# Common Instructions

## Q1 in every session (worth 20 marks) was only graded by the autograder. Manual grading was used only for questions worth 40 marks (Q2 and Q3) and the question worth 70 marks (Q4)

There are 3 components of the marking scheme (apart from penalty):

1. **Autograde Score**: given by autograder based on performance on test cases
    a. out of 20 for 40 marks questions (2 + 2 + 2 + 2 + 3 + 3 + 3 + 3 for 8 test cases)
    b. out of 28 for 70 marks questions (3 + 3 + 3 + 3 + 4 + 4 + 4 + 4 for 8 test cases)
2. **Manual Score**: given by human graders
    a. out of 15 for 40 marks questions (see questions for break up)
    b. out of 35 for 70 marks questions (see questions for break up)
3. **Good Coding Score**: given by human graders for indentation, proper variable names, comments
    a. out of 5 for 40 marks questions (2 indentation + 2 variable names + 1 comments)
    b. out of 7 for 70 marks questions (3 indentation + 2 variable names + 2 comments)
4. **Penalty Score:** given for illegal library use and hardcoding.
    a. Solutions that indulge in **hard-coding get a straight zero overall** even if passing some test cases. Hard-coding is a form of cheating where someone writes 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 read from evaluation/submission window or else guessed.
    b. Only libraries permitted are stdio.h, string.h, math.h and stdlib.h. If any other libraries are used (e.g. ), a penalty is given
    **ILLEGAL LIBRARY USAGE PENALTY IS 5 MARKS FOR 40 MARKS QUESTIONS AND 10 MARKS FOR 70 MARKS QUESTIONS**
    c. Use of all programming techniques 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, is allowed. No penalties for using these.

# Rotate Then Rotate Code (40 marks)

**Problem Statement** In your input you will be given two strings containing only lower-case English alphabets (no spaces, punctuation marks). Mr C suspects that the second string was obtained by taking the first string and performing two rotation operations on the first string. The operations are described below

First, the English alphabet was rotated to the right by k letters (k is not known to you). For example if k = 2, a would become c, b would become d, y would become a, z would become b, and so on. The clone first used this rotated alphabet to rewrite the original string. For example, if the original string was abc and k = 2 then the string now becomes cde

The new string is itself rotated to the right by l locations (l is not known to you). For example, if l = 2, the new string cde, rotated one location becomes ecd and rotated two locations becomes dec.

If the second string can be obtained from the first string by performing the above two rotations, you have to print in your output the word "YES" (without quotes) followed by a space followed by the value of k followed by a space followed by the value of l. If there is no way to obtain the second string from the first string using the rotations described above, print "NO" (without quotes) in your output.

## Rubric

1. Give **3 marks** for using a function (even if incorrectly) which takes two character arrays as input among other parameters and either returns the verdict or prints the verdict itself. Do not cut marks if the function prototype is not exactly the same as that in the question.
2. Give **3 marks** for checking for the case of unequal string lengths.
3. Give **3 marks** for going over all possible shifts of the alphabet (or something equivalent).
4. Give **3 marks** for going over all possible rotations of the string (or something equivalent).
5. Give **3 marks** for correctly checking whether the rotated string with the rotated alphabet matches the original string or not.

# The enigma that was Enigma (40 marks)

**Problem Statement** The first line of the input will give you a strictly positive number n. The next line will give you n digits i.e 0, 1, 2, ..., 9 (no spaces). These digits actually stand for a secret message but there could be many such secret messages, as described below. Your job is to print all secret messages in lexicographic order, i.e. in the order these messages would appear in a dictionary. Suppose n = 4 and the 4 digits are 1234. Then we can try to interpret this message as a sequence of alphabets with a = 1, b = 2, c = 3, d = 4 and so on. Thus, one obvious interpretation of this message is as follows [1] [2] [3] [4] => abcd However, there exist other interpretations as well such as [1][23][4] => awd and [12][3][4] => lcd.

Note that [12][34] is not a valid interpretation since the English alphabet contains only 26 characters and there is no 34-th character. For the same reason, [123][4], [1][234], [1234] are also invalid. Now let us see what to do when there are zeros in the digits. Suppose n = 2 and the 2 digits are 10. Then one valid interpretation is [10] => j However, the interpretation [1][0] is not valid since there is no 0-th character in the English alphabet. Thus, the only interpretations that are valid are those in which each chunk corresponds to a number between 1 and 26 (both included).

## Rubric

1. Give **3 marks** for using a function (even if incorrectly) recursively calls itself to solve the problem. Do not cut marks if the function prototype is not exactly the same as that in the question.
2. Give **3 marks** for correct invocation of the function from the main function.
3. Give **3 marks** for correctly handling the base case of the recursion.
4. Give **6 marks** for correctly handling the recursive case of the function
    a. 2 marks for correctly handling the lexicographic ordering
    b. 2 marks for correctly deciding which position is to be output next
    c. 2 marks for checking if something is an illegal interpretation e.g. [01] or [34].

If a student has not written recursive code, then point 1 marks will not be awarded. However, point 2-4 will have to be interpreted accordingly.

