Spring 2014 ACM Programming Contest

Overview

Welcome to the Spring 2014 ACM Programming Competition! This is a contest wherein you compete amongst your fellow students, and professors, in an effort to solve a set of programming problems as fast as possible.

Problems

There are seven problems of (roughly) increasing difficulty. The team who solves the most problems is declared the winner. In the event of a tie, the team that finished their last problem earliest wins.

Each problem has a general description, details on the formats for input and output, and example input and output. They also have code to read the input in C++ and Python. You may use this code as the starting point for your solution, but don't have to. Each problem also comes with a Scratch file that has the sample input pre-loaded.

Input is done from standard input and output is done to standard output. *Do not open any files inside of your program.* Output of your program must match that of the correct output *exactly*. You can solve the problems in C, C++, Haskell, Java, PHP, Python, Racket, Ruby, R, or Scratch. Up to date versions of these languages will be used to judge your entries. You should save your code using the standard extension for your language of choice (.c .cpp .hs .java .php .py .rkt .rb .R .sb respectively).

Rules

You are allowed to have any printed material such as books or printouts. You are also allowed to access standard library documentation for your language of choice. *No other use of the internet is permitted.* You are also not allowed to copy and paste code from any source be it a thumb drive, or another file that you have. Only one computer is allowed to a team.

Logging In

To log in to the system, direct your browser to `http://71.63.64.96`. You will then need to enter your team name and password which are provided for you. This will bring up the interface for you to submit solutions to the problems, ask questions (which will be broadcast to all participants), and see how your team is doing.

Submitting

To submit a solution, first make sure that you select the correct problem letter on the top of the site. Then click the "Choose File" button. Then choose the file that contains your source code (be sure it has the correct extension). After that it will be submitted for judging. It will receive an automatic judging response which is marked pending. Your solution will then be checked manually and a final response will be assigned.

DO NOT TURN OVER THIS SHEET UNTIL THE CONTEST STARTS

Problem A: Keys

As Link works his way through dungeons, he often comes across locked doors to which he needs to find keys. Sometimes he finds keys in other parts of the dungeon, and sometimes he finds them after defeating enemies. He needs to find at least as many keys as there are locked doors in order to get out of the dungeon. After using a key, i is destroyed and can't be used again. Any key can be used in any door (locksmithing in Hyrule is not very advanced). Having extra keys is also handy as the next part of the dungeon may require them.

You should write a program that takes in the number of keys Link has available, in addition to the number of locked doors. You should then decide if Link has enough keys to get through the dungeon, as well as the number of extras he has (if any). If Link does not have enough keys to get out of the dungeon, you should print the number he needs to find.

Input

The first line of input is the number of keys Link has. The second line is the number of locked doors he must get through.

Output

If Link has exactly enough keys to get through the locked doors, you should output "Link has the correct number of keys."

If Link has more keys than he needs, you should output "Link gets through the dungeon with X extra keys." where X is the number of keys he will have left over after opening each door in the dungeon.

If Link does not have enough keys to escape the dungeon, you should output "Link needs X more keys to get through the dungeon." where X is the number of additional keys he would need to find to open all of the doors.

Sample Input

```
11
8
```

Sample Output

```
Link gets through the dungeon with 3 extra keys.
```

C++ Input Code

```
#include <iostream>
using namespace std;

int main( ) {
    int keys, locks;
    cin >> keys >> locks;


    return 0;
}
```

Python Input Code

```
keys = int(input( ))
locks = int(input( ))
```

Problem B: Choosing Socks

Princess Zelda is being held captive in a tower by Ganon. After a few months, Link comes to rescue her in the middle of the night. The tower is dark and they cannot turn on their lantern so as not to alert the guards.

Princess Zelda needs to put socks on, but she needs to grab socks from her drawer without being able to look at them. She has socks in several different varieties, and would still like to have a perfect match.

You should write a program that decides the minimum number of socks she needs to grab to guarantee a perfect match, based on the number and variety of socks she has.

Input

The first line of input gives the number of varieties of socks Zelda has. Next will be one line of input for each variety giving the number of socks of that variety.

