Recursion 2

Advanced Programming | Lab | 19CLC4-8

Objective

Practice recursion in C/C++

Regulations

- This is individual work.
- **You must do by yourself first.**
- If you're stuck, request help from your friends, tutors or Google.
    - You only ask for some guide, explanation, …, not for source code.
    - Do it again by yourself.
- Submit your works to Moodle.

About the exercises

- The exercises in this document are collected from some resources on the internet and eBooks. The detail information is listed in Reference. (Note that, the references are just written in the short text, not following any format.)
- Tutors added some hints or guides to help you better understand.

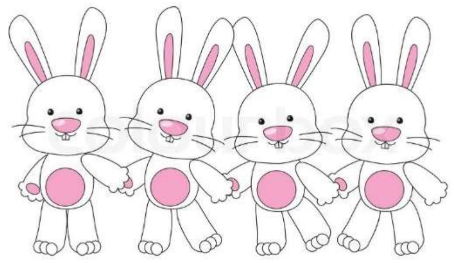
Tutors' Contact

1. Trương Tấn Khoa (truongtankhoa1190@gmail.com)
2. Lê Ngọc Thành (lnthanh@fit.hcmus.edu.vn)

Exercises

EX2.1

- We have bunnies standing in a line, numbered 1, 2, ...
    - The odd bunnies (1, 3, ...) have the normal 2 ears.
    - The even bunnies (2, 4, ...) we'll say have 3 ears, because they each have a raised foot.
- Recursively return the number of "ears" in the bunny line 1, 2, ... n (without loops or multiplication).
- For example:
    - bunnyEars2(0) → 0
    - bunnyEars2(1) → 2
    - bunnyEars2(2) → 5



EX2.2

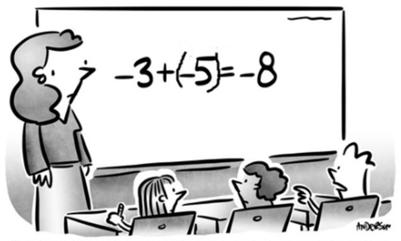
- Given a string, compute recursively the number of times lowercase "hi" appears in the string, however do not count "hi" that have an 'x' immedately before them.
- For example:
  - countHi2("ahixhi") → 1
  - countHi2("ahibhi") → 2
  - countHi2("xhixhi") → 0

EX2.3

- We have triangle made of rectangle blocks.
  - The topmost row has 1 block, the next row down has 2 blocks, the next row has 3 blocks, and so on.
- Compute recursively (no loops or multiplication) the total number of blocks in such a triangle with the given number of rows.
- For example:
  - triangle(0) → 0
  - triangle(1) → 1
  - triangle(2) → 3

EX2.4

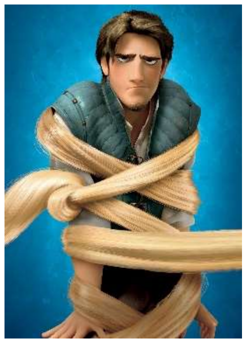
- Given a string that contains a single pair of parentheses, compute recursively a new string made of only of the parenthesis and their contents, so "xyz(abc)123" yields "(abc)".
- For example:
  - parenBit("xyz(abc)123") → "(abc)"
  - parenBit("x(hello)") → "(hello)"
  - parenBit("(xy)1") → "(xy)"



"So by adding them together we ended up with less than we started with? See, this is why people hate math."

