Universidad de San Carlos de Guatemala
Facultad de Ingenieria
Lenguajes Y compiladores 1
Primer Semestre 2020

Manual Técnico Practica 2

Nombre: Juan José Ramos
Carnet: 201801262
Fecha: 20/4/2020

CONTENIDO

Analisis Lexico

Para el desarrollo de la aplicación se conto con un analizador lexico que trabaja de la siguiente forma

Basicamente se compone de una función que recibe el texto de entrada y hace un for para separar carácter por carácter.

```
analizerThisText(textInput: string) {

    var state: number = 0;
    var column: number = 0;
    var row: number = 1;;
    textInput = textInput + "\n"; //se agrega el hash del final
    for (let i = 0; i < textInput.length; i++) {
        var letra = textInput[i];
```

Una vez obtenida la letra, entra a un switch y verifica que lo que viene.

```
if (this.IsLetter(letra) == true) {···
}
//VERIFICA SI VIENE NUMERO
else if (this.IsDigit(letra)) {
    state = 2;
    this.auxiliar += letra;
}
//VERIFICA SI ES PUNTUACION
else if (this.IsPunctuation(letra)) {···
}
else if( this.IsSymbol(letra) ) {···
}
//SI VIENE SALTO DE LINEA
else if (letra == '\n') {···
}
//VERIFICA ESPACIOS EN BLANCO
else if (letra == ' ' || letra.charCodeAt(0) == 13) {···
}
else {
    this.controller.InsertError(row, column, letra.toString(), "TK_Desconocido");
    state = 0;
}
```

Básicamente cuenta con 6 opciones, si el carácter es letra, si es digito, si es símbolo, si es signo de puntuación, si es salto de línea o si es espacio en blanco. Si no es ninguno, es error lexico y se envia como error.

Depenediendo de lo que venga, se dirige a cada uno de los casos del switch, si es letra o digito concatena hasta encontrar un espacio en blanco u otro símbolo respectivamente.

```
const reservada = ['int', 'string' ,'double', 'char', 'bool', 'public', 'class',
'static', 'void', 'main', 'return', 'true', 'false', 'for', 'if', 'while', 'else', 'const',
'switch', 'case', 'break', 'null', 'default', 'do', 'console', 'write', 'continue', 'abstract',
'writeline', 'continue', 'decimal', 'try', 'catch', 'float'];

if (reservada.includes(this.auxiliar.toLowerCase())) {
    this.controller.InsertToken(row, (column - this.auxiliar.length - 1), this.auxiliar.toLowerCase(), "PR_" + this.au
}
else {
    this.controller.InsertToken(row, (column - this.auxiliar.length - 1), this.auxiliar, "Identificador");
}
```

Para las letras, cuando se encuentra un delimitador, se verifica si la cadena formada es palabra reservada o no, si lo es se guarda como reservada, si no es un identificador común.

Se cuentan con unas funciones especiales que determinan el tipo del símbolo a evaluar, dichas funciones son las siguiente.

```
    IsLetter( str: string ): boolean {
        var char=str.toLowerCase();
        return (char>='a' && char<='z');
    }
    IsDigit( str: any ): boolean {
        var charCodeZero = "0".charCodeAt(0);
        var charCodeNine = "9".charCodeAt(0);
        return (str.charCodeAt(0) >= charCodeZero && str.charCodeAt(0) <= charCodeNine);
    }

    IsPunctuation( str: string) : boolean {
        if ( this.PUNTUACION.includes(str) ) {
            return true;
        }
        return false;
    }

    IsSymbol( str: string) : boolean {
        if ( this.SYMBOL.includes(str) ) {
            return true;
        }
        return false;
    }
```

Analisis Sintactico

Para el análisis sintactico se procedio a la creación de una gramática LL1, dicha gramática cuenta con la siguiente estructura.

