(x_{2i+1}x_{2i}, c)
else
  cz_o ← PROC0(x_o,y)
  for i ← 1 to (n-1)/2 do
    cz_{2i}z_{2i-1} ← PROC1(x_{2i}x_{2i-1}, c)

z_n ← c
```

2. (**Seeing the MCGW problem for the last time**) Previously we use 4-tuple [E/W, E/W, E/W, E/W] to represent the locations of the man, cabbage, goat and wolf. In this problem, we use a different data abstraction. We use two sets: one for the east and one for the west. For example, initially the two sets are {M, C, G, W} and {} (i.e., all of them are on the east and none on the west). The final state is simply reverse of the two sets.

   a. [5 marks] Write down all the set representations for the 10 legal states.

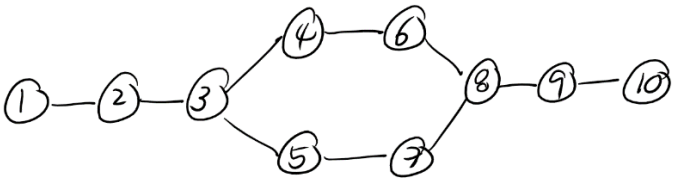
   b. [5 marks] By labeling the states by 1 to 10, draw the graph for this problem in which the nodes are the 10 states.

   Solution:

   a. [5 marks]

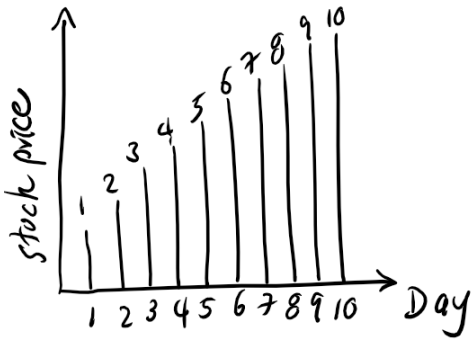
1. {M,C,G,W}{ }
2. {C,W}{M,G}
3. {M,C,W}{G}
4. {W}{M,C,G}
5. {C}{M,G,W}
6. {M,G,W}{C}
7. {M,C,G}{W}
8. {G}{M,C,W}
9. {M,G}{C,W}
10. { }{M,C,G,W}

   b. [5 marks]



3. (**How often is the stack empty?)** In this question we consider the improved algorithm for the stock price problem. We are interested in counting the times of having an empty stack after day 1 is pushed into the stack.

   a. [2 marks] Consider the price-day graph below. How many times the stack is empty in running *stockUp()* after pushing day 1 into the stack?



   b. [3 marks] Repeat (a) for the graph below.

   c. [5 marks] Design an algorithm in pseudocode that will count the number of times of encountering an empty stack when running *stockUp()* for *n*

days. The inputs are *n* and the list of *n* prices, and the output is the count.

Solution:
a. [2 marks] Day 2 will find an empty stack, so is day 3 and thereafter. Therefore, 9 times.
b. [3 marks] Day 4 and day 7 will find an empty stack. Therefore, 2 times.
c. [5 marks]

```
count = 0
maxDay = 1
for day ← 2 to n do
    if stock(day) ≥ stock(maxDay) then
        count += 1
        maxDay = day
return count
```

4. (**Playing with the stack**) Consider the pseudocode for a function called *eval()* which evaluates an arithmetic expression.

```
eval(aList) → a number
input: aList contains either operators (+, –, ×, and /) or operands (numbers or variables).
output: a number

create a stack called aStack
for item in aList do
    if item is an operand, then
        aStack.push(item)
    else
        opr2 = aStack.pop()
        opr1 = aStack.pop()
        result ← cal(item, opr1, opr2)
        aStack.push(result)

return aStack.pop()
```

For *cal(item, opr1, opr2)*, *item* could be "+", "−", "×", and "/". *cal()* evaluates "*opr1 item opr2*" and returns the value. For example, *cal(*"/", 4, 2) evaluates 4/2 and returns 2.0.

a. [2 marks] If *aList* = [1, 2, 3, "×", "+"], what would be the value returned by *eval(aList)*? Explain your answer.
b. [3 marks] If *aList* = [1, 2, "+", 3, "×", 4, "+"], what would be the value returned by *eval(aList)*? Explain your answer.
c. [5 marks] If *aList* = [6, 3, "/", 2, 4, "+", "×", 5, "/"], what would be the value returned by *eval(aList)*? Explain your answer.

Solution:
a. [2 marks] The result is 1 + (2 × 3) = 7.
b. [3 marks] The result is (1 + 2) × 3 + 4 = 13.
c. [5 marks] The result is 6 / 3 × (2 + 4) / 5 = 2.4.

5. (**What are the worse cases?**) Consider below *stockUp()* that returns both the up periods and the number of *while()* executed when *aStack* is not empty.

