Slot **2** – Question **2 [Pizzeria]** Marks 20

Gist: It is a rephrased version of the parentheses balancing problem. There cannot be a closing parenthesis unless there is a matching open parenthesis. The ‘(‘ is replaced by P here and ‘)’ is replaced by O.

Tasks:

- Print the correct recursion for balanced parentheses.
- Algorithm:
  Keep track of counts of open and close brackets.
  1. Initialize these counts as 0.
  2. Recursively call the _printParenthesis() function until open bracket count is less than the given n.
     1. If open bracket count becomes more than the close bracket count, then put a closing bracket and recursively call for the remaining brackets.
     2. If open bracket count is less than n, then put an opening bracket and call _printParenthesis() for the remaining brackets.

Grading Scheme: MANUAL

ANY FORM OF HARD-CODING ATTRACTS FULL PENALTY.

[4 Marks]: Correct recursion idea for step 2.1

[4 Marks]: Correct recursion idea for step 2.2

[2 Marks] : Correct termination condition.

======= GOLD CODE ================

```
#include<stdio.h>
#include<stdlib.h>

#define LP 'P'
#define RP 'O'

void _printParenthesis(int pos, int n, int open, int close);
static char *str;

/* Wrapper over _printParenthesis()*/
void printParenthesis(int n)
{
  if(n > 0)
     _printParenthesis(0, n, 0, 0);
  return;
}

void _printParenthesis(int pos, int n, int open, int close)
{
  if(close == n)
  {
    printf("%s \n", str);
    return;
  }
  else
  {
    if(open > close) {
        str[pos] = RP;
        _printParenthesis(pos+1, n, open, close+1);
    }
    if(open < n) {
       str[pos] = LP;
       _printParenthesis(pos+1, n, open+1, close);
    }
  }
}

/* driver program to test above functions */
int main()
{
  int n;
  scanf("%d", &n);
  // we need a str of size 2n+1 to store null char.
  // using calloc guarantees that 2n+1-th char is \0.

  str = (char*) calloc(2*n+1, sizeof(char));
  printParenthesis(n);
  return 0;
}
```