EX2.5

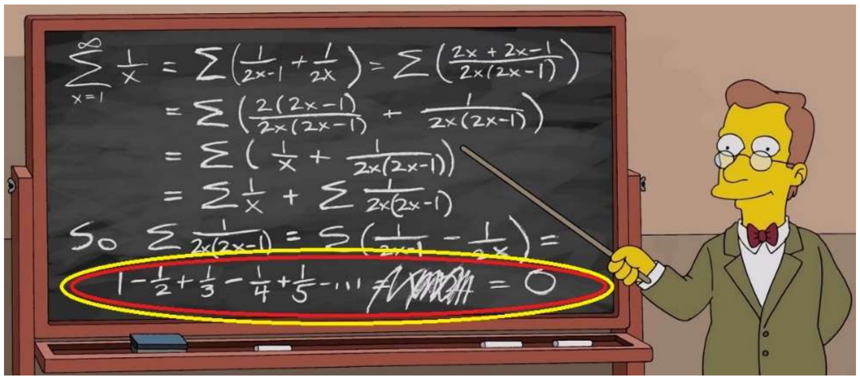
- Given a string and a non-empty substring ***sub***, compute recursively the largest substring which starts and ends with ***sub*** and return its length.
- For example:
  - strDist("catcowcat", "cat") → 9
  - strDist("catcowcat", "cow") → 3
  - strDist("cccatcowcatxx", "cat") → 9



EX2.6

- Given an array of ints, is it possible to choose a group of some of the ints, such that the group sums to the given target?
    - The caller can specify the whole array simply by passing start as 0.
    - No loops are needed -- the recursive calls progress down the array.
- For example:
    - groupSum(0, [2, 4, 8], 10) → true
    - groupSum(0, [2, 4, 8], 14) → true
    - groupSum(0, [2, 4, 8], 9) → false

boolean **groupSum**(int start, int[] nums, int target);



EX2.7

- Given an array of ints, is it possible to divide the ints into two groups, so that the sums of the two groups are the same.
    - Every int must be in one group or the other.
- Write a recursive *helper method* that takes whatever arguments you like, and make the initial call to your recursive helper from splitArray(). (No loops needed.)
- For example:
    - splitArray([2, 2]) → true
    - splitArray([2, 3]) → false
    - splitArray([5, 2, 3]) → true

EX2.8

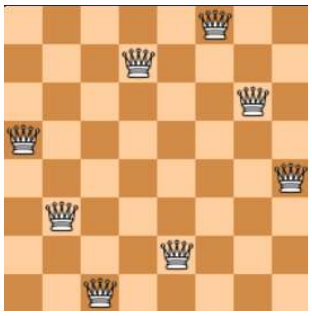
- Given following three values, the task is to find the ***total number of maximum chocolates*** you can eat.

    - money: Money you have to buy chocolates
    - price: Price of a chocolate
    - wrap: Number of wrappers to be returned for getting one extra chocolate.
- It may be assumed that all given values are positive integers and greater than 1.
- You can write a *helper method* that takes whatever arguments you like.
- No loops are needed.
- For example:
    - Input 1: money = 16, price = 2, wrap = 2
        - Output 1: 15
        - Explain: Price of a chocolate is 2. You can buy 8 chocolates from amount 16. You can return 8 wrappers back and get 4 more chocolates. Then you can return 4 wrappers and get 2 more chocolates. Finally, you can return 2 wrappers to get 1 more chocolate.

    - Input 2: money = 15, price = 1, wrap = 3
        - Output 2: 22
        - Explain: We buy and eat 15 chocolates. We return 15 wrappers and get 5 more chocolates. We return 3 wrappers, get 1 chocolate and eat it (keep 2 wrappers). Now we have 3 wrappers. Return 3 and get 1 more chocolate. So total chocolates = 15 + 5 + 1 + 1
    - Input 3: money = 20, price = 3, wrap = 5
        - Output 3: 7

EX2.9* (NOT REQUIRED)

- The N queens puzzle is the problem of placing N chess queens on an N×N chessboard so that no two queens threaten each other.
    - Thus, a solution requires that no two queens share the same row, column, or diagonal.
    - You can use loop without nested.
- Display result on Console screen to show queens' place. This function don't need recursive.



Extend: Because this exercise have many solutions, you try to think how to find other solutions.

References

[1] https://codingbat.com/java
[2] https://www.geeksforgeeks.org/recursion-practice-problems-solutions/