Advanced MindBlock (AMB)

Regular Expressions

Keywords:

- START_PROGRAM
- END_PROGRAM.
- START_SUB
- END_SUB.
- GOSUB
- CODE
- IF
- THEN
- ELSE
- END_IF
- WHILE
- DO
- END_WHILE
- INT
- STRING
- PRINT
- INPUT_INT
- INPUT_STRING

Symbols:

- **softOpen** → (
- **softClose** →)
- **hardOpen** → [
- **hardClose** →]
- **semi** → ;
- **assignment** → :=
- **colon** → :
- **multOp** → * | /
- **addOp** → + | −
- **compOp** → < | > | =< | => | = | != ***(NOTE: NO SPACE)***

Symbol Collections:

- *nonZeroDigit* → 1|2|3|4|5|6|7|8|9
- *digit* → 0 | nonZeroDigit
- *naturalNumber* → nonZeroDigit digit*
- *negativeNumber* → −naturalNumber ***(NOTE: NO SPACE)***
- **number** → 0 | naturalNumber | negativeNumber
- *character* → [a−z] | [A−Z] | digit
- **characterString** → "(character|space)*" (***NOTE: spaces are included until end quote, also quotes cannot be within quote***)
- **label** → ([a−z] | [A−Z]) character*

Grammar

Program ⇒ `START_PROGRAM` *VariableList*

VariableList ⇒ *Variable VariableList*

VariableList ⇒ `CODE` *SubList*

Variable ⇒ `INT label;`

Variable ⇒ `STRING label;`

Variable ⇒ `[`*ArrayVariable*

ArrayVariable ⇒ `INT] label [number];`

ArrayVariable ⇒ `STRING] label [number];`

SubList ⇒ `START_SUB label:` *CodeList SubList*

SubList ⇒ `END_PROGRAM.`

CodeList ⇒ *CodeLine CodeList*

CodeList ⇒ `END_SUB.`

CodeLine ⇒ *LineLabel*

CodeLine ⇒ *Condition*

CodeLine ⇒ *Loop*

CodeLine ⇒ `PRINT(`*Expression*`);`

CodeLine ⇒ `GOSUB label;`

LineLabel ⇒ `label` *Assignment*

Assignment ⇒ `:=` *ExpressionOrInput*`;`

Assignment ⇒ `[number] :=` *ExpressionOrInput*`;`

Assignment ⇒ `[label] :=` *ExpressionOrInput*`;`

Condition ⇒ `IF` *Expression* `compOp` *Expression* `THEN` *ThenCodeList*

ThenCodeList ⇒ *CodeLine ThenCodeList*

ThenCodeList ⇒ `ELSE` *ElseCodeList*

ThenCodeList ⇒ `END_IF`

ElseCodeList ⇒ *CodeLine ElseCodeList*

ElseCodeList ⇒ `END_IF`

Loop ⇒ `WHILE` *Expression* `compOp` *Expression* `DO` *WhileCodeList*

WhileCodeList ⇒ *CodeLine WhileCodeList*

WhileCodeList ⇒ `END_WHILE`

ExpressionOrInput ⇒ *Expression*

ExpressionOrInput ⇒ `INPUT_INT`

ExpressionOrInput ⇒ `INPUT_STRING`

Expression ⇒ *Term TermTail*

TermTail ⇒ `addOp` *Term TermTail*

TermTail ⇒ ε

Term ⇒ *Factor FactorTail*

FactorTail ⇒ `multOp` *Factor FactorTail*

FactorTail ⇒ ε

Factor ⇒ `(` *Expression* `)`

Factor ⇒ `number`

Factor ⇒ `characterString`

Factor ⇒ `label` *PossibleArray*

PossibleArray ⇒ `[` *ArrayNumberOrLabel*

PossibleArray ⇒ ε

ArrayNumberOrLabel ⇒ `number]`

ArrayNumberOrLabel ⇒ `label]`

Execution Rules

- All Variables must be declared in the first section of code.
- You will begin execution in a subroutine label `main`. If `main` does not exist, then you will receive a runtime error. (Note, `main` is not part of the grammar. You can include it anywhere in your subroutine list.)
- Upon completing a subroutine you will return to the line of code you called GOSUB from. (Exactly how functions work, except there are no parameters, no return statement, and all variables are global scope)
- When you declare an array, using the `[TYPE] label [number]` syntax. The entire array is of the given TYPE and the size of the array is given by the value in the `[number]`. If the value in `[number]` is negative, then you will receive a compile time error.
- The PRINT command will display the evaluated expression of whatever is in ()
- Mathematical Expressions are evaluated via normal INTEGER math
- If you add an integer expression to a string, that evaluation of the integer expression is concatenated to the string expression.
- If you multiply an integer expression by a string, then the string is repeated however many times the integer expression multiplies by
- If you attempt to divide or subtract a string, then the you will receive a run time error.
- There cannot be two subroutine labels that are the same
- Multiple subroutine statements with the same name will cause a run-time error.
- Two variables cannot have the same name. Violation of this will cause a run time error.
- `INPUT_INT` and `INPUT_STRING` will return there respective types