2016 UTS Programming Competition

Saturday 2 April, 2016

Laboriously laid out for you by the UTS Programmers' Society (ProgSoc)

Sponsored by
The Faculty of Engineering & IT and WiseTech Global

*Contains the **competition question set** and an **Appendix** following the questions.*

uts.edu.au
progsoc.org
wisetechglobal.com

Problem 1: ASCII Steps
Run time limit: 2 seconds

Problem Description

On a bright sunny day, four score and seven weeks ago, reluctant adventurer Uter the Scared, set out on an epic quest to find the Jade Monkey before the next blue moon. Uter encountered many obstacles on his adventure, vanquishing many a manxome foe with his vorpal sword along the way.

Uter's most formidable challenge came in the form of a cliff in the enchanted forest of Ascii. In the valley below was the Jade Monkey waiting to be found.

Uter, unfortunately, has a fear of heights and his rock climbing skills leave a lot to be desired. If only there was some other way to descend into that valley, thought Uter.

"Greetings!", you, the Wizard of Ascii, shout out to Uter, whom you have noticed out of the corner of your eye. "What brings you to my enchanted forest!?"

"I come to seek the Jade Monkey!", says Uter.

"Well then", you say. "Answer me these riddles three, then you'll be free to cross the valley, where you will see the Jade Monkey!"

Uter promptly answers your questions relating to bird speed and Assyrian capitals and you, in turn, divine a set of steps for Uter by typing the necessary incantations into your enchanted computer, as outlined below.

Data Specification

Input

A single integer, N, where $1 \leq N \leq 40$, denoting the number of steps to be built.

Output

Each step shall be formed as follows:

Start with the footing.

For the footing, use three underscore characters (_), followed by a newline character.

The vertical face of the step shall be formed with two pipe characters (|) on two separate lines, one on top of the other.

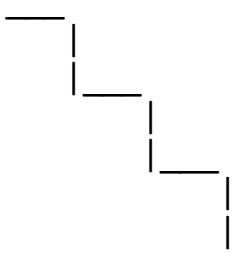
Print enough SPACE characters to the left of each pipe (NOT to the right), such that the vertical face aligns with the footing (i.e. no space between the footing and vertical face, and no overhang of the footing over the vertical face).

Do not terminate with a newline character on the second pipe, UNLESS you have built the FINAL step, since you want the footing of the next step to be placed immediately after the vertical face of the current step.

Sample input

```
3
```

Sample output



Problem 2: Libra Solidus Denarius
Run time limit: 2 seconds

Problem Description

The year is 1961, and you are a cashier at Umbert Threapleton & Sons' department store in the city.

As a reward for customers who buy products in bulk, management have decreed that a discount be applied to each bulk purchase. The exact discount as a percentage varies from product to product.

All prices in the store are stated in pounds, shillings and pence. A pound is subdivided into 20 shillings and each shilling in turn is subdivided into 12 pence. Not exactly the most convenient of currencies to add together nor apply discounts to.

Umbert Threapleton, a forward-thinking man who likes to embrace modern technologies, has installed a UNIVAC computer in the storage area of the department store for the purpose of computing complex calculations. All employees, including cashiers, have access to the computer. It is on this computer that you will run your program to calculate the discounted price for the customer.

Data Specification

Input

A single test case on a single line, with five integers separated by a space.

The first three numbers represent the unit price of a product in pounds ($0 \leq L \leq 1000$), shillings ($0 \leq S \leq 19$) and pence ($0 \leq P \leq 11$), respectively.

The fourth number is the quantity of units ($0 \leq Q \leq 1000$) to be purchased.

The fifth number is the discount ($1 \leq D \leq 99$) to be applied as a percentage.

Output

Three lines with three integers separated by a space, representing a calculated price in pounds, shillings and pence, respectively. Round fractions of pence to the nearest integer value (e.g rounding 0.5 yields 1).

The first line is the total price prior to the discount being applied.

The second line is the total price after the discount is applied.

The third line is the unit price after the discount is applied.

