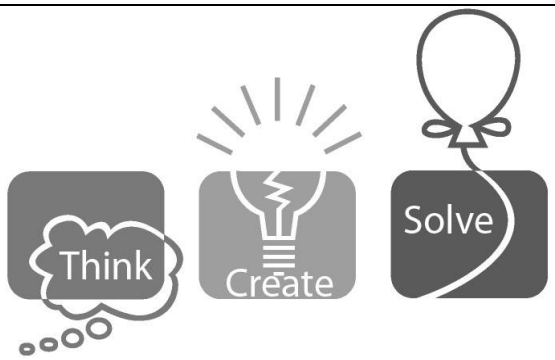
University of Central Florida



2021 (Fall) "Practice" Local Programming Contest

Problems			
Problem#	Difficulty Level	Filename	Problem Name
1	Easy	jersey	Special Jersey Numbers
2	Easy	medal	Medal Ranking
3	Easy	brownie	Brownies vs. Candies vs. Cookies
4	Medium	lemonade	Lemonade Stand
5	Medium	rain	Rain Gauge
6	Medium	balance	Balanced Strings
7	Medium-Hard	hanoi	Towers of Hanoi Grid

Call your program file: filename.c, filename.cpp, filename.java, or filename.py

For example, if you are solving Towers of Hanoi Grid:

Call your program file: hanoi.c, hanoi.cpp, hanoi.java, or hanoi.py

UCF "Practice" Local Contest — Aug 28, 2021

Special Jersey Numbers

filename: `jersey`
Difficulty Level: `Easy`
Time Limit: `5 seconds`

The jersey number 17 has been worn by several famous athletes, e.g., John Havlicek (NBA), Don Meredith (NFL), and Felipe Alou (MLB). The jersey number 18 has also been worn by several famous athletes, e.g., Peyton Manning (NFL), Joe Morgan (MLB), and Phil Jackson (NBA). These two jersey numbers are special to Dr. Orooji as well but for other reasons! So, whenever Dr. O is watching sports, he looks for these two special numbers.

The Problem:

Given a list of 10 numbers, determine which special jersey number is in the list.

The Input:

There is one input line, it consists of exactly 10 single-space-separated distinct integers (each integer between 11 and 99 inclusive) giving the jersey numbers for the players.

The Output:

Print one of four messages (`17`, `18`, `both`, `none`), indicating which special jersey number is in the list.

Sample Input	Sample Output
11 99 88 17 19 20 12 13 33 44	17
11 12 13 14 15 16 66 88 19 20	none
20 18 55 66 77 88 17 33 44 11	both
12 23 34 45 56 67 78 89 91 18	18

UCF "Practice" Local Contest — Aug 28, 2021

Medal Ranking

filename: `medal`
Difficulty Level: `Easy`
Time Limit: `5 seconds`

When different countries compete against each other (e.g., in the Olympics), they receive gold/silver/bronze medals. The countries can then be ranked in one of two ways: by "count" which is based on the total number of medals (regardless of the medal colors) or by "color" which is based on the number of gold medals (and silver medals if tied in gold medals, and bronze medals if tied in gold and silver).

The Problem:

Given the gold/silver/bronze medal counts for USA and Russia, you are to determine if USA wins in these two ranking methods.

The Input:

There is one input line, it consists of 6 integers (each integer between 0 and 500 inclusive); the first three integers represent (respectively) the gold, silver, and bronze medal counts for USA; the last three integers provide this info for Russia (in same order).

The Output:

Print one of four messages (`count, color, both, none`), indicating how USA can win. USA will win by `count` if its total medal count is higher than the total for Russia. USA will win by `color` if it has more gold medals than Russia (if tied in gold, then USA must have more silver; if tied in gold and silver, then USA must have more bronze).

Sample Input	**Sample Output**
10 5 15 10 1 0	both
10 5 15 10 6 10	count
12 5 10 5 20 30	color
10 0 15 10 5 30	none
10 5 15 10 5 15	none

UCF "Practice" Local Contest — Aug 28, 2021

Brownies vs. Candies vs. Cookies

filename: `brownie`
Difficulty Level: `Easy`
Time Limit: `5 seconds`

Everyone is welcome to the UCF Programming Team practices, and many students take advantage of this opportunity. The main benefit is that these students improve their problem solving and programming skills. Another benefit is that the students enjoy the refreshments Dr. Orooji brings every week! Dr. O usually brings candies but sometimes he brings cookies or brownies. Brownies are very popular and don't usually last long, so Dr. O has to come up with some clever trick to make the brownies last longer (so that the students stay for the entire practice!). Well, the easiest solution is to cut the brownies in half; that will double the number of brownies.

