* **Generate Paranthesis:** [Imp Ques]

Input : n → pairs of paranthesis ()

i.e n → opening brackets,
n → closing brackets.

we have to generate all possible combinations.

i.e n = 1, (→ 1 ob
) → 1 cb.

possible combinations are ["()"] i.e 1.

)(→ is invalid

n = 2 (→ 2 ob
) → 2 cb

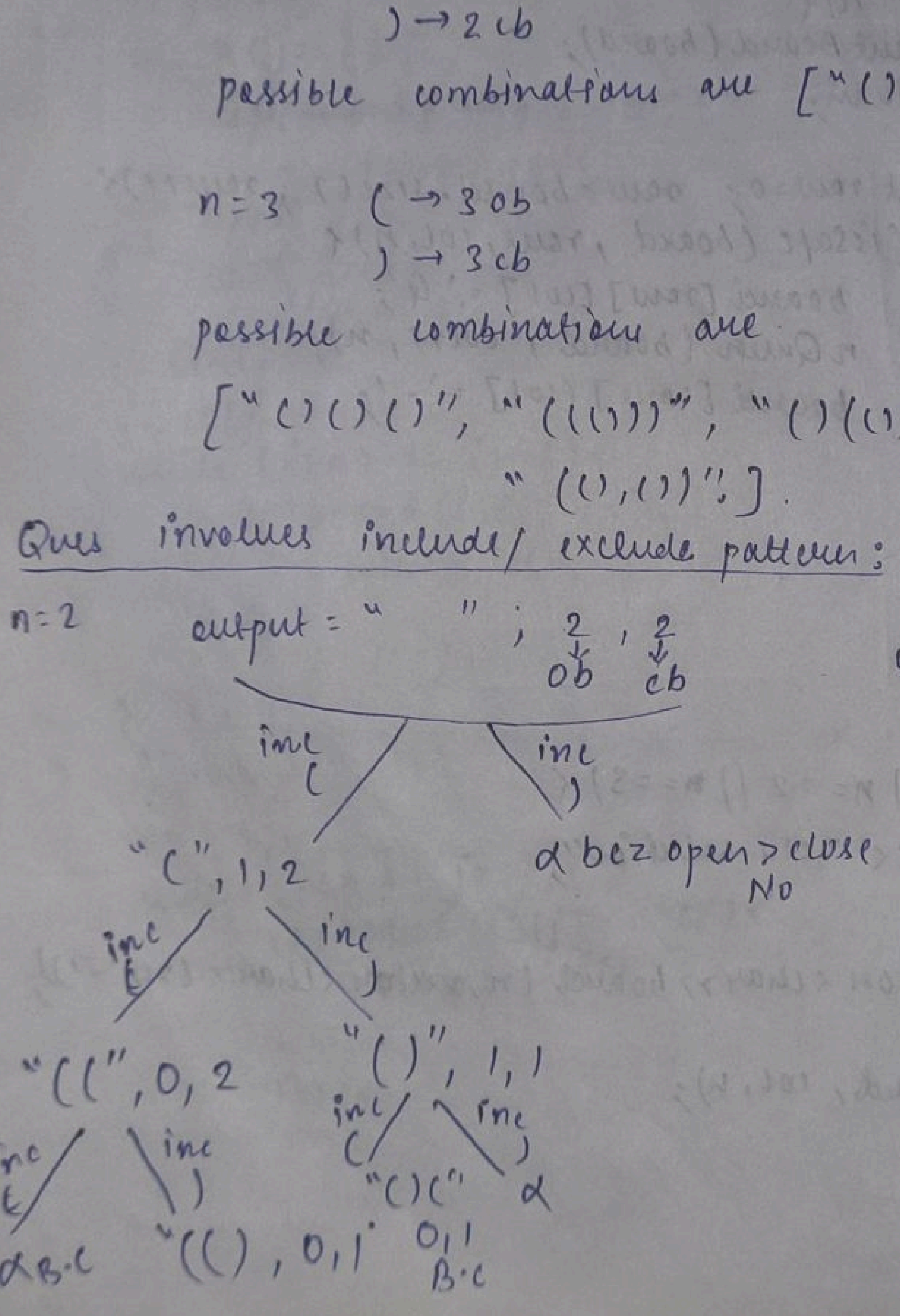
possible combinations are ["()()", "(())"] i.e 2

n = 3 (→ 3 ob
) → 3 cb

possible combinations are

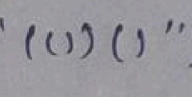
["()()()", "((()))", "()(())", "(())()",
"(()())"].