# How Mr C reads your code (70 marks)

**Problem Statement** 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 0-9, the plus symbol + and the multiplication symbol *. Let us call it the pattern 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 message string. You have to find out whether the message string follows the pattern given in the pattern string or not.

The pattern and message strings will contain no more than 99 characters. The pattern 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

First, take the pattern and output it in reduced form. Then print whether the message string obeys the pattern string or not.

1. The reduced pattern string cannot have '0' or '+' as a repetition character for any token.
2. In a reduced pattern string, if two consecutive tokens have the same first character, then the first token must have repetition character as a digit and the second token must have the repetition character as *. For example a6a* is a reduced pattern but a+, a5a+, a4a2a*, a+a* are not.
3. If a message obey the pattern string it must obey the reduced pattern string and vice versa.

## Rubric

1. Give **2 marks** for using a structure for solving the problem.
2. Give **3 marks** for translating standalone + e.g. a+ => a1a*.
3. Give **3 marks** for handling translations of + followed by digit tokens a+a5 => a6a*
4. Give **3 marks** for handling translations of + followed by + tokens a+a+ => a2a*
5. Give **3 marks** for handling translations of + followed by * tokens a+a* => a1a*
6. Give **3 marks** for handling translations of digit followed by digit tokens a5a4 => a9
7. Give **3 marks** for handling translations of combinations of the form a5a+ => a6a*
8. Give **3 marks** for handling translations of combinations of the form a5a* => a5a*
9. Give **3 marks** for handling translations of combinations of the form a*a5 => a5a*
10. Give **3 marks** for handling translations of combinations of the form a*a+ => a1a*
11. Give **3 marks** for handling translations of combinations of the form a*a* => a*
12. Give **3 marks** for correctly checking if the message string obeys the original or reduced string or not.

# Pretty Patterns (40 marks)

**Problem Statement** 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

## Rubric

13. Give **3 marks** for using a function (even if incorrectly) which takes two character arrays as input parameters and either returns the verdict or prints the verdict itself. Do not cut marks if the function prototype is not exactly the same as that in the question.
14. Give **3 marks** for correctly handling the tokens with a digit.
15. Give **3 marks** for correctly checking the tokens with +.
16. Give **3 marks** for correctly checking the tokens with *.
17. Give **3 marks** for correctly checking if the string contains illegal characters at the beginning or the end of the string e.g. a+b+ cannot generate “dab” or “abc” since the characters d an c are extra.

# Trivial Tic-Tac-Toe (40 marks)

**Problem Statement** We will give you the incomplete tic-tac-toe board as input. In your output, you have to generate all possible ways the X player, can win this game if the two of them resume playing. We will be playing a slightly simplified version of the tic-tac-toe game in this question for your convenience. In our version, X starts playing in the first round and places an X in some blank cell. Then O places an O in some blank cell. This continues till all the cells are filled in. Given the filled board, if any row is completely filled with X, then X is declared a winner. Beware that this is not the standard tic-tac-toe game.

Your output should print every winning board for Mr C on a separate line. A board is printed by first printing the first row then the second row then the third row. Print the winning combinations in lexicographic order.

## Rubric

1. Give **3 marks** for using a function (even if incorrectly) recursively calls itself to solve the problem. Do not cut marks if the function prototype is not exactly the same as that in the question.
2. Give **3 marks** for correct invocation of the function from the main function.
3. Give **3 marks** for correctly handling the base case of the recursion.
4. Give **6 marks** for correctly handling the recursive case of the function
    a. 2 marks for correctly handling the lexicographic ordering
    b. 2 marks for correctly deciding which position is to be decided next
    c. 2 marks for checking that number of X in the final board is 5 and the number of O is 4.

If a student has not written recursive code, then point 1 marks will not be awarded. However, point 2-4 will have to be interpreted accordingly.

# Malloc Mystery (70 marks)

**Problem Statement** This question will require us to simulate malloc and free calls on a toy system with only 1000 bytes of memory. As I mentioned in the 03 November lecture, not so long ago, this was actually how much memory that was available on computers. Create an integer array called MEM of length 1000. This array will act as our memory bytes. If we set MEM[i] = 0 then it will indicate that the i-th byte is not allocated to any variable. If we set MEM[i] = 1, it will mean that some variable has already been allocated this memory location. Recall that i can take valid values between 0 and 999. The first line of the input will give you N, the number of malloc or free commands that we will give you in the next N lines. Each command will be either

1. malloc <iden> <bytes>
2. free <iden>

## Rubric

1. Give **2 marks** for using a structure for solving the problem.
2. Give **3 marks** for maintaining a MEM array to keep track of allotted memory
3. Give **5 marks** for handling memory leak situations correctly (MEM LEAK SUCCESS)
4. Give **5 marks** for handling failure malloc commands correctly (FAILURE)
5. Give **5 marks** for handling successful malloc commands correctly (SUCCESS)
6. Give **5 marks** for handling successful free commands correctly (SUCCESS)
7. Give **5 marks** for handling failure free commands correctly (FAILURE)
8. Give **5 marks** for correctly keeping track of used memory