The Problem:

Given the original number of brownies and the students wanting brownies, you are to keep track of the brownie count as Dr. O cuts them in half.

The Input:

The input provides the information about a practice. The first input line for the practice contains two integers (separated by a space): the number of students (between 1 and 30 inclusive) in the practice and the number of brownies (between 60 and 600 inclusive) Dr. O has brought that day. The next input line for the practice contains a positive integer, *m*, indicating how many groups of students approach the refreshment table to take brownies. This is followed by the number of students in each group, one number per line. Assume that the input values are valid, e.g., the number of students in a group will be at least 1 and it will not be greater than the number of students in the practice.

If a group of students is approaching the refreshment table and Dr. O notices that the number of remaining brownies is less than or equal to the number of students in the group, Dr. O cuts the brownies in half to be sure they won't be all gone after each student in the group grabs one brownie. Note that, if needed, Dr. O will cut the brownies more than once (as many times as needed). For example, if there are 3 brownies left and 24 students are approaching the table, Dr. O has to cut the brownies four times ($3 \rightarrow 6 \rightarrow 12 \rightarrow 24 \rightarrow 48$) to be sure the brownies won't be all gone after each student in the group grabs one.

The Output:

Print one output line for each group of students approaching the refreshment table. Each output line should provide the number of students in a group approaching the table and the number of brownies left after each of these students has grabbed one brownie (note that cutting in halves may occur before grabbing).

Sample Input	Sample Output
20 60 8 15 10 20 18 9 12 2 10	15 45 10 35 20 15 18 12 9 3 12 12 2 10 10 10
15 100 4 1 2 3 5	1 99 2 97 3 94 5 89

UCF "Practice" Local Contest — Aug 28, 2021

Lemonade Stand

filename: `lemonade`
Difficulty Level: `Medium`
Time Limit: `1 second`

You are running a lemonade stand and have the good fortune of knowing exactly how many cups of lemonade customers are going to want to buy on each day that you run the lemonade stand. You hate to turn any customer away, so you would like to make sure that you always have enough lemons and sugar to make the appropriate number of cups of lemonade on each day. Unfortunately, the cost of lemons and sugar change daily, so you have to choose on which days you buy each, and how much of each to buy. You can buy individual lemons and five pound bags of sugar. (Note that there are 16 ounces in one pound.) On the days you choose to buy ingredients, you buy them in the morning, before any sales are made. (You're an early riser, so you can always get to the store and back before any customers would come.) Note that you can buy as little or as much as you wish on any day to minimize your overall cost, i.e., you have enough startup money (capital) to buy as much as you wish on any day.

The Problem:

Given that you always want to have enough lemons and sugar to serve each customer, determine the minimum cost of buying those lemons and sugar.

The Input:

The first input line will have three space-separated positive integers: d *(*$1 \le d \le 1000$*)*, the number of days you'll run the lemonade stand, x *(*$1 \le x \le 10$*)*, the number of lemons needed to make a single cup of lemonade, and s *(*$1 \le s \le 10$*)*, the number of ounces of sugar needed to make a single cup of lemonade. The following d lines will contain data for days 1 through d, respectively. Each of these lines will have three integers separated by spaces: c *(*$1 \le c \le 1000$*)*, the number of cups sold for that day, p_l *(*$1 \le p_l \le 50$*)*, the price of a single lemon in cents for that day, and p_s *(*$1 \le p_s \le 500$*)*, the price of a five pound bag of sugar in cents for that day. Note that the extra sugar and lemon from each day carry over to the next day.

The Output:

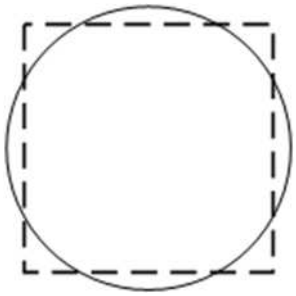
Print the minimum cost of supplies (in cents) necessary to make sure that no customer who wants a cup of lemonade gets turned away.

