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

Program	→ *FunDef FunDef**
Type	→ *Type* **[ ]**
	→ **bool**
	→ **int**
	→ **unit**
VarDecl	→ *Type id* **;**
FunDef	→ **def** *Type id* **(** *FormalList* **) {** *VarDecl* Statement** **}**
FormalList	→ *Type id FormalRest**
	→
FormalRest	→ **,** *Type id*
Statement	→ **{** *Statement** **}**
	→ *id* **=** *Exp* **;**
	→ *PrimaryExp* **[** *Exp* **]** **=** *Exp* **;**
	→ **if (** *Exp* **)** *Statement* **else** *Statement*
	→ **while (** *Exp* **)** *Statement*
	→ **output** *Exp* **;**
	→ **outchar** *Exp* **;**
	→ **return** *Exp* **;**
	→ *id* **(** *ExpList* **) ;**
Exp	→ *PrimaryExp op PrimaryExp*
	→ *PrimaryExp* **[** *Exp* **]**
	→ *PrimaryExp* **. length**
	→ *id* **(** *ExpList* **)**
	→ *PrimaryExp*
PrimaryExp	→ *INTEGER_LITERAL*
	→ **true**
	→ **false**
	→ *id*
	→ **new** *Type* **[** *Exp* **]**
	→ **!** *PrimaryExp*
	→ **isnull** *PrimaryExp*
	→ **(** *Exp* **)**
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: **`&& || < == / + - *`**

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

INTEGER_LITERAL is a non-empty sequence of decimal digits optionally prefixed with a minus sign.

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