Lexer y Parser para un lenguaje basado en pseudocódigo

Traductores

Guillermo Licea Sandoval

```
/*  Gramática del Pseudocódigo
 *
    <Programa> ::= inicio-programa <Enunciados> fin-programa
    <Enunciados> ::= <Enunciado> <Enunciados> | Vacio
    <Enunciado> ::= <Asignacion> | <Leer> | <Escribir> | <Si> | <Mientras>
    <Asignacion> ::= <Variable> = <Expresion>
    <Expresion > ::= <Valor> | <Operacion>
    <Operacion> ::= <Valor> <Operador-aritmetico> <Valor>
    <Valor> ::= <Variable> | <Numero>
    <Operador-aritmetico> ::= + | - | * | /
    <Leer> ::= leer <Cadena> , <Variable>
    <Escribir> ::= escribir <Cadena> | escribir <Cadena> , <Variable>
    <Si> ::= si <Comparacion> entonces <Enunciados> fin-si
    <Mientras> ::= mientras <Comparacion> <Enunciados> fin-mientras
    <Comparacion> ::= ( <Valor> <Operador-relacional> <Valor> )
    <Operador-relacional> ::= < | > | == | <= | >= | !=
 *
 */
```

```java
public class PseudoParser {
    ArrayList<PseudoLexer.Token> tokens;
    int tokenIndex = 0;

    public boolean parse(ArrayList<PseudoLexer.Token> tokens) {
        this.tokens = tokens;

        System.out.println("\n\n**********  Reglas de producción  **********\n");
        return programa();
    }

    private boolean token(String name) {
        if (tokens.get(tokenIndex).type.name().equals(name)) {
            System.out.println(tokens.get(tokenIndex).type.name() + " " + tokens.get(tokenIndex).data);
            tokenIndex++;
            return true;
        }

        return false;
    }

    // <Programa> ::= inicio-programa <Enunciados> fin-programa
    private boolean programa() {
        System.out.println("<Programa> --> inicio-programa <Enunciados> fin-programa");

        if (token("INICIOPROGRAMA"))
            if (enunciados())
                if (token("FINPROGRAMA"))
                    if (tokenIndex == tokens.size())
                        return true;

        return false;
    }
```

```java
// <Enunciados> ::= <Enunciado> <Enunciados> | Vacio
private boolean enunciados() {
    System.out.println("<Enunciados> --> <Enunciado> <Enunciados> | Vacio");

    if (token("FINPROGRAMA")) {
        tokenIndex--;
        return true;
    }

    if (token("FINSI")) {
        tokenIndex--;
        return true;
    }

    if (token("FINMIENTRAS")) {
        tokenIndex--;
        return true;
    }

    if (enunciado())
        if (enunciados())
            return true;

    return false;
}
```

```java
// <Enunciado> ::= <Asignacion> | <Leer> | <Escribir> | <Si> | <Mientras>
private boolean enunciado() {
    System.out.println("<Enunciado> --> <Asignacion> | <Leer> | <Escribir> | <Si> | <Mientras>");

    if (enunciadoAsignacion()) {
        System.out.println("<Enunciado> --> <Asignacion>");
        return true;
    }

    if (enunciadoLeer()) {
        System.out.println("<Enunciado> --> <Leer>");
        return true;
    }

    if (enunciadoEscribir()) {
        System.out.println("<Enunciado> --> <Escribir>");
        return true;
    }

    if (enunciadoSi()) {
        System.out.println("<Enunciado> --> <Si>");
        return true;
    }

    if (enunciadoMientras()) {
        System.out.println("<Enunciado> --> <Mientras>");
        return true;
    }

    return false;
}
```

```java
// <Asignacion> ::= <Variable> = <Expresion>
private boolean enunciadoAsignacion() {
    System.out.println("<Asignacion> --> <Variable> = <Expresion>");

    if (token("VARIABLE"))
        if (token("IGUAL"))
            if (expresion())
                return true;

    return false;
}

// <Expresion > ::= <Valor> | <Operacion>
private boolean expresion() {
    System.out.println("<Expresion > --> <Operacion> | <Valor>");

    if (operacion())
        return true;

    if (valor())
        return true;

    return false;
}
```

```java
// <Valor> ::= <Variable> | <Numero>
private boolean valor() {
    System.out.println("<Valor> --> <Variable> | <Numero>");

    if (token("VARIABLE"))
        return true;

    if (token("NUMERO"))
        return true;

    return false;
}

// <Operacion> ::= <Valor> <Operador-aritmetico> <Valor>
private boolean operacion() {
    System.out.println("<Operacion> --> <Valor> <Operador-aritmetico> <Valor>");

    int tokenIndexAux = tokenIndex;

    if (valor())
        if (token("OPARITMETICO"))
            if(valor())
                return true;

    tokenIndex = tokenIndexAux;
    return false;
}
```

```java
// <Leer> ::= leer <Cadena> , <Variable>
private boolean enunciadoLeer() {
    System.out.println("<Leer> --> leer <Cadena> , <Variable>");

    if (token("LEER"))