Sample Input	Sample Output
3 3 2 200 10 399 300 8 499 400 12 499	31977
2 5 10 9 10 199 8 20 99	1347

UCF "Practice" Local Contest — Aug 28, 2021

Rain Gauge

filename: `rain`
Difficulty Level: `Medium`
Time Limit: `1 second`



When going to your internship you got a nice apartment with a skylight. However, one crazy party later and you now have a square-shaped hole where your skylight used to be. Rather than telling the landlord, you decided you would "fix" it by putting a circular pot to collect the water but, as the saying goes, round peg square hole. You need to determine how much of the square the circle covers to help you determine if you should buy a larger pot. Don't worry about the area not covered; you can do multiplication and subtraction easily in your head.

The Problem:

Given the radius of a circular pot and the length of the square skylight, calculate the amount of skylight rain area covered by the pot assuming the two shapes have the same center (i.e., are coaxial) with respect to the direction rain falls from (up). In other words, the center of the square will be directly above the center of the circle. See the picture for an example; let up be the direction from above the page, while the dotted square is the skylight and the solid circle is the pot to collect water.

The Input:

There is one input line, it contains a pair of integers, s, r ($1 \le s \le 100$, $1 \le r \le 100$), which represents the length of the side of the skylight and the radius of the pot, respectively.

The Output:

Print a single decimal representing the area under the skylight that is covered by the pot. Round the answers to two decimal places (e.g., 1.234 rounds to 1.23 and 1.235 rounds to 1.24). For this problem, use 3.14159265358979 as the value of pi.

Sample Input	**Sample Output**
1 1	1.00
8 5	62.19
10 4	50.27

UCF "Practice" Local Contest — Aug 28, 2021

Balanced Strings

filename: `balance`
Difficulty Level: `Medium`
Time Limit: `1 second`

Being an individual obsessed with balancing and trying to find strings that meet that criteria, you have begun a new endeavor to study the curious structure of what we will call balanced strings.

Balanced strings are strings that maintain an equal ratio of consonants to vowels in all of their substrings. What? You don't know the difference between a consonant and a vowel? Vowels are the characters 'a', 'e', 'i', 'o', 'u' and sometimes 'y'. Actually, you don't like the character 'y' because this makes the problem much harder. What was the difficulty level of this problem again? Oh yeah Medium! We can't have a problem about consonants and vowels that makes 'y' sometimes a vowel! That would make the problem a Hard and too many hard problems is a very bad thing. Being obsessed with balance, you have decided that 'y' will be included with the vowels. That way, there are 6 vowels and 20 consonants. A nice balanced even number of both! Oh! I almost forgot! A consonant is any letter that is not a vowel. Also to simplify things (this is a Medium problem after all!), we will consider strings composed of lowercase letters only.

Now you are ready to understand balanced strings! A balanced string is a string that has an equal number of consonants and vowels in all of its substrings. Well… not all of its substrings. Just the substrings of even length! For example, the string "orooji" has the following set of even-length substrings: {"or", "ro", "oo", "oj", "ji", "oroo", "rooj", "ooji", "orooji"}. In that set the following substrings are unbalanced: {"oo", "oroo", "ooji", "orooji"}. That is, the substrings do not contain an equal number of vowels and consonants. So, the string "orooji" is not balanced. But a string like "arup" is balanced. The requisite even-length substrings are: {"ar", "ru", "up", "arup"} and all these substrings are balanced (have the same number of vowels and consonants), thus the string "arup" is balanced. Note that a balanced string may contain an odd number of characters, e.g., the string "ali" is balanced since all its even-length substrings ({"al", "li"}) are balanced.

Now you want to take words and erase some of the letters and replace them with new letters to form balanced strings. Since this is a Medium problem, we've taken the liberty of erasing some of the letters for you and replaced them with '?' characters. See! The problem is already halfway solved!

The Problem:

Given a string of lowercase letters and '?' characters, count the number of ways to replace all the '?' with lowercase letters such that the string results in a balanced string. Two ways are considered different if there exists some i such that the i^{th} character in one string differs from the i^{th} character of the other string. Note that all the '?' do not have to be replaced by the same letter.

The Input:

There is one input line, it contains a string. The string contains only lowercase letters and/or '?' characters. Assume the input string has at least one character and at most 100 characters.

The Output:

Print the number of ways to replace the question marks with lower case letters such that the string results in a balanced string. It is guaranteed that the output value will fit in a signed 64-bit integer for the string we provide.

