
OPHP (4)

P74143_es

Concurso on-line 7, OIE-10 (2010)

Para una secuencia de *tokens* (como los de la salida del problema OPHP 3) tu programa deberá interpretarlos, y decir si representan una expresión OPHP correcta o no. Te recordamos que los *tokens* que puedes recibir son:

- VAR: Una variable
- NUM: Un número
- OP+: Una operación con prioridad aditiva (+, -).
- OP*: Una operación con prioridad multiplicativa (*, /, %).
- (,), =: Paréntesis abierto, cerrado o asignación.

Las expresiones OPHP se definen a partir de las siguientes reglas,

- $TERM \leftarrow VAR \mid NUM \mid (\text{EXPR2})$,
- $EXPR1 \leftarrow TERM \text{ OP* } EXPR1 \mid TERM$,
- $EXPR2 \leftarrow EXPR1 \text{ OP+ } EXPR2 \mid EXPR1$,
- $ASSIG \leftarrow VAR = EXPR2$.

El significado de estas reglas es el siguiente: si tienes una secuencia de *tokens* como las que aparecen en la parte derecha de la regla, puedes substituir dicha secuencia por el correspondiente *token* de la parte izquierda de la regla. El símbolo | es un separador: por ejemplo, puedes obtener un TERM a partir de un NUM, de un VAR, o de una secuencia de 3 *tokens* (, EXPR2 y).

Te pedimos que escriba un programa que determine, dada una secuencia de *tokens* básicos (VAR, NUM, OP+, OP*, (,), =), si es posible transformar la secuencia entera en un *token* de tipo ASSIG, o si por el contrario, la expresión dada no es una asignación válida de OPHP.

Entrada

Una secuencia de líneas, cada una de las cuales contiene una secuencia de *tokens* básicos como los descritos.

Salida

Para cada línea de la entrada, escribe Ok si la expresión puede transformarse en un *token* ASSIG, y No si esto no es posible.

Pista

Aunque este problema puede resolverse usando un atajo, te recomendamos que lo resuelvas programando funciones recursivas del tipo

- `int ASSIG(int from, const vector<string>& tokens);`
- `int EXPR2(int from, const vector<string>& tokens);`
- `int EXPR1(int from, const vector<string>& tokens);`
- `int TERM(int from, const vector<string>& tokens);`

(o el equivalente en el lenguaje de programación que utilices) que devuelvan un entero i tal que `tokens[from], tokens[from+1], ..., tokens[i-1], tokens[i]` es la secuencia más larga de *tokens* que puede ser transformada en un *token* de tipo `ASSIG`, `EXPR2`, `EXPR1` y `TERM` respectivamente. Como estas funciones recursivas se llamarán mutuamente, si programas en C o en C++ tendrás que *declarar* las cabeceras de las funciones antes de *definirlas*.

Ejemplo de entrada 1

```
VAR = NUM
VAR = VAR
VAR = ( ( NUM ) )
VAR = NUM OP+ VAR OP+ NUM
VAR = NUM OP* ( VAR OP+ NUM )
VAR = VAR OP+ VAR OP+ NUM OP+ VAR OP+ NUM
VAR = VAR OP+ VAR OP* NUM OP+ VAR OP+ NUM
VAR = ( NUM OP+ ( NUM OP* VAR ) OP+ NUM )
```

Ejemplo de salida 1

```
Ok
Ok
Ok
Ok
Ok
Ok
Ok
Ok
Ok
```

Ejemplo de entrada 2

```
VAR =
VAR = ( NUM ) )
VAR = NUM NUM
VAR = VAR ( NUM OP+ NUM )
NUM = NUM
= VAR
NUM OP+ NUM
VAR = ( ( NUM ) ) )
VAR = ( ( ( ( NUM ) ) ) )
VAR = ( )
VAR = NUM OP* VAR OP+ NUM OP+
VAR = OP* NUM OP* ( VAR OP+ NUM )
VAR = VAR OP+ VAR OP+ OP+ NUM OP+ NUM
VAR = ( NUM OP+ VAR ) ( NUM OP+ VAR )
VAR = ( VAR OP+ ( NUM OP* VAR ) OP+ ) OP+
```

Ejemplo de salida 2

```
No
No
No
No
No
No
No
No
No
No
No
No
No
No
No
No
No
No
No
```

Información del problema

Autor : Omer Giménez

Generación : 2014-01-31 14:23:28

© Jutge.org, 2006–2014.

<http://www.jutge.org>