LAB-PRAC-13_STRUC-NUM

Too tired to create a story - part I (p1v1d1)

Too tired to create a story - part I [20 marks]

Problem Statement
The first line of the input will give you N, a strictly positive number telling you the number of employees in a company. The next N lines will give you details of those N employees in the following format. For every employee, we will first give that employee's roll number in the form of a strictly positive integer, then a space, then the birth date of the person as a strictly positive integer, then the character '/' (without quotes) then the birth month as a strictly positive integer, then a space, then the year the person joined the company as a strictly positive integer.

ROLL D/M Y

The last line of the output will be just a roll number in the form of a strictly positive integer. Let us call this the query roll number.

1. If that roll number does not appear on the list at all, print "NOT FOUND" in the output and that is it.
2. If the query roll number does belong to an employee on the list, then find out the roll numbers of all employees who joined the company in the same year as the person in the query roll number, as well as share a birth month with the person with the query roll number.
3. If there is no such person on the list who has the same joining year and birth month as the query person (other than the query person themself of course), print "NO ONE" in the output.
4. If there are multiple people who share the joining year and birth month with the query person, output their roll numbers in the order they appeared on the list in the input. Output one roll number on each line.

Caution

1. The date, month, and year numbers will not have leading zeros.
2. Roll numbers on the list will be unique i.e. no roll number will appear twice on the list.
3. Of course the query roll number, if present on the list, will also share the joining year and birth month with themself. However, you should not print the query roll number itself in the output.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS: Use a structure to store and process the employee data (this is not compulsory though)

```
struct Employee{
int roll;
int dateBirth;
int monthBirth;
int yearJoining;
};
```

EXAMPLE 1:
INPUT
2
100100 23/5 2011
100101 28/5 2011
100100

OUTPUT:
100101

Explanation: 100101 joined the company in the same year as 100100 did and the two share their birth month.

EXAMPLE 2:
INPUT
2
10 11/12 2018
11 10/10 2017
12

OUTPUT:
NOT FOUND

Explanation: roll number 12 is not on the list at all

EXAMPLE 3:
INPUT
5
98834 18/3 2015
98393 4/10 2017
88575 31/8 2016
18167 23/4 2013
26181 23/4 2018
18167

OUTPUT:
NO ONE

Explanation: roll number 18167 does not share the joining year and birth month with anybody else on the list.

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.

All Test Cases (Visible + Hidden)

Input	Output

5 98834 18/3 2015 98393 4/10 2017 88575 31/8 2016 18167 23/4 2013 26181 23/4 2018 18167	NO ONE
20 60938 7/10 2014 78712 11/10 2015 45524 9/1 2017 29337 13/8 2013 15364 29/12 2013 48318 30/10 2017 41322 27/7 2018 17594 1/2 2013 81456 18/1 2016 66058 16/7 2015 42366 10/12 2017 39990 18/7 2018 36066 7/10 2018 83872 17/7 2017 35048 25/12 2015 98906 18/10 2017 96010 17/1 2018 45660 19/3 2016 82769 20/11 2013 69495 27/1 2017 48318	98906
2 101 20/9 2014 102 20/9 2015 103	NOT FOUND
50 11821 29/5 2018 36544 13/5 2018 83907 22/5 2018 28597 5/5 2018 40946 1/5 2018 29137 18/5 2018 81821 12/5 2018 40919 6/5 2018 98406 20/5 2018 70825 3/5 2018 91710 12/5 2018 55249 14/5 2018 41951 7/5 2018 47453 10/5 2018 33598 17/5 2018 49469 8/5 2018 29738 29/5 2018 44557 16/5 2018 67261 20/5 2018 60137 23/5 2018 99550 31/5 2018 42380 21/5 2018 54322 25/5 2018	11821 36544 83907 28597 40946 29137 81821 40919 98406 70825 91710 55249 41951 47453 33598 49469 29738 44557 67261 60137 99550 42380 54322 12691

12691 18/5 2018
73241 8/5 2018
94328 8/5 2018
63296 13/5 2018
23013 1/5 2018
28865 11/5 2018
37810 26/5 2018
99643 25/5 2018
90895 16/5 2018
60892 11/5 2018
27296 12/5 2018
89053 25/5 2018
75370 9/5 2018
93717 26/5 2018
96306 10/5 2018
31718 23/5 2018
47313 9/5 2018
50846 17/5 2018
70033 19/5 2018
87679 10/5 2018
32847 22/5 2018
95379 8/5 2018
58673 11/5 2018
48554 16/5 2018
47235 23/5 2018
90199 18/5 2018
98346 6/5 2018
32847

73241
94328
63296
23013
28865
37810
99643
90895
60892
27296
89053
75370
93717
96306
31718
47313
50846
70033
87679
95379
58673
48554
47235
90199
98346

100
27336 2/6 2015
14432 23/7 2013
48074 6/2 2017
63665 27/4 2014
15680 1/12 2018
22862 5/2 2018
61239 17/7 2015
99245 28/3 2018
13284 14/2 2017
55553 20/1 2018
29517 11/11 2017
77159 4/12 2016
40800 31/10 2016
86596 5/4 2018
74156 22/8 2016
55683 13/12 2018
30902 2/10 2013
89695 20/3 2018
11386 6/1 2013
54794 3/11 2016
77893 2/6 2018
80356 20/1 2014
46886 26/12 2014
78625 25/7 2013
20286 22/9 2015
88481 8/9 2014
74499 4/12 2015
98140 9/2 2014

84194
69621
99976

21649 11/2 2016
37831 14/5 2016
79523 30/1 2016
11614 31/9 2018
38278 11/5 2016
90314 11/9 2014
96723 13/11 2016
27044 6/1 2017
42690 20/7 2015
56183 13/6 2014
77703 30/4 2017
47242 31/7 2015
48513 10/12 2014
21470 22/3 2015
41419 9/6 2017
80323 1/12 2016
15865 8/5 2017
38056 18/6 2014
19330 27/1 2017
73000 7/3 2014
94796 31/2 2013
71745 22/5 2015
77900 19/9 2017
84194 3/8 2017
30365 24/12 2017
73388 8/4 2016
72313 21/12 2017
82951 7/3 2015
37107 11/6 2015
14066 28/9 2016
23763 29/5 2018
56141 24/5 2013
94889 11/1 2018
52511 24/9 2015
41653 8/12 2014
80023 12/5 2013
47201 23/1 2014
13623 3/2 2018
83383 20/12 2018
55463 3/6 2017
45763 3/3 2018
85000 24/8 2015
69621 7/8 2017
56998 24/8 2013
90447 23/10 2013
63422 25/3 2016
38754 19/1 2016
28718 15/10 2017
54140 11/7 2015
74762 31/5 2013
71332 14/5 2013
25591 20/7 2014
45436 28/9 2015
41657 7/4 2013
20387 24/6 2015
17200 13/8 2018
90325 4/12 2015
40854 16/7 2017

