```
# numlistv1.plcc
# Language specification for a list of numbers
#
# Lexical spec: becomes values for an enumerated type
#
skip WHITESPACE '\s+'
NUMBER '\d+'
LPAREN '\('
RPAREN '\)'

# End of lexical spec
%

# Grammar rules start here, start rule first.
# Putting a token type in angle brackets causes its value to be saved.

<numSeq> ::= LPAREN <numbers> RPAREN
<numbers>:NonEmptyNumbers ::= <NUMBER> <numbers>
<numbers>:EmptyNumbers ::=

# Abstract classes created
# Numbers
#   subclcasses NonEmptyNumbers, EmptyNumbers
#
# Classes created: NumSeq, NonEmptyNumbers, EmptyNumbers
#
# NumSeq
#   fields numbers(Numbers)
# NonEmptyNumbers, extends Numbers
#   fields number(Token), numbers(Numbers)
# EmptyNumbers, extends Numbers
#   no fields!

# End of syntax spec
%

# Custom code would go here.
```

```
# numlistv2.plcc
#
# Alternate form: uses a special iterative construct in PLCC

skip WHITESPACE '\s+'
NUMBER '\d+'
LPAREN '\('
RPAREN '\)'

# End of lexical spec
%

# Grammar rules start here, start rule first.
# Putting a token type in angle brackets causes its value to be saved.

<numSeq> ::= LPAREN <nums> RPAREN
<nums> **= <NUMBER>

# Classes created: Numseq, Nums
#
# Lon
#   field nums(Nums)
# Nums
#   field numberList(List<Token>)

# End of syntax spec
%

# Custom code would go here.
# numlistv3.plcc
#
# Custom code added to tree root class

# Lexical specification
skip WHITESPACE '\s+'
NUM '\d+'
LPAREN '\('
RPAREN '\)'
%
# Grammar
<numSeq> ::= LPAREN <nums> RPAREN
<nums>   **= <NUM>
%

NumSeq
%%%

    public String toString() {
        String ret = "( ";
        for (Token tok: nums.numList) {
            ret += tok + " ";
        }
        return ret + ")";
    }

%%%
```

```
# numlistv4.plcc
#
# Note printing of token type.

skip WHITESPACE '\s+'
NUMBER '\d+'
LPAREN '\('
RPAREN '\)'

# End of lexical spec
%

# Grammar rules start here, start rule first.
# Putting a token type in angle brackets causes its value to be saved.

<numSeq> ::= LPAREN <nums> RPAREN
<nums>   **= <NUMBER>

# Classes created: NumSeq, Nums
#
# NumSeq
#   field nums(Nums)
# Nums
#   field numList(List<Token>)

# End of syntax spec
%

# Custom code


NumSeq

%%%
@Override
public String toString() {
    String result = "(( ";
    for (Token tok: nums.numberList) {
        result += "(" + tok.val + ')' + tok.str + " ";
    }
    return result + "))";
}
%%%
```

```
# numlistv5.plcc
#
# The tree nodes print the integers in the list through recursive descent
# of the tree.

# Lexical specification
#
skip WHITESPACE '\s+'
NUM '\d+'
LPAREN '\('
RPAREN '\)'
COMMA ','

%

# Grammar

<numSeq>  ::= LPAREN <nums> RPAREN
<nums>    **= <number> +COMMA
<number>  ::= <NUM>

%

NumSeq
%%%

    public String toString() {
        return nums.toString();
    }

%%%

Nums
%%%

    public String toString() {
        String ret = "( ";
        for (Number number: numberList) {
            ret += number + " ";
        }
        return ret + ")";
    }

%%%

Number
%%%

    public String toString() {
        return num.str;
    }

%%%
```

```
# numlistv6.plcc
#
# Change semantic action to print the minimum value of the list.
# NOTE that this is the recursive grammar.

# Lexical specification
skip WHITESPACE '\s+'
LPAREN '\('
RPAREN '\)'
NUM '\d+'
%
# Grammar
<numSeq>        ::= LPAREN <NUM> <nums> RPAREN
<nums>:NumsNode ::= <NUM> <nums>
<nums>:NumsNull ::=
%

# add a toString method to the NumSeq class
NumSeq
%%%
    public String toString() {
        int minSoFar = Integer.parseInt(num.str);
        int m = nums.min(minSoFar); // get the overall minimum
        return "minimum value = " + m;
    }
%%%

Nums
%%%
    public abstract int min(int minSoFar);
%%%

NumsNode
%%%
    public int min(int minSoFar) {
        return 0; // How to implement?
    }
%%%

NumsNull
%%%
    public int min(int minSoFar) {
        return minSoFar;
    }
%%%
```