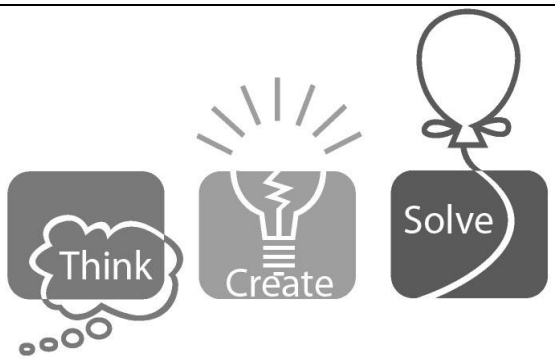
University of Central Florida



2020 (Fall) "Practice" Local Programming Contest

Problems			
Problem#	Difficulty Level	Filename	Problem Name
1	Easy	vowel	Vowel Count
2	Easy	soccer	Soccer Standings
3	Easy	frog	Jumping Frog
4	Medium	sushi	Fujiyama Thursday
5	Medium	email	Chain Email
6	Medium	microwave	Faster Microwaving

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

For example, if you are solving Fujiyama Thursday:

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

UCF "Practice" Local Contest — Aug 29, 2020

Vowel Count

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

Dr. Orooji noticed that his name has more vowels than consonants. Since he likes meeting people like himself, he has asked you to write a program to help him identify such names.

The Problem:

Given a name, determine whether or not it has more vowels than consonants. Assume the vowels are "aeiou".

The Input:

There is one input line, it contains a name. The name starts in column 1 and consists of 1-20 lowercase letters (assume the name will not contain any other characters).

The Output:

Print a 1 (one) or 0 (zero) indicating whether or not the name has more vowels than consonants.

Sample Input	**Sample Output**
`ali`	`1`
`arup`	`0`
`travis`	`0`
`orooji`	`1`

UCF "Practice" Local Contest — Aug 29, 2020

Soccer Standings

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

In a soccer match, a team either earns a win, tie or loss. A win is worth 3 points, a tie is worth 1 point, and a loss is worth 0 points. Unfortunately, due to poor record-keeping, some leagues have only saved the number of total matches played and the number of points each team has earned. One of these leagues has asked you to write a program to recreate the possible combinations of wins, ties and losses for a team in the league.

The Problem:

Given the number of games played by a soccer team in a season and the number of points earned by the team, list each possible combination of wins, ties and losses that the team could have gotten to achieve the given total points.

The Input:

There is one input line, it contains two space separated integers: g *(0 < g ≤ 100),* and p *(0 ≤ p ≤ 300)*, representing the number of games played and the total points earned by the team, respectively. It is guaranteed that there is at least one possible combination of wins, ties and losses that is consistent with the given information for the team.

The Output:

Print the possible records, each on a separate line with the format:

`w-t-l`

where `w` is the number of wins, `t` is the number of ties and `l` is the number of losses. Print these by descending order of wins.

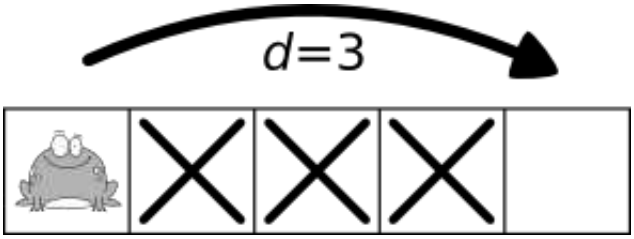
Sample Input	**Sample Output**
`6 10`	`3-1-2` `2-4-0`
`1 3`	`1-0-0`
`4 4`	`1-1-2` `0-4-0`

UCF "Practice" Local Contest — Aug 29, 2020

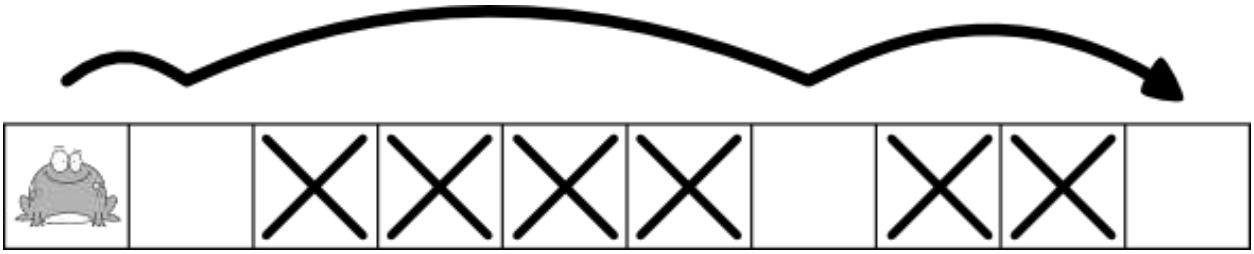
Jumping Frog

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

Freddy the frog is trying to go get a bite to eat. The problem is Freddy lives far away from all the restaurants he likes. Freddy is a capable frog, able to jump over a large number of cells. Unfortunately, on a given day, Freddy may be too hungry to jump very far. On a given day, he can only jump over at most *d* cells (note that he can also jump over fewer cells).