80321 27/5 2016 16827 22/10 2014 19543 20/3 2017 69712 26/7 2013 33305 20/9 2015 22138 21/8 2013 71877 28/11 2015 41826 24/7 2018 74949 17/11 2014 91223 28/8 2017 95537 17/11 2018 26229 27/10 2016 19323 10/1 2016 99976 16/8 2017 91223	
20 60938 7/10 2014 78712 11/10 2015 45524 9/10 2017 29337 13/8 2013 15364 29/12 2013 48318 30/10 2017 41322 27/7 2017 17594 1/2 2013 81456 18/1 2016 66058 16/7 2015 42366 10/10 2017 39990 18/7 2018 36066 7/10 2018 83872 17/7 2017 35048 25/12 2015 98906 18/10 2017 96010 17/1 2017 45660 19/3 2016 82769 20/11 2013 69495 27/1 2017 48318	45524 42366 98906

Too tired to create a story - part II (p1v2d1)

Too tired to create a story - part II [20 marks]

Problem Statement
The input to the problem will be a single strictly positive integer n, followed by a floating point number eps denoting the *precision* we want in your output. You have to output the square root of that integer. However, you have to calculate the square root using the Babylonian method (often called the Hero's method as well). The method is described below. Use float variables for all your calculations.

Recall that you are given a strictly positive integer n, and a precision parameter eps which will be a floating point number. Store n in an int variable and eps in a float variable. The algorithm starts off by taking an initial guess of the square root, say x0 (the method to find x0 is given below). At each step, the algorithm updates the guess according to the following rule.

x1 = (x0 + n/x0)/2

i.e. the next guess is the average of the previous guess and the ratio of the number n and the previous guess. This is repeated till we have abs(xt - n/xt) < eps i.e. the absolute difference between the guess and the ratio of the number n and the guess is strictly less than eps, the precision parameter we gave you in the input.

You can show that once this happens, xt is very close to the square root of n. More precisely, one can prove a theorem that shows that actually if xt satisfies the above condition, then we must have abs(xt - sqrt(n)) < eps. You have to choose the initial guess x0 from the set {2.000, 20.000, 200.000, 2000.000, â€¦.} i.e. two times powers of 10. The guess x0 is chosen such that its square is closest to n. In case squares of two numbers from this set are equally close to n, choose the smaller one as the initial guess x0.

In the first line of the output, print the value of the initial guess as a floating point number to three digits of precision (use the %0.3f flag in printf to achieve this). In the next line, print the square root you have computed, correct to 3 decimal places (use the %0.3f flag in printf to achieve this).

Caution

1. The initial guess must also be printed as a floating point number with 3 digits of precision.
2. Even though your inputs are integers, your output is not an integer. Use float variables for all calculations.
3. Do not try to cheat by using the math.h sqrt() function to calculate the square root. We are looking for the answer given by the Babylonian which will have errors depending on the precision parameter given to you. The math.h sqrt() function will give you a highly accurate output and hence will not match the expected output.
4. There are two lines in your output with no spaces.

EXAMPLE:
INPUT
441 1.0

OUTPUT:
20.000
21.025

Explanation: 400 is closest to 441 among (4, 400, 40000, etc). Note that the square root of 441 is exactly 21.000 but due to the precision constant being high, Babylonian method is returning an answer with error. This error will go down if we decrease the epsilon parameter.

Grading Scheme:
Total marks: **[20 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). Each visible test case is worth 1 point and each hidden test case is worth 2 points. There are 2 visible and 4 hidden test cases.

All Test Cases (Visible + Hidden)

Input	Output
441 1.0	20.000 21.025

54794 0.1	200.000 234.099
6.25 0.5	2.000 2.500
2 0.01	2.000 1.417
400 0.8	20.000 20.000
4738512 1.0	2000.000 2176.826

Too tired to create a story - part III (p1v3d1)

Too tired to create a story - part III [20 marks]

Problem Statement
In the input, you will be given a strictly positive integer K denoting the length of the strings you have to print. In your output, on each line, you have to print a string of length K using only the characters '0' and '1' (without quotes). The strings must be printed in lexicographically increasing order i.e. if you think of these strings as numbers, the numbers should appear in increasing order.

The only property these strings must satisfy is that no two consecutive characters in the strings you generate can be the character '0'. Two or more consecutive characters can be '1' but two consecutive characters cannot be '0'.

Caution

1. The first character in the string can freely be 0 or 1 since there is no previous character to cause consecutive 0. However, second character onwards, we must have a 0 only if the previous character was not 0 to avoid consecutive 0.
2. Be sure to print all leading 0 in the output. Every string you print must contain k characters.
3. We will not penalize you for extra newlines at the end of your output but do not have extra spaces anywhere in your output.

HINTS: Write a function of the form
void genStrings(char *str, int k, int done)

1. str: the character array that contains the (possibly incomplete) string
2. k: the length of the string
3. done: how many characters have we generated yet

The base case can be when done = k in which case you can just print the string. Otherwise you need to set the character at position done appropriately and call the function recursively. The function may be initially invoked as
char str[k+1];
str[k] = '\0';
genStrings(str, k, 0);

EXAMPLE:
INPUT
2

OUTPUT:
01
10
11

Explanation: 00 is an illegal string since it has two consecutive zeros.

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.

