








































Practice Arena

Practice problems aimed to improve your coding skills.

-  PRACTICE-02_SCAN-PRINT
-  PRACTICE-03_TYPES
-  LAB-PRAC-02_SCAN-PRINT
-  LAB-PRAC-01
-  PRACTICE-04_COND
-  BONUS-PRAC-02
-  LAB-PRAC-03_TYPES
-  PRACTICE-05_COND-LOOPS
-  LAB-PRAC-04_COND
-  LAB-PRAC-05_CONDLLOOPS
-  PRACTICE-07_LOOPS-ARR
-  LAB-PRAC-06_LOOPS
-  LAB-PRAC-07_LOOPS-ARR
-  LABEXAM-PRAC-01_MIDSEM
-  PRACTICE-09_PTR-MAT
-  LAB-PRAC-08_ARR-STR
-  PRACTICE-10_MAT-FUN
-  LAB-PRAC-09_PTR-MAT
-  LAB-PRAC-10_MAT-FUN
-  PRACTICE-11_FUN-PTR
-  LAB-PRAC-11_FUN-PTR
-  LAB-PRAC-12_FUN-STRUC
 -  Point Pairing Party
 -  Verify the family tree of Mr C
 -  Simple Sudoku
 -  The Family Tree of Mr C Part Three
 -  The Post offices of KRville
 -  Matrix Mandala
 -  Mango Mania
 -  Recover the Rectangle
 -  Crazy for Candy
 -  A Brutal Cipher Called Brutus
 -  Triangle Tangle
 -  Basic Balanced Bracketing
-  LABEXAM-PRAC-02_ENDSEM
-  LAB-PRAC-13_STRUC-NUM
-  LAB-PRAC-14_SORT-MISC

Basic Balanced Bracketing

LAB-PRAC-12_FUN-STRUC

Basic Balanced Bracketing [20 marks]

Problem Statement

Brackets are essential to coding. We use brackets to indicate blocks of statements, for instance, in loops, conditional statements, functions etc. However it is very important to have *balanced bracketing*. This means that whenever an opening bracket (is introduced, its sister bracket) must also be introduced sometime later. Thus, the following are examples of illegal bracketing (there are more open brackets than closing brackets)

```
(
((
(((
```

However, the reverse is also true. Every closing bracket) must have her sister bracket specified somewhere. Thus, the following are also examples of illegal bracketing (there are more closing brackets than opening brackets)

```
)
))
)))
```

Moreover, a closing bracket can come only after her sister opening bracket has already come. Thus, in no point in the string, when read from left to right, should we have more closing brackets than opening brackets. This has to hold even if the total number of opening and closing brackets in the overall string is the same. Thus, the following are also examples of illegal bracketing (even though total number of opening brackets is the same as the total number of closing brackets, some of the closing brackets come before their sister opening brackets have come)

```
) (
) ( (
) ( (
```

Any bracketing that does not violate the above rules i.e. has the same number of opening and closing bracket, and makes sure that at no point have we closed more brackets than opened brackets, is a valid bracketing. Examples include

```
((()))
()()()
()()()
()(())
```

In the input you will be given a number n. You have to print all possibly ways of performing valid bracketing using n opening brackets and n closing brackets. Print each valid way in a separate line. You have to print the valid bracketings in lexicographically increasing order. For this, we will consider that the opening bracket (has a "smaller" value than the closing bracket). Say we assign the opening bracket a value of 1 and the closing bracket a value of 2. Then the valid bracketings mentioned above would look like

```
((())) 111222
()()() 121212
()()() 112212
()(()) 121122
```

However, it is clear that these have not been specified in increasing order. The correct order would have been

```
((())) 111222
()()() 112212
()(()) 121122
```

((())) 121212

Caution

1. You only have to output the bracketings - do not output the number equivalents.
2. We will not penalize you for extra newlines at the end of your output. However, do not have extra spaces at the end of any line of the output.
3. Make sure that you output the bracketings in correct order. If you output all possible bracketings but in incorrect order then the autograder may give you zero marks since partial marks are given only when the correct line is output at the correct location.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS: You may want to solve this problem using recursion. Define a function of the form
void generateBalanced(char *str, int n, int pos, int open, int close)

1. str: a string in which we will write the bracketing
2. n: tells us how many opening and closing brackets are needed in total
3. pos: tells us which is the next position in the string where characters have to be written
4. open: tells us how many opening brackets have we written so far
5. close: tells us how many closing brackets have we written so far

Make use of the variables open and close to make sure that you never have more closing brackets than open brackets at any point in the string. The recursion can be started off by calling the function as

generateBalanced(str, n, 0, 0, 0);

EXAMPLE:

INPUT

3

OUTPUT:

((()))

((()())

((())())

()((()())

()()())

Grading Scheme:

Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you

have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

 **Start Solving!** (</editor/practice/6241>)