
# UAB CIS High School Programming Contest

## May 17, 2014

Each problem in this packet contains a brief description, followed by two example test cases of a successful implementation.

The example input and output shown for each question should be regarded only as examples. We will test your programs on the examples provided with each problem, as well as several other test cases generated by the judges. **Be sure to follow the input and output formatting exactly, printing each output answer on its own line.** Extraneous or malformed output will result in a failed submission.

If your program fails, a result will be returned by the submission system stating the counterexample test case that caused your program to be judged incorrect. Each incorrect answer will be incur a 20 minute time penalty, but this penalty will only be applied to your score if the problem is eventually answered correctly. As noted in the rules, the overall ranks will be determined first by the number of problems completed, and then by your time score (including penalties).

Please pay close attention to the directions in each problem description. In some cases, assumptions are stated about the limitations of the input, which are designed so that you do not have to consider difficult cases or perform input validation.

If you get stuck on a problem, you are encouraged to jump around and try different problems, as an eventual correct answer on a problem is better than no answer at all.

**A. Simple Checksum**

A large part of communicating messages over a network is being able to verify that the message has arrived on the other end intact. To accomplish this, the sender of the message calculates a checksum and sends it with the original message, allowing the recipient to check the message integrity by calculating the checksum of the message they receive and comparing it to the sent checksum. As part of a new RFC protocol, you have been tasked to implement a program that determines whether or not a given message is valid, based on the provided message and checksum.

This checksum can be calculated by using the first and last characters of the string, followed by the string length. Your goal is to determine the validity of the provided message by calculating its checksum and comparing it to the one provided.

Input:
Each test case consists of one line containing a one word message followed by a space and the checksum string. The word will always contain at least 2 characters and the checksum will never be empty.

Output:
Print 'Valid', if the checksum you calculate is identical to the one provided, otherwise print 'Invalid'.

**Sample Inputs:**
IHAVEPIZZA IA10
THISISNOTVALID TD123

**Sample Outputs:**
Valid
Invalid

**B. WERTYU**


A common typing error is to place your hands on the keyboard one row to the right of the correct position. Then "Q" is typed as "W" and "J" is typed as "K" and so on. Your task is to decode a message typed in this manner.

Input:
Input for each test case consists of one line of text. Each line may contain digits, spaces, uppercase letters (except "Q", "A", "Z"), or punctuation shown above [except back-quote (')]. Keys labeled with words [Tab, BackSp, Control, etc.] are not represented in the input. All input strings will have at least one character.

Output:
You are to replace each letter or punctuation symbol by the one immediately to its left on the QWERTY keyboard shown above. Spaces in the input should be echoed in the output.

**Sample Inputs:**
O S, GOMR YPFSU/
HPPF ;IVL PM YJR VPMYRDY/

**Sample Outputs:**
I AM FINE TODAY.
GOOD LUCK ON THE CONTEST.

**C. Jacobi's Grader**

While your professor, Dr. Jacobi, is a brilliant mathematician, he has often expressed his frustrations with spending his limited free time having to grade assignments. For his online courses, the university has generously provided him with Chalkboard™, which will allow him to create quizzes and homework assignments for his students to complete online. He can then download their answers for manual grading. Unfortunately, the designers of this system didn't think to include a grading system in their product!

Dr. Jacobi, knowing that you're a whiz with computers, has agreed to give you bonus points if you can automate his grading by calculating the class average for the assignment so he can get back to proving theorems.

Input:
Each test case will consist of several lines of input: a line indicating N, the number of students in the class, a line containing the answer key as a list of comma separated values, and N lines containing a comma separated list of student answers, where each line is one student. You can assume all questions are weighted equally with a perfect exam being worth 100 points, all student solutions have the same number of questions answered as the key, and that blank answers are denoted by a *. All quizzes will have at least one question.

Output:
Print out the class average as a percentage, flooring (discarding the decimal portion) to the nearest whole number. Do not do any rounding before printing the output.

