The Halting Problem

By Pedro Demasi Brazil

Timelimit: 1

The halting problem is a classic Computer Science decision problem which consists basically in determining whether a given program will always stop (ie, terminate its execution) for a given arbitrary input or will run forever. Alan Turing proved in 1936 that it is impossible to solve the halting problem generalizing for any pair program-input. In this problem, however, given the description of a simple language, a program written in this language and an entry to this program, you must determine if the given program stops with the given input, and if so, what is the output produced.

This language only works with integer numbers from 0 to 999 (inclusive). Thus, the successor of 999 is 0, and the predecessor of 0 is 999. Furthermore, it has ten variables (R0 to R9), where R0 is always assigned the value of program call (ie, the input parameter) and R9 is always assigned the output value (the return). At the beginning of the program, is assigned the value 0 to all variables, except for R0 that receives the input parameter.

The basics operations are: attribution (MOV), sum (ADD), subtraction (SUB), multiplication (MUL), integer division (DIV) and rest of integer division (MOD). All these operation have a syntax COMMAND OPERATOR1,OPERATOR2 (without spaces between the comma and the operators), where COMMAND is one of these operations, OPERATOR1 is one of the ten variables (R0 to R9) and OPERATOR2 could be one of the ten variables or an integer value (between 0 and 999). All operations change the value of OPERATOR1, thus MOV R4,100 is equivalent to assigned 100 to R4, as MUL R3,R8 is equivalent to multiply R3 by R8 and assigned the result to R3. The operation DIV, as well as MOD, returns 0 (zero) if OPERATOR2 is 0 or if the equivalent variable has the value 0. Thus, DIV R4,0 is equivalent as MOV R4,0. For integer division, we understand the integer part of quotient of the division (no fractional part). For example, the integer division of 7 by 2 is 3 (the rest being 1).

There are six decision flow commands: IFEQ (if equal), IFNEQ (if different), IFG (if greater), IFL (if less), IFGE (if greater or equal) e IFLE (if less or equal). The syntax for all them is COMMAND OPERATOR1,OPERATOR2 (without spaces between the comma and the operators), where OPERATOR1 and OPERATOR2 could be variables (R0 to R9) or integer values (between 0 to 999). Thus, the command IFEQ R4,123 is the equivalent to test is R4 is equal 123. If the tested condition is true, the program continues to run the next line of the decision command normally. If the condition is false, the program turns to run the next line of the nearest ENDIF. **All** of the decision commands must have an ENDIF command correspondent.

Finally, there are the commands CALL and RET, both with the syntax COMMAND OPERATOR, where OPERATOR is a variable (R0…R9) or direct value (between 0 and 999). The command CALL calls the program again, given the OPERATOR as an input parameter, assigning the value of the OPERATOR to the variable R0. The command RET finishes the program execution returning the value of the OPERATOR as an output result. **The last line of the program will always be a command RET**. Note that if the program calls itself by the command CALL, when the program execution returns the value of R9 will be changed with the output value returned by the program. Note also that **all** of the variables (R0…R9) are locals, it means that, when the program calls itself, it cannot change the values stored in the variables of the previous instance, in exception of the value of R9 that receives the return of the called instance.

The following example illustrates a program that calculates the factorial of a number.

Linha	Comando
1	IFEQ R0,0
2	RET 1
3	ENDIF
4	MOV R1,R0
5	SUB R1,1
6	CALL R1
7	MOV R2,R9
8	MUL R2,R0
9	RET R2

1st line: Check if the value of R0 is equal 0, if so runs the next line else jump to the 4th line (nearest ENDIF).

2nd line: Return 1 as output of the program.

3rd line: Mark the end of the decision block initiated at the first line.

4th line: Assign the value from R0 to R1 (R1 ← R0).

5th line: Decrement 1 from R1 (R1 ← R1 – 1).

6th line: Call the program passing R1 as input parameter.

7th line: Save the value of R9 (returned by the previous call) in R2 (R2 ← R9).

8th line: Multiply the value of R2 by R0 (R2 ← R2 * R0).

9th line: Return the value of R2 as output of the program.

The following table brings the summary of the commands for reference:

Comando	Sintaxe	Significado
MOV	MOV OP1,OP2	OP1 ← OP2
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	if OP1 > OP2
IFL	IFL OP1,OP2	if OP1 < OP2
IFGE	IFGE OP1,OP2	if OP1 >= OP2
IFLE	IFLE OP1,OP2	if OP1 <= OP2
ENDIF	ENDIF	Marca fim do bloco de execução condicional
CALL	CALL OP	Chama o programa com OP como entrada
RET	RET OP	return OP

Input

The input contains several test cases. Each test case initiates with two integers, **L** and **N**, representing respectively the numbers of lines of the program ($1 \leq \mathbf{L} \leq 100$) and the value of the input parameter of the program ($0 \leq \mathbf{N} \leq 100$). The following **L** lines contain the program. You can assume that it is always

syntactically correct according to the rules defined above. All of the command (as well as the variables name) will only contain uppercase letters. The end of the input is marked by the case of **L** = **N** = 0 and should not be processed.

Output

For each test case your program must print a single line, containing an integer that represents the output value (return) for the given input **N**, or an asterisk (*) in case of the program never ends.

Sample Input	Sample Output

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
	720
	*

ACM/ICPC South America/Brazil Regional Contest 2006.