Ques involves include/ exclude pattern:



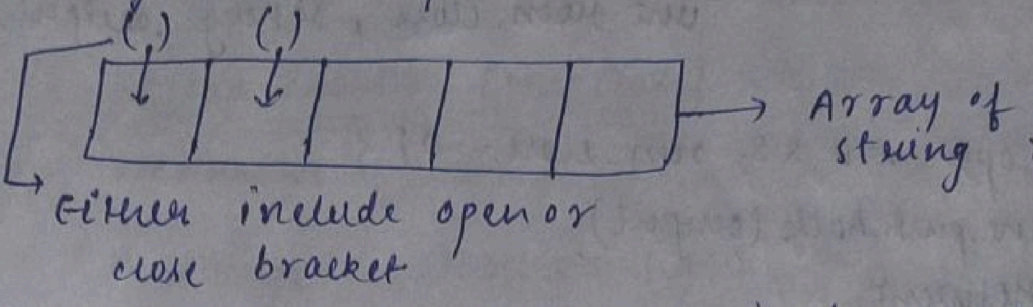
we can include closing bracket if we have opening bracket for it in the left side.

i.e



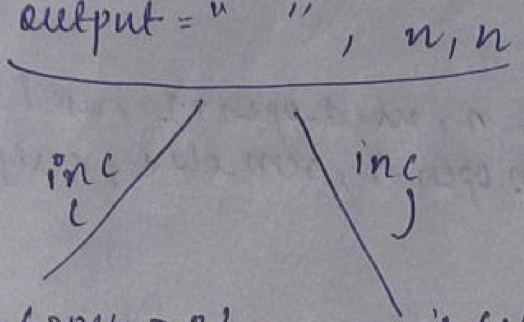
open count > close count.

we can observe that for n → open bracket & n → closing brackets generates 2^n-1 combinations.



