Pràctica prat de Compiladors (LP): El Llenguatge dels Plots

Cal fer un compilador per interpretar un llenguatge de definició de dades que poden ser visualitzades en un plot de dues dimensions. El llenguatge permet definir datasets de dues dimensions, operar amb ells i visualitzar-los amb comandes bàsiques. També es disposa de mètodes per operar iterativa o condicionalment amb ells, així com a funcions per normalitzar, verificar que estiguin ben formats, o arreglar datasets que no estiguin ben formats. El següent exemple mostra el llenguatge:

```
DP = [<2,3>, <3,4>]           // Definim el dataset DP, on el primer punt correspon al
                              // 2 (eix de les Xs), i 3 (eix de les Ys), etc ...

DP1 = [<2,5>]·[<3,3>,<7,4>]·[<12,3>]   // Operador de concatenació
PLOT(DP)                               // Imprimeix el plot corresponent al dataset DP
DP2 = NORMALIZE(DP1)                    // Aplica la funció de normalització sobre DP1
                                       // és a dir, resta el mínim dels components en cada dimensió
                                       // DP2 = [<0,2>,<1,0>,<5,1>,<10,0>]
PLOT(DP2)
LOGPLOT(DP2)                           // Plot logarítmic
DP3 = DP2·[<2,4>,<13,3>]·DP
DP4 = POP(DP3)                         // Elimina l'ultim punt a DP3 i el resultat ho copia a DP4
DP5 = PUSH(POP(DP3),<7,6>)             // Seqüència de push i pop, resultat copia a DP5
PLOT(NORMALIZE(DP5·[<3,4>,<23,13>]))   // Expressió complexa dintre d'un plot
WHILE (NOT EMPTY(DP5))                 // Bucle utilitzant com a condició funció empty
  DP5 = POP(DP5)
  PLOT(DP5)
ENDWHILE
IF (NOT CHECK(DP3))                    // Condicional per decidir si DP3 esta ben format
  DP6 = AMEND(DP3)                     // Com no ho està (té 2 definicions del punt 2 a les Xs)
                                       // ho arreglarà eliminant el darrer dels punts conflictius
ENDIF
WHILE (ITH(3,DP4) != ITH(2,DP1))       // Funció per seleccionar l'element i-éssim d'un dataset
  DP4 = POP(DP4)                       // També es pot comparar amb '>', '<' i '=='
ENDWHILE                               // Tots els operadors de comparació operen amb parells d'enters
```

1. Gramàtica

Defineix la part lèxica (tokens) i sintàctica (gramàtica). Fes la gramàtica per a que PCCTS pugui reconèixer-la i decorar-la per generar l'AST mostrat al dos últims fulls. Mira tots els comentaris en l'exemple per acabar de comprendre el llenguatge.

Les condicions del `WHILE` i el `IF`, poden tenir una expressió booleana amb conjuncions, disjuncions i negacions (i paréntesis), amb les prioritats usuals. Per altra banda, els operados `<` i `>` comparen els parells lexicogràficament, és a dir comparen la primera component per decidir si és més petit o més gran i, només en cas d'igualtat, comparen la segona component.

Heu de considerar també que si un operadors espera un plot en algun argument, aquest pot rebre una expressió qualsevol que tingui com a resultat un plot.
La regla inicial de la gramàtica és:

```
plots: linterpretation "@"! <<#0=createASTstring(_sibling,"DataPlotsProgram");>>;
```

on la funció `createASTstring` crea un node tipus llista amb l'etiqueta passada com a segon paràmetre. L'heu d'implementar.