```
def stockUp(n,stock):
    up = n*[0]
    count = n*[0]
    aStack = stack.Stack()
    up[0] = 1
    aStack.push(0)

    for k in range(1,n):
        while not aStack.isEmpty() and stock[aStack.peek()] <= stock[k]:
            lastPopped = aStack.pop()
            count[lastPopped] += 1
        if aStack.isEmpty():
            up[k] = k+1
        else:
            count[aStack.peek()] += 1
            up[k] = k - aStack.peek()
        aStack.push(k)
```

```
    return up,count
```

Recall that there are three sources of comparisons for this program:

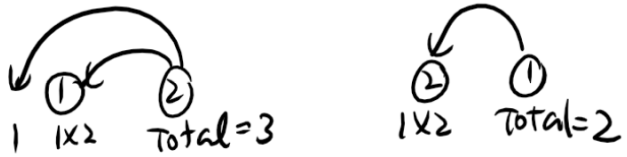
i. Checking the if-condition for each day except day 1, therefore $n-1$ comparisons in total.
ii. When *aStack* is empty, it takes 1 comparison.
iii. When *aStack* is not empty, it takes two comparisons.

For the worst case for (iii), we know that the two highest stock prices must be on the first and last days, and the total number of comparisons is $2 \times (2n - 3)$. In this question, we consider **the worst case for (ii) and (iii) together**. As before, all n stock prices are assumed distinct.

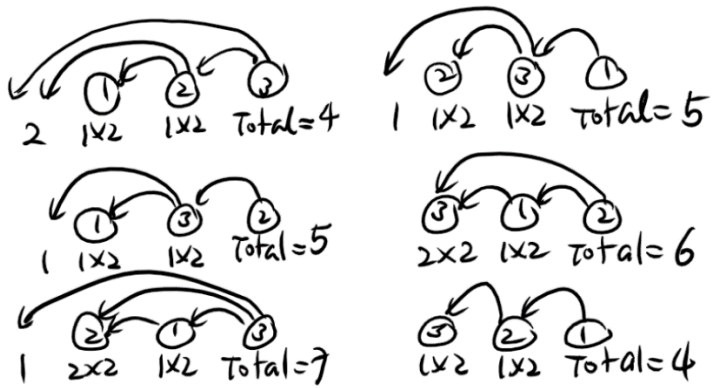
a. [2 mark2] For $n = 2$, there two permutations. What are the numbers of comparisons for these two cases? Which is the worst case?
b. [3 marks] For $n = 3$, there are six possible permutations of the three prices. Which one is the worst case?
c. [5 marks] By modifying your program in A9 or applying some sound argument, find out the worst-case total number of comparisons for $n > 1$?

Solution:

a. [2 marks] From the diagram below, (1,2) is the worst case.



b. [3 marks] From the diagram below, (2,1,3) is the worst case.



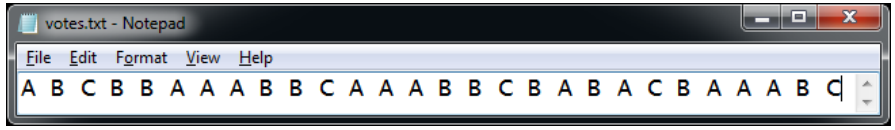
c. [5 marks] A worst case is when the second highest price is in day 1, the highest price is in day n, and the rest is ordered increasingly. Therefore, the total number is $2 \times (2n - 3) + 1 = 4n - 5$.
To prove this result, we introduce a virtual day 0 (that corresponds to the empty stack) that has the highest price among all days. Therefore, according to the nonempty stack case, these n+1 prices is a worst case, and the number of comparisons is $2 \times (2(n + 1) - 3) = 4n - 2$, assuming each case requires two comparisons. However, we need to subtract 2, because day 1 does not contribute any comparisons because it is first pushed into the stack. We also need to subtract 1 because there is only 1 comparison for day n. As a result, the total number of comparison is $4n - 2 - 2 - 1 = 4n - 5$.

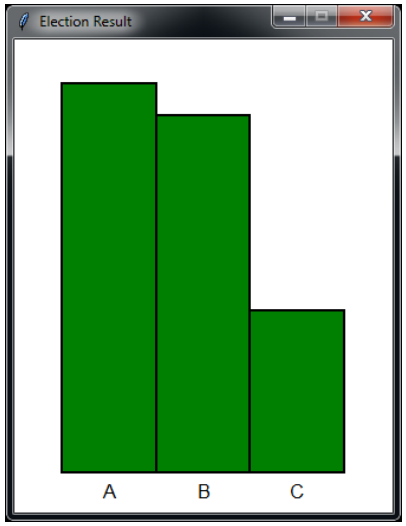
6. (**An ancient Chinese puzzle**) Write a program that uses **brute force approach** to solve a classic ancient Chinese puzzle: "We count 35 heads and 94 legs among the chickens and rabbits in a farm. How many rabbits and how many chickens do we have?" The program should display the number of rabbits and chickens respectively. If you obtain the answer by solving the two simultaneous equations, you will receive no marks.

Solution: In the USB

7. (**Counting the votes**) Suppose you work for the Electoral Affairs Commission, and you are going to write a program to count the votes of each district in District Council Elections and depict the result using a histogram. Write a program that first accepts a list of votes from the file "vote.txt", stored in the same folder of the program. Assume there are three candidates (i.e., A, B and C) only.

   Here are the sample input and output of the program:





   You may choose the dimension of the GUI window, as long as the histogram depicts the results clearly.

   Solution: In the USB

8. (**Floyd's triangle**) Floyd's triangle is a right-angled triangular array of positive integers. It is defined by filling the rows of the triangle with consecutive integers, starting with a 1 in the top left corner. For example,