All Test Cases (Visible + Hidden)

Input	Output
2	01 10 11
4	0101 0110 0111 1010 1011 1101 1110 1111
1	0 1
3	010 011 101 110 111
10	0101010101 0101010110 0101010111 0101011010 0101011011 0101011101 0101011110 0101011111 0101101010

0101101011
0101101101
0101101110
0101101111
0101110101
0101110110
0101110111
0101111010
0101111011
0101111101
0101111110
0101111111
0110101010
0110101011
0110101101
0110101110
0110101111
0110110101
0110110110
0110110111
0110111010
0110111011
0110111101
0110111110
0110111111
0111010101
0111010110
0111010111
0111011010
0111011011
0111011101
0111011110
0111011111
0111101010
0111101011
0111101101
0111101110
0111101111
0111110101
0111110110
0111110111
0111111010
0111111011
0111111101
0111111110
0111111111
1010101010
1010101011
1010101101
1010101110
1010101111
1010110101
1010110110
1010110111
1010111010
1010111011
1010111101
1010111110

1010111111
1011010101
1011010110
1011010111
1011011010
1011011011
1011011101
1011011110
1011011111
1011101010
1011101011
1011101101
1011101110
1011101111
1011110101
1011110110
1011110111
1011111010
1011111011
1011111101
1011111110
1011111111
1101010101
1101010110
1101010111
1101011010
1101011011
1101011101
1101011110
1101011111
1101101010
1101101011
1101101101
1101101110
1101101111
1101110101
1101110110
1101110111
1101111010
1101111011
1101111101
1101111110
1101111111
1110101010
1110101011
1110101101
1110101110
1110101111
1110110101
1110110110
1110110111
1110111010
1110111011
1110111101
1110111110
1110111111
1111010101
1111010110

	1111010111 1111011010 1111011011 1111011101 1111011110 1111011111 1111101010 1111101011 1111101101 1111101110 1111101111 1111110101 1111110110 1111110111 1111111010 1111111011 1111111101 1111111110 1111111111
8	01010101 01010110 01010111 01011010 01011011 01011101 01011110 01011111 01101010 01101011 01101101 01101110 01101111 01110101 01110110 01110111 01111010 01111011 01111101 01111110 01111111 10101010 10101011 10101101 10101110 10101111 10110101 10110110 10110111 10111010 10111011 10111101 10111110 10111111 11010101 11010110 11010111 11011010

	11011011
	11011101
	11011110
	11011111
	11101010
	11101011
	11101101
	11101110
	11101111
	11110101
	11110110
	11110111
	11111010
	11111011
	11111101
	11111110
	11111111

Point Proximity (p2v1d1)

Point Proximity [20 marks]

Problem Statement
The first line of the input will give three strictly positive integers n, k, q, separated by a space. In the next n lines, we will give you a list of k-dimensional point. On each of the n lines we will give you the k integer coordinates of a point and a code name corresponding to that point. All points will be k-dimensional and the code name will be a string with no spaces and at most 99 characters. q will be an integer between 1 and n (both included).

Look at the q-th point in the list (i.e. if q = 1, look at the first point in the list and if q = 2 then look at the 2nd point in the list). Call this the *query point*. Your job is to find the point in the list that is farthest from the query point in terms of squared Euclidean distance. In your output, you will have two lines. In the first line, print the code name of the point farthest from the query point and in the second line, print the squared Euclidean distance between the query point and the point farthest from it. If there are multiple farthest points in the list, print the code name of the point that appears first on the list.

We can represent a k-dimensional point as an array. Given two k-dimensional points as two integer arrays (remember coordinates of the points are all integers) a and b, the squared Euclidean distance between the two is calculated as
$$
sqEuclid(a,b) = \sum_{i=0}^{k-1} (a[i] - b[i])^2
$$

Caution

1. All code names will be unique i.e. no two lines will have the same code name.
2. q will be strictly positive i.e. it will look like a human-readable position in the list that starts from 1, not an array index that starts from 0.
3. Coordinates of the points can be negative integers too.
4. Since your coordinates are integers, the squared Euclidean distances (in particular the maximum squared Euclidean distance) will be an integer too.
5. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE:
INPUT
6 3 1
1 1 1 IITKANPUR
11 2 3 IITMADRAS
-1 10 1 IITBOMBAY
-10 2 1 IITDELHI
5 -5 5 IITKHARAGPUR
-10 2 1 IITD

OUTPUT:
IITDELHI
122

Explanation: Both IITDELHI and IITD are equally far from IITKANPUR but IITDELHI appears first on the list.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are two lines in your output. Printing each line correctly, in the correct order, carries 50% 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.

All Test Cases (Visible + Hidden)

Input	Output
6 3 1 1 1 1 IITKANPUR 11 2 3 IITMADRAS -1 10 1 IITBOMBAY -10 2 1 IITDELHI 5 -5 5 IITKHARAGPUR -10 2 1 IITD	IITDELHI 122
10 5 3 6 4 9 3 3 APPLE 5 3 8 0 4 MANGO 5 7 1 2 4 GRAPE 2 9 1 5 7 STRAWBERRY 4 5 4 4 9 PINEAPPLE 1 3 2 3 3 BANANA 0 3 5 9 1 BLUEBERRY 5 8 6 4 0 ORANGE 4 9 8 0 3 PEACH 6 0 5 0 0 WATERMELON	BLUEBERRY 115
11 5 10 6 4 9 3 3 APPLE	BLUEBERRY 115

