**Recursion**

# Section 1: Simple problems

1. Write a recursive function that prints string (passed as parameter) in reverse.
2. Write a recursive function to detect if a parameter string is palindrome or not. Two pointers are parameters to this function. The first one points to the beginning of the string. The second one points to the last non-null character in the string.
3. Write a recursive function to calculate the number of set bits in a integer that is passed as a parameter to the function.
4. Write a recursive function to print the steps to solve the tower of Hanoi problem. Use appropriate parameters.
5. Write a recursive function to print only the ith step of the tower of Hanoi problem. Use appropriate parameters.
6. Write a recursive function int FibSum(int n) that returns the sum of first *n* Fibonacci numbers.

# Section 2: Hard problems

**Problem# 1**

**Brackets Matching**

In this problem, you need to find whether an input string of brackets is error free or not. For a string of brackets to be error free, each opening bracket must have a matching closing bracket of same type and there wouldn't be any extra unmatched closing brackets. Y**ou need to solve this problem using recursion.**

The input contains a string *S* consisting of symbols from the set {'(', '{', '[',')', '}', ']'}. If *S* is error free, then print “Matched” in a line. Otherwise report the first position (from left) where error occurs.

|  |  |
| --- | --- |
| **Sample Input(s)** | **Corresponding Output(s)** |
| {()[]}([])  (){}[[]]  {(})  ([])}  ({[])}  )(  ({}(([]{})[]){()})  ({}(([]{)}[]){()})  ({}(([] | Matched  Matched  Error at position 3  Error at position 5  Matched  Error at position 5  Error at position 1  Matched  Error at position 9  Error at position 8 |

**Problem #2**

**Count Valid Parenthesization**

An integer, *n*, is given as input. Output the number of valid parenthesizations that can be formed by using *n* pair of parentheses. For example, 5 valid parenthesizations can be formed using 3 pair of parentheses; namely “((()))”, “(()())”, “(())()”, “()(())” and “()()()”.

**Do not use arrays for this task.** Use a recursive method with the following prototype:

int validParenthesizationCount(int n)

|  |  |
| --- | --- |
| **Sample Input(s)** | **Corresponding Output(s)** |
| 1  2  3  4  6  8 | 1  2  5  14  132  1430 |

**Problem #3**

**Count Binary String**

A binary string is a string containing only two characters: ‘0’ and ‘1’. For this problem, you are given an integer, *n*, as input. Output the number of binary strings of length *n*, that do not contain three or more consecutive 1’s. For example, there are 16 binary strings of length 4. Three of them, namely “0111”, “1110” and “1111” contain three or more consecutive 1’s. So there are 13 binary string of length 4, that do not contain three or more consecutive 1’s.

**Arrays cannot be used for this task.** Use a recursive method with the following prototype:

int binaryStringCount(int n)

|  |  |
| --- | --- |
| **Sample Input(s)** | **Corresponding Output(s)** |
| 1  2  3  4  6  8 | 2  4  7  13  44  149 |

**Problem #4**

**Count Bacteria Population**

A certain species of bacteria reproduces according to the following rules:

* A bacteria born in day *n* does not give birth to new bacteria in day *n* and day *n+1.*
* A bacteria born in day *n* gives birth to 1 bacteria in day *n+2.*
* A bacteria born in day *n* gives birth to 2 bacteria in day *n+k,* for each *k>2*, i.e., a bacteria born in day *n* gives birth to 2 bacteria from the *n+3rd* day onwards.
* A bacteria does not die.

Let *Pn*denote the population of bacteria on day *n*. It is provided that *P0=0* and *P1=1*. In this problem, *n* is given as input, you have to output *Pn*. For example, *P2=1*, because the bacteria born in day 1 does not replicate in day 2. In day 3, a bacteria is born of the 1st bacteria. So, *P3=2*. In day 4, 2 bacteria are born of the 1st bacteria and none of the 2nd. So, *P4=4*. **Arrays cannot be used for this task.** Use a recursive method with the following prototype:

int bacteriaCount(int n)

|  |  |
| --- | --- |
| **Sample Input(s)** | **Corresponding Output(s)** |
| 1  2  3  4  5  6  8  10 | 1  1  2  4  7  13  44  149 |

**Problem# 5**

**Permutations**

Given integer input *n*, print all permutations of the first *n* letter of English alphabet (small case). The permutations should be printed in lexicographical order.

**Restrictions: One way to solve this problem is to generate all the permutations in no particular order, then store them in an array and then sort the array. But this approach is not allowed in this task. Instead, formulate a recursive solution such that the permutations get generated in lexicographical order. As soon as a permutation is generated, print that out.**

|  |  |
| --- | --- |
| **Sample Input(s)** | **Corresponding Output(s)** |
| 3 | abc  acb  bac  bca  cab  cba |

**Problem #6**

**Subsequence**

A subsequence is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements. For example, <a, b, d> is a subsequence of <a, b, c, d, e, f>. Also, an empty sequence is a subsequence of any other sequence. Given 2 input strings *S* and *S'*, you have to determine whether *S'* is a subsequence of *S*. Implement a recursive method called *IsSubsequence(S, S')* that returns non-zero (true) if *S'* is a subsequence of *S*; otherwise it returns 0 (false). **Global memory, static variables and loops cannot be used in this task**. Design parameters of your function appropriately and write necessary *main*() function.

The input contains *S* in line 1 and *S'* in line 2. In the output, print “YES” if S' is a subsequence of S; “NO” otherwise.

|  |  |
| --- | --- |
| **Sample Input(s)** | **Corresponding Output(s)** |
| abababa  abaa | YES |
| abcdefg | YES |
| pqr | NO |
|  | YES |
| saifur rahman  siam | YES |
| programming contest  practice | NO |