# Name : Shrinivas Hatyalikar

# Div: TY-B (B2)

# Roll No: 26

**Q) Valid parenthesization**

Input: n (indicates the number of pairs of left and right brackets)  
  
Output: generate and print all valid parenthesizations of n pais or brackets.  
  
Example:  
n = 1 --> ()  
n = 2 --> ()(), (())  
n = 3 --> ((())), ()(()), (())(), (()()), ()()()

**Approach:**

Some observations that can be made in the above problem are as follows:-

* The first index i.e ind = 0 is always "("
* The last index i.e ind = n-1 is always ")"
* For each step we must ensure that "(" is always less than ")".This is the most missed out part.
* For each recursion step before adding ")" we must ensure that (open < close) also we ensure (open > 0) before adding "(" and (close > 0) before adding ")".

**Code:**

#include <stdio.h>

#include <stdlib.h>

void generateParenthesisRecursive(char\* current, int open, int close, int n) {

    if (open == n && close == n) {

        current[open + close] = '\0';

        printf("%s\n", current);

        return;

    }

    if (open < n) {

        current[open + close] = '(';

        generateParenthesisRecursive(current, open + 1, close, n);

    }

    if (close < open) {

        current[open + close] = ')';

        generateParenthesisRecursive(current, open, close + 1, n);

    }

}

void generateParenthesis(int n) {

    if (n <= 0) {

        return;

    }

    char\* current = (char\*)malloc(2 \* n \* sizeof(char));

    generateParenthesisRecursive(current, 0, 0, n);

    free(current);

}

int main() {

    int n;

    printf("Enter the number of pairs of brackets (n): ");

    scanf("%d", &n);

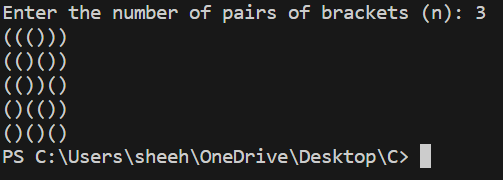
    generateParenthesis(n);

    return 0;

}

**Output:**

****

****