Exam 2 PLC

Nathan Mone

1. **(15 Points)** Define the rules for recognizing all lexemes as their proper token, and clearly define integer token codes for each token required for this language.

* Tokens:
  + tiny, medi, big, huge (for 1byte, 2byte, 4byte, and 8byte integers)
  + +, -, \*, /, %, <, >, <=, >=, ==, !=, =, (, ) (operations)
  + start, finish, | (beginning and end of program, and statement separator)
  + conloop, forloop, perform, ? (while loop, for loop, do while loop, if)
* Regex:
  + (start)|(finish)|(tiny)|(medi)|(big)|(huge)|(conloop)|(forloop)|(perform)|(\?)|(\+)|(-)|(\\*)|(\/)|(%)|(<=)|(>=)|(<)|(>)|(==)|(\!=)|(=)|(\()|(\))|(\|)|(\{)|(\})|([0-9]+)|([a-zA-Z\_]{6,8})
  + I separate each lexeme into groups because it is easier to define token codes this way in python, even if the regex isn’t very readable. When is regex ever readable though.
* Token Codes (in order of the groups in the RegEx):
  + 0 : "program beginning",
  + 1 : "program ending",
  + 2 : "1 byte integer declaration",
  + 3 : "2 byte integer declaration",
  + 4 : "4 byte integer declaration",
  + 5 : "8 byte integer declaration",
  + 6 : "while loop",
  + 7 : "for loop",
  + 8 : "do while loop",
  + 9 : "if statement",
  + 10: "add",
  + 11: "sub",
  + 12: "multiply",
  + 13: "divide",
  + 14: "modulus",
  + 15: "lessthan or equal",
  + 16: "greaterthan or equal",
  + 17: "lessthan",
  + 18: "greaterthan",
  + 19: "equal to",
  + 20: "not equal to",
  + 21: "variable assignment",
  + 22: "left paren",
  + 23: "right paren",
  + 24: "separator",
  + 25: "left bracket",
  + 26: "right bracket",
  + 27: "integer",
  + 28: "variable name”

1. **(15 Points)** Define production rules for implementing the mathematical syntax of operators and operands, loops, variable declaration, selection statements.

* **Mathematical Syntax of Operators and Operands:**
  + Instead of PEMDAS, this would be PASEMDO (Parentheses, Addition, Subtraction, Exponent, Multiply, Division, Modulus)
  + <Variable> → `some valid name`
  + <Expr> → <Expr> % <Division> | <Division>
  + <Division> → <Division> / <Multiply> | <Multiply>
  + <Multiply> → <Multiply> \* <Exp> | <Exp>
  + <Exp> → <Exp> ^ <Sub> | <Sub>
  + <Sub> → <Sub> - <Add> | <Add>
  + <Add> → <Add> + <Term> | <Term>
  + <Term> → ( <Expr> ) | `some number`
* **Loops, selection:**
  + Production rules for conloop, forloop, perform, ?, and variable declaration. Connects to above production rules.
  + <Statement> → <?> | <Conloop> | <Forloop> | <Perform> | <Variable>
  + <?> → `?` `(` <Compare> `)` `{` <Statement> `}`
  + <Conloop> → `conloop` `(` <Compare> `)` `{` <Statement> `}`
  + <Forloop> → `forloop` `(` <Variable> `=` <Expr> `|` <Variable> `=` <Expr> `)` `{` <Statement> `}`
  + <Perform> → `perform` `{` <Statement> `}` <Conloop>
  + <Compare> → <brel> { (`!=`|`==`) <brel> }
  + <brel> → <bexpr> { (`<=`|`>=`|`<`|`>`) <bexpr> }
  + <bexpr> → <expr> % <division> | <Division> | [!] <division>

1. **(10 points)** Show whether every rule set in your language conforms to the standard of an LL Grammar.