

Pràctica LP. Part Compiladors: el llenguatge avançat de les llistes de nombres!

Cal fer un compilador per interpretar el llenguatge *NumListLang2*. Aquest és una evolució de *NumListLang*, un llenguatge simple de definició i tractament de llistes heterogènies de nombres. En la nova evolució, s'incorporen noves funcionalitats i instruccions de control de fluxe. El següent exemple mostra el llenguatge:

```
L = []                // L conte la llista buida
L2 = [1,2,3]          // L2 conte una llista amb 3 numeros
L3 = L#L2              // L3 es la concatenacio de L i L2, es a dir L2
L4 = [[ [1,2], 3], 4] // L4 es una llista heterogenea
L5 = lreduce + L4      // L5 te un unic element: la suma dels nombres de L4
L6 = lmap - 1 L4       // L6 s'obte a partir de L4, restant 1 als nombres
print L5              // S'imprimeix [10]
print L6              // S'imprimeix [[ [0,1], 2], 3]
L7 = lfilter != 1 L4   // L7 es una copia d'L4 treient els nombres igual a 1
print L7              // S'imprimeix [[ [2], 3], 4]
L8 = lfilter > 2 L7    // L8 sera [[ [], 3], 4]
flatten L4            // L4 esdevé [1,2,3,4]
if (L2 > L4) then      // Conditional. Operator ">" is for flattened lists.
  print L2             // printing of a list
endif

while (not empty(L2)) do // Loop. function to test the emptiness of a list
  print head(L2)        // First element
  pop(L2)               // Remove first element
endwhile
```

Part 1: Defineix la part lèxica (tokens) i sintàctica (gramàtica). En quant als operadors, inclou suma, resta i multiplicació. En quant als operadors relacionals pel filtratge, inclou major, menor, igual i diferent. Assumeix que els operadors `lreduce`, `lmap` i `lfilter` s'apliquen només sobre identificadors de llistes. La concatenació s'aplica només sobre identificadors i associa a l'esquerra. Les condicions booleanes poden tenir AND, OR i NOT (amb les prioritats estàndard) i usar operacions relacionals `>`, `>=`, `<`, `<=`, `==` o `empty`. Els operadors relacionals només s'apliquen a llistes (no necessàriament identificadors) aplanades i el `>` i `<`, s'interpreten com la comparació lexicogràfica (d'esquerra a dreta) dels elements de la llista. Per exemple, `[2,8,3]` és més gran que `[2,6,1,8]` i `[12,5,4]` és més gran que `[12,5]`. Fes la gramàtica per a que PCCTS pugui reconèixer-la i decorar-la per generar l'AST mostrat a l'anvers de la pàgina.

La regla inicial de la gramàtica és:

```
lists: (list_oper)* <<#0=createASTlist(_sibling);>> ;
```

Part 2: Interpretació. Feu un interpret per al llenguatge *NumListLang2*, que rep el codi del programa i, després de mostrar el AST (al final de l'enunciat trobareu el que s'ha de mostrar per a l'exemple que us hem donat), executa el codi del programa. Heu de mostrar el resultat de les instruccions `print` per la sortida estàndard (mostreu les llistes amb “[” a l'inici i “]” al final i amb una coma separant els elements mostrats).

```

list
  \__=
  |  \__L
  |  \__[
  |  \__=
  |  \__L2
  |  \__[
  |      \__1
  |      \__2
  |      \__3
  |  \__=
  |  \__L3
  |  \__#
  |      \__L
  |      \__L2
  |  \__=
  |  \__L4
  |  \__[
  |      \__[
  |          |  \__[
  |          |      \__1
  |          |      \__2
  |          |      \__3
  |          \__4
  |  \__=
  |  \__L5
  |  \__lreduce
  |      \__+
  |      \__L4
  |  \__=
  |  \__L6
  |  \__lmap
  |      \__-
  |      \__1
  |      \__L4
  |  \__print
  |      \__L5
  |  \__print
  |      \__L6
  |  \__=
  |  \__L7
  |  \__lfilter
  |      \__!=
  |      |  \__1
  |      \__L4
  |  \__print
  |      \__L7
  |  \__=
  |  \__L8
  |  \__lfilter
  |      \__>
  |      |  \__2
  |      \__L7
  |  \__flatten
  |      \__L4
  |  \__if
  |      \__>
  |      |  \__L2
  |      |  \__L4
  |      \__list
  |          \__print
  |          \__L2
  |  \__while
  |      \__not
  |      |  \__empty

```

```

|          \_L2
\_list
  \_print
    |      \_head
    |          \_L2
  \_pop
    \_L2

```