ITCR – IC5701.COMPILADORES E INTÉRPRETES

TAREA #2 – CORRECCIÓN DE ERRORES: FASE LÉXICA - MODO PÁNICO

Estudiantes:
Antonio Fernández García
Marco Rodríguez Vargas
Josué Mena González

Profesor:
Emmanuel Ramírez S.

II Semestre 2024

FASE LÉXICA

- **Descripción**

El modo de pánico se activa en la fase léxica. Esto sucede cuando en la entrada se detecta un error en alguno de los tokens que se está generando. Esto puede ser algún carácter inválido, tamaño de token inválido, formato de entero incorrecto, entre otros. Al topar alguno de estos casos el Lexer entraría en modo pánico y procede a realizar sus procedimientos con tal de quitarse el error de encima.

MODO PÁNICO

- **Corrección de errores**

Cuando se encuentra alguno de los casos mencionados (carácter inválido, tamaño de token inválido, formato de entero incorrecto, entre otros), se entra en modo pánico. En este modo el lexer va desechando o ignorando varios caracteres de entrada hasta encontrar un punto de token válido de sincronización. Por lo general estos son puntos y comas o palabras reservadas como "end". La idea de esto es llegar a un punto donde el programa este bien léxicamente y ahí reanudar el análisis. Lo ideal es anotar el error y reportarlo al programador después de haber sido identificado para posteriormente poder recuperarse.

[ALGORITMO]

- **Algoritmo**

  1. Detección de Error Léxico: Al leer la entrada si se encuentra una cadena que no es reconocida con ninguna de las reglas definidas para los tokens se detecta un error léxico.

  2. Activación del Modo Pánico: Se reporta el error indicando la posición en el código y se ignoran los caracteres hasta encontrar un punto de sincronización.

  3. Puntos de Sincronización: Los puntos de sincronización pueden ser caracteres como saltos de línea, punto y coma u otros símbolos propios del lenguaje.

  4. Reanudación del Análisis: Al llegar a un punto de sincronización, se reanuda con el análisis léxico desde ese punto.

MODO PÁNICO

- **Algoritmo**

```
def analizarEntrada():
        while not finDeEntrada():
    token = obtenerSiguienteToken()
// aca empieza el modo
  if token == no se reconoce:
        manejarError()
// sino se continua como normalmente se hace
    else:
        procesarToken(token)
```

```
def obtenerSiguienteToken():
 IgnorarEspacios()
   if finDeEntrada():
       return Token
   if hacerrTokenValido():
       return tokenArreglado
   else:
// no es posible continuar
      return none
```

```
def manejarError():
// esto es para que salga la linea respectiva del codigo
  reportarError(posActual)
  activarModoPanico()

def activarModoPanico():
   while caracterActual no es puntoDeSincronizacion y no se ha llegado al finDeEntrada():
      avanzarAlSiguienteCaracter()
```

[GRAMATICA]

- **Gramática utilizada**

Gramática en formato EBNF, L1:

programa -> expresion ; { programa } ;
expresion -> identificador = expresion | termino { (+ | -) termino } ;
termino -> factor { (* | /) factor } ;
factor -> identificador | numero | (expresion) ;
numero -> digito { digito } ;
digito -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

[CÓDIGO]

- **Código**

```
1 H0LA = 45-34;
2 CALCULO = [(5-6)*4]/5:
3 EL3VