ECOO 2016

Programming Contest Questions

Final Competition (Round 3)

May 14, 2016

Sheridan | Get Creative

Questions made possible in part through the support of the Sheridan College Faculty of Applied Science and Technology.

(Continued on the next page)

Problem 1: Nerd Poker

Nerd Poker is a podcast where you get to listen in on a group of people playing a game of Dungeons and Dragons. There is a lot of dice rolling in this game, using a lot of different types of dice. There are dice with 4, 6, 8, 10, 12 and 20 sides that are regularly used to play the game. In combat and at other critical junctures of the game, low rolls are often bad.

The Nerd Poker players are superstitious, so before entering combat they take their dice and “roll out the ones”. They start with their 20 sided dice, then do their 12 sided, 10 sided, etc. They roll all the dice of the same type at once, set aside any that show a 1, then roll the remaining dice, set aside the 1’s again, and continue this process until they have rolled a 1 on every die.

For example, Brian has 3 four-sided dice. On the first roll, they show a 1, a 3 and a 3, so he sets aside the 1 and rolls the other two again. This time he gets a 1 and a 4, so he sets aside the 1 and continues to roll the last die. It takes him 3 more rolls before he gets a 1, for a total of 5 rolls. But how many rolls should Brian *expect* to have to make? Or to put it another way, if Brian rolled out the ones on his 3 four-sided dice an arbitrarily large number of times, what would be the average number of rolls required?

DATA11.txt (DATA12.txt for the second try) will contain 10 test cases. Each test case consists of a single line with 2 positive integers N and S where N is the number of dice and S is the number of sides on each die ($1 \leq N \times S \leq 500$). Your job is to output a line containing a single integer for each test case, representing the *expected* number of rolls required to roll out all the 1’s. Your answers should always be rounded up.

Note that the sample input below contains only 3 test cases but the data files will contain 10.

Sample Input

```
1 20
125 4
25 20
```

Sample Output

```
20
20
75
```

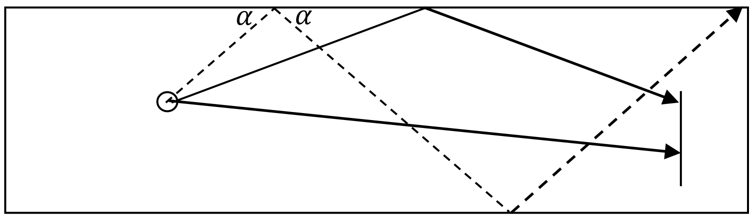
Question Development Team
Sam Scott (Sheridan College)
Kevin Forest (Sheridan College)
Stella Lau (University of Cambridge)
Reyno Tilikaynen (University of Waterloo)
David Stermole (ECOO-CS President)
John Ketelaars (ECOO-CS Communications)

(Continued on the next page)

Problem 2: Target Practice

You wake up to find yourself in an airless, frictionless, gravity-less arena armed with an object that bounces perfectly off any surface. Targets are appearing at one end of the room and you have to hit as many as you can. You can throw the ball directly at a target, or bounce it off the side walls before hitting the target. But it doesn't count if you hit the wall behind the target before hitting the target.

The diagram below shows three different attempts to hit a target. Two of them are hits and one of them (the dashed line) is a miss. The ball always bounces perfectly, which means the angle with which it hits the wall is the same as the angle with which it leaves the wall (one example is shown as α on the diagram) and it never loses any speed on a bounce.



For specifying locations and speeds, you can consider the arena to be on a Cartesian plane with the origin at the bottom left corner. You can treat the target and walls of the arena as line segments and the ball as a point on the plane.

DATA21.txt (DATA22.txt for the second try) will contain 10 test cases. Each test case consists of 6 lines. The first line contains 6 integers $A_w, A_h, B_x, B_y, S_x, S_y$ separated by spaces. These integers represent an arena and a single throw of the ball. A_w and A_h represent the width and height of the arena ($100 \leq A_w, A_h \leq 1000$), B_x and B_y represent the initial position of the ball you are throwing ($1 \leq B_x \leq A_w/2$ and $1 \leq B_y \leq A_h/2$), and S_x and S_y represent the X and Y components of the speed with which you are throwing the ball ($0 \leq S_x \leq 100$ and $-1000 \leq S_y \leq 1000$). The next 5 lines each contain 3 integers T_h, T_x, T_y separated by spaces. Each set of integers represents a possible target. T_x and T_y represent the location of the top of the target and T_h represents its height ($1 \leq T_h \leq A_h/4$ and $B_x + 1 \leq T_x \leq A_w - 2$ and $A_h/4 \leq T_y \leq 3A_h/4$). For each test case, your program should output a single line containing an H or M character for each of the five targets in the order they appear. Output an H if the ball would hit the target or M if it would miss.