Sample Input	**Sample Output**
orooji	0
al?	6
a?i	20
g?ha	6
a?u?	400
????????????????	11179524096000000000
arup	1

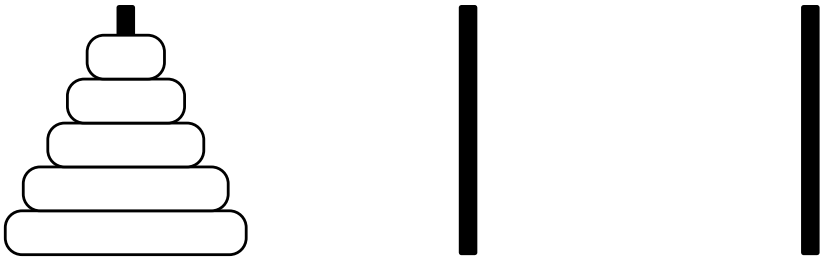
Note: The first Sample Input and the last Sample Input do not have any '?'. The first string is not balanced so there is no way of replacing all '?' to make it balanced, thus the output is zero. However, the last string is balanced so there is one way of replacing all '?' to make it balanced, thus the output is 1.

UCF "Practice" Local Contest — Aug 28, 2021

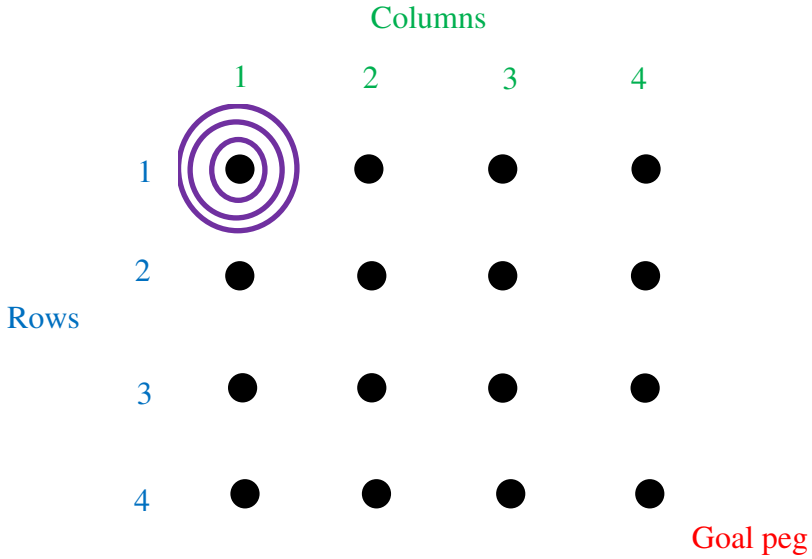
Towers of Hanoi Grid

filename: `hanoi`
Difficulty Level: `Medium-Hard`
Time Limit: `3 seconds`

Towers of Hanoi is a rather famous problem for computer scientists as it is an excellent exercise in recursion. For those of you unfamiliar, here is the classic problem. You are given three pegs. On the first peg, there are d disks placed in decreasing order of size (as placed on the peg). The objective of the game is to move the entire tower from the first peg to the last peg. In each move, you are only allowed to move a single disk from the top of one stack to another stack. For the entire game, no disk of larger size is ever allowed to be placed on top of a disk of smaller size. The goal of the puzzle is to move the tower in the minimum number of moves.



In our problem, we will instead have an $n \times n$ grid of pegs. The rows are numbered top to bottom from 1 to n, while the columns are similarly labeled, from left to right, 1 to n. The original tower is placed on the top left peg $(1, 1)$. The goal is to move the tower to the bottom right peg (n, n) in the minimum number of moves possible.



Our game will have some different but related rules:

- For a peg (r, c) at row r and column c, you may only move the top-most disk from peg (r, c) to peg $(r + 1, c)$ or peg $(r, c + 1)$, in a single move and only if such a pair of pegs exists.
- Only pegs $(1, 1)$ and/or (n, n) may have more than one disk at any time; all other pegs may contain at most one disk.
- You can choose any peg for each move.
- You still may never place a larger disk on a smaller disk.

The Problem:

Given the number of disks on the starting peg and the number *n* described above, determine the minimum number of moves to solve our Tower of Hanoi Grid puzzle.

The Input:

There is one input line, it contains two integers, *d* ($2 \leq d \leq 100$) and *n* ($2 \leq n \leq 100$), representing the number of disks and the dimensions of the grid, respectively.

The Output:

Print the minimum number of moves to solve the Tower of Hanoi Grid puzzle. If it is not possible to move the disks from peg $(1,1)$ to peg (n, n), output "impossible" (without quotes).

Sample Input	**Sample Output**
2 2	4
100 8	impossible
3 100	594