








































# 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
  -  Name the Clones
  -  The Race of the Clones
  -  Partial Palindrome
  -  Growth Curve
  -  The Family Tree of Mr C
  -  Timely Tasks
  -  Plenty of Palindromes
  -  Count and Say Sequence
  -  Orbiting Indices
  -  Zig-zag Numbers
  -  Parent Palindrome
  -  Leaderboard
-  LAB-PRAC-12\_FUN-STRUC
-  LABEXAM-PRAC-02\_ENDSEM
-  LAB-PRAC-13\_STRUC-NUM
-  LAB-PRAC-14\_SORT-MISC

# Name the Clones

LAB-PRAC-11\_FUN-PTR

## Name the Clones [20 marks]

---

### Problem Statement

Mr C has a large number of clones that come to life when you call functions. He wants to give these clones, names for which he wants your help. In the input, you will be given two strictly positive integers  $n$  and  $k$ , separated by a space. You will have to generate names for the clones of Mr C. These names should all contain exactly  $k$  characters and all of these characters must be from the first  $n$  lower-case English alphabet characters (no upper-case characters, spaces or punctuation marks allowed). However, the allowed characters can repeat any number of times in a name.

Mr C has requested that the names that you suggest be in *lexicographically* increasing order. We explain lexicographic ordering below. Thus, you have to first output the lexicographically smallest string of length  $k$  you can construct out of the first  $n$  lower-case letters of the English alphabet. Then find the lexicographically next smallest string of length  $k$  and so on till you have generated all such names possible.

### Lexicographic ordering

Just as given two digit sequences, say 1923 and 3122, we can say which digit sequence is "smaller" and which is "larger" (by interpreting the digit sequences as numbers), two sequences of alphabets, i.e. strings, can also be compared and given two sequences of alphabets, we can say which one is smaller and which one is larger.

The rules of doing so are pretty simple if the two strings are of the same length, which will be the case in this question. We first declare that the character 'a' is smaller than the character 'b', which is in turn smaller than the character 'c' and so on. To compare two strings, simply look at their first characters - the string with the larger character wins. If both strings have the same first character, then the second characters of the two strings are compared and so on.

Thus, we have "cat" = "cat" since the two strings are exactly the same but we have "cap" < "cat" since 'p' is smaller than 't'. Also, we have "mat" > "cat" since 'm' is larger than 'c' and also "aazd" < "abbb" since 'a' is smaller than 'b' (The third characters 'z' and 'b' do not get compared at all since "abbb" wins when the second characters get compared).

### Caution

1. Using recursion is not compulsory in this question. However, you will have to write much less code if you use recursion.
2. Do not output names in incorrect order. The autograder will heavily penalize you if you do this since it will give you marks only if a correct name is output in its correct location.
3. We will not penalize you for stray newlines at the end of your output. However, do not have stray spaces at the end of each line of your output. You will not pass test cases if you have these.

### HINTS:

This problem may seem very complicated if you try to write a loop to solve the problem but it can be very elegantly solved using recursion as we explain below. Suppose we want to generate all names of length 3 out of the first two alphabets, in lexicographic order. This can be solved if we first

generate all names starting with a and then generate all names starting with b (since names starting with a come before those starting with b in lexicographic order).

However, notice that generating all names starting with a, and that too in lexicographic order, we simply need to find all names of length 2 in lexicographic order, and simply add an a before them. But this is just a smaller version of the original problem which is why recursion can be used. Indeed, all names of length 2 (formed out of the first two alphabets) in alphabetical order are simply

aa  
ab  
ba  
bb

and if we prepend (add to the beginning) an a to all these names, we get all names starting with a in alphabetical order!

aaa  
aab  
aba  
abb

Write a function that takes partially filled in names and recursively calls itself to complete the names.

Write a function `generateNames(char* name, int k, int n, int left)` which takes in four arguments

1. a character array `name` of length `k+1` (`k` characters for the names and one for the NULL character)
2. the value of `k` (will help you know what is the length of the array)
3. the value of `n` (which all letters of the English alphabet can you use)
4. the value of how many letters are left to be filled in

The base case of the recursion can be `left = 0` which means we have a complete name which can simply be printed. To print all names starting with the letter 'a', you could do something like the following:

```
char name[k+1];  
name[k] = '\0'; // Do not forget the NULL character  
name[0] = 'a';  
generateNames(name, k, n, k - 1); // one character already filled and k-1 left to be filled
```

Use these hints to completely solve the problem

-----  
**EXAMPLE:**

INPUT

2 3

OUTPUT:

aaa  
aab  
aba  
abb  
baa  
bab  
bba  
bbb  
-----

**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/6215>)