```
1
2  3
4  5  6
```

Write a program that accepts the number of rows for the Floyd's triangle and prints out the triangle. Immediately below the Floyd's triangle prints an image of the triangle but the integers on each row are reversed. After that, print the sum of the integers of odd-numbered rows (i.e., 1st row, 3rd row, 5th row and so on). Below are the sample inputs and outputs:

```
>>> ================================ RESTART ================================
>>>
Enter number of rows: 3
1
2 3
4 5 6
6 5 4
3 2
1
The sum of integers in odd row(s) is 21
>>> main()
Enter number of rows: 6
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
16 17 18 19 20 21
21 20 19 18 17 16
15 14 13 12 11
10 9 8 7
6 5 4
3 2
1
The sum of integers in odd row(s) is 231
>>>
```

Solution: In the USB

9. (**An O(n log n) sorting algorithm**) The following program implements a sorting algorithm called merge sort. The only missing part is the implementation of the `merge()` function. Given two lists of data sorted in ascending order. The function returns a list which contains all the data items from the two lists and they are sorted in ascending order. For example, the two lists are [1, 4, 6, 10] and [5, 7, 8, 11, 12]. The list to be returned is [1, 4, 5, 6, 7, 8, 10, 11, 12]. Without using any functions from built-in or imported modules, other than the function `len()`, implement the function `merge()`.

```
def merge(lefthalf, righthalf):
    #
    # Implement your code here
    #

def mergesort(x):
    if len(x) == 0 or len(x) == 1:
        return x
    else:
        middle = int(len(x)/2)
        a = mergesort(x[:middle])
        b = mergesort(x[middle:])
        return merge(a,b)

def main():

    aList = [10, 5, 2, 9, 6, 3, 4, 8, 1, 7]
    print(mergesort(aList))

    list1 = [1, 4, 6, 10]
    list2 = [5, 7, 8, 11, 12]
    print(merge(list1, list2))

main()
```

Expected output of the program:

```
>>> ================================ RESTART ================================
>>>
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 4, 5, 6, 7, 8, 10, 11, 12]
>>>
```

Solution: In the USB

10. (**Let no errors escape!**) You are given a function below, where both `low` and `high` are integers. It prints all the integers between `low` and `high`, inclusive.

```
def printARange(low, high):
    for i in range(low, high + 1):
        print(str(i), end="")
        if i != high:
            print (", ", end="")
```

Enhance this function, so that it will raise two types of errors: (1) If `low` and `high` are not both integers, it will raise a `TypeError`, and (2) if `low` is greater than or equal to `high`, it will raise a `ValueError`.

Write also a `main()` that will use `try-except` blocks to test out all the cases below and displays the corresponding correct outputs/error messages:

```
printARange(1, 10)
printARange("x", 10)
printARange(1, "y")
printARange("a", "b")
printARange(5, 5)
printARange(6, 7)
```

The output of the program should look like below:

```
>>> ================================ RESTART ================================
>>>
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
unorderable types: str() >= int()
unorderable types: int() >= str()
Can't convert 'int' object to str implicitly
The low must not be greater than or equal to the high.
6, 7
>>>
```

Solution: In the USB