5 3 8 0 4 MANGO 2 9 1 5 7 STRAWBERRY 4 5 4 4 9 PINEAPPLE 1 3 2 3 3 BANANA 0 3 5 9 1 BLUEBERRY 5 8 6 4 0 ORANGE 4 9 8 0 3 PEACH 6 0 5 0 0 WATERMELON 5 7 1 2 4 GRAPE 0 3 5 9 1 BURBERRY	
14 7 4 1 7 3 3 3 4 2 CRICKET 11 2 8 9 10 1 2 BASEBALL 2 2 3 4 5 7 8 BASKETBALL 10 6 1 1 2 2 5 FOOTBALL 80 1 2 5 1 5 6 BEACHVOLLEYBALL 80 1 2 5 1 5 6 VOLLEYBALL 10 12 3 6 3 7 9 TENNIS 7 11 5 8 3 4 BEACHBADMINTON 9 12 9 3 4 41 1 CHESS 9 12 9 3 4 41 1 ATHLETICS 16 13 20 3 3 9 1 SHOOTING 5 8 9 8 9 6 5 KABADDI 9 12 9 3 4 41 1 FUNATHLETICS 10 12 3 6 3 7 9 LAWNTENNIS	BEACHVOLLEYBALL 4953
12 6 7 65 41 15 34 70 49 TOKYO 10 38 2 81 68 51 NEWYORK 33 39 24 83 20 35 MEXICO 40 8 30 81 44 12 SAOPAOLA 54 100 58 43 86 70 LOSANGELES 62 2 10 16 43 67 SHANGHAI 78 64 35 25 34 65 MUMBAI 89 46 21 70 79 60 OSAKA 47 24 14 89 26 45 LONDON 24 31 28 39 88 82 CALCUTTA 66 79 20 77 6 86 PARIS 36 61 79 89 73 85 KARACHI	NEWYORK 10877
13 6 13 -1 -10 -10 -1 -1 -10 DELHI 65 41 15 34 70 49 TOKYO 10 38 2 81 68 51 NEWYORK 33 39 24 83 20 35 MEXICO 40 8 30 81 44 12 SAOPAOLA 54 100 58 43 86 70 LOSANGELES 62 2 10 16 43 67 SHANGHAI 78 64 35 25 34 65 MUMBAI 89 46 21 70 79 60 OSAKA 47 24 14 89 26 45 LONDON 24 31 28 39 88 82 CALCUTTA 66 79 20 77 6 86 PARIS 36 61 79 89 73 85 KARACHI	DELHI 36932

The Bisection Method (p2v2d1)

The Bisection Method [20 marks]

Problem Statement
Let $f: \mathbb{R} \rightarrow \mathbb{R}$ be a continuous real-valued function. Then if there exist 2 points $a, b \in \mathbb{R}$ such that $f(a)f(b) < 0$, then there exists at least one root of $f(x) = 0$ in the interval (a, b). This can be formally proven using the intermediate value theorem (IVT).

In this problem, we will use the bisection method to find a root of a polynomial function using the bisection method. Suppose $f(a) > 0$ and $f(b) < 0$. Then the bisection method proceeds as follows

1. Find the midpoint of a & b. Call this mid point m. Then the following cases are possible
2. Case 1: If $f(m) = 0$, then we have found a root. We end the algorithm here itself.
3. Case 2: $f(m) < 0$, then applying the IVT again tells us that a root must lie between in the interval (a,m). Repeat the procedure on this new interval i.e. with the setting $a = a$ and $b = m$.
4. Case 3: $f(m) > 0$, then applying the IVT again tells us that a root must lie between in the interval (m,b). Repeat the procedure on this new interval i.e. with the setting $a = m$ and $b = b$.
5. Keep repeating the above procedure till we have $b - a < eps$ (remember that we always have b > a in our algorithm) where eps will be given to you in the input. When b and a are indeed strictly closer than eps, simply output $(a+b)/2$ as our approximation to the root.

We can mathematically show that this algorithm will always give us a number which is no farther than eps from a true root of the function. It can also be shown that the above algorithm will always stop in less than about $log\frac{1}{eps}$ steps.

The first line of the input will give you n, the degree of a polynomial, a, b and eps, all separated by a space. n will be a strictly positive integer whereas a, b, eps will be floating point numbers. Use float variables to store them and also use float variables in all your calculations. The second line will give you the n+1 coefficients of the polynomial starting from the zero-order coefficient and moving on to higher powers. All coefficients will be integers. For example, if n = 3, and the polynomial is
\[
f(x) = a\cdot x^3 + b\cdot x^2 + c\cdot x + d,
\]
then we will give you the coefficients in the following order
d c b a

If we have given you invalid inputs, i.e. if $f(a) \cdot f(b) > 0$, then print "INVALID INPUT" in the output. If we have given you inputs such that $f(a) \cdot f(b) = 0$, then either a or b is a root. If both are roots then output the smaller root, correct to two decimal places using the %0.2f flag in printf, else if only one of a and b is a root then output that root, correct to two decimal places using the %0.2f flag in printf. In the general case, give the output correct to two decimal places using the %0.2f flag in printf.

Caution

1. Although we know that checking for equality with floating point numbers is not good, in order to simplify this question, we are asking you to check for exact equality when you check for $f(m) == 0$.
2. Note that we will not always ensure that $f(a) > 0$ in the input. We may give you a case where $f(a) < 0$ and $f(b) > 0$. Even that is a valid case and you have to process it accordingly by applying the IVT.
3. Be careful that whereas eps will be a strictly positive floating point number, a,b can be zero or negative as well.
4. The coefficients of the function can be zero or negative as well.
5. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE:
INPUT
3 1 5 0.1
-27 0 0 1

OUTPUT:
3.00

Grading Scheme:
Total marks: **[20 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). Each visible test case is worth 1 point and each hidden test case is worth 2 points. There are 2 visible and 4 hidden test cases.

All Test Cases (Visible + Hidden)

Input	Output
3 1 5 0.1 -27 0 0 1	3.00
3 4 7 0.1 -27 0 0 1	INVALID INPUT
5 2 5 0.1 -96 0 0 0 0 3	2.00
3 0 5 0.01 -8 6 -4 1	2.89
3 -1 5 0.1 -4 -13 -4 1	-0.39
29 -0.5 0 0.01 10 9 -41 -12 -13 -47 33 18 -31 38 7 -3 16 37 -42 -12 31 6 42 -31 1 45 49 12 -17 23 18 34 46 -13	-0.43

The pace is too fast (p2v3d1)

The pace is too fast [20 marks]

Problem Statement
Mr C thinks that the pace of the ESC101 course is too fast and that the interest of the students in the course just wont last. Help Mr C find out how to teach this course. In the input, you will be given two strictly positive integers n and T, separated by a space. n is the number of weeks in the course and T is the maximum number of topics that can be covered in the course in total. We assure you that n will be strictly positive and that T will be strictly positive but strictly less than 10 (i.e. T can be 1, 2, ... 9).

