2006 Quinnipiac University Programming Competition for High School Students

Quinnipiac University

April 8, 2006

Instructions

1. Submit a program by attaching it to an e-mail to the address

   `cscsubmit@.quinnipiac.edu`

   Include the number of the problem you are solving in the subject line of your e-mail.

2. The entire program must be contained in a single file.

3. Your feedback on a submitted problem will be e-mailed in a reply and will be limited to one of the following:

   (a) COMPILE ERROR: The program did not compile

   (b) PROGRAM NOT RUNNABLE: The program compiled, but did not run as a stand-alone program.

   (c) CRASH: The program crashed or threw an exception when it was run

   (d) INCORRECT OUTPUT

   (e) TIMED OUT: The program did not finish in a reasonable amount of time

   (f) ACCEPTED

   If your program crashes, produces incorrect output, or times out, no information will be given as to the input that caused the problem.

1 DNA and RNA

Deoxyribonucleic Acid (DNA) and Ribonucleic Acid (RNA) each consist of a sequence of nucleotides. Possible nucleotides for DNA include adenine, cytosine, guanine, and thymine, abbreviated A, C, G, and T, respectively. RNA contains the same nucleotides except that thymine is replaced by uracil, abbreviated U.

To form a protein from a strand of DNA, Ribonucleic Acid (RNA) attaches itself to the DNA. Each nucleotide in the DNA is matched with a corresponding nucleotide of RNA. Adenine in DNA corresponds to Uracil in RNA, Thymine in DNA corresponds to Adenine in RNA, and Cytosine and Guanine correspond to each other. The nucleotides in the RNA are then split into groups of three, and these groups each correspond to amino acids which build the proteins.

Write a program which takes a sequence of between one and fifty DNA nucleotides as input and prints the corresponding sequence of RNA nucleotides, broken into groups of three with each group separated by a single space. There may be fewer than three nucleotides in the final group. For example, if the input is

```
CATTTCGAAG
```

the output should be

```
GUA AAG CUU C
```

You may assume the input consists of a single string which only includes the characters A, C, T, and G.

2 Evaluating a polynomial

A *polynomial* is a function in the form

$$f(x) = a_0x^0 + a_1x^1 + a_2x^2 + \cdots + a_nx^n$$

where each coefficient a_i is a real number. For example, the function

$$f(x) = -3 + 5x^3 - 2x^4$$

is a polynomial whose coefficients are $a_0 = -3$, $a_1 = a_2 = 0$, $a_3 = 5$, and $a_4 = -2$. We can evaluate a polynomial at a particular value by plugging in that value for x and computing the result. For example, evaluating the above polynomial at 2 gives

$$f(2) = -3 + 5 \cdot (2^3) - 2 \cdot (2^4) = -3 + 5 \cdot 8 - 2 \cdot 16 = 5.$$

Write a program which takes as input a sequence of coefficients that represents a polynomial on a single line, followed by a sequence of lines which each contains a single nonzero integer, followed by a zero indicating the end of the input file. Your program should then evaluate the polynomial for each input value *including the zero*, and print out each result on a separate line. The line that represents the polynomial starts with a coefficient for x^0 and proceeds in increasing order of the exponent, using zeroes to represent omitted terms. For example, if the input contains

```
-3 0 0 5 -2
2
1
-1
-2
3
0
```

the polynomial is the example given above, and the output should be

```
5
0
-10
-75
-30
-3
```

You may assume that all coefficients and all input values in this problem are integers between -10 and 10 (inclusive), and that there are between one and nine coefficients (so the largest significant power of x is x^8). A polynomial with one coefficient is a constant. It produces the same output value for every input value.

3 Winning the Lottery

The Powerball Lottery is played in 28 States, Washington D.C. and the US Virgin Islands. A drawing consists of selecting five out of a collection of fifty-five white balls, numbered from 1 to 55, and one of forty-two red balls, numbered from 1 to 42. The five white balls must all be different, but the red ball may have the same number as one of the white balls. In the simplest version of the game, prizes are paid depending on the number of white balls that match and whether the red ball matches according to the following table:

Number of white balls matched	**Prize if red ball does not match**	**Prize if red ball matches**
0	0	$3.00
1	0	$4.00
2	0	$7.00
3	$7.00	$100.00
4	$100.00	$10,000.00
5	$200,000.00	**JACKPOT**

For example, a player who matches three of the white balls wins $7.00 if an incorrect red ball is chosen, and $100.00 if the red ball does match.