Sample input	**Sample output**
`3 16 8 150 7`	`575 0 0`

Problem 3: WidgCo
Run time limit: 2 seconds

Problem Description

WidgCo is a company that produces various widgets.

Write a program that will assist the stock manager in determining what components need to be ordered from suppliers.

Data Specification

Input

Input consists of two parts for a single test case.

First, the catalogue of widgets produced by the factory and their components. A number is given on its own line indicating how many different widgets are produced. For each widget in the catalogue, there will be a line starting with an integer indicating how many different components are required, and a widget name. Each component then appears on a line describing the number of that component required to make a single widget, and the name of that component. Note that widget and component names are each a single word with an initial capital letter followed by a number. They all have unique letters.

Second, the orders for widgets. A number is given indicating the number of order lines to follow. Each has a number for the number of widgets required, and a widget name.

Output

Output consists of the components that should be purchased from WidgCo's suppliers to fill the orders for the given test case. Components to be ordered are to be given in alphabetical order. Each component to order must appear on its own line, with the number to be ordered followed by the name of the component. Note that if a component should not be ordered (0 are required), your program must not include that component in the output.

Sample input	**Sample output**
2 3 X20 4 C6 2 A4 3 B3 2 Y900 5 D1 4 C6 4 2 Y900 2 X20 3 X20 1 Y900	10 A4 15 B3 32 C6 15 D1

Problem 4: Mindmelt
Run time limit: 2 seconds

Problem Description

You are on a quest to create the smallest, Turing-complete language interpreter ever!

Your cute little language is comprised of a mere eight characters, each with a significant meaning.

Conceptually, your language works like this:

- You have a large number of cells in memory (10000 cells ought to be more than sufficient for this problem) and you have an 'active' cell. You may change the active cell using the commands '>' and '<'. You perform operations on the active cell using '+' and '-'.

- You also have input and output. '.' will print the contents of the current cell as an ASCII character, while ',' will read input and place its value into the active cell.

- Finally you have loops. '[' opens the loop and ']' closes it. The loop is like a **while** loop, where the condition being tested is '**while** the active cell is not zero'.

Command	Meaning
+	Add 1 to the active cell
-	Subtract 1 from the active cell
>	Move the active cell one step to the right
<	Move the active cell one step to the left
[	Start **while** loop (while the active cell is non-zero)
]	End **while** loop
.	Print the active cell as an ASCII character
,	Read from standard input into the active cell

Data Specification

Input

A single test case.

The first line represents standard input to be read in by the interpreter using the ',' command. This line may be empty i.e. no characters other than an end of line marker.

The second and subsequent lines, if any, represent a stream of characters in the printable ASCII range that are to be interpreted. Your interpreter will process one character at a time, acting upon those characters described above and ignoring all other characters (including whitespace).

Cease execution upon encountering the end-of-file (EOF) marker.

Output

Display the output from executing your Mindmelt program.

Sample input	Sample output
`HELLO` `,.,.,.,.,. please print HELLO`	`HELLO`

Problem 5: The Resistance Against the Invasion of Our Ohm Land
Run time limit: 10 seconds

Problem Description

You have been hired by Unparalleled Technological Services to help reduce the number of resistors used in their mass-produced electrical circuits, which will reduce manufacturing costs.

The humble resistor is a small, but crucial component in every electrical circuit. It plays a major role in regulating the flow of electrons (*current*) throughout a circuit by converting electrical energy into kinetic energy (heat), and dissipating that heat, in a controlled and quantified manner. We refer to this energy conversion as *resistance*.

The unit of measurement to quantify resistance is *ohms*. The higher the ohm value, the higher the resistance. When deciding on the number and type of resistors to be used, we need to first consider how much current we want within a path of a circuit, and also the potential energy needed to transfer electrons from one point along that path to another (*voltage*). We relate current, voltage and resistance, using a very simple formula known as Ohm's Law:

V = **IR** (where **V** is voltage, **I** is current and **R** is resistance).