You have to generate all possible ways in which topics can be covered in the course by printing, for each

week, how many topics have been covered till that week. Obviously, number of topics covered till week k has to be greater than or equal to the number of topics covered till week k-1. However, Mr C insists that the number of topics covered within week k must be less than or equal to the number of topics covered in week k-1 so that the number of topics covered in a certain week never goes up (stays the same or goes down) as the weeks pass by. Thus, if week k covers 3 topics, week k+1 can cover 0, 1, 2 or 3 topics but not 4 or more topics.

The total number of topics covered in the course must not exceed T but the course may cover 0 topics in total and that is fine. You have to print each schedule on a different line of the output by printing, for the n weeks, how many topics were covered till the end of that week. There should be no spaces within a schedule i.e. the schedule will look like an n-digit number since $0 < T < 10$. Print the schedules in lexicographically increasing order.

Caution

1. You have to print the leading zeros in any schedule as well
2. You have to print the number of topics covered uptil the various weeks, not the number of topics covered in those weeks.
3. We wont penalize you for extra newlines at the end of your output but do not have stray spaces in your output.

HINTS: Write a recursive function of the following form to solve this problem.
void genCourse(int *topics, int n, int T, int pos)

1. topics: an integer array (possibly partially filled) storing how many topics were covered uptil a certain week
2. n: number of weeks
3. T: number of topics
4. pos: which position in the array needs to be filled next

EXAMPLE:
INPUT
2 5

OUTPUT:
00
11
12
22
23
24
33
34
35
44
45
55

Explanation There are 2 weeks and at most 5 topics can be covered in the course. If 0 topics were covered in the first week, at most 0 topics can be covered in the second week as well. Thus, 00 is a possible schedule. If 1 topic was covered in the first week then at most 1 topic can be covered in the second week which is why

11 and 12 are possible schedules. However 13 is not valid since it requires 2 topics to be covered in the 2nd week when only 1 topic was covered in the first week. 36, 46 etc are also not valid schedules since there are at most 5 topics in the course.

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.

All Test Cases (Visible + Hidden)

Input	Output
2 5	00 11 12 22 23 24 33 34 35 44 45 55
3 2	000 111 122 222
3 3	000 111 122 123 222 233 333
3 4	000 111 122 123 222 233 234 244 333 344 444
4 3	0000

	1111 1222 1233 2222 2333 3333
4 6	0000 1111 1222 1233 1234 2222 2333 2344 2345 2444 2455 2456 2466 3333 3444 3455 3456 3555 3566 3666 4444 4555 4566 4666 5555 5666 6666

A Question on Quadrilaterals (p3v1d1)

A Question on Quadrilaterals [20 marks]

Problem Statement
In the input, you will be give you the coordinates of 4 points on the 2D plane. The coordinates will be all be integers and will be separated by a space. The format is given below
x1 y1 x2 y2 x3 y3 x4 y4
These form the points p1 = (x1, y1), p2 = (x2, y2), p3 = (x3, y3) and p4 = (x4, y4). The four points will form either a square or a rectangle or a parallelogram or a trapezium.

In the first line of the output, tell us whether both pairs of opposite sides are parallel (print "2" without quotes in this case) or is only one pair of opposite sides of this quadrilateral parallel (print "1" without quotes in this case). In the second line of the input, print whether any angle of the quadrilateral is a right angle or not. If any one of the four angles is 90 degrees, print "YES" (without quotes) else if none of the angles is 90 degrees, print "NO" (without quotes). In the last line of the output, print the following

1. If the quadrilateral is a square, print "SQUARE" (without quotes)

2. If the quadrilateral is not a square but is a rectangle, print "RECTANGLE" (without quotes)
3. If the quadrilateral is not a rectangle but is a parallelogram, print "PARALLELOGRAM" (without quotes)
4. If the quadrilateral is not a parallelogram, print "TRAPEZIUM" (without quotes)

We assure you that the points will be given to you in counterclockwise order i.e. starting from p1 if you move around the quadrilateral in a counter clockwise manner, you will encounter first p2, then p3, then p4 and then back to p1. This hint is very important as it will help you figure out the edges of the quadrilateral. Since points are given in counter clockwise order (p1 p2) will be an edge of the quadrilateral, as will be (p2 p3) but (p1 p3) will be a diagonal of the quadrilateral.

Caution

1. The squares, rectangles etc may be rotated and not be axis aligned.
2. Coordinates will be integers but may be negative or zero.
3. Be careful of vertical and horizontal lines
4. The question may require you to compare floating point numbers like slopes etc. Since comparing floating point numbers for equality is dangerous, we will consider two floating point numbers to be the same if their difference in absolute terms is less than 0.0001. Use the fabs() function from math.h to get the absolute value of floating point numbers.
5. We will never give you a test case where any three points are collinear or where the quadrilateral is a scalene quadrilateral (i.e. neither of the cases mentioned above).
6. Be careful not to have any missing spaces or extra newlines in your output.

HINTS:
Write functions to check whether two lines are parallel to each other and whether two lines are perpendicular to each other. Lines are represented using two Point variables (see below). Also write a function to compute the Euclidean distance between two Point variables. Use these functions to decide which type of quadrilateral we have given you.

```
struct Point{
int x, y;
};
```

EXAMPLE:
INPUT
1 4 10 4 20 5 7 5

OUTPUT:
1
NO
TRAPEZIUM

Explanation: Only one pair of opposite sides is parallel, no corner angle is 90 degrees and the quadrilateral is a trapezium.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are three lines in your output. Printing each line correctly, in the correct order, carries 33% 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.

All Test Cases (Visible + Hidden)

Input	Output
0 0 10 1 9 11 -1 10	2 YES SQUARE
1 4 10 4 20 5 7 5	1 NO TRAPEZIUM
4 4 7 7 7 12 4 9	2 NO PARALLELOGRAM
1 10 10 1 30 21 21 30	2 YES RECTANGLE
-5 -5 0 -5 0 0 -5 0	2 YES SQUARE
0 0 10 0 10 5 5 5	1 YES TRAPEZIUM

The Trapezoidal Technique (p3v2d1)

The Trapezoidal Technique [20 marks]

Problem Statement
Mr C has just learnt the trapezoidal technique for calculating the area under a curve. Given a function $f: \mathbb{R} \rightarrow \mathbb{R}$, two points $a, b \in \mathbb{R}$ such that $a \leq b$ and a resolution parameter n, we first chop up the interval $[a,b]$ into n equal pieces (n will always be a strictly positive integer) in the following manner.