        if (token("CADENA"))
            if (token("COMA"))
                if (token("VARIABLE"))
                    return true;

    return false;
}

// <Escribir> ::= escribir <Cadena> | escribir <Cadena> , <Variable>
private boolean enunciadoEscribir() {
    System.out.println("<Escribir> --> escribir <Cadena> | escribir <Cadena> , <Variable>");

    int tokenIndexAux = tokenIndex;

    if (token("ESCRIBIR"))
        if (token("CADENA"))
            if (token("COMA"))
                if (token("VARIABLE"))
                    return true;

    tokenIndex = tokenIndexAux;

     if (token("ESCRIBIR"))
        if (token("CADENA"))
            return true;

    return false;
}
```

```
//  <Si> ::= si <Comparacion> entonces <Enunciados> fin-si
private boolean enunciadoSi() {
    System.out.println("<Si> -- > si <Comparacion> entonces <Enunciados> fin-si");

    if (token("SI"))
        if (comparacion())
            if (token("ENTONCES"))
                if (enunciados())
                    if (token("FINSI"))
                        return true;

    return false;
}

// <Comparacion> ::= ( <Valor> <Operador-relacional> <Valor> )
private boolean comparacion() {
    System.out.println("<Comparacion> --> ( <Valor> <Operador-relacional> <Valor> )");

    if (token("PARENTESISIZQ"))
        if (valor())
            if (token("OPRELACIONAL"))
                if (valor())
                    if (token("PARENTESISDER"))
                        return true;

    return false;
}

// <Mientras> ::= mientras <Comparacion> <Enunciados> fin-mientras
private boolean enunciadoMientras() {
    System.out.println("<Mientras> --> mientras <Comparacion> <Enunciados> fin-mientras");

    if (token("MIENTRAS"))
        if (comparacion())
            if (enunciados())
                if (token("FINMIENTRAS"))
                    return true;

    return false;
}
```

```
    //  <Si> ::= si <Comparacion> entonces <Enunciados> fin-si
    private boolean enunciadoSi() {
        System.out.println("<Si> -- > si <Comparacion> entonces <Enunciados> fin-si");

        if (token("SI"))
            if (comparacion())
                if (token("ENTONCES"))
                    if (enunciados())
                        if (token("FINSI"))
                            return true;

        return false;
    }

    // <Comparacion> ::= ( <Valor> <Operador-relacional> <Valor> )
    private boolean comparacion() {
        System.out.println("<Comparacion> --> ( <Valor> <Operador-relacional> <Valor> )");

        if (token("PARENTESISIZQ"))
            if (valor())
                if (token("OPRELACIONAL"))
                    if (valor())
                        if (token("PARENTESISDER"))
                            return true;

        return false;
    }

    // <Mientras> ::= mientras <Comparacion> <Enunciados> fin-mientras
    private boolean enunciadoMientras() {
        System.out.println("<Mientras> --> mientras <Comparacion> <Enunciados> fin-mientras");

        if (token("MIENTRAS"))
            if (comparacion())
                if (enunciados())
                    if (token("FINMIENTRAS"))
                        return true;

        return false;
    }
}
```

```
    PseudoLexer pLexer;
    PseudoParser pParser;

    public PseudoCompiler() {
        pLexer = new PseudoLexer();

        String input = "";

        try {
            FileReader reader = new FileReader("ejemplo2.alg");
            int character;

            while ((character = reader.read()) != -1) {
                input += (char) character;
            }
            reader.close();

        } catch (IOException e) {
            e.printStackTrace();
        }

        System.out.println("**********  Programa fuente  **********\n");
        System.out.println(input);

        ArrayList<PseudoLexer.Token> tokens = pLexer.lex(input);

        System.out.println("\n\n**********  Tokens  **********\n");

        for (PseudoLexer.Token token : tokens)
            System.out.println(token);

        pParser = new PseudoParser();
        System.out.println("\nSintaxis correcta: " + pParser.parse(tokens));
    }

    public static void main(String[] args) {
        new PseudoCompiler();
    }
}
```

```
inicio-programa
    leer "Cuantas calificaciones", n
    prom = 0

    i = 0
    mientras (i < n)
        leer "Da una calificacion", cal
        prom = prom + cal
        i = i + 1
    fin-mientras

    prom = prom / n

    si (prom > 5) entonces
        escribir "Aprobado con: ", prom
    fin-si

    si (prom == 10) entonces
        escribir "Excelente"
    fin-si
fin-programa
```