Output

You should output "Zelda would need to grab X socks.", where X is the minimum number of socks Zelda needs to grab in order to guarantee a matching pair.

Sample Input

```
3
7
1
9
```

Sample Output

```
Zelda would need to grab 4 socks.
```

C++ Input Code

```
#include <iostream>
using namespace std;

int main( ) {
    int num;
    int types[100];
    cin >> num;
    for (int i = 0; i < num; i++) {
        cin >> types[i];
    }

    return 0;
}
```

Python Input Code

```
num = int(input( ))
types = []
for i in range(num):
    types.append(int(input( )))
```

Problem C: A Game of Chance

There is a popular game of chance in Hyrule wherein the player pays a certain amount of Rupees to be able to open one of a set of chests (less than 100) containing Rupees. If the players is lucky, they will open a chest that has *more* Rupees than they paid.

A Hyrulian gambler would like to know if the *expected value* they would win is less than, or greater than the cost to play. The expected value is calculated by multiplying the probability of each possible value by its magnitude, then summing them up. For example, if there are three chests, which contain 1, 50, and 300 Rupees respectively, then the expected winnings would be:

```
    ⅓ * 1 + ⅓ * 50 + ⅓ * 300 = 117
```

To be conservative, the gambler would like to round the expected winnings down if it is not an integer.

Input

The first line of input is an integer representing the cost of playing the game. The second line of input is the number of chests in the game. Subsequent lines of input will contain the amounts of Rupees in each chest - one for each chest in the game.

Output

Output should consist of two lines. The first is "The expected winnings are X." where X is the expected winnings.

The second line is either "The game is worth playing!" or "The game is NOT worth playing!" based on if the expected winnings are greater than the cost. If they are equal, print that the game is not worth playing.

Sample Input

```
150
4
0
10
150
500
```

Sample Output

```
The expected winnings are 165.
The game is worth playing!
```

C++ Input Code

```
#include <iostream>
using namespace std;

int main( ) {
    int cost;
    int num;
    int chests[100];
    cin >> cost >> num;
    for (int i = 0; i < num; i++) {
        cin >> chests[i];
    }

    return 0;
}
```

Python Input Code

```
# read input
cost = int(input( ))
num  = int(input( ))
chests = []
for i in range(num):
    chests.append(int(input( )))
```

Problem D: Choosing Gloves

Princess Zelda is being held captive in a tower by Ganon. After a few months, Link comes to rescue her in the middle of the night. The tower is dark and they cannot turn on their lantern so as not to alert the guards.

Princess Zelda needs to put gloves on, but she needs to grab gloves from her drawer without being able to look at them. She has pairs of gloves in several different varieties, and would still like to have a perfect match.

You should write a program that decides the minimum number of gloves she needs to grab to guarantee a perfect match, based on the number and variety of gloves she has.

Note that gloves are different from socks in that gloves are for a specific hand. So getting two gloves of the same variety is no help if they are both left-handed.

Input

The first line of input gives the number of varieties of gloves Zelda has. Next will be one line of input for each variety giving the number of **pairs of** gloves of that variety.

Output

You should output "Zelda would need to grab X gloves.", where X is the minimum number of gloves Zelda needs to grab in order to guarantee a matching pair.

Sample Input

```
3
5
3
2
```

Sample Output

```
Zelda would need to grab 11 gloves.
```

C++ Input Code

```
#include <iostream>
using namespace std;

int main( ) {
    int num;
    int types[100];
    cin >> num;
    for (int i = 0; i < num; i++) {
        cin >> types[i];
    }

    return 0;
}
```

Python Input Code

```
num = int(input( ))
types = []
for i in range(num):
    types.append(int(input( )))
```

Problem E: Messages From the Past

There are several ancient artifacts around Hyrule that contain text in the ancient Hylian language. Sometimes Link needs to read the messages on these artifacts on his travels. For example, some palaces have inscriptions in ancient Hylian that describe how to enter.