Note that the sample data below contains only 2 test cases, but the data files will contain 10 each.

Sample Input

```
116 178 53 41 7 16
54 101 73
32 64 128
13 62 119
57 55 98
44 74 54
137 122 11 24 7 16
47 123 78
51 130 85
56 29 30
23 79 67
32 39 66
```

Sample Output

```
MMMHM
HHMHM
```

(Continued on the next page)

Problem 3: CamelCase

Many programmers use CamelCase when naming variables, functions, classes and other entities. In CamelCase, when a name consists of multiple words concatenated together (such as "myawesomevariablename"), the first letter of every distinct word is capitalized. In Upper CamelCase, all words are capitalized ("MyAwesomeVariableName"). In Lower CamelCase the first word is not capitalized but the others are ("myAwesomeVariableName"). Sometimes there is more than one way to turn a string of letters into CamelCase.

DATA31.txt (DATA32.txt for the second try) will contain a dictionary, followed by 10 test cases. The dictionary starts with an integer N (where $1 \leq N \leq 200000$) followed by N lines, each containing a word. A word consists of a string of lowercase English letters. Each of the 10 lines following the dictionary will contain a single test case. Each test case consists of a string of lowercase English letters of length 2000 or less, created by concatenating words from the dictionary. Your program should output 10 integers, one per line, representing the minimum number of capitalizations required to convert each test word to Lower CamelCase, so that it can be read as a string of legal words from the dictionary.

Note that the sample input below contains only 4 test cases, but the real data files will contain 10.

Sample Input

```
26
aid
all
app
apple
brown
come
country
crab
crabapple
dogs
for
fox
good
is
jumps
lazy
men
now
of
orchard
over
quick
the
their
time
to
apple
appleorchard
crabapple
thequickbrownfoxjumpsoverthelazydogs
```

Sample Output

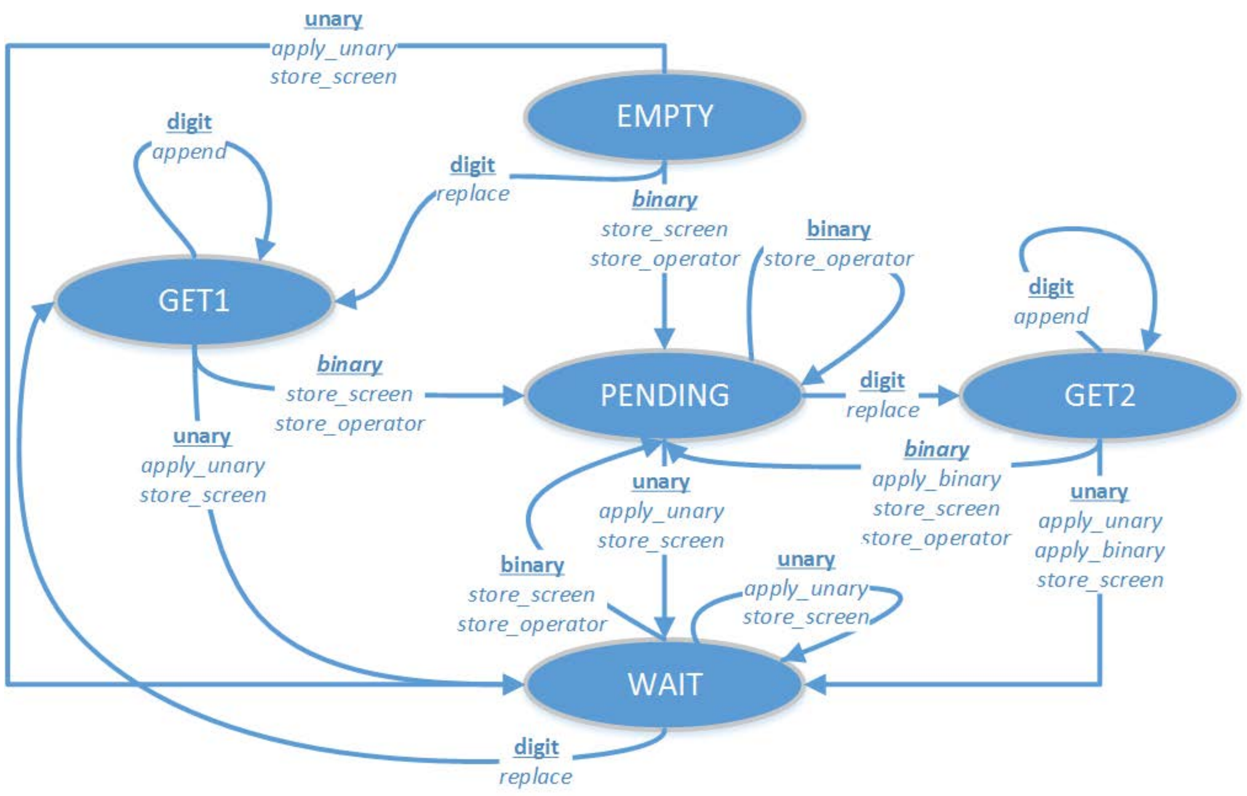
```
0
1
0
8
```

(Continued on the next page)

