VINUNIVERSITY

Extra Lab Practice: Final Exam Review

COMP1010 Introduction to Programming **Week 15**

Extra Lab Practice Submission Instructions:

- This is optional extra lab practice for whom want extra practice at home.
- There is NO due date, and NO Canvas submission is needed.
- Use your Autograder CMS system to check your answer later. The test cases are available for Problem 1, Problem 2, and Problem 3. We will also publish all solutions on Canvas at the end of the day.
- You are welcome to discuss with your peers but do not copy code from your peers as it does not help you at all.

Problem 1 - Factorization

Write a program to input an integer number n, write an efficient function to print all prime factors of n. For example, if the input number is 12, then output should be “2 2 3”. And if the input number is 315, then output should be “3 3 5 7”.

- Solve this problem using `while-loop`
- Solve this problem using `recursion`

```
Input an integer: 12
12 = 2 2 3


Input an integer: 315
315 = 3 3 5 7

Input an integer: 12600
12600 = 2 2 2 3 3 5 5 7
```

Figure 1: Examples for Input/Output

For `while-loop`, following are the steps to find all prime factors.

1) While n is divisible by 2, print 2 and divide n by 2.
2) After step 1, n must be odd. Now start a loop from i = 3 to square root of n. While i divides n, print i and divide n by i, increment i by 2 and continue.
3) If n is a prime number and is greater than 2, then n will not become 1 by above two steps. So print n if it is greater than 2.

For `recursion`, find k the smallest divisor of n, then print k and recursion on n/k.

Sample Solution for Problem 1

Listing 1: Sample Solution for Problem 1.

```
1  import math
2
3  # A function to print all prime factors of
4  # a given number n
5  # Using while-loop method
6  def primeFactors(n):
7      # Print the number of two\'s that divide n
8      while n % 2 == 0:
```

```
 9          print(2, end=' ')
10          n = int(n / 2)
11
12      # n must be odd at this point
13      # so a skip of 2 ( i = i + 2) can be used
14      for i in range(3, int(math.sqrt(n)) + 1, 2):
15
16          # while i divides n , print i ad divide n
17          while n % i == 0:
18              print(i, end=' ')
19              n = int(n / i)
20
21              # Condition if n is a prime
22      # number greater than 2
23      if n > 2:
24          print(n)
25
26  # Recursive method
27  def primeFactorsRecur(n):
28      # Print the number of two\'s that divide n
29      if n % 2 == 0:
30          print(2, end=' ')
31          primeFactorsRecur(n // 2)
32          return
33
34      # n must be odd at this point
35      # so a skip of 2 ( i = i + 2) can be used
36      for i in range(3, int(math.sqrt(n)) + 1, 2):
37
38          # while i divides n , print i ad divide n
39          if n % i == 0:
40              print(i, end=' ')
41              primeFactorsRecur(n // i)
42              return
43
44              # Condition if n is a prime
45      # number greater than 2
46      if n > 2:
47          print(n, end=' ')
48
49
50  while True:
51      #n = 12600
52      n = int(input('\nInput an integer: '))
53      print(n, end=' = ')
54      primeFactorsRecur(n)
55      print()
```

```
56      break
```

Problem 2 - Interleaving

Find all interleavings of given strings that can be formed from all the characters of first and second string where order of characters is preserved. See example of input, output below.

```
Enter the first string: 12
Enter the second string: ABC
All possible interleavings:
1A2BC AB12C AB1C2 1ABC2 A1B2C A1BC2 12ABC ABC12 1AB2C A12BC
```

Figure 2: Examples for Input/Output

Hint: We can easily solve this problem using recursion. The idea is to append first or last character of X and Y in the result one by one and recur for remaining substring.

Listing 2: Problem 2 Solution.

```
1
2  Input: X[1..m], Y[1..n]
3  fun(X, m, Y, n) = fun(X, m-1, Y, n) + X[m]
4                    fun(X, m, Y, n-1) + Y[n]
5  fun(X, 0, Y, n) = Y[n]     // X is empty
6  fun(X, m, Y, 0) = X[m]     // Y is empty
```

Note: See the sample output below (and convince yourself about the output).