For example, let's say that on a given path in our circuit, we have 44558 volts applied and need 10 amperes of current through that path. By applying Ohm's Law (and rearranging the equation to make **R** the subject), we determine that we need to place a resistor along that path whose resistance is 4455.8 ohms. Problem solved, right?

Wrong.

Unfortunately, UTS only assembles circuits, it does not manufacture the components. This includes resistors. In fact, most electronics companies rely on pre-manufactured resistors. Because of this, there has been a need for standardisation of resistor values. UTS makes use of the E-12 standard range of resistors, so called because there are 12 standard base resistor values that all resistors in that range make use of, namely:

10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, 82.

This is referred to as the first *decade* of E-12 resistor values (measured in ohms). The second decade is:

100, 120, 150, 180, 220, 270, 330, 390, 470, 560, 680, 820.

The third and subsequent decades can easily be derived by multiplying each base value by the appropriate power of 10.

The resistances of E-12 resistors are usually only approximately equal to their nominal value but UTS have found a supplier that guarantees exact resistances. UTS wishes to use combinations of these exact resistors to achieve close approximations to actual desired resistances while using the fewest number of resistors. Resistors are always to be connected in series so that the resistance value of a set of resistors is the sum of their resistances. To measure the closeness of an approximation, UTS define the error as the distance of the approximate value (the sum of the resistances) from the target value expressed as a percentage

of the approximate value. They wish to ensure that that error is at most 1%. For example, if we wish to approximate 4455.8 ohms, we could choose the following set of resistors:

3900, 470, 82

as they sum up to 4452 ohms. The error is only $|(4455.8 - 4452)| * 100/4452 = 0.085\%$ which is well within the desired accuracy of 1%. However, a better choice would be:

3900, 560.

While the total resistance of 4460 ohms is not as accurate as that achieved with the previous choice, the error is still well under 1% and, importantly, this combination uses one less resistor (remember, manufacturing costs add up on a large scale).

Your task is to write a program that, given a voltage and current, chooses the best set of resistors to provide the required amount of resistance to within the 1% error as defined above.

Data Specification

Input

A single test case.

The input has two integer values V ($1 \leq V \leq 10^9$) and I ($1 \leq I \leq 10^7$). V is the voltage and I is the current.

Output

Output a series of E-12 resistor integer values, from largest value to smallest value, separated by a space, which consists of the lowest number of resistors that approximates the target resistance with an error of at most 1%.

You can use the same resistor value more than once. If you find two or more resistor sets with the same number of resistors that are within the error range, output the set whose sum is closest to the target value. If there are two sets that are the same distance from the target value, output the set whose sum is smaller. If there are still ties, output the set of resistors which is lexicographically least (when the resistors are ordered from largest to smallest).

If there are no possible sets of resistors that allow this resistance, output `Impossible` instead.

Sample input 1	**Sample output 1**
`50000 5`	`10000`
Sample input 2	**Sample output 2**
`44558 10`	`3900 560`
Sample input 3	**Sample output 3**
`1 1`	`Impossible`

Problem 6: Sator Squares
Run time limit: 20 seconds

Problem Description

Take a look at the following Latin text (the meaning of the words are irrelevant):

S A T O R
A R E P O
T E N E T
O P E R A
R O T A S

You will see that there are five words of five characters each, that may be read in one of four ways:

1. left-to-right, top-to-bottom;
2. right-to-left, bottom-to-top;
3. top-to-bottom, left-to-right, or;
4. bottom-to-top, right-to-left,

and whichever way you read them, you end up with the same five words in the same order.

This is the original Sator Square dating back to Ancient Roman times, so called because of the first word in the square, *sator*.

Here is another Sator square, of size 3:

a r e
r 6 r
e r a

Your task is to write a program that finds as many Sator squares as you can in a large grid of text.

Trivial Sator squares (i.e. of size 1, or a single character) are not to be considered.

Data Specification

Input

A single test case.

The first line denotes the horizontal (H) and vertical (V) dimensions, respectively, of the grid of characters to search, separated by a single space
($1 \leq H \leq 200$, $1 \leq V \leq 200$, H,V are integers).