This mean we will use inc/exc pattern

→

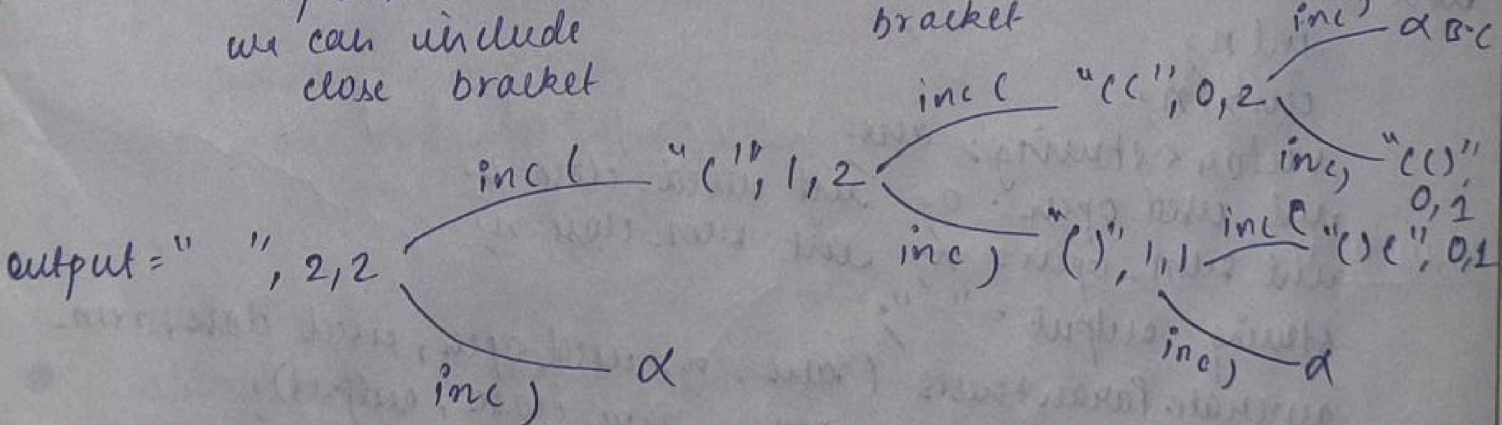


if (open>0)
agar apke pass opening brackets nai then you will include

if (close>0)
we can't use this condition because including or starting parantheses with) brackets is invalid.

ex: (()()()
open=4
closed=3
open> closed
we can include close bracket

(()())
open=3
close=3
we can't include close bracket



* Code

```
void generateParantheses(vector<string> &ans , int n , int
                         used-open , int used-close , int rem-open,
                         int rem-close , string output){
    //bc
    if(rem-open==0 && rem-close==0){
        ans.push-back(output);
        return;
    }
    if(rem-open>0){
        output.push-back('(');
        generateParantheses(ans, n, used-open+1, used-close,
                            rem-open-1, rem-close, output);
        output.pop-back();
    }
    if(used-open > used-close){
        output.push-back(')');
        generateParantheses(ans, n, used-open, used-close+1,
                            rem-open, rem-close-1, output);
        output.pop-back();

    }
}
int main(){
    int n;
    cin>>n;
    vector<string> ans;
    int used-open=0; int used-close=0;
    int rem-open=n; int rem-close=n;
    string output="";
    generateParantheses(ans, n, used-open, used-close, rem-
                        open, rem-close, output);

    for(int i=0; i<ans.size(); i++){
        cout<<ans[i]<<" ";

    }
}
```

output : 3
((())) , (()()) · (())() · ()(()) · ()()() 5 combination

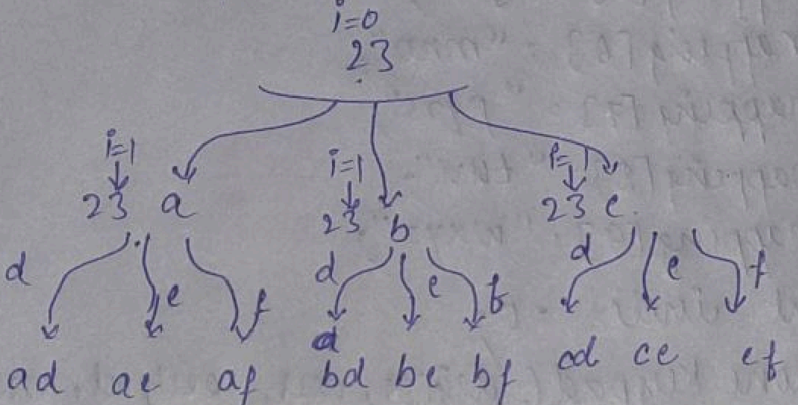
≡*. Phone - keypad Problem [Imp Ques]
[Letter combination of Phone]

Input : strings - containing digits from 2 to 9.

Output : all possible letter combination. that the number can be formed through

2 →	"abc"
3 →	"def"
4 →	"ghi"
5 →	"jkl"
6 →	"mno"
7 →	"pqrs".
8 →	"tuv"
9 →	"wxyz".

input = "23"



Code

```
void phoneKeyPad( string digits, vector<string> &ans , string
                  output, vector <string> mapping , int index){

   if (index >= digits. length()) {
       ans. push_back (output);
       return;
   }
   int digits = digits [index] - '0'.
   string value = mapping [digit].
   for ( int i=0 . i< value. length(); i++){
          char ch = value [i].
          output. push-back (ch).
          phone Keypad ( digits, ans, output, mapping, index+1).
          output. pop_back ().
```