Define $\Delta x = \frac{b-a}n$ and let $x_i := a + i \cdot\Delta x$. The first piece is $[x_0, x_1] = [a, a+\Delta x]$, the second piece is $[x_1, x_2] = [a + \Delta x, a + 2\Delta x]$ and so on. On every piece, say $[x_i, x_{i+1}]$ we can define a trapezoid with heights at the end points as $f(x_i)$ and $f(x_{i+1})$ respectively.

The area under the curve in the interval $[a,b]$ is approximated by adding up the areas of all the trapezoids. Our function will always be a cubic polynomial of the form
\[
f(x) = p \cdot x^3 + q \cdot x^2 + r \cdot x + s
\]
The input will first give you p, q, r, s as integers on the first line, all separated by a space. Then we will give you a, b as integers on the second line, separated by a space. Finally we will give you n as a strictly positive integer on the third line. Your output should be the area of the curve from a to b, calculated as above.

Give your output rounded off to 4 decimal places using the %0.4lf format specifier in printf. Use double variables for all calculations.

Caution

1. Even though your inputs are integers, your output is not an integer. Use double variables for all calculations.
2. Do not try to cheat by doing the area calculation yourself using a definite integral and printing it directly. We are looking for the area given by the trapezoidal method which will have errors depending on the resolution errors. A definite integral will have no errors and hence will not match the expected output.
3. Area under the x axis is considered negative area as usual.
4. In case you get a situation where $f(x_i) < 0$ and $f(x_{i+1}) < 0$, count that entire trapezium as negative area.
5. In case $f(x_i) < 0$ and $f(x_{i+1}) > 0$ (or the other way round), your "trapezium" will look like a combination of two triangles, one below the x axis and one above the x axis. The area of the trapezium in this case will be found by adding the area of the triangle above the x axis and subtracting the area of the triangle below the x axis.
6. There is only one line in your output with no spaces.

EXAMPLE:
INPUT
0 1 0 0
-2 2
5

OUTPUT:
5.7600

Explanation: the function is $f(x) = x^2$. Note that the exact integral is 5.3333... but the expected output is 5.7600 since the trapezoidal method makes mistakes due to low resolution (5 is a very small number). The trapezoidal method output will approach the true integral output as the number 5 is increased to larger values like 5000 or so.

Grading Scheme:
Total marks: **[20 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). Each visible test case is worth 1 point and each hidden test case is worth 2 points. There are 2 visible and 4 hidden test cases.

All Test Cases (Visible + Hidden)

Input	Output
0 0 -2 1 0 2 10	-2.0000
0 1 0 0 -2 2 5	5.7600

0 1 0 0 -2 2 50	5.3376
1 1 1 1 0 10 500	2893.3440
1 1 1 1 -2 1 20	0.7444
1 -2 2 -3 -5 6 10000	-81.5833

Constrained Candy Crush (p3v3d1)

Constrained Candy Crush [20 marks]

Problem Statement
Recall from a lab problem earlier this week that Mr C is under treatment for candy addiction. He was instructed by his doctor to take only k grams of candy every day. Everyday, Mr C goes to the candy store to get the required amount of candy but yesterday, due to end-sem lab exam preparations, he forgot to buy candy. Now he finds that he has only n pieces of candy in his home, all scattered in mixed up.

Help Mr C find out how to get his daily quota from the candy he has at home. The first line of the input will give you two strictly positive integers n and k, separated by a space. The next line will give you n strictly positive integers, separated by a space. These integers will not be in any specific order but will not repeat i.e. no integer will be given to you twice. These integers represent the weights of the candies Mr C has at his home, in grams.

In your output you have to print all possible ways Mr C can eat k grams of candy using the candy he has at home. Note that Mr C has only one piece of each candy. Suppose k = 5 and Mr C has 4 pieces of candy of weight 4, 3, 2, and 1 grams. Then clearly there are two ways in which he can eat 5 grams of candy - eat the 4 gram and the 1 gram candy (represent this as the string 1001 to indicate that the first and the last candy are to be eaten) and eat the 3 gram and the 2 gram candy (represent this as the string 0110 to indicate that the second and the third candy are to be eaten).

Thus, every combination looks like a long number with 0 and 1 digits. Print these numbers in decreasing order i.e. in the above example, your output should be
1001
0110

If there is no way to eat k grams of candy using the candy provided, print "MR C IS DOOMED" in the output.

Caution

1. Mr C cannot break any candy into two - then creating combinations, the entire candy must be eaten or the entire candy must be set aside.
2. Mr C needs exactly k grams of candy - not one gram more, not one gram less.
3. Note that candy are not given to you increasing order of weights - they are all mixed up.

4. We will not penalize you for extra newlines at the end of your output but do not have extra spaces at the end of any line of your output.

HINTS: Solve this problem using recursion. Write a function of the form
void getCombinations(int *weights, int *choices, int n, int k, int sum, int pos, int *ptr)

1. weights: array of weights given to you as input
2. choices: did you choose the i-th candy or not?
3. n: number of candy
4. k: total weight of candy to be eaten by Mr C
5. sum: how much weight of candy has already been eaten due to choices already made
6. pos: which position in the weight array (i.e. which piece of candy) is it with respect to which a decision now needs to be made?
7. ptr: a pointer to a integer flag variable - set this flag to 1 (using *ptr = 1) the moment any combination is found which adds up to k grams of candy.

Initially, the choices array is all zeros. Gradually fill it up with 1 by choosing various pieces of candy. You may invoke this function as follows:
int isFound;
getCombinations(weights, choices, n, k, 0, 0, &isFound);

EXAMPLE:
INPUT
4 5
4 3 2 1

OUTPUT:
1001
0110

Explanation: see example above.

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.