V lines of H characters then follow.

Output

Lines of the form S X Y, where:

S is the size of a found square (S ≥ 2), and;
X,Y are the co-ordinates of the top-leftmost character of the found square, relative to the grid.

Sort all found squares in ascending order, first by size, then by horizontal and vertical co-ordinates.

If no squares have been found, simply print 0 on a single line.

Sample input	**Sample output**

`6 8`
`AweFAR`
`vanACA`
`3naRAF`
`f6#rtg`
`67g33%`
`764ef^`
`*&#fEH`
`!!@feG`
	`2 1 1`
	`2 4 0`
	`3 0 3`

Problem 7: Sudoku
Run time limit: 2 seconds

Problem Description

A 9x9 sudoku puzzle has 9 rows, 9 columns, and 9 blocks (3x3 subgrids), each of 9 cells, where each cell can contain a digit in the range [1, 9]. Each row, column and block should have a cell with each digit once only. Write a program to solve any Sudoku puzzle.

Note that Sudoku puzzles can often have multiple solutions for a given input. These test cases will all have enough cells completed such that there will be only one possible solution to the puzzle.

Data Specification

Input

Input consists of a single test case containing the initial puzzle state.

Known cells are given as digits 1-9, and unknown cells are given as 0. Each digit on a line is separated by a single SPACE.

There are nine lines in total.

Output

Output consists of the solution to the given puzzle, ie. with all the zeroes filled in with digits 1-9, digits separated by a single SPACE, nine lines in total.

Sample input

```
4 0 0 3 5 7 9 6 0
0 0 5 2 0 6 3 0 4
0 0 9 0 0 0 0 2 5
5 2 0 9 0 8 1 4 7
0 0 6 0 7 0 2 0 0
1 9 7 5 0 2 0 3 8
6 5 0 0 0 0 8 0 0
9 0 8 6 0 5 4 0 0
0 7 1 8 3 4 0 0 6
```

Sample output

```
4 8 2 3 5 7 9 6 1
7 1 5 2 9 6 3 8 4
3 6 9 4 8 1 7 2 5
5 2 3 9 6 8 1 4 7
8 4 6 1 7 3 2 5 9
1 9 7 5 4 2 6 3 8
6 5 4 7 2 9 8 1 3
9 3 8 6 1 5 4 7 2
2 7 1 8 3 4 5 9 6
```

Problem 8: AMazeIng
Run time limit: 20 seconds

Problem Description

A maze is a convoluted pathway than bends and winds its way around a confined area.

Within a maze you may have regions that are connected to the outside world, either via a direct escape portal or to another region not separated by a wall. These regions form a chain, or path, towards an escape portal.

You might also have regions that are completely surrounded by walls and are therefore not linked to the outside world.

Your task is for each cell in a maze, determine if it is eventually connected to the outside world through any available path. It must not pass through any walls.

Data Specification

Input

A single test case.

The first line will contain two integers. The first integer represents the maze width ($1 < W < 9999$). The second integer represents the maze height ($1 < H < 9999$).

This is followed by H lines of W cells. Each cell may contain any or none of the characters N, S, E, W, separated by a TAB character ('\t').

The characters N, S, E W represent open connections. The absence of a letter means there is a wall in that direction:

North: A connection to the cell above
South: A connection to the cell below
East: A connection to the cell to the right
West: A connection to the cell to the left

If a cell has a 'N', then the northern cell is guaranteed to have a corresponding 'S'. The characters will always be in alphabetical order (ENSW).

Output

Reprint the maze, replacing each cells' contents with:

a '#' if a cell is connected to the outside world, or;

a '*' if it is not.

Do not use a separating character (e.g. whitespace) in between cells.

