Universidad Autónoma de Yucatán
Facultad de Matemáticas

Proyecto

El proyecto final consiste en construir un evaluador de expresiones aritméticas utilizando listas simplemente ligadas para la representación de las pilas empleadas en el desarrollo. Las expresiones aritméticas considerarán las siguientes condiciones:

- Las operaciones serán sobre números enteros únicamente (0, 1, 2, … , 9).
- Los únicos caracteres válidos serán números enteros, operadores aritméticos (‘^’, ‘*’, ´/´, ´+´, ´-´) y los paréntesis.
- Las operaciones posibles son suma, resta, multiplicación, división y potencia.
- Los operadores serán +, -,*, / y ^ correspondientemente para las operaciones de suma, resta, multiplicación, división y potencia.
- Los operadores adicionales que se considerará son los paréntesis ‘(‘, ‘)´.
- La prioridad de los operadores se conserva.
    - () mayor
    - ^
    - * , /
    - +, - menor
- La expresión se capturará como una cadena.
- En caso de que la expresión sea correcta la salida será el resultado de la evaluación de la misma.
- En caso de que la expresión sea incorrecta se determinará el error que puede ser léxico o sintáctico:
    - Se considerará léxico cuando la expresión contiene algún carácter fuera de los válidos. Ejemplos de caracteres inválidos:
        - 2.22, %, #, 3.3.3,;,},
    - Se considerará sintáctico cuando la expresión tiene algún error en la escritura de la expresión. Ejemplos:
        - 2++2, (2*3)+3), ++23*3,(3+3(+3)
    - Los errores léxicos tienen prioridad sobre los sintácticos.
    - Si la expresión tiene algún error solo se reportará en el primero que se encuentre.

Entrada	Salida	Observaciones
2+(2/(2-3))	0	
(2+3)	5	
2+3*5	17	
2.3+3	Error Léxico	Error en el la secuencia 2.3
2/3+2%3	Error Léxico	Error en el carácter %
2++2	Error Sintáctico	Error en la secuencia ‘++’

2+(2/())	Error Sintáctico	Error en la secuencia '()'

El sistema básicamente generará una transformación de notación infija (la notación habitual para las expresiones, operando operador y operando) a notación posfija (primer operando, segundo operando, operador). Los algoritmos se detallan en las páginas posteriores

Notación Infija y Postfija.

1. Conversión de notación infija a postfija

El siguiente algoritmo en pseudocódigo traduce una expresión en notación infija a notación postfija:

- **Entrada:** Una lista que contiene los términos de la ecuación en notación infija (la notación habitual).
- **Salida:** Una lista que contiene los términos de la ecuación en notación postfija.
- **Datos locales:** Una pila, que va a contener operadores y paréntesis izquierdos.

```
INICIO
   Crear pila y la lista de salida, inicialmente vacias.
   MIENTRAS lista de entrada no este vacia y
            no se ha encontrado ningun error HACER
     Extraer el primer termino de la lista (lo llamaremos E)
     SEGUN-SEA E
       CASO E es número :
         Insertar E al final de la lista de salida
       CASO E es la variable x :
         Insertar E al final de la lista de salida
       CASO E es un paréntesis izquierdo :
         Insertar E en la pila
       CASO E es un paréntesis derecho :
         MIENTRAS La pila no este vacía y
                  su cima no sea un paréntesis izquierdo HACER
           Extraer elemento de la pila
           Insertarlo al final de la lista de salida
         FIN-MIENTRAS
         SI Encontramos el parentesis izquierdo ENTONCES
           Extraerlo de la pila y destruirlo
         SINO
           Se ha detectado un ERROR 2
         FIN-SI
         Destruir E
       CASO E es un operador :
         MIENTRAS La pila no este vacía y
                  su cima sea un operador
                  de precedencia mayor o igual que la de E HACER
           Extraer elemento de la pila
           Insertarlo al final de la lista de salida
         FIN-MIENTRAS
         Insertar E en la pila
     FIN-SEGUN-SEA
  FIN-MIENTRAS
  MIENTRAS Pila no esté vacía HACER
    Extraer elemento de la pila
    Insertarlo al final de la lista de salida
  FIN-MIENTRAS
  Destruir pila
FIN
```

A continuación se muestra el estado de la lista de entrada, la lista de salida y la pila en cada iteración del bucle principal del algoritmo al dar como entrada la lista de términos correspondiente al ejemplo del enunciado de la práctica.

```
-4 * ( -5 - 2 ) / ( -1 * x - -3 ) = -1 / -2
```

```
* ( -5 - 2 ) / ( -1 * x - -3 ) = -1 / -2       -4
( -5 - 2 ) / ( -1 * x - -3 ) = -1 / -2       -4       *
-5 - 2 ) / ( -1 * x - -3 ) = -1 / -2       -4       ( *
- 2 ) / ( -1 * x - -3 ) = -1 / -2       -4 -5       ( *
2 ) / ( -1 * x - -3 ) = -1 / -2       -4 -5       - ( *
) / ( -1 * x - -3 ) = -1 / -2       -4 -5 2       - ( *
/ ( -1 * x - -3 ) = -1 / -2       -4 -5 2 -       *
( -1 * x - -3 ) = -1 / -2       -4 -5 2 - *       /
-1 * x - -3 ) = -1 / -2       -4 -5 2 - *       ( /
* x - -3 ) = -1 / -2       -4 -5 2 - * -1       ( /
x - -3 ) = -1 / -2       -4 -5 2 - * -1       * ( /
- -3 ) = -1 / -2       -4 -5 2 - * -1 x       * ( /
-3 ) = -1 / -2       -4 -5 2 - * -1 x *       - ( /
) = -1 / -2       -4 -5 2 - * -1 x * -3       - ( /
= -1 / -2       -4 -5 2 - * -1 x * -3 -       /
-1 / -2       -4 -5 2 - * -1 x * -3 - /       =
/ -2       -4 -5 2 - * -1 x * -3 - / -1       =
-2       -4 -5 2 - * -1 x * -3 - / -1       / =
       -4 -5 2 - * -1 x * -3 - / -1 -2       / =
```

```
-4 -5 2 - * -1 x * -3 - / -1 -2 / =
```

Nota: El algoritmo anterior no puede detectar errores relacionados con la ausencia de paréntesis de cierre.

ALGORITMO PARA EVALUAR UNA EXPRESION POSFIJA

Repetir

Tomar un caracter de la expresión posfija.

Si (caracter es un operando) entonces

Colocarlo en la pila.

Sino {Es un operador}

Tomar los dos valores del tope de la pila, aplicar el operador y colocar el resultado en el nuevo tope de la pila. (Se produce un error en caso de no tener los 2 valores)

Hasta encontrar el fin de la expresión RPN.

NOTA:

Algoritmos tomados de

http://www.infor.uva.es/~cvaca/asigs/AlgInfPost.htm