The Book of Mudora is the only known source of inforation on ancient Hylian. The book contains the information needed to translate between ancient Hylian and the modern language of Hyrule, as well as translations of ancient proverbs. Upon finding this book, Link would like a program that could translate messages for him automatically.

Ancient Hylian actually uses the same words and grammar as the modern version, but the way they are written is quite different. Rather than write the characters of a message in the order they appear, ancient Hylian is written by indicating all the positions a character appears.

Each character in the messages is given only once. After the character appears, an integer is given indicating which positions in the message have that character. If there are more than one, they are spearated with semi-colons. For example, the message:

```
Hello!
```

Would be wrritten:

```
!6e2H1l3;4o5
```

This means that the exclamation point is the 6th character, the e is the second character, the H is the first character, l is the third and fourth character, and o is the fifth character.

It's not surprising that this way of writing messages was abandoned!

Ancient Hylian messages never include numbers or semi-colons as these have special meanings in the way that the messages are written.

Each character can only be specified once in ancient Hylian. So, for example, the message "Hello!" could not have been written:

```
!6e2H1l3o5l4
```

Since "l" appears twice.

Typically, all punctuation is given first, then the letters are given in alphabetical order. Some writers do not follow this rule, however.

Input

The input to your program will be one line of ancient Hylian text.

Output

Output should be the modern translation of the ancient text.

Sample Input

```
!22 5;9;16A11D4;21;E8;14F1H7I2M10N3O19R15;20S12;17T6;13W18
```

Sample Output

```
FIND THE MASTER SWORD!
```

C++ Input Code

```
#include <iostream>
using namespace std;

int main( ) {
    string line;
    getline(cin, line);
    return 0;
}
```

Python Input Code

```
line = input( )
```

Problem F: Smithy's Dilemna

The Smithy in Kakariko Village forges weapons and armor, and also fixes and tempers existing equipment. He is one of the only Blacksmiths in Hyrule and is extremely busy (especially since his partner mysteriously disappeared).

He has a number of different jobs that each take a different number of days to complete. He can only work on one job at a time, and each job takes a whole number of days.

The Smithy also prices his work such that customers get a discount if they have to wait for him to begin their job. He does this by giving a discount of a certain amount of Ruppees per day before starting a given job.

He has asked you to write a program that will help him maximize his total profits by ordering the jobs in such a way that he minimizes the discounts he has to give out.

Input

The first line of input is an integer, N, that gives the number of jobs Smithy has. Following that are N lines which each contain a pair of integers: the number of days the job will take, and the number of Ruppees discount per day that Smithy has agreed to.

Output

Your program should output the sequence of jobs Smithy should work on in order to minimize the discounts he gives out. The output should consist of the numbers 1 through N, where each number references the position of the job in the input.

If there are multiple solutions with the same total discount, you should output the first one in a lexicographic order (e.g. between 2,3,4,1 and 1,4,3,2; choose 1,4,3,2.)

Sample Input

```
4
3 4
1 1000
2 2
5 5
```

Sample Output

```
2 1 3 4
```

C++ Input Code

```
#include <iostream>
using namespace std;

int main( ) {
    int jobs[1000][2];
    int count;
    cin >> count;

    for (int i = 0; i < count; i++) {
        cin >> jobs[i][0] >> jobs[i][1];
    }

    return 0;
}
```

Python Input Code

```
count = int(input( ))
jobs = []
for i in range(count):
        jobs.append(list(map(int, input( ).split( ))))
```

Problem G: Triforce

The Triforce is a sacred artifact left by the three Golden Goddesses when Hyrule was first created. The Triforce is extremely powerful, whoever touches it will have their deepest wishes granted.

The Triforce is a set of three inter-locked golden triangles: each one symbolizing one of the Golden Goddesses. You should write a program that prints a representation of the Triforce.

Input

Input is a single integer giving the size of one side of the Triforce. This number will always be even and greater than 4.

Output

Your program should output three equilateral triangles in the form of the Triforce. The Triforce should be composed of pound characters (#) and have space between each pound character. (See Sample Output for example).