**Sample Inputs:**

```
3
A,B,C,D
A,*,C,D
A,B,C,D
*,*,*,*

5
1,2,A,F
1,2,B,F
1,2,B,*
1,2,A,F
```

1,2,A,F
1,2,A,F

**Sample Outputs:**

58
85

**D. The 3n+ 1 Problem**

Consider the following algorithm to generate a sequence of numbers. Start with an integer n. If n is even, divide by 2. If n is odd, multiply by 3 and add 1. Repeat this process with the new value of n, terminating when n = 1. For example, the following sequence of numbers will be generated for n = 22:

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured (but not yet proven) that this algorithm will terminate at n = 1 for every integer n. Still, the conjecture holds for all integers up to at least 1,000,000. For an input n, the cycle-length of n is the number of numbers generated up to and including the 1. In the example above, the cycle length of 22 is 16. Given any two numbers i and j, you are to determine the maximum cycle length over all numbers between i and j, including both endpoints.

Input:
Each test case will contain one line that will consist of a pair of integers i and j, where
$0 < i <= j <= 1{,}000{,}000$.

Output:
For each pair of input integers i and j, output the maximum cycle length for integers between and including i and j.

**Sample Inputs:**
1 10
100 200

**Sample Outputs:**
20
125

**E. A Quest for Fame and Glory: Tic-Tac-Toe**

Artificial Intelligence programs have often been used to play complex games on levels that human players are incapable of, such as in the games of Chess and Reversi. You recently heard that there's a regional Tic-Tac-Toe AI tournament, and thus you've decided to compete in it for personal fame and glory. As a first step, you've determined that your AI needs to be able to determine the current state of the board in order to know whether or not you should continue processing potential moves.

Input:
Each test case will consist of several lines. The first line will tell contain an odd number N, where the board's dimensions are NxN and 1 <= N <= 51. The next N lines will consist of N characters, one for each space on the board. There are three possible board characters: X's and O's are used to indicate player positions, and B indicates a blank (or open space). A player has won if they have N pieces across a row, column or diagonal (as in traditional Tic-Tac-Toe), and in each input there will at most be one winner.

Output:
Print only one of 4 possible 1 character outputs:
X- the X's have won
O - the O's have won
T - Tie, all spaces have been played and there is no winner
I - Incomplete, neither player has won and there are still open spaces.

**Sample Inputs:**
3
OOO
OOO
OOX

5
BBBBB
BBBBB
BBXBB
BBBBB
BBBBO

**Sample Outputs:**

O

I

**F. Out Cold**

In an attempt to escape your writer's block, you have accepted a job from Mr. Kubrick to be the Winter Groundskeeper in the illustrious Overlook Hotel. In addition to your usual duties such as regulating the broiler and repairing the roofing, you have also been asked to redesign the hotel's world famous hedge maze. Mr. Kubrick has only asked that you carefully consider your layout and ensure that there is at least one valid path through this maze. Knowing that it's better to work smarter rather than harder, you've decided to write a program to check if your proposed mazes have at least one valid path from the Starting location (the upper left hand corner) to the Ending location (the bottom right corner) of the proposed rectangular mazes.

Input:
Each test case will consist of one line that contains two numbers, indicating the rectangular dimensions of the maze N and M, where $1 < N <= 20$ and $1 < M <= 20$. This will be followed by N lines consisting of M characters. S indicates the start space, which will always be (0,0), and E indicates the end space, which will always be the bottom right most corner of the maze. B's, indicate an open space, while W's indicate a wall that is impassable. No maze will have B Blocks with an area larger than 25.

Output:
Print 'Yes' if there is a valid path through the maze from S to E using only 4 directional movement (LEFT, RIGHT, UP, DOWN), otherwise print 'No' if no such path is found.

**Sample Inputs:**
2 2
SW
WE

5 3
SBB
WWB
BBB
BWW
BBE

**Sample Outputs:**
No

Yes

*This page intentionally left blank for scratch paper.*

*This page intentionally left blank for scratch paper.*