Ejercicios

- Modificar el analizador sintáctico para detectar la correctitud del programa de acuerdo a las modificaciones a la sintaxis del pseudocódigo.

  variables enteras : i, j, n

  variables flotantes : x, y, z

  repite (i = 1, n)

  …

  fin-repite

```
(INICIOPROGRAMA  "inicio-programa")
(LEER  "leer")
(CADENA  ""Cuantas calificaciones"")
(COMA  ",")
(VARIABLE  "n")
(VARIABLE  "prom")
(IGUAL  "=")
(NUMERO  "0")
(VARIABLE  "i")
(IGUAL  "=")
(NUMERO  "0")
(MIENTRAS  "mientras")
(PARENTESISIZQ  "(")
(VARIABLE  "i")
(OPRELACIONAL  "<")
(VARIABLE  "n")
(PARENTESISDER  ")")
(LEER  "leer")
(CADENA  ""Da una calificacion"")
(COMA  ",")
(VARIABLE  "cal")
(VARIABLE  "prom")
(IGUAL  "=")
(VARIABLE  "prom")
(OPARITMETICO  "+")
(VARIABLE  "cal")
(FINMIENTRAS  "fin-mientras")
(VARIABLE  "prom")
(IGUAL  "=")
(VARIABLE  "prom")
(OPARITMETICO  "/")
(VARIABLE  "n")
(SI  "si")
(PARENTESISIZQ  "(")
(VARIABLE  "prom")
(OPRELACIONAL  ">")
(NUMERO  "5")
(PARENTESISDER  ")")
(ENTONCES  "entonces")
(ESCRIBIR  "escribir")
(CADENA  ""Aprobado con: "")
(COMA  ",")
(VARIABLE  "prom")
(FINSI  "fin-si")
(SI  "si")
(PARENTESISIZQ  "(")
(VARIABLE  "prom")
(OPRELACIONAL  "==")
(NUMERO  "10")
(PARENTESISDER  ")")
(ENTONCES  "entonces")
(ESCRIBIR  "escribir")
(CADENA  ""Excelente"")
(FINSI  "fin-si")
(FINPROGRAMA  "fin-programa")
```

```
<Programa> --> inicio-programa <Enunciados> fin-programa
INICIOPROGRAMA inicio-programa
<Enunciados> --> <Enunciado> <Enunciados> | Vacio
<Enunciado> --> <Asignacion> | <Leer> | <Escribir> | <Si> | <Mientras>
<Asignacion> --> <Variable> = <Expresion>
<Leer> --> leer <Cadena> , <Variable>
LEER leer
CADENA "Cuantas calificaciones"
COMA ,
VARIABLE n
<Enunciado> --> <Leer>
<Enunciados> --> <Enunciado> <Enunciados> | Vacio
<Enunciado> --> <Asignacion> | <Leer> | <Escribir> | <Si> | <Mientras>
<Asignacion> --> <Variable> = <Expresion>
VARIABLE prom
IGUAL =
<Expresion > --> <Operacion> | <Valor>
<Operacion> --> <Valor> <Operador-aritmetico> <Valor>
<Valor> --> <Variable> | <Numero>
NUMERO 0
<Valor> --> <Variable> | <Numero>
NUMERO 0
<Enunciado> --> <Asignacion>
<Enunciados> --> <Enunciado> <Enunciados> | Vacio
<Enunciado> --> <Asignacion> | <Leer> | <Escribir> | <Si> | <Mientras>
<Asignacion> --> <Variable> = <Expresion>
VARIABLE i
IGUAL =
<Expresion > --> <Operacion> | <Valor>
<Operacion> --> <Valor> <Operador-aritmetico> <Valor>
<Valor> --> <Variable> | <Numero>
NUMERO 0
<Valor> --> <Variable> | <Numero>
NUMERO 0
<Enunciado> --> <Asignacion>
<Enunciados> --> <Enunciado> <Enunciados> | Vacio
<Enunciado> --> <Asignacion> | <Leer> | <Escribir> | <Si> | <Mientras>