<Inicio> ::= **<Clase>**
<Clase> ::= public class identificador llave_izquierda **<ListaDeclaracionGlobal>** llave_derecha

<ListaDeclaracionGlobal> ::= <DeclaracionVariableGlobales>
| **<MetodoPrincipal>**
| **<DeclaracionMetodo>**
| **E**

<DeclaracionGlobal> ::= **<Declaracion> <OtraDeclaracionGlobal>**
<Declaracion> ::= **<MetodoPrincipal>**
| <TipoDatoGlobal>()
| <TipoDatoGlobal>,
|<TipoDatoGlobal>=12,
<TipoDatoGlobal>::= int identificador

<OtroMetodo> ::= **<Declaracion><OtraDeclaracionGlobal>**

<DeclaracionVariableGlobal> ::= <TipoDato><Funcion><ListaParametro>

<MetodoPrincipal> ::= void Main paréntesis_izquierda **<ParametroMain>** paréntesis_derecho llave_izquierda **<ListaDeclaracion>** llave_derecha
<TipoDato> ::= int
| doublé
| char
| bool
| string

<Inicio> ::= **<Clase>**
<Clase> ::= public class identificador llave_izquierda **<MetodoPrincipal>** **<DeclaracionMetodo>** llave_derecha

<MetodoPrincipal> ::= static void Main paréntesis_izquierda **<ParametroMain>** paréntesis_derecho llave_izquierda **<ListaDeclaracion>** llave_derecha
<ParametroMain> ::= string corchete_izquierdo corchete_derecho identificador
| E
<DeclaracionMetodo> ::= **<Metodo> <OtroMetodo>**
<Metodo> ::= void Identificador paréntesis_izquierda **<ParametrosMetodos>** paréntesis_derecho llave_izquierda **<ListaDeclaracion>** llave_derecha
| **<TipoDatoFuncion>** Identificador paréntesis_izquierda <ParametrosMetodos> paréntesis_derecho llave_izquierda **<ListaDeclaracion>** **<Retorno>** llave_derecha
<TipoDatoFuncion> ::= int
| doublé
| char
| bool
| string
<OtroMetodo> ::= **<Metodo><OtroMetodo>**
|E

<DeclaracionParametros> ::= **<TipoVariable> <ListaParametro>**
|E
<TipoVariable> ::= int
| doublé
| char
| bool
| string
<ListaParametro>::= identificador**<MasParametros>**
<MasParametros> ::= coma **< ListaParametro >**
|E

<Retorno> ::= return identificador;
<DeclaracionComentario> ::= **<Comentario><OtroComentario>**
<Comentario> ::= ComentarioLinea
| ComentarioMultilinea
|E
<OtroComentario> ::= **<Comentario><OtroComentario>**
|E

<ListaDeclaracion> ::= **<DeclaracionAsignacion>**
| <DeclaracionIf>
|
<DeclaracionAsignacion> ::= <Asignacion><OtraAsignacion>
<Asignacion> ::= **<TipoVariable> <ListaAsignacion><AsignacionVariable>** punto_coma
<TipoVariable> ::= int
| doublé
| char
| bool

| string
<ListaAsignacion>::= identificador**<MasElementos>**
<MasElementos> ::= coma **<ListaAsignacion>**
|E
<AsignacionVariable> ::= igual **<ValorVariable>**
|E
<ValorVariable> ::= digito
| cadena
| caracter
| identificador
| true
| false
<OtraAsignacion> ::= **<Asignacion><OtraAsignacion>**
|E
<DeclaracionIf> ::= if paréntesis_izquierdo **<CondicionIf>** paréntesis_derecho llave_izquierda **<Else> <ListaDeclaracion>**
<Else> ::= else **<TipoElse>**
|E
<TipoElse> ::= **<DeclaracionElseIf>**
| llave_izquierda <ListaDeclaracion> llave_derecha
<DeclaracionElseIf> ::= **<ElseIf><OtroElseIf>**
<OtroElseIf> ::= **<ElseIf><OtroElseIf>**
|E
<ElseIf> ::= if paréntesis_izquierdo <Condicion> paréntesis_derecho llave_izquierda **<ListaDeclaracion>** llave_derecha
| else llave_izquierda **<ListaDeclaracion>** llave_derecha
|E
<OtroElseIf> ::= **<ElseIf><OtroElseIf>**
|E

<Condicion> ::= **<TipoVariable><OperacionRelacional><TipoVariable>**
<TipoVariable> ::= identificador
| numero
| carácter
| cadena