2. Semàntica

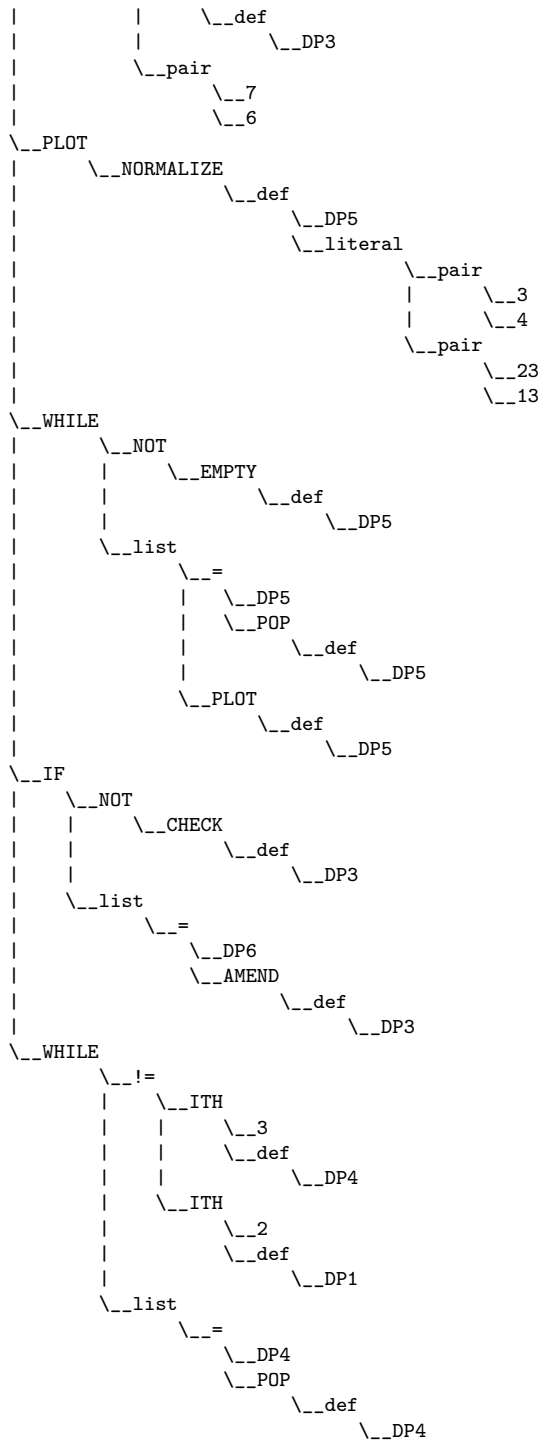
Fes un programa que donat una entrada com la de l'exemple, recorri el corresponent arbre per interpretar l'entrada. La sortida ha de ser primer l'AST (**exactament** com es mostra en les següents dues fulles) i el resultat de les impressions posant `PLOT` o `LOGPLOT` (segons el cas) seguit de la llista de parells.

Per això pot ser útil tenir un diccionari com a variable global, que relaciona strings i llistes de parells on es manté l'estat actual de cada dataset:

```
map<string, list<pair<int,int> > > dps;
```

Assumeix també que les funcions `PUSH`, `POP`, `NORMALIZE`, `AMEND` NO modifiquen el paràmetre.

```
DataPlotsProgram
 \__list
        \__=
        |   \__DP
        |   \__def
        |         \__literal
        |                   \__pair
        |                   |      \__2
        |                   |      \__3
        |                   \__pair
        |                          \__3
        |                          \__4
        \__=
        |   \__DP1
        |   \__def
        |         \__literal
        |         |         \__pair
        |         |                \__2
        |         |                \__5
        |         \__literal
        |         |         \__pair
        |         |         |      \__3
        |         |         |      \__3
        |         |         \__pair
        |         |                \__7
        |         |                \__4
        |         \__literal
        |                   \__pair
        |                          \__12
        |                          \__3
        \__PLOT
        |      \__def
        |            \__DP
        \__=
        |   \__DP2
        |   \__NORMALIZE
        |               \__def
        |                     \__DP1
        \__PLOT
        |      \__def
        |            \__DP2
        \__LOGPLOT
        |         \__def
        |               \__DP2
        \__=
        |   \__DP3
        |   \__def
        |         \__DP2
        |         \__literal
        |         |         \__pair
        |         |         |      \__2
        |         |         |      \__4
        |         |         \__pair
        |         |                \__13
        |         |                \__3
        |         \__DP
        \__=
        |   \__DP4
        |   \__POP
        |         \__def
        |               \__DP3
        \__=
        |   \__DP5
        |   \__PUSH
        |          \__POP
```


__def
__DP3
__pair
__7
__6
__PLOT
__NORMALIZE
__def
__DP5
__literal
__pair
__3
__4
__pair
__23
__13
__WHILE
__NOT
__EMPTY
__def
__DP5
__list
__=
__DP5
__POP
__def
__DP5
__PLOT
__def
__DP5
__IF
__NOT
__CHECK
__def
__DP3
__list
__=
__DP6
__AMEND
__def
__DP3
__WHILE
__!=
__ITH
__3
__def
__DP4
__ITH
__2
__def
__DP1
__list
__=
__DP4
__POP
__def
__DP4