Irish Collegiate Programming Competition 2016

Practice Problem Set

University College Cork ACM Student Chapter

April 2, 2016

Instructions

Rules

- All mobile phones, laptops and other electronic devices must be powered off and stowed away for the duration of the contest.
- The only networked resource that teams are permitted to access is the submission system.
- If a team discovers an ambiguity or error in a problem statement they should tell an organiser. If the organisers agree that an ambiguity or error exists, a clarification will be issued to all teams.
- No multi-threading is allowed, and no sub-processes.
- No file input/output is allowed. All input to your program will be provided via standard input (stdin) and all output should be printed to standard output (stdout). Examples of how to do this in each of the languages is provided in the resources section of the submission site.

Submission Instructions

- The submission system URL is: `http://4c245.ucc.ie:8080/`
- Your password will be provided by the organisers. Please notify an organiser if you are unable to log in.
- Submissions should consist of a single source file, **not a compiled executable**.
- To submit, click the "Submit a Solution" link, complete the submission form, upload your source file, and click save.
- Your submission should now be listed in your submission queue.
- Java solutions should be a single source file and should not include the package header. The testing script will also be renaming your file to `Pn.java` (where n is the problem number) and as such the main class for problem 1 should be defined as: `public class P1 {...}` and will be called with: `java P1 < input-file`

- C and C++ submissions will be compiled with the flags `-lm -O2`. C# submissions will be run using mono since the test environment is running on linux.
- Haskell submissions will be compiled with the flag `-O2`.

Testing and Scoring

- Solutions should be submitted in the form of source code, which will be compiled on the test environment. A solution will be accepted if it compiles on the test environment and produces the correct output for the sample inputs given in the question.
- The output from your program should be terminated by a new line and should not contain any additional whitespace at the beginning or end of lines.
- A solution will be tested against 10 separate test cases which are designed to test the limits and corner cases of the inputs. One point is given for each correct output.
- Programs are limited to one second of CPU time and 256MB of RAM.
- If a solution is submitted while an earlier solution to the same problem is still in the queue, the earlier submission will not be tested. The earlier submission will be marked with the message: "Old, not going to be tested".
- In the event of a tie, the winning team will be the one who's last scoring submission was submitted earliest.

1 Adding Integers

Did you know that two integers can be added together to form a single integer? It's true! To verify that you can compile, run, and submit a solution write a program that reads two integers and outputs their sum.

Input The input consists of a single line. The line contains two integers N and M, $0 \leq N, M \leq 10000$, the integers to be added together.

Output The output of your program should be the sum of the two input numbers. The answer should immediately be followed by a single newline character.

Sample Input 1

```
5 2
```

Sample Output 1

```
7
```

Sample Input 2

```
19 23
```

Sample Output 2

```
42
```

2 Look and Say

A 'Look and Say' sequence consists of a series of lines where each line *describes* the previous one; beginning with a starting sequence. Lines are described by counting continuous occurrences of characters. For example:

Line	Description	Sequence
1	starting sequence	1
2	one-1	11
3	two-1s	21
4	one-2, one-1	1211
5	one-1, one-2, two-1s	111221
6	three-1s, two-2s, one-1	312211
	...	

Your task is to implement a program to compute a 'Look and Say' sequence. You will be given the first line as input and asked to output the n^{th} line of the sequence.

Input The input is a single line consisting of the starting string sequence, s; followed by a space; and an integer $1 \leq n \leq 100$ which is the index of the line that should be calculated. Sequences for all inputs and expected outputs are guaranteed to only contain digits 0..9 and have a length between $1 \leq |s| \leq 1000$. Therefore, the sequence should not be stored as an integer.

Output The output should consist of a single line containing the string representation of the n^{th} line of the sequence. Characters of the sequence should not be separated by any space and should be terminated by a new line character.

Sample Input 1

```
1 7
```

Sample Output 1

```
13112221
```

Sample Input 2

```
1234 4
```

Sample Output 2

```
13211231133114
```

3 Boolean Postfix

Logical Boolean expressions are typically represented using *infix* notation, where the operators ($\wedge, \vee$) are placed between the operands. For example, $((a \wedge b) \vee \neg c)$ states that for the expression to be *true*, both a and b should be *true*, or c should be *false*. In her mathematics, Mary Lucy Margret uses an alternate form, where the operator is placed after the operands, called *postfix* notation. For example, the above expression could be written in postfix notation as: $(a\ b \wedge c \neg \vee)$.

Your task is to write a program to parse and evaluate Mary Lucy Margret's Boolean expressions, which are represented in postfix notation. You can be confident in her mathematics; you are guaranteed to be given correct and valid expressions.

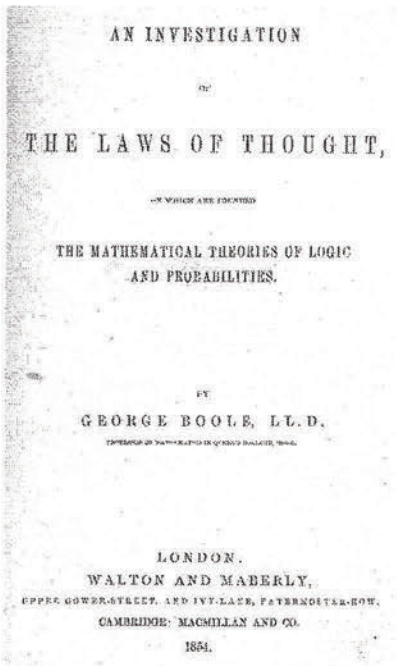
AN INVESTIGATION
OF
THE LAWS OF THOUGHT,
ON WHICH ARE FOUNDED
THE MATHEMATICAL THEORIES OF LOGIC
AND PROBABILITIES.
BY
GEORGE BOOLE, LL.D.
LONDON:
WALTON AND MABERLY,
UPPER GOWER-STREET, AND IVY-LANE, PATERNOSTER-ROW.
CAMBRIDGE: MACMILLAN AND CO.
1854.

Figure 1: *An Investigation of the Laws of Thought* by George Boole, Professor of Mathematics in Queen's College, Cork, published 1854.

Input The first line of input contains a single integer T representing the number of expressions. Each of the next T lines contains a single Boolean formula in postfix notation. The line starts with a single integer n representing the number of tokens, with the remainder of the line containing n tokens, each separated by a single space.

The tokens 1 and 0 are used to denote Boolean truth values *true* and *false* respectively, and uppercase characters are used to denote the operators. Thus, the set of possible tokens are:

1 for Boolean *true*,	A for logical *and*,	X for logical *exclusive-or*,
0 for Boolean *false*,	R for logical *or*,	N for logical negation.

Output The output should consist of T lines where each line contains a 1 if the corresponding expression evaluates to *true*, or 0 otherwise.

Sample Input 1

```
3
3 0 1 R
3 0 0 R
7 1 0 A 1 R N N
```

Sample Output 1

```
1
0
1
```

Sample Input 2

```
4
9 0 0 A 0 N 0 N A R
9 0 1 A 0 N 1 N A R
9 1 0 A 1 N 0 N A R
9 1 1 A 1 N 1 N A R
```

Sample Output 2

```
1
0
0
1
```