Another complication for Freddy is the path he jumps across is always under construction. Some of the cells are blocked off! Freddy doesn't want to land in a cell under construction but is allowed to jump over them.



Freddy starts off in the first cell and must travel to the last cell where his destination restaurant resides. He would like to know if he can reach the last cell and how quickly he can reach it. Freddy always jumps towards his destination.

The Problem:

Given a description of cells, determine the minimum number of jumps Freddy can make to reach the restaurant.

The Input:

The first input line contains two integers, c $(2 \leq c \leq 50)$ and d $(0 \leq d \leq 50)$, representing (respectively) the number of cells in the path from Freddy's home to the restaurant (including his home and the restaurant) and the maximum number of cells Freddy can jump over in a single jump on that day. The second input line is a string consisting of c characters containing only '`.`'

and ‘X’ characters where ‘.’ represents that the cell is okay for Freddy to occupy and ‘X’ represents that the cell is blocked by construction. The first and last characters of the string represent Freddy’s home and the restaurant, respectively; these two locations will never be blocked.

The Output:

Print the minimum number of jumps it takes Freddy to reach the restaurant. If it is not possible to reach the restaurant, print the number 0 instead.

Sample Input	**Sample Output**
8 3 .XX.X.X.	2
3 50 ...	1
8 1 ...XX...	0
10 4 ..XXXX.XX.	3

UCF "Practice" Local Contest — Aug 29, 2020

Fujiyama Thursday

filename: `sushi`
Difficulty Level: `Medium`
Time Limit: `5 seconds`

During the past year, the UCF programming team members started a weekly dinner at Fujiyama Sushi. Usually, the programming team members wait for everyone to arrive to start eating, but as a wise programming team coach from the north of campus always says: "regionals is coming." Heeding this ominous warning, the programming team members need to spend more time to prepare. To spend more time practicing, the programming team members will start eating as they arrive. However, they will still wait until everyone is done eating to leave Fujiyama.

Each car the team members are taking holds exactly four people. As each car may take a different amount of time to arrive at Fujiyama and each team member may have different eating speeds, it is important to assign team members to cars in a careful manner. Any team member can be assigned to any car as all team members can drive any of the cars.

The Problem:

Given the time it takes for cars to arrive and eating speeds of the team members, determine the minimum amount of time needed for all the team members to finish eating if the team members are assigned to cars optimally.

The Input:

The first input line for the trip contains a single integer, c *(*$1 \leq c \leq 50$*)*, representing the number of cars going to Fujiyama. The second input line contains c integers, d_i *(*$1 \leq d_i \leq 45$*)*, representing the time (in minutes) it takes for the i^{th} car to arrive to Fujiyama. The third input line contains $4*c$ integers, t_j *(*$1 \leq t_j \leq 75$*)*, representing the time (in minutes) it takes for the j^{th} team member to finish eating.

The Output:

Print a single integer representing the minimum amount of time (in minutes) for all the team members to finish eating.

Sample Input	Sample Output
1 40 1 2 3 4	44
2 10 20 5 6 3 4 8 9 1 2	24
3 15 20 20 10 10 10 10 20 20 20 20 30 30 30 30	45

UCF "Practice" Local Contest — Aug 29, 2020

Chain Email

filename: `email`
Difficulty Level: `Medium`
Time Limit: `5 seconds`

A chain email is an email that people receive and then forward to all of their friends. This sort of email is very common amongst elderly people, who have notably bad memories. Elderly people's memories are so bad that if they ever receive a chain email they will forward it to all of their contacts. This can become very problematic when elderly people continually send the same email to each other. For instance, if two people have each other in their contacts and if either of them receive a chain email they will continually send the email to each other back and forth forever. Email companies are worried that this will result in a massive amount of storage loss on their servers and have asked you to determine if a specific person were to start a chain email, who would receive that email forever.

The Problem:

Given each elderly person's contacts and which elderly person will be starting a chain email, determine who will be indefinitely receiving emails.