<DeclaracionFor> ::= if paréntesis_izquierdo **<InicializacionFor>** punto_coma **<CondicionFor>** punto_coma **<IncrementoFor>** paréntesis_derecho llave_izquierda **<ListaDeclaracion>** llave_derecha **<ListaDeclaracion>**

<DeclaracionWhile> ::= while llave_izquierda **<ListaDeclaracion>** llave_derecha **<ListaDeclaracion>**

<DeclaracionSwitch> ::= if paréntesis_izquierdo identificador paréntesis_derecho llave_izquierda **<CuerpoSwitch>** llave_derecha **<ListaDeclaracion>**
<CuerpoSwitch> ::= **<Case><OtroCase><Default>**

<Case> ::= case **<TipoSwitch>** dos_puntos **<ListaDeclaracion>** **<Break>**
<TipoSwitch> ::= identificador
| numero
| carácter
| cadena
<OtroCase> :: **<Case><OtroCase>**
|E
<Break> ::= break punto_coma
|E
<Default> ::= default: **<ListaDeclaracion>** break punto_coma
|E
<OperacionRelacional> ::= menor
| menor igual
| mayor
| mayor igual
| igual igual
| exclamación igual
<DeclaracionDoWhile> ::= do llave_izquierda **<ListaDeclaracion>** llave_derecha while paréntesis_izquierdo **<Condicion>** paréntesis_derecho punto_coma

<DeclaracionIf> ::= if paréntesis_izquierdo **<Condicion>** paréntesis_derecho llave_izquierda **<ListaDeclaracion>** llave_derecha
<DeclaracionElseIf><DeclaracionElse>
<DeclaracionElseIf> ::= **<ElseIf><OtroElseIf>**
<ElseIf> ::= else if paréntesis_izquierdo <Condicion> paréntesis_derecho llave_izquierda **<ListaDeclaracion>** llave_derecha
| else llave_izquierda **<ListaDeclaracion>** llave_derecha
|E
<OtroElseIf> ::= **<ElseIf><OtroElseIf>**
|E
< **DeclaracionElse**> :: else llave_izquierda <ListaDeclaracion> llave_derecha
|E

Al llevarla a código, se crearon diferentes funciones para validar la gramática. Pero la función mas importante con la que se cuenta es la siguiente.

```
public Parea(token:string)
{
    let validar:Boolean;
    if(this.currentToken!=null){
        if (this.currentToken.getDescription()!=token)
        {
            //ERROR SI NO VIENE LO QUE DEBERIA
            this.strError = this.strError + "\n" + "*Error Sintactico: Se esperaba '"+
            token.replace("TK_", "") + "' en lugar de '" + this.currentToken.getLexema()
            + "', Linea: " +this.currentToken.getRow() + ", Columna: " + this.currentToken.getColum();
            //RECUPERACION MODO PANICO
            for (let index = this.index; index < this.arrayListToken.length; index++) {
                this.currentToken = this.arrayListToken[this.index];
                if(this.currentToken.getDescription() == "TK_PuntoComa"
                || this.currentToken.getDescription() == "TK_LlaveDerecha"
                || this.currentToken.getDescription() == "TK_LlaveIzquierda"
                || this.currentToken.getDescription() == "PR_void"
                || this.currentToken.getDescription() == "PR_int"
                || this.currentToken.getDescription() == "PR_string"
                || this.currentToken.getDescription() == "PR_double"
                || this.currentToken.getDescription() == "PR_char"
                || this.currentToken.getDescription() == "PR_bool") {
                    this.currentToken = this.arrayListToken[this.index];
                    validar = true;
                    break;
                }
                this.index += 1;
            }
        }
        //FLUJO CORRECTO
        if (this.currentToken.getDescription()==token)
        {
            console.log(this.currentToken.toString())
            this.index += 1;
            this.currentToken = this.arrayListToken[this.index];
        }
    }
```

La función de Parea, se utiliza para analizar cada token de entrada, y verificar si va en el flujo correcto de cada producción de la gramática.

Cuando entra un token, verifica si el token actual de la gramática es igual al token de entrada, si lo es, continua con el flujo correcto de la producción, y avanza en esta. Si el token de entrada no coincide con el token esperado por la producción entra en modo pánico y empieza la recuperación de este, marcando el error sintactico correspondiente.