<Asignacion> --> <Variable> = <Expresion>
<Leer> --> leer <Cadena> , <Variable>
<Escribir> --> escribir <Cadena> | escribir <Cadena> , <Variable>
<Si> -- > si <Comparacion> entonces <Enunciados> fin-si
<Mientras> --> mientras <Comparacion> <Enunciados> fin-mientras
MIENTRAS mientras
<Comparacion> --> ( <Valor> <Operador-relacional> <Valor> )
PARENTESISIZQ (
<Valor> --> <Variable> | <Numero>
VARIABLE i
OPRELACIONAL <
<Valor> --> <Variable> | <Numero>
VARIABLE n
PARENTESISDER )
<Enunciados> --> <Enunciado> <Enunciados> | Vacio
<Enunciado> --> <Asignacion> | <Leer> | <Escribir> | <Si> | <Mientras>
<Asignacion> --> <Variable> = <Expresion>
<Leer> --> leer <Cadena> , <Variable>
LEER leer
CADENA "Da una calificacion"
COMA ,
VARIABLE cal
```

```
<Enunciado> --> <Leer>
<Enunciados> --> <Enunciado> <Enunciados> | Vacio
<Enunciado> --> <Asignacion> | <Leer> | <Escribir> | <Si> | <Mientras>
<Asignacion> --> <Variable> = <Expresion>
VARIABLE prom
IGUAL =
<Expresion > --> <Operacion> | <Valor>
<Operacion> --> <Valor> <Operador-aritmetico> <Valor>
<Valor> --> <Variable> | <Numero>
VARIABLE prom
OPARITMETICO +
<Valor> --> <Variable> | <Numero>
VARIABLE cal
<Enunciado> --> <Asignacion>
<Enunciados> --> <Enunciado> <Enunciados> | Vacio
FINMIENTRAS fin-mientras
FINMIENTRAS fin-mientras
<Enunciado> --> <Mientras>
<Enunciados> --> <Enunciado> <Enunciados> | Vacio
<Enunciado> --> <Asignacion> | <Leer> | <Escribir> | <Si> | <Mientras>
<Asignacion> --> <Variable> = <Expresion>
VARIABLE prom
IGUAL =
<Expresion > --> <Operacion> | <Valor>
<Operacion> --> <Valor> <Operador-aritmetico> <Valor>
<Valor> --> <Variable> | <Numero>
VARIABLE prom
OPARITMETICO /
<Valor> --> <Variable> | <Numero>
VARIABLE n
<Enunciado> --> <Asignacion>
<Enunciados> --> <Enunciado> <Enunciados> | Vacio
<Enunciado> --> <Asignacion> | <Leer> | <Escribir> | <Si> | <Mientras>
<Asignacion> --> <Variable> = <Expresion>
<Leer> --> leer <Cadena> , <Variable>
<Escribir> --> escribir <Cadena> | escribir <Cadena> , <Variable>
<Si> -- > si <Comparacion> entonces <Enunciados> fin-si
SI si
<Comparacion> --> ( <Valor> <Operador-relacional> <Valor> )
PARENTESISIZQ (
<Valor> --> <Variable> | <Numero>
VARIABLE prom
OPRELACIONAL >
<Valor> --> <Variable> | <Numero>
NUMERO 5
PARENTESISDER )
ENTONCES entonces
<Enunciados> --> <Enunciado> <Enunciados> | Vacio
<Enunciado> --> <Asignacion> | <Leer> | <Escribir> | <Si> | <Mientras>
<Asignacion> --> <Variable> = <Expresion>
<Leer> --> leer <Cadena> , <Variable>
<Escribir> --> escribir <Cadena> | escribir <Cadena> , <Variable>
```