Sample Solution for Problem 2

Listing 3: Sample Solution for Problem 2.

```
1
2  def findInterleavings(X, Y, result=set(), curr=''):
3      # insert curr into set if we have reached end of both Strings
4      if not X and not Y:
5          result.add(curr)
6          #print(curr)
7          return result
8
9      # if X is empty, append its first character not in the
10     # result and recur for remaining substring
11     if X:
12         result = findInterleavings(X[1:], Y, result, curr + X[0])
13
14     # if Y is empty, append its first character not in the
15     # result and recur for remaining substring
16     if Y:
17         result = findInterleavings(X, Y[1:], result, curr + Y[0])
18
19     return result
20
21 while True:
22     X = input('Enter the first string: ')
23     Y = input('Enter the second string: ')
24
25     # use set to handle duplicates
26     result = findInterleavings(X, Y)
27     print('All possible interleavings:')
28     for k in result:
29         print(k, end=' ')
30     print()
31     break
```

Problem 3

Prompt user to enter two strings. Use for-loop (or recursion) to find the longest common substring. See the following example:

```
Enter the first string: ABCDFGH
Enter the second string: XBCDYZ
The longest common substring: BCD
Enter the first string: ABCD
Enter the second string: EFG
There is no common substring!
```

Listing 4: Problem 3 Sample Solution.

```
# Problem 3 Sample Solution
def print_lcsub(X, Y):
    for m in reversed(range(1, len(X))):
        for k in range(len(X)-m):
            if Y.find(X[k:k+m]) != -1:
                print(f'The longest common substring: {X[k:k+m]}')
                #print(k, m)
                return
    print('There is no common substring!')

X = input('Enter the first string: ')
Y = input('Enter the second string: ')

print_lcsub(X, Y)
```

Problem 4

Prompt user to enter a multi-level nested list of number. Store this input as a string. Using `json.loads()` to convert this string into a multi-level nested list. Finally, use `recursion` method to find the sum of all elements in this nested list. See the following example:

```
Enter a multi-level nested list: [1, 2, [3, 4], 5]
The sum of all elements in [1, 2, [3, 4], 5] is 15

Enter a multi-level nested list: [1, 2, [3, 4, [1, 2]], 5]
The sum of all elements in [1, 2, [3, 4, [1, 2]], 5] is 18
```

Listing 5: Problem 4 Sample Solution.

```
# Problem 4 Sample Solution
import json

def list_sum(data_list):
    total = 0
    for element in data_list:
        if isinstance(element, list):
            total = total + list_sum(element)
        else:
            total = total + element

    return total

in_str = input('\nEnter a multi-level nested list: ')

# Converting string to list
in_lst = json.loads(in_str)

s = list_sum(in_lst)
print(f'The sum of all elements in {in_str} is {s}')
```

Problem 5

First, create a class named `Family` with the following properties:

(i) A class variable `count` and set to 0 by default.

(ii) A constructor that accepts three parameters (`son_name`, `father_name`, and `mother_name`). Also, increase the class variable `count` whenever a `Family` object is created.

(iii) Convert Python object into string by using `__str__` to output the family number as show below (Hint: use the class variable `count`).

Next, create classes `Father` and `Mother` which inherit from the class `Family` with class method `show_father` and `show_mother`, respectively. The output format are shown in the example below.

Lastly, create a class `Son` that inherits both classes `Father` and `Mother`. This class has the following properties:

(i) A constructor that accepts 06 parameters (`son_name, son_age, father_name, father_age, mother_name, mother_age`). Invoke the base class constructor and set the attributes for `son_age`, `father_age`, and `mother_age`.

(ii) A class method `show_parent` which calls `show_father` and `show_mother` to show the parents info (see example below).

(iii) A class method `show_son` to show son's name and age (see example below).

Prompt user to enter (i) the number of families, and (ii) information for each family: son_name, son_age, father_name, father_age, mother_name, mother_age, separated by comma (','). Then, create `Son` instance for each family (assuming there is one son in each family), and printout family number info, and call `show_son` and `show_parent` to output the info as shown in the following sample output:

```
Enter number of families: 2
Enter Family Info (son_name,son_age,father_name,father_age,mother_name,mother_age):
   Tu, 1, Ali Baba, 60, Micky Mimi, 20
--------------Family No.1 Info--------------
Son's Name: Tu (1-year-old)
Father's Name: Ali Baba (60-year-old)
Mother's Name: Micky Mimi (20-year-old)

Enter Family Info (son_name,son_age,father_name,father_age,mother_name,mother_age):
   Wong, 50, Harry Potter, 30, Janet Wewe, 80
--------------Family No.2 Info--------------
Son's Name: Wong (50-year-old)
Father's Name: Harry Potter (30-year-old)
Mother's Name: Janet Wewe (80-year-old)
```

Problem 6 – Numpy

Given the matrix below:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

(a) Create transpose of A using Numpy array.

(b) Find the smallest element in A.

(c) Find the maximum element in each column of a matrix A.

Listing 6: Problem 6 Sample Solution.

```
1  import numpy as np
2
3  _a = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
4
5  a = np.array(_a)
6  # Create transpose of A
7  a_t = np.transpose(a)
8
9  print(a_t)
10 # Find the smallest element in A
11 print(np.min(a))
12
13 # Find the maximum element in each column of a matrix A.
14 print(np.max(a, axis=0)) # max of each column
```