Sample input

```
3 3
   NS
ES ENW   W
N
```

Sample output

```
*#*
###
#**
```

1. How to compile and run your programs:

Java

```
$ javac problem1.java
$ java problem1
```

N.B. the name of your class should also be `problem1`, with a single method:

```
public class problem1
{
        public static void main(String args[]) {
            // code
        }
}
```

C

```
$ gcc -lm problem1.c -o problem1
$ ./problem1
```

C++

```
$ g++ -lm problem1.cpp -o problem1
$ ./problem1
```

`-lm` links in the maths library that contains various maths-related functions. You will still need to include the appropriate header files e.g. `math.h`. Java has no command line options for maths functions, but you will need to import the appropriate class e.g. `java.lang.Math`

2. How to parse standard input in C:

How to read in an undetermined number of lines from standard input:

Basically, you set up a while loop that tests the result returned by the function used for input (`scanf(), fgets(), sscanf(),` etc.), and terminate the loop if the result returned is zero or `EOF` (end of file).

The simplest is `scanf()`, which is useful if each line of input is of known fixed format and length, e.g.

```
/* read 3 integers from each line of standard input, and print the sum
for each line */
#include <stdio.h>
int main() {
int a,b,c;
  while (scanf("%d %d %d",&a,&b,&c)!=EOF) {
        printf("%d\n",a+b+c);
  }
  return 0;
}
```

If you don't know how many items there will be on each line, use `fgets()` to read the line into a buffer, then use `strtok()`or `sscanf()` on that buffer to read in individual items, e.g.

```
/* read up to 10 integers from each line of standard input and print the
sum for each line */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main() {
char linebuf[1024];  /* A 1024 character buffer is more than enough for
this                        problem */
char* token;
int sum;

  while (fgets(linebuf, sizeof(linebuf), stdin) !=0) {
        sum=0;
        token = strtok(linebuf," \t");  /* we are using space and tab as
                                           delimiters */
        while (token != NULL) {
                sum += atoi(token);
                token = strtok(NULL, " \t");
        }
        printf("%d\n",sum);
  }
return 0;
}
```

When you are testing your program by typing from the terminal, type `control-d` to signal end of file (`EOF`).

Read the `man` pages of `sscanf(), fgets(), strtok()`, and `atoi()` (available on your machine) to understand how these system functions work, what their return values signify, and what libraries you need to include.

3. How to parse standard input in C++:

There are several ways to do this, including using the C code above. The most common way is to use `cin` and `cout`.

```
// Read in an integer, a string, and a floating point number, and print
them out
#include <iostream>
#include <string>
using namespace std;

int main() {
int x;
double y;
string s;
  cin >> x >> y >> s;
  cout << s << endl;
  cout << x << " " << y << endl;
  return 0;
}
```

You can also use `getline()`. Make sure you know how to use the delimiter argument. Depending on how you write your code, you may have to call `getline()` just to throw away the newline at the end of a line of input.

```
// reading strings from standard input with getline()
#include <iostream>
#include <string>
using namespace std;

int main () {
  string s1,s2,s3;
  getline(cin,s1,' ');
  getline(cin,s2,' ');
  getline(cin,s3);
  cout << s1 << endl;
  cout << s2 << endl;
  cout << s3 << endl;
  return 0;
}
```

How to read in an undetermined number of lines from standard input:

Basically, you set up a while loop that tests the result returned by the method `cin.operator>>`, and terminate the loop if the result returned is zero.

```
while (cin >> x) {
        // process input
}

while (getline(cin, linebuf)) {
        // process input
}
```

4. How to parse standard input in Java:

Again there are several ways to do this. Here we present only one - using the `Scanner` class.

```
import java.util.*;
import java.io.*;

public class foo
{
  public static void main (String args[])
  {
        Scanner in = new Scanner(System.in);
        String s, linebuf;

        int n = in.nextInt();
        double x = in.nextDouble();
        double y = in.nextDouble();
        char c = in.next().charAt(0);
        s = in.next(); // read in the next token (tokens are separated
by
                       //whitespace by default)
        in.nextLine(); // throw away the rest of the line
        linebuf = in.nextLine();

        System.out.println(n + " " + x + " " + y + " " + c );
        System.out.print(s+" ");
        System.out.println(linebuf);
  }
}
```

No guarantee is made regarding the accuracy of information in this appendix. We hope you are already familiar with processing standard input yourself.