Sample Input

```
16
```

Sample Output

```
               #
              # #
             # # #
            # # # #
           # # # # #
          # # # # # #
         # # # # # # #
        # # # # # # # #
       #               #
      # #             # #
     # # #           # # #
    # # # #         # # # #
   # # # # #       # # # # #
  # # # # # #     # # # # # #
 # # # # # # #   # # # # # # #
# # # # # # # # # # # # # # # #
```

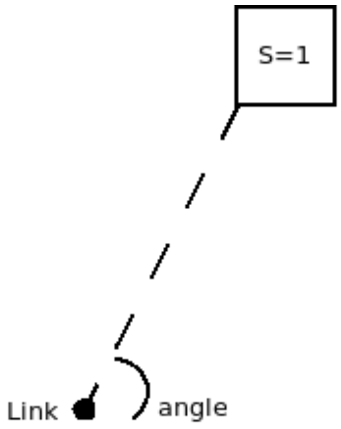
C++ Input Code

```
#include <iostream>
using namespace std;

int main( ) {
    int size;
    int >> size;

    return 0;
}
```

Python Input Code

```
size = int(input( ))
```

Problem H: Target Practice



Link keeps his archery skills sharp by practicing with his bow and arrow on targets near his home. He uses his bow to defend himself and slay enemies and needs to stay good.

You will write a program that, given the location of Link, the angle he fires at, and the location of his target, will compute whether or not Link will hit his target.

This problem can be seen as looking down on Link and his target from a bird's eye view. Link's position can be represented as a point in two-dimensional space. The target can be represented as a square with each side one meter long. The square target is always aligned on the cardinal directions. The angle Link fires at is given in degrees (where 0 degrees is due east).

You can assume that Link fires his arrow at the right height to hit his target, and only need to consider whether he hits it in two dimensions.

Input

The first line of input is the X and Y coordinates of Link. The second line of input is the X and Y coordinates of the center of the target. The third line of input contains the angle which Link shoots at.

All values will be non-negative, and the angle will be less than 360.

Output

You should output either "Link hits the target!" or "Link does not hit the target!".

Sample Input

```
5.0 5.0
25.0 25.0
45.0
```

Sample Output

```
Link hits the target!
```

C++ Input Code

```
#include <iostream>
using namespace std;

int main( ) {
    double linkX, linkY, targetX, targetY, angle;
    cin >> linkX >> linkY >> targetX >> targetY >> angle;

    return 0;
}
```

Python Input Code

```
linkX, linkY = tuple(map(float, input( ).split( )))
targetX, targetY = tuple(map(float, input( ).split( )))
angle = float(input( ))
```

Problem I: Two Worlds

The *Dark World* was transformed into a twisted version of the *Light World* by Ganon using the Triforce. The light world and dark world are different versions of the same world; each location in one world has a corresponding location in the other. Features such as rivers, towns, mountains and trees are similar between the two worlds, but there are some differences between the two.

There are portals in the light world that transport whoever steps on them into the dark world. Portals always transport one to the corresponding location in the dark world There are no portals from the dark world back into the light world, leaving many unfortunate souls trapped in the dark world. Luckily, Link has a magic mirror which allows him to transport himself back to the light world any time he chooses. Like portals, the magic mirror transports Link back into the corresponding location in the light world.

Sometimes, to get to a particular location, Link must pass between the two worlds multiple times. For example, if there is an impassable ridge in the light world, Link can find a portal to the dark world, pass over where the ridge would be, then use the mirror to get back to the light world.

Your task is to help Link find his way from one point to another (if possible). Link's starting point and destination could be in either the light or dark world, and Link may need to pass between worlds multiple times to get to his destination. Your program will report whether or not it is possible for Link to get from the start position to the end position.

Link can only move in cardinal directions (not diagonally).

Input

The first line of input will be the number of rows in the map for each world. Next there will be one line of input for each row in the map. These represent the map for the light world. They are made up of the following characters:

Character	Meaning
-	A tile Link can walk on, such as grass.
L	Link's starting position.
D	Link's intended destination.
T	A tree, which Link cannot walk on.

W	A body of water, which Link cannot walk on (as he doesn't have Zora's flippers yet).
M	A mountain, or ridge, that Link cannot walk on.
P	A portal to the dark world. These only appear in the light world.

