RE description of Extended PL/0 lexical structure

Program ::= (Token | Whitespace)*

Token ::= Id | Integer | Keyword | Operator | Punctuation

Punctuation ::= **;** | **:** | **.** | **,** | **(** | **)** / ***[*** / ***]***

Keyword ::= **module** | **procedure** | **begin** | **end** | **const** | **var** | **int** | **if** | **then** | **while** | **do** | **input** | **output** | **odd** / ***else*** / ***for*** / ***to*** / ***break*** / ***return*** / ***array*** / ***of*** / ***bool*** / ***true*** / ***false*** / ***and*** / ***or***

Operator ::= **:=** | ***** | **/** | **+** | **-** | **=** | **<>** | **<=** | **<** | **>=** | **>**

Integer ::= Digit$^+$

Id ::= Letter AlphaNumeric*

AlphaNumeric ::= Letter | Digit

Digit ::= **0** | ... | **9**

Letter ::= **a** | ... | **z** | **A** | ... | **Z** / **_**

Whitespace ::= <space> | <tab> | <newline> / *Comment*

Comment *::=* ***#*** *not(<newline>)** *<newline>*

EBNF description of Extended PL/0 syntax

Program ::= **module Id ;** Block **Id .**

Block ::= { Decl **;** } **begin** StmtList **end**

Decl ::= ConstDecl | ProcDecl | VarDecl

ConstDecl ::= **const** ConstDeclItem { **,** ConstDeclItem }

ConstDeclItem ::= **Id :** Type **=** ConstExpr

ConstExpr ::= *Expr*

VarDecl ::= **var** VarDeclItem { **,** VarDeclItem }

VarDeclItem ::= **Id :** Type

ProcDecl ::= **procedure Id (** [FormalDecl { **,** FormalDecl }] **)** *[**:** Type]* **;** Block **Id**

FormalDecl ::= *[**var**]* **Id :** Type

Type ::= **int** *| **bool** | **array [** ConstExpr **] of** Type*

StmtList ::= { Stmt **;** }

Stmt ::= CallStmt | AssignStmt | OutStmt | IfStmt | WhileStmt *| ForStmt |*
BreakStmt | ReturnStmt

CallStmt ::= **Id (** [Exprs] **)**

AssignStmt ::= LValue **:=** Expr

LValue ::= **Id** *| LValue **[** Expr **]***

OutStmt ::= **output :=** Expr

IfStmt ::= **if** *Expr* **then** StmtList *[**else** StmtList]* **end**

WhileStmt ::= **while** *Expr* **do** StmtList **end**

*ForStmt ::= **for Id :=** Expr **to** Expr **do** StmtList **end***

*BreakStmt ::= **break***

*ReturnStmt ::= **return** [Expr]*

Exprs ::= Expr { **,** Expr }

*Expr ::= Conjunction { **or** Conjunction }*

*Conjunction ::= Test { **and** Test }*

Test ::= **odd** Sum | Sum Relop Sum | Sum

Relop ::= **<=** | **<>** | **<** | **>=** | **>** | **=**

Sum ::= Term { (**+** | **-**) Term }

Term ::= Factor { (***** | **/**) Factor }

Factor ::= **-** Factor | LValue | ***Id (** [Exprs] **)*** | **Integer** | ***true*** | ***false*** | **input** | **(** Expr **)**