Advanced Problem

the Halting Problem

14 March 2009

Source File	`halting.{java,c,cc}`
Input File	`halting.in`
Output File	*standard output*

The *Halting Problem* is a classic decision problem in computer science which basically requires one to determine whether a given program will always stop (or terminate its execution) for an arbitrary given input or will execute forever. Alan Turing proved, in 1936, that it is impossible to solve the halting problem generalizing for any program-input pair. In this problem, however, given the description of a simple language, a program written in the language and an input for this program, you must determine whether the given program stops with the given input and, in the positive case, what output will be produced.

Values and Variables

This language only works with integers from 0 to 999 (inclusive). For the purposes of arithmetic, the successor of 999 is 0, and the predecessor of 0 is 999. Moreover, it has ten variables (`R0` to `R9`), among which `R0` is always assigned the calling value of the program (that is, the input argument) and `R9` is always assigned the exit (return) value. At the beginning of execution of the program, the value zero is assigned to all these variables, with the exception of `R0`, which receives the input parameter.

Arithmetic

The arithmetic operations are

- `MOV` – assignment,
- `ADD` – addition,
- `SUB` – subtraction,
- `MUL` – multiplication,
- `DIV` – integer division, and
- `MOD` – remainder of integer division.

All these operations have the syntax

COMMAND DESTINATION,*OPERAND*

(without spaces between the comma and the operands), where

- *COMMAND* is one of the given operations above,
- *DESTINATION* is one of the ten variables (`R0` to `R9`), and
- *OPERAND* is an integer value (between 0 and 999) or any one of the ten variables.

All the operations modify the value of *DESTINATION*, consequently

```
MOV R4,100
```

is equivalent to assigning the value 100 to `R4`, while

```
MUL R3,R8
```

is equivalent to multiplying `R3` by `R8` and assigning the result to `R3`. The operation `DIV`, as well as `MOD`, returns zero if *OPERAND* is 0 or the given variable conatins the value zero. That is,

```
DIV R4,0
```

is equivalent to

```
MOV R4,0
```

By integer division, we mean the integral part of the quotient of the division (without the fractional part). For example, the integer division of 7 by 2 is 3 (with remainder 1).

Conditionals

There exist six decision flow commands:

- `IFEQ` o,o' ... `ENDIF` – if equal,
- `IFNEQ` o,o' ... `ENDIF` – if different,
- `IFG` o,o' ... `ENDIF` – if greater,
- `IFL` o,o' ... `ENDIF` – if less,
- `IFGE` o,o' ... `ENDIF` – if at least,
- `IFLE` o,o' ... `ENDIF` – if at most.

The syntax of all of them is

```
IFTEST OPERAND,OPERAND′
...
ENDIF
```

(without spaces between the comma and the operands), where each operand may be a variable (`R0` to `R9`) or integer values. Thus, the command

```
IFEQ R4,123
```

is equivalent to testing whether `R4` is equal to 123. When the tested condition is true, the program continues to execute normally the line subsequent to the decision command. When the condition is false, the program proceeds to execute the line subsequent to the nearest following `ENDIF`. All the decision commands must have a corresponding `ENDIF` command.

Subroutines

Finally, there exist the commands `CALL` and `RET`, both with the syntax

COMMAND OPERAND

where *OPERAND* is a variable (`R0` to `R9`) or a direct value (between 0 and 999). The `CALL` command calls the program recursively, passing its operand value as the input parameter, that is, assigning the value of *OPERAND* to variable `R0`. The `RET` command terminates the current execution of the program, returning the value of *OPERAND* as the output result. The last line of the program will always be a command `RET`. It can be observed that, if the program calls itself through the command `CALL`, when execution of the recursive call returns (via `RET`, the value of `R9` is going to be changed to contain the value returned by recursive call. Note also that all the variables (`R0` to `R9`) are local, that is, a subsequent call to the program cannot change values saved in the variables of previous instance, with the exception of, naturally, the value of R9, which receives the return value of the called instance.

The following example illustrates a program that calculates the factorial of a number, annotated with line numbers and comments to aid understanding.

```
1  IFEQ R0,0     check if the value of R0 is 0
2  RET 1         if it is, return 1
3  ENDIF         otherwise continue
4  MOV R1,R0     remember argument
5  SUB R1,1      subtract 1 from argument
6  CALL R1       and recurse with it
7  MOV R2,R9     get result
8  MUL R2,R0     multiply by result
9  RET R2        and return product
```

The following table holds a summary of the commands for reference.

Command	Syntax	Meaning
`MOV`	`MOV` *`DST,OP`*	*DST* ← *OP*
`ADD`	`ADD` *`OP1,OP2`*	*OP1* ← *OP1* + *OP2*
`SUB`	`SUB` *`OP1,OP2`*	*OP1* ← *OP1* - *OP2*
`MUL`	`MUL` *`OP1,OP2`*	*OP1* ← *OP1* * *OP2*
`DIV`	`DIV` *`OP1,OP2`*	*OP1* ← *OP1* / *OP2*
`MOD`	`MOD` *`OP1,OP2`*	*OP1* ← *OP1* % *OP2*
`IFEQ`	`IFEQ` *`OP1,OP2`*	IF *OP1* == *OP2*
`IFNEQ`	`IFNEQ` *`OP1,OP2`*	IF *OP1* ≠ *OP2*
`IFG`	`IFG` *`OP1,OP2`*	check if *OP1* > *OP2*
`IFL`	`IFL` *`OP1,OP2`*	check if *OP1* < *OP2*
`IFGE`	`IFGE` *`OP1,OP2`*	check if *OP1* ≥ *OP2*
`IFLE`	`IFLE` *`OP1,OP2`*	check if *OP1* ≤ *OP2*
`ENDIF`	`ENDIF`	End of a conditional execution block
`CALL`	`CALL` *`OP`*	Call the program with *OP* as input in `R0`
`RET`	`RET` *`OP`*	return *OP* in `R9`

Input

The input contains several test cases. Each test case starts with two integers, L and N, separated by a space, representing the number of lines of the program ($1 \leq L \leq 100$) and the value of the input argument to the program ($0 \leq N < 1000$), respectively. The following L lines contain the program. It can be assumed that it is always syntactically correct in accordance with the rules defined above. All the commands (as well as the variables) only contain capital letters. The end of the input is marked by a case where $L = N = 0$ which should not be processed.

Output

For each test case, your program should produce one line containing an integer which represents the exit (return) value for the given input N, or an asterisk (`*`) in the case that the program never terminates.

Output is emitted to standard output, with no leading or trailing spaces.

C, C++	`stdout`
C++	`cout`
Java	`System.out`

Example

Sample input and output are given in figures 1 and 2, respectively.

```
9␣6
IFEQ␣R0,0
RET␣1
ENDIF
MOV␣R1,R0
SUB␣R1,1
CALL␣R1
MOV␣R2,R9
MUL␣R2,R0
RET␣R2
2␣123
CALL␣R0
RET␣R0
0␣0
```

Figure 1: Input

```
720
*
```

Figure 2: Output