After the map for the light world, there will be a blank line. Next, there is a set of lines representing the map of the dark world. It is the same size as the light world, and is composed of the same characters (except that the dark world has no portals).

There will only be one "L" tile and only one "D" tile, but each one may either be in the light world or the dark world. Each map will have 100 or fewer rows.

Whenever Link is transported from one world to the other, he goes to the corresponding location. For example, if he steps on a portal in the third row and fourth column of the light world, he will appear in the third row and fourth column of the dark world.

Output

Your program should simply output whether or not Link can make it from his starting point to his destination. You should output "Link can make it to his destination!" if he can, and "Link can NOT make it to his destination!" if he cannot.

Sample Input

```
8
---W-----T---M-P
-L-W---------M--
---W----TT---M--
---W----TT------
----W--------MMM
----W--------M--
-T--W--T-----M-D
P--W---------M--

---W-----T---M--
-------------M--
---W----TT---M--
---W----TT---M--
----W--------M--
----W--------M--
-T--W--T-----M--
---W---------M--
```

Sample Output

```
Link can make it to his destination!
```

C++ Input Code

```
#include <iostream>
#include <string>
using namespace std;

int main( ) {
    // read and height
    int height;
    cin >> height;

    // read in the light world
    string light[100];
    for (int i = 0; i < height; i++) {
        cin >> light[i];
    }

    // read in the dark world
    string dark[100];
    for (int i = 0; i < height; i++) {
        cin >> dark[i];
    }

    return 0;
}
```

Python Input Code

```
# read height
heigh = int(input( ))

# read the light world in
light = []
for i in range(height):
    light.append(input( ))

# skip the blank line
input( )

# read the dark world in
dark = []
for i in range(height):
```

```
dark.append(input( ))
```

Problem J: Wolf Transformations

When Link enters in area shrouded in Twilight, he is transformed into Wolf form. When in Wolf form, some things are more difficult for Link, whereas other things are easier. Link spends some time trapped in Wolf form, but eventually recovers the Master sword which allows him to transform at will.

In Wolf form, Link is able to run faster than he can in is normal, Hylain form. However, he is able to open doors much faster as himself (due to his hands). He also takes some amount of time to transform back and forth.

If Link needs to travel along a path with doors, he wants to know what the fastest way to do it would be.

As a wolf, Link can run 10 meters per second, while he can only run 4 meters per second normally. It takes Link 8 seconds to open a door in Wolf form, and only 1 second normally. It takes him 5 seconds to transform to and from Wolf form.

You should write a program that, given the distance Link needs to travel, and the locations of all of the doors in that distance, the shortest time he could cover the distance, along with how many transformations he would make.

Link always begins in normal, Hylian form.

Input

The first line of input is the distance, in meters that Link needs to travel. The second line will contain the number of doors. There will be between 1 and 15 doors. The third line contains the locations of the doors along that path, separated by spaces.

Output

Your output should be of the form:

```
Link could travel this distance in X seconds, by transforming Y times.
```

Where X is the minimum number of seconds that Link could travel the distance and Y is the number of times he would need to transform to do this. If Y is equal to 1, you should output "1 time" instead of "1 times".

Small rounding errors in the time will be accepted for this problem.

Sample Input

```
100.0
4
10.0 80.0 85.0 90.0
```

Sample Output

```
Link could travel this distance in 28.500000 seconds, by transforming 2 times.
```

C++ Input Code

```cpp
#include <iostream>
using namespace std;

int main( ) {
    double distance;
    int ndoors;
    double doors[15];

    /* read all input in */
    cin >> distance;
    cin >> ndoors;
    for (int i = 0; i < ndoors; i++) {
        cin >> doors[i];
    }

    cout << distance << endl << ndoors << endl;
    for (int i = 0; i < ndoors; i++) {
        cout << doors[i] << " ";
    }

    return 0;
}
```

Python Input Code

```python
distance = float(input( ))
ndoors = int(input( ))
doors = list(map(float, input( ).split( )))
```