All Test Cases (Visible + Hidden)

Input	Output
4 5 4 3 2 1	1001 0110
7 18 3 5 7 111 123 12 33	MR C IS DOOMED

10 20 1 13 15 7 9 11 3 17 19 5	1001101000 1000011001 1000000010 0101000000 0010000001 0000110000 0000001100
8 53 15 22 14 26 32 9 16 8	11000010 10100011 01100101
9 10 1 2 3 4 5 6 7 8 9	111100000 110000100 101001000 100110000 100000001 011010000 010000010 001000100 000101000
10 190 1 2 3 4 5 6 7 8 90 100	0000000011

Major Mobile Madness (p4v1d1)

Major Mobile Madness [20 marks]

Problem Statement
All of Mr C's clones have the latest mobile phone in the market and he does not want to be left out. However, he is in a dilemma. Some phones have a better camera whereas others have better battery life. Moreover, Mr C's parents gave him only a limited budget to purchase a phone. Help Mr C choose a phone to purchase. Mr C has collected a list of phones available in the market including their costs and ratings in 5 different categories - Camera, Performance, Battery, Hardware and Design. Mr C wants to buy a phone within his budget with the highest average rating across the 5 categories.

The first line of the input will contain two strictly positive integer, n and b, separated by a space. n will denote the number of phones in the market and b will denote Mr C's budget. In the next n lines, we will give you the specifications of each of the phones. Each specification will consist of 2 integers and 5 floating point numbers as shown below. All numbers will be separated by a space. The phone id will be unique, i.e. no two phones will have the same id.
id price cam perf bat hard des

In your output, you have to print the id of the phone which has the highest average rating among phones that Mr C can buy, i.e. phones whose price is less than or equal to Mr C's budget. If there is no phone Mr C can buy, print -1 in your output. If there is more than one phone with highest average rating among those Mr C can buy, choose the phone with the lowest price. If there is more than one phone with lowest price and highest average rating among those Mr C can buy, choose the phone that appears earliest in the list of phones. Use a structure to maintain details about the phone to help simplify your code.

Caution

1. Phone ids will not be given to you in any particular order, nor will the phones be listed such that prices or ratings are in any particular order.
2. The price of a phone will be a non-negative integer but may be zero.
3. The ratings of a phone will be a non-negative floating point number but may be zero.
4. The id of a phone will be a strictly positive integer.
5. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE:
INPUT
3 100
89 40 8.0 8.0 8.0 8.0 6.5
93 99 8.0 5.0 5.0 5.0 10.0
84 50 8.1 8.0 8.2 8.2 6.0

OUTPUT:
89

Explanation: All phones are affordable and phone ids 89 and 84 have equal average rating of 7.7. However, phone id 89 is cheaper.

Grading Scheme:
Total marks: **[20 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). Each visible test case is worth 1 point and each hidden test case is worth 2 points. There are 2 visible and 4 hidden test cases.

All Test Cases (Visible + Hidden)

Input	Output
3 100 89 40 8.0 8.0 8.0 8.0 6.5 93 99 8.0 5.0 5.0 5.0 10.0 84 50 8.1 8.0 8.2 8.2 6.0	89
2 100 19 200 8.0 7.0 4.0 6.3 9.1 21 400 0.3 0.4 10.0 10.0 8.0	-1
3 100 15 90 0.8 9.0 9.4 8.1 3.2 17 100 0.8 9.0 9.4 8.0 8.1 32 0 8.4 0.5 0.5 0.5 0.5	17
10 10000 10597 10449 7.1 4.9 4.2 2.9 2.8 47362 10060 4.5 9.5 9.5 7.0 0.6 75627 9514 6.4 0.5 8.0 5.2 7.8 16996 10107 2.5 4.9 8.1 0.2 7.1 63272 9970 1.1 7.5 0.8 9.4 0.1 49904 9022 1.4 0.2 9.9 3.8 3.1 55699 10114 6.1 4.9 5.8 3.8 2.5 8750 9607 4.1 4.5 7.4 0.5 8.2	75627

89776 10512 7.5 5.0 1.4 8.4 0.3 68852 9435 4.1 0.3 6.6 3.9 5.6	
12 10000 10597 10449 7.1 4.9 4.2 2.9 2.8 47362 10060 4.5 9.5 9.5 7.0 0.6 75627 9514 6.4 0.5 8.0 5.2 7.8 16996 10107 2.5 4.9 8.1 0.2 7.1 63272 9970 1.1 7.5 0.8 9.4 0.1 49904 9022 1.4 0.2 9.9 3.8 3.1 756272 9510 6.4 0.5 8.0 5.2 7.8 55699 10114 6.1 4.9 5.8 3.8 2.5 8750 9607 4.1 4.5 7.4 0.5 8.2 89776 10512 7.5 5.0 1.4 8.4 0.3 68852 9435 4.1 0.3 6.6 3.9 5.6 756271 9510 6.4 0.5 8.0 5.2 7.8	756272
50 10000 81251 2619 0.4 0.7 5.3 7.4 2.2 12989 4888 5.5 6.1 9.2 6.6 8.3 19741 16656 4.4 4.3 7.4 3.6 8.0 2079 13978 6.4 8.0 4.6 4.3 4.9 29095 4316 6.8 4.4 2.2 8.6 3.3 43842 15797 3.0 6.9 8.7 5.5 1.2 65841 3116 9.3 0.5 3.4 5.0 1.1 47539 6998 9.8 9.1 9.2 7.0 4.9 94249 11254 9.9 0.6 0.5 8.6 8.0 61911 14172 1.9 4.8 2.4 4.4 1.2 35878 5206 6.5 0.7 6.1 4.3 9.5 37747 15115 0.1 7.1 8.1 9.6 0.7 54617 10560 7.7 7.4 0.4 6.9 1.1 4898 10912 0.2 8.7 3.0 5.2 9.3 94980 9770 2.0 2.8 2.9 5.6 0.5 34288 4900 5.5 5.9 2.4 0.4 5.3 67690 7696 9.1 2.3 9.5 7.0 0.0 13310 9796 1.8 2.9 5.6 5.0 8.4 75916 15091 5.1 6.1 6.1 1.6 4.2 11906 2370 5.9 8.8 2.0 7.9 3.1 70859 15819 5.4 7.1 5.1 1.8 9.9 71083 13158 5.4 2.6 10.0 8.0 6.2 57831 12738 7.2 4.3 6.3 3.7 6.6 54841 17562 4.0 9.2 5.4 6.7 3.3 25683 3542 8.6 9.1 3.4 1.1 9.0 78017 12711 8.1 0.3 2.8 7.2 9.2 44168 10449 0.7 2.6 1.5 1.5 2.7 73212 15927 8.9 4.4 3.5 3.4 1.5 53662 12233 5.7 6.5 0.2 4.2 4.3 33947 14007 1.8 4.3 5.8 1.2 7.2 48407 5133 3.8 4.4 8.1 3.9 9.6 50117 10347 3.9 7.0 2.4 1.2 8.3 33745 2257 6.8 2.0 4.1 4.6 0.2 2878 3378 7.2 0.4 1.9 3.8 2.1 17504 10267 3.8 8.2 3.3 9.4 0.7 33102 2189 1.7 5.4 7.6 1.0 6.7 56512 5723 3.9 9.9 2.6 7.6 5.4 34958 12163 5.4 3.7 0.3 3.1 6.7 42231 6124 8.8 7.8 8.4 9.8 1.0 20550 6466 9.7 9.6 1.0 6.3 4.7 45919 17490 7.3 6.3 6.2 0.6 2.3	47539

