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_CONDLOOPS
- 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
- LABEXAM-PRAC-02_ENDSEM
  - Meanie Numbers
  - Rotate Then Rotate Code
  - The enigma that was Enigma
  - Save the Date
  - Pretty Patterns
  - Trivial Tic-Tac-Toe
  - How Mr C reads your code
  - Malloc Mystery
- LAB-PRAC-13_STRUC-NUM
- LAB-PRAC-14_SORT-MISC

Pretty Patterns

LABEXAM-PRAC-02_ENDSEM

Pretty Patterns [40 marks]

Problem Statement
Mr C wanted to generate pretty patterns according to some rules. He asked one of his clones to generate a pattern. However, that clone frequently makes mistakes. Help Mr C verify if the clone generated a pattern according to the rules or not.

The first line of the input will be a string that will only contain the following characters: upper and lower case English characters i.e a-z and A-Z, digits 1-9, the plus symbol + and the multiplication symbol *. Let us call it the rule string. The second line of the input will be another string which will only contain upper and lower case English characters i.e a-z and A-Z. Let us call it the pattern string. You have to find out whether the pattern string follows the rules given in the rule string or not. If it follows the rules, print "YES" (without quotes) else print "NO" (without quotes). The rule and pattern strings will contain no more than 99 characters.

The rule string will always give you an alphabet character (either upper or lower case) followed by either a single digit, or else the symbols + or *. Let us call this pair a token. The pattern string will basically be a sequence of such tokens. For example a4c+a* has 3 tokens [a4][c+][a*]. The above pattern is interpreted as follows

1. The token [a4] means that the character 'a' (without quotes) should appear exactly 4 times
2. The token [c+] means that the character 'c' (without quotes) should appear one or more times
3. The token [a*] means that the character 'a' (without quotes) should appear zero or more times

Thus, the second character of the token tells us how many times the first character must repeat. This is why we call the second character a repetition character. Thus, the above rule is interpreted as a command "a should appear exactly 4 times followed by one or more appearances of c followed by zero or more appearances of a". Suppose the pattern string is "aaaac" (without quotes). Then it is clear that this message follows the command as it has four 'a' followed by one 'c' (the third token asks for zero or more 'a' after the 'c's are over and this string has 0 'a's after the 'c's are over which is fine). However, the string "aacccaa" does not follow the rule since it starts off with 2 'a' whereas the rule demands that the pattern start with four 'a'. Some other patterns that satisfy the rule are "aaaacccaaaa" and "aaaacccc". Some other patterns that violate the rule are "aaaa" (there is no c but there should be at one or more c after the 4 a), "aaaaa" (missing c), "Aaaaca" (case error) and "abaaca" (the rule does not allow b to appear anywhere in the string).

We promise you that we will never give you a rule string where two consecutive tokens have the same first character. For example, we will never give you the rule string a2a+b5c* since a appears in two consecutive tokens.

Compulsory function usage in your code
In your code, you should write a function in the following format that takes two strings, the rule string and the pattern string and returns 1 if the pattern string does follow the rules given in the rule string and 0 otherwise. Be warned that not using such a function to write your code will cause you to lose a small number of manual grading marks.

```
int checkRule(char *rule, char* pattern){
   // Write your code here
   // If pattern follows the rule, return 1 else return 0
}
```

Problem-specific Words of Caution:

1. **Do not forget to submit your code**. You can submit multiple times. Your last submission will get graded.
2. If the repetition character will never be '0'.
3. There is only one line in your output.
4. Consecutive tokens in the rule string will have different first characters but they may have the same repetition character. Moreover, two or more tokens in the rule string may have the same first character, just that two consecutive tokens will never have the same first character. This means that a5b+a* is a valid rule string that can be given to you.

General Grading Policy

1. **TOTAL MARKS OF THE EXAM** 20 + 40 + 40 + 70 = 170
2. **TOTAL DURATION OF THE EXAM** 3 hours 30 minutes
3. See below for question-specific details of how partial marking would be done by the autograder in this question
4. Your submissions will be inspected by the autograder as well as a human grader
5. Human graders will (among other things) allot marks for the following

    1. Neatly structured code that uses at least one function other than the main function to process the input. The questions will usually suggest how to use functions to process the input. Submissions that ignore these suggestions and use only the main function to solve the entire problem, will lose a small fraction of marks.

    2. Proper and meaningful variable names

    3. Nice looking and consistent indentation

    4. At least a couple of comments explaining to the human grader what are you doing, especially when the steps are not obvious

    5. Comments, good indentation and meaningful variable names are very important for the human grader to understand what are you doing and why. If they cannot understand your code, do not expect them to give you (partial) marks either.

6. Solutions that indulge in hard-coding **will get a straight zero** even if they are passing some test cases. Hard-coding is a form of cheating strategy where someone write code of the form "if(input == A) printf(B)" without doing any calculations on A to obtain B. The values of A and B are either read from the evaluation/submission window or else guessed.
7. Be careful about extra/missing lines and extra/missing spaces if you do not want to lose autograder marks
8. Proportion of marks allotted to autograder (in particular, weightage to visible and hidden test cases) and human grader will be revealed when marks and grading rubrics are released
9. You are allowed to use the libraries stdio.h, math.h, string.h, stdlib.h **but not any other library**. Use of unpermitted libraries will carry a penalty. You may use any programming tools that we have discussed in lectures/tutorials or in lab questions such as arrays (1D, 2D, 3D, arrays of arrays etc), strings, loops, structures, functions, recursion, pointers, linked lists,

stacks, queues, graphs, enumerations, flags, conditionals, global, static and shadowed variables.

EXAMPLE:
INPUT
a4c+a*
aaaac

OUTPUT:
YES

Explanation: see example above

Grading Scheme:
Total marks: **[40 Points]**

There will be no partial grading in this question. An exact match will receive full marks whereas an incomplete match will receive 0 points. Please be careful of missing/extra spaces and missing/lines (take help of visible test cases). There are 4 visible and 4 hidden test cases.

Start Solving! (/editor/practice/6249)