Problem 4: Cuthbert's Calculator

Your friend Cuthbert has a broken calculator. When it was working properly, it had buttons for all 10 decimal digits (0 through 9), plus three binary operators ($+, -, \times$), two unary operators ($+/-$, and x^2), two memory functions (*store* and *recall*) and a clear button. Binary operators require two operands to produce an answer (e.g. $12 + 4 = 16$). Unary operators require only one operand: the $+/-$ button changes the sign of the number on the screen and displays the result; the x^2 button squares the number on the screen and displays the result. The calculator cannot display more than 8 digits.

A diagram that describes the basic operation of Cuthbert's calculator is shown below. The calculator starts in the EMPTY state with "0" showing on the screen. When the user presses a key, if it is a digit key, the calculator moves to the GET1 state. If it's a binary operator ($+, -, \times$), it goes to the PENDING state. If it's a unary operator ($+/-$, x^2) it goes to the WAIT state. In addition, there are actions the calculator takes on each state transition. For example, if the user presses a unary operator key while the calculator is in the GET2 state, the calculator performs the *apply_unary*, *apply_binary* and *store_screen* operations, in that order, and then moves into the WAIT state. A table that explains these operations is shown on the next page.



(Continued on the next page)

OPERATION	DESCRIPTION
replace	Replace the contents of the screen with the digit just pressed.
append	Add the digit just pressed to the right of the screen.
store_screen	Store the contents of the screen in an internal register (X) for later use. If there is already something in X, it will be replaced.
store_operator	Store the operator just pressed in an internal register (OP) for later use. If there is already something in OP, it will be replaced.
apply_unary	Apply the unary operator to the value currently showing on the screen, and update the screen to show the result.
apply_binary	Perform the operation in the OP register using the value in the X register as the left operand and the value currently on the screen as the right operand, then put the result on the screen. For example if X = 45, OP = "–" and 23 is showing on the screen, perform 45 – 23 and put the result (68) on the screen.

The calculator also has two memory buttons MS and MR. The MS button will store the value currently on the screen in a memory register (M). The MR button will copy the value from M to the screen. The MS button does not change the state of the calculator. The MR moves the calculator to GET1 from EMPTY or WAIT and to GET2 from PENDING. Otherwise, it does not change the state.

Finally, there is an AC button which clears the screen to "0" and changes the calculator state to EMPTY.

Unfortunately, the calculator is broken. Only some of the buttons are working at the moment.

DATA41.txt (DATA42.txt for the second try) will contain 10 test cases. Each test case will consist of two lines. The first line contains the available buttons (some combination of $1, 2, 3, 4, 5, 6, 7, 8, 9, 0, +, -, *, n$, and s separated by spaces, where *n* means negation and *s* means square. The ac, ms and mr buttons will always be available so they will not be listed). The second line contains an integer N that can be produced on the screen using only the available buttons (where $1 \leq N \leq 1000$) . Your job is to output a sequence of the available keys (in lowercase) such that when all the keys have been pressed in the order you specify, the integer N will be shown on the screen.

SPECIAL NOTES:

1. Your output needs to go to a file named OUTPUT41.txt (OUTPUT42.txt for the second try) and will be scored by an automated judging program.
2. The solutions shown below are not the only possible solutions.
3. The sample input below contains only 3 test cases but the input files will contain 10.

Sample Input

```
012345+-*sn
53
456*-s
```

```
78
94s-n
101
```

Sample Output

```
5 3
5 4 4 - 4 6 6 *
9 - 4 s ms 9 4 - mr -
```

(Continued on the next page)

Further Information

Here is a table to help explain the solution to the final sample input above:

INPUT	STATE	STORED SCREEN	STORED OPERATOR	MEMORY	SCREEN
	EMPTY				0
9	GET1				9
–	PENDING	9	–		9
4	GET2	9	–		4
S	WAIT	-7	–		-7
MS	WAIT	-7	–	-7	-7
9	GET1	-7	–	-7	9
4	GET1	-7	–	-7	94
–	PENDING	94	–	-7	94
MR	GET2	94	–	-7	-7
–	PENDING	101	–	-7	101

(Continued on the next page)