Write a program which takes several lines as input on which each line has six integers. On each line, the first five integers correspond to white balls in any order, and the sixth integer corresponds to a red ball. The first line represents the actual numbers drawn, and each subsequent line except the last represents the numbers chosen by a player. The last line consists of six zeroes.

The output for the program should give the winnings of each player in the same order in which they appear in the input. Dollar amounts should be displayed as integers or as the word "JACKPOT" all in uppercase. There should be no line of output corresponding to the line of zeroes.

Here is an example:

Input	**Output**
`26 3 17 8 11 11`	
`5 10 15 20 25 11`	`3`
`11 40 20 6 3 5`	`0`
`17 40 6 26 3 11`	`100`
`3 8 1 17 26 11`	`10000`
`3 8 11 17 26 11`	`JACKPOT`
`0 0 0 0 0 0`	

Note that in each case, the red ball selected is the last integer on the line.

You may assume that each row consists of exactly six integers and nothing else, and the first five integers on each row except the last are distinct.

4 Mini-Sudoku

To solve a Mini-Sudoku puzzle, one must fill in a 4×4 grid with the integers from 1 to 4 in such a way that each integer appears once in each row, once in each column, and once in each 2×2 subgrid. (Normally Sudoku is played on a 9×9 grid). For example, the following would be a solution.

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

Write a program which takes four rows of four integers as input. If the grid represents a legitimate solution, your program should print the words "`Valid Solution`"; otherwise, it should print one or more of the following, depending on which criteria fail:

- Invalid row
- Invalid column
- Invalid subgrid

Your program should display each reasons which is responsible for a failure on a separate line, but the same reason should not be given more than once. For example, it might print both "Invalid row" and Invalid subgrid" if both of these reasons apply, but it should print "Invalid row" only once, regardless of how many rows actually fail.

You may assume the input consists only of integers from 1 to 4, and that there are exactly four rows, each with four integers.

Here are some sample inputs, along with the corresponding output.

Input	**Output**

`1 2 3 4`
`3 4 1 2`
`2 1 4 3`
`4 3 2 1`
	`Valid Solution`

`1 2 3 4`
`4 3 2 1`
`1 2 3 4`
`4 3 2 1`
	`Invalid column`

`2 2 2 2`
`1 1 1 1`
`3 3 3 3`
`4 4 4 4`
	`Invalid row`
	`Invalid subgrid`

`3 3 3 3`
`3 3 3 3`
`3 3 3 3`
`3 3 3 3`
	`Invalid row`
	`Invalid column`
	`Invalid subgrid`

5 3D Maze

You find yourself in the northwest corner of a three-dimensional maze. You want to get to the exit at the southeast corner. The maze has no walls, but each square of the maze can be at a different level. You may move from one square to an adjacent square as long as you do not move up or down more than one level.

The input begins with two integers on a single line that represent the number of rows and columns in the maze, followed by rows of integers (no spaces) that represent the height of each square in the maze. The height of each square is between 0 and 9.

Your output should show a path to the exit (if one exists) using the characters `^`, $>$, `v`, and $<$ to show when you go north, east, south, and west, respectively to get out of the maze. Use dots to show a square that is not part of the path and use an asterisk at the exit. Print the words "`No Solution`" if it is impossible to reach the exit. In this case, you should not print the maze.

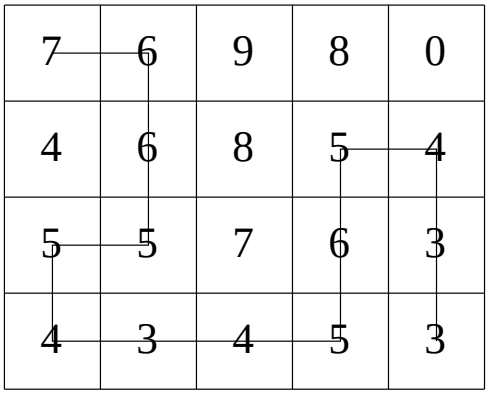
For example, if the input is as follows

```
4 5
76980
46854
55763
43453
```

the output should be

```
>v...
.v.>v
v<.^v
>>>^*
```

The input and output correspond to the following maze.