88595 14853 9.3 0.6 7.1 8.9 0.9 56428 5998 6.7 5.1 3.1 4.2 10.0 74941 16409 8.2 0.3 6.8 5.3 1.6 37923 11265 1.0 6.4 6.3 3.3 6.2 6866 9148 5.9 7.6 3.4 7.4 3.4 16275 16560 4.4 4.0 9.0 1.2 0.1 93167 15155 2.2 7.6 7.9 7.3 3.8 41995 5434 10.0 3.4 4.6 5.4 9.0 73870 12902 8.2 2.8 4.9 7.3 1.0	

The Newton Raphson Method (p4v2d1)

The Newton Raphson Method [20 marks]

Problem Statement
The Newton-Raphson method is a popular method for finding the roots of functions. It is a precursor to the Newton method for optimizing non-linear functions. Given a real-valued function $f: \mathbb{R} \rightarrow \mathbb{R}$, and an initial guess of the root x_0, the NR method iteratively improves this guess using the following update rule
\[
x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}
\]
Then, using x_1, it obtains a (hopefully) better estimate of the root x_2 as
\[
x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}
\]
Thus, you can get x_{t+1} using x_t. You have to stop updating when $abs(x_t - x_{t+1}) < eps$ i.e. the absolute difference between two successive estimates is strictly smaller than eps where eps will be given to you. When the above happens, simply output x_{t+1} as your output.

The first line will contain n, a strictly positive number indicating the degree of the polynomial, followed by a space, followed by eps a floating point number (store it as a double). The second line will contain n+1 integers (may be zero or negative or positive), containing the coefficients of the polynomial from zero degree to max degree i.e. if the polynomial is a cubic (i.e. n = 3)
\[
f(x) = a\cdot x^3 + b\cdot x^2 + c\cdot x + d
\]
then we will give you the coefficients as
d c b a
The last line will contain x_0 the initial guess you should use. x_0 will not be an integer and you should store it as a double. Run the NR algorithm as shown above and give your output correct to 2 decimal places, using the %0.2lf flag in printf.

Caution

1. All coefficients of the polynomial will be integers but they may be zero or negative too.
2. You may use the fabs() function from math.h to compute the absolute value of a non-integral number.
3. The roots and intermediate values in your computations may be non-integral. Use double variables for all your computations.
4. Be careful while computing the derivative polynomial.
5. We assure you that if you follow the above rules correctly, you will never run into a divide-by-zero situation.

6. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE:
INPUT
2 0.01
1 -2 1
3.00

OUTPUT:
1.01

Grading Scheme:
Total marks: **[20 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). Each visible test case is worth 1 point and each hidden test case is worth 2 points. There are 2 visible and 4 hidden test cases.

All Test Cases (Visible + Hidden)

Input	Output
2 0.01 1 -2 1 3.00	1.01
3 0.0001 -3 2 0 0 15.00	1.50
2 0.001 1 -2 1 8.00	1.00
3 0.001 1 1 -3 2 0.4	-0.40
3 0.0001 1 1 -3 2 -5.00	-0.40
3 0.0001 1 5 -5 1 0.00	-0.17

The Palindrome Decomposition (p4v3d1)

The Palindrome Decomposition [20 marks]

Problem Statement

Any string can be decomposed into a sequence of palindrome strings (since a single letter is always a palindrome). For example, the string abc can be decomposed as [a][b][c] which we will represent as 123 denoting the fact that [a] is the first palindrome, [b] is the second palindrome and [c] is the third palindrome. On the other hand, abab can be decomposed in the following three ways (the corresponding numerical representations are also given.

[aba][b] => 1112
[a][bab] => 1222
[a][b][a][b] => 1234

You will be given a string with 9 or less characters (all characters will be English lowercase alphabets). You have to print the numerical representations of all possible palindromic decompositions of the string in the output. Each decomposition should be printed on a separate line with no spaces anywhere. Notice that each decomposition seems like a number with k digits where k is the length of the string we gave you. Print these numerical representations in increasing order

Caution

1. Even if the same palindrome repeats in the string, use a different counter to denote its occurrence in the decomposition. For instance, you may decompose the string abaaba as [aba][aba]. However, this will correspond to the representation 111222 and not 111111 since the second aba is a part of a different palindrome and not the first palindrome.
2. The first palindrome in the decomposition is to be denoted the number 1, not the number 0.
3. We will not penalize you for extra new lines at the end of your output but do not have extra spaces any where in your output.

HINTS: Write a recursive function printPalinDecomp(char *str, int *decomp, int len, int done, int counter) to solve this problem.

1. str: string given as input
2. decomp: an integer array to store the numerical representation of the decomposition
3. len: lenght of the string
4. done: how many positions of the string have we processed so far?
5. counter: how many palindromes have we generated so far?

EXAMPLE:
INPUT
abab

OUTPUT:
1112
1222
1234

Explanation: see above.

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.

All Test Cases (Visible + Hidden)

Input	Output
abcde	12345
abaaba	111111 111222 111234 122223 123345 123444 123456
abab	1112 1222 1234
banana	122222 122234 123334 123444 123456
malayalam	111111111 122222223 122234445 122234567 123333345 123444567 123456667 123456789
kanpur	123456