The Input:

The first line of input will have two single-space-separated integers, p ($1 \leq p \leq 50$), indicating the number of people who use the email service and, s ($1 \leq s \leq p$), indicating the source of the chain email, where each person is labeled from 1 to p. Following this will be a single line with the names of all of the people, from person 1 to person p, who use the email service, each separated by exactly one space. All names will contain alphabetic characters only and be between 1 and 19 characters (inclusive) in length. Following this will be p lines. The i^{th} line will describe the contact list of the i^{th} person. This description will consist of an integer, m ($0 \leq m < p$), indicating the number of contacts this person has, followed by the 1-based index of each of the contacts, each separated by exactly one space. It's guaranteed that no one will contain themselves as a contact.

The Output:

Print the names of all of the people who will infinitely receive chain emails, assuming that everyone continually forwards the email to all of their contacts. Each name should be followed by a space. List these contacts in the order that they appear in the input. If no one will infinitely receive chain emails, then print "`Safe chain email!`" instead.

Sample Input	Sample Output
3 1 James Sarah John 2 2 3 2 1 3 2 1 2	James Sarah John
3 1 James Sarah John 2 2 3 0 0	Safe chain email!
6 3 Ali Matt Glenn Sumon Arup Chris 2 3 5 0 1 4 1 1 1 2 2 5 4	Ali Matt Glenn Sumon Arup

UCF "Practice" Local Contest — Aug 29, 2020

Faster Microwaving

filename: `microwave`
Difficulty Level: `Medium`
Time Limit: `5 seconds`

Chris likes getting his food quickly so he cooks a lot using the microwave. However, he is frustrated by how microwave makers seem not to communicate well with makers of microwavable foods. For example, many microwaves have a "popcorn" button, but most microwave popcorn instructions say "Do not use the 'popcorn' button." To avoid any chance of ruining his food, Chris uses only timed cooking, entering the time to cook each item in minutes and seconds (in MM:SS format) on the numbered buttons of the microwave. Since there is one digit per button, he has to press one digit at a time and move his finger between different digits, which is tedious and annoying because he's hungry. One nice thing is that there is no button for the ":" as the microwave always interprets the last 2 digits as seconds, and inserts the ":" appropriately—however, it does not enforce a restriction that the last two digits are 59 seconds or less; if Chris presses 1, then 9, then 0 for a time of "1:90" it will cook for 1 minute and 90 seconds, which is the same as 2 minutes and 30 seconds.

Chris would like to be able to enter the cooking times more quickly. He notices that it takes 1 "moment" (a unit of time just under half a second) to press each digit's button firmly, and it also takes 1 moment (the same unit of time) to move his finger away from one digit's button to find the button for a different digit. Therefore, to enter a time "4:00" takes 4 moments in total—one to press "4", one to move from "4" to "0", one to press "0", and one to press "0" again immediately, without having to find the button. It also takes 4 moments to enter "4:45" (press 4, then 4 again, then move from 4 to 5, then press 5), but it takes 5 moments to enter "4:30" (4, then move, then 3, then move, then 0).

After some experimentation, Chris devises the following plan to enter faster cooking times that are reasonably close to the recommended times in the cooking instructions for each item:

1. Based on the microwavable item type, consider using a range of proposed cooking times that are each within a small percent above or below the recommended cooking time. For example, using 10% with a recommended cooking time of 2 minutes and 30 seconds (2:30), the proposed cooking times would be the range of times from 2:15 to 2:45 inclusive.
2. Find the sequence of digits (buttons) that takes the lowest total moments to enter out of *any* of the proposed cooking times in the range.
3. If there are multiple sequences of digits that have the same lowest total moments, choose the sequence that yields an actual cooking time that is closest to the recommended cooking time.

Chris has verified that the above plan always results in a unique answer so you may assume so.

The Problem:

Given the recommended cooking time for a microwavable item, and a percent to use for the range of proposed cooking times, output the sequence of digits that Chris should press in order to start the microwave as fast as possible, according to his plan.

The Input:

The first input line contains only the recommended cooking time for the microwavable item, which consists of exactly 5 characters in the format *MM*:*SS* with 2-digit minutes *MM* $(00 \leq MM \leq 20)$ and 2-digit seconds *SS* $(SS \in \{00, 15, 30, 45\})$. The recommended cooking time will be at least 00:15 (15 seconds). The second input line contains only an integer p $(2 \leq p \leq 10)$, which is the percent of the recommended cooking time that defines the range of lower and higher proposed cooking times.

The Output:

Print the exact digits that should be pressed, in the order they should be pressed.

Note that the seconds are always integers and the time must be "within" the percent range. For example, for a recommended cooking time of 00:45 and 10%, the range of proposed cooking times is 41 seconds to 49 seconds (45 ± 4, because 40 and 50 are not within 10% of the recommended time).

Sample Input	**Sample Output**
01:30 4	88
00:30 10	33
06:00 8	555