You may assume that both the number of rows and the number of columns are between 2 and 40 inclusive.

6 Simulating A Simple Computer

This problem requires you to simulate a simple computer. The computer has 8 one-byte registers (This is a very simple computer!) numbered from 1 to 8. The only data the computer knows about is positive integers from 0 to 255 (the values that can fit in one byte).

Write a program that reads in a positive integer specifying the number of assembly instructions (no more than 255), followed by that number of instructions for this computer, one per line with no leading spaces. The instructions are then executed in the order listed, except that three instructions (`SKIPEQ`, `SKIPLT`, and `JUMP`) cause execution to jump to a new location. The program finishes when the `END` instruction executes or when a runtime error occurs. In either case, print out the final values of the eight registers on a single line, with the values separated by spaces. If there is a runtime error in the program, print the words `RUNTIME ERROR` on a separate line.

When the program starts, all registers contain an unknown value. Print the letter '`U`' for any register with an unknown value at the end of the program.

Instruction set

Here is a description of each instruction for this computer. You may assume a program contains only the following instructions, all in uppercase, and in each case, R_1, R_2, and R_3 all have values between 1 and 8. If an instruction requires two or more registers, those registers need not be distinct. For example, the instruction `SUB 3 3 3` sets register 3 to zero if the previous value of that register was known. Otherwise, it generates a `RUNTIME ERROR`.

`SET` R_1 `#`	Set the value of register R_1 equal to the integer represented by the # sign. The value will be between 0 and 255.
`COPY` R_1 R_2	Copy the value in register R_1 to register R_2.
`ADD` R_1 R_2 R_3	Add the values in registers R_1 and R_2; store the result in register R_3.
`SUB` R_1 R_2 R_3	Subtract the value in register R_2 from R_1; store the result in register R_3.
`SKIPEQ` R_1 R_2	Skip one instruction in the program if the value in register R_1 equals the value in R_2. Otherwise, continue to the next instruction.
`SKIPLT` R_1 R_2	Skip one instruction in the program if the value in register R_1 is less than the value in R_2. Otherwise, continue to the next instruction.
`JUMP #`	Jump up or down in the program code to a new instruction. If the integer represented by the # sign is positive, jump down that number of steps. If that number is negative, jump up. The instruction `JUMP 0` jumps to the current line (this creates an infinite loop).
`END`	This instruction represents the end of the program. Any instruction after `END` is executed is ignored.

(Continued on next page)

Error conditions

Generate a `RUNTIME ERROR` whenever one of the following occurs:

- An `ADD` or `SUB` instruction causes a register to acquire an illegal value (either less than 0 or greater than 255). In this case, you should perform the operation before halting and your output should show the illegal value in this register.
- A program executes 255 instructions and has not executed an `END` instruction (assume this is caused by an infinite loop).
- A program attempts to copy an unknown value into another register or perform an addition, subtraction or comparison using an unknown value. Note that it is legal to replace an unknown value with a known value.
- A program attempts to execute an instruction either before the beginning of after the end of the code (either because of a misdirected `JUMP` or because the program fails to `END`).

Examples

Here are sample inputs along with their corresponding output. The first program is an infinite loop that run 127 times before stopping, the second adds the numbers from 1 to 10 into register 8, and the third sorts numbers in the first three registers.

Input	4 SET 3 200 SET 4 1 SUB 3 4 3 JUMP -1	13 SET 8 0 SET 2 1 SET 1 1 SET 3 10 SKIPLT 1 3 JUMP 2 JUMP 3 SKIPEQ 1 3 JUMP 4 ADD 1 8 8 ADD 1 2 1 JUMP -7 END	18 SET 1 30 SET 2 20 SET 3 10 SKIPLT 2 1 JUMP 4 COPY 1 8 COPY 2 1 COPY 8 2 SKIPLT 3 2 JUMP 4 COPY 2 8 COPY 3 2 COPY 8 3 SKIPLT 1 2 JUMP -9 SKIPLT 2 3 JUMP -6 END
Output	U U 73 1 U U U U RUNTIME ERROR	11 1 10 U U U U 55	10 20 30 U U U U 20