Mapl grammar (*N** denotes 0, 1 or more repetitions of *N*)

Program → *ProcDecl MethodDecl**
MethodDecl → *ProcDecl*
→ *FunDecl*
ProcDecl → **proc** *id* **(** *FormalList* **)** **{** *Statement** **}**
FunDecl → **fun** *Type id* **(** *FormalList* **)** **{** *Statement** **return** *Exp* **;** **}**
FormalList → *Type id FormalRest**
→
FormalRest → **,** *Type id*
Type → *Type* **[ ]**
→ **boolean**
→ **int**
Statement → *Block*
→ **local** *Type id* **;**
→ *Var* = *Exp* **;**
→ *PrimaryExp* **[** *Exp* **]** = *Exp* **;**
→ **if (** *Exp* **)** **then** *Statement* **else** *Statetment*
→ **while (** *Exp* **)** **do** *Statement*
→ **output** *Exp* **;**
→ **outchar** *Exp* **;**
→ *id* **(** *ExpList* **)** **;**
Block → **{** *Statement** **}**
Exp → *PrimaryExp op PrimaryExp*
→ *PrimaryExp* **[** *Exp* **]**
→ *PrimaryExp* **. length**
→ *PrimaryExp*
PrimaryExp → *INTEGER_LITERAL*
→ **true**
→ **false**
→ *Var*
→ **new** *Type* **[** *Exp* **]**
→ *id* **(** *ExpList* **)**
→ **!** *PrimaryExp*
→ **isnull** *PrimaryExp*
→ **(** *Exp* **)**
Var → *id*
ExpList → *Exp ExpRest**
→
ExpRest → **,** *Exp*

See overleaf for definitions of *op, id, INTEGER_LITERAL* and the comment syntax.

op is one of the following binary operators: **`and < == div + - *`**

id is a sequence of letters, digits and underscores, starting with a letter.

INTEGER_LITERAL is a sequence of decimal digits. [Note that this means that negative numbers are *not* integer literals.]

Comments: these can either be placed between `/*` and `*/` or make up the remainder of a line beginning with `//`