PLC Exam 2 ReadMe

**Variable Declaration**

datatype <variable name> = <expression>

**Conditions**

**Instead of using the word “if” for an if else statement, my language will use “incase”.**

**“eincase” would be the same as “else if”.**

incase <condition> then

# statements

eincase <condition> then

# statements

else then

# statements

end

**Print statements**

print <expression>

**Functions**

fun <function name> [ <argument1, argument2>, ... ]

<body>

return <expression>

end

**Loops**

#LOOP

LOOP <variable> in <lower\_bound>..<upper\_bound>

# statements

End

**Instead of a while loop my language will use a “when loop”**

# when LOOP

when <condition> then

# statements

end

**Data Types**

**Numeric**

small example = 123

this would be 2 bytes

numer example = 123543

this would be 4 bytes

large example = 1234567891

this would be 8 bytes

**Text**

symbol = ‘a’

(similar to a char in java this would be 1 byte)

text = “hello world”

(similar to a string from java, this would be 11 bytes)

**Logical**

logic = true

(similar to a boolean in java this would be 2 bytes)

**Rules for Recognizing Lexemes as Tokens**

Comment {"\\#.\*"}

LineBreak {"[\\n\\r]"}

Whitespace {"[\\s\\t]"}

Keyword {"(incase|eincase|else|then|end|print|fun|return|when|small|large|numer|LOOP|)(?=\\s|$)"}

GroupDivider {"(\\[|\\]|\\,|\\{|}|[.]{2})"}

Logical {"(true|false)(?=\\s|$)"}

Numeric {"([-]?(?=[.]?[0-9])[0-9]\*(?.]{2})[.]?[0-9]\*)"}

Null {"(null)(?=,|\\s|$)"}

Text {"\"([^\"]\*)\""}

Operator {"([+]|[-]|[\*]|[/]{1,2}|[%]|>=|>|<=|[<]{1,2}|[=]{1,2}|!=|[!]|[:]{2}|[(]|[)]|(new|and|or)(?=\\s|$))"}

VariableName {"[a-zA-Z\_]+[a-zA-Z]{6,9}\*"}

**Production Rules in EBNF format**

**Mathematical Expressions**

<expression> 🡪 <term> { ("+" | “-“|)} <term>

<term> 🡪 <factor> { ("\*"|"/") } <factor>

<factor>🡪 <constant> | <variable> | "(" <expression> ")"

<variable>🡪 "x" | "y" | "z"

<constant>🡪 <digit> {<digit>}

<digit>🡪”0” | “1” | “...” | “9”

**Variable Declaration**

<variable> 🡪 <datatype> <variable name> = <expression>

<datatype> 🡪 small|numer|large|symbol|text|logic

<variable name> 🡪 {[a-zA-Z\_]+[a-zA-Z]{6,9}\*}

**Statements/Selection Statements**

<statement> 🡪<incase-statement>|<when-statement>|<variable>

<incase-statement> 🡪 incase(<condition>)then <statement>| incase(<condition>)eincase <statement>|

**LOOPS**

<LOOP> 🡪<variable> in (<lower\_bound>...<upperbound>)<statement>

<when-LOOP> 🡪 when <LOOP> <condition> then <statement>

**Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Value** | **Precedence** |
| Assignment | = | 1 |
| Logical OR | or | 2 |
| Logical AND | and | 3 |
| Left Paren | ( | 4 |
| Right Paren | ) | 4 |
| Equals | == | 5 |
| Not Equals | != | 5 |
| Greater Than or Equals | >= | 5 |
| Greater Than | > | 5 |
| Less Than or Equals | <= | 5 |
| Less Than | < | 5 |
| Addition | + | 6 |
| Division | / | 6 |
| Multiplication | \* | 7 |
| Modulo | % | 7 |
| Subtraction | - | 7 |
| NOT | ! | 8 |

**Mathematical Precedence**

This language will follow a PADMES order of operation shown by the following binary tree.

![Diagram

Description automatically generated

**Keywords**

incase

eincase

else

then

end

print

fun

return

LOOP

numer

large

small

text

logic

symbol