L

Любая программа на языке L - набор *statement*, которые выполнются последовательно. *Statement* делятся на несколько типов:

(1) объявление функций и переменных
(2) присваивание переменных
(3) выражение
(4) управляющие конструкции
(5) input-output

После всех statement (кроме управляющих конструкций) обязательно должены идти точка с запятой (';')

Объявление функций:

```
fun <FUNCTION_NAME> (<PARAM_NAME1>, <PARAM_NAME1>, ...) {
    <STATEMENT_1>
    <STATEMENT_2>
    <STATEMENT_3>
    ...
}
```

.

При этом множество параметров и *statement* может быть пустым, а название функции должно содержать только цифры, латинские буквы и нижние подчеркивание, и при этом не должно начинаться с цифры.

Объявление переменных:

```
var <VAR_NAME>
```

.

или

```
var <VAR_NAME> = <EXPRESSION>
```

.

При этом на имена переменных действуют те же ограниения, что и на имена функций.

Присваивание переменных:
По ссылке:

```
<VAR_NAME> := <EXPRESSION>
```

.

По значению:

```
<VAR_NAME> = <EXPRESSION>
```

.

Управляющие конструкции:
Циклы:

```
while (<EXPRESSION>) {
    <STATEMENTS>
}
```

.

Условные операторы:

```
if (<EXPRESSION>) {
    <STATEMENTS>
}
if (<EXPRESSION>) {
    <STATEMENTS>
} else {
    <OTHER_STATEMENTS>
}
```

.

Выражения (expression):

(Вставка кода из antlr с комментариями, т.к. кажется, что это наиболее удобный для чтения формат):

Общее описание выражения

```
expression: binaryExpression | '(' expression ')' | atomicExpression;
```

.

```
functionCall: Identifier '(' (arguments)? ')';
arguments: expression (',' expression)*;
```

.

Примитивное выражение, имеет наивысший приоритет
Здесь Identifier - название переменной, а Literal - число

```
atomicExpression: functionCall | Identifier | Literal;
```

.

```
binaryExpression: eqExpression | boolExpression | mdExpression | pmExpression;
```

.

Каждое из последующих видов бинарных выражений является праворекурсивным и, слеовательно, однозначным
Так же для каждого типа выражения строится Helper, который содержит все возможные выражения с более высоким приоритетом

```
boolExpression: boolExpressionHelper BoolOp (boolExpressionHelper | boolExpression);
boolExpressionHelper: eqExpressionHelper | eqExpression;

eqExpression: eqExpressionHelper EqOp (eqExpressionHelper | eqExpression);
```

```
eqExpressionHelper: pmExpressionHelper | pmExpression;

pmExpression: pmExpressionHelper PmOp (pmExpressionHelper | pmExpression);
pmExpressionHelper: mdExpressionHelper | mdExpression;

mdExpression: mdExpressionHelper MdOp (mdExpressionHelper | mdExpression);
mdExpressionHelper: atomicExpression | '(' expression ')';
```

.

Input-Output:

```
read <VAR_NAME>
```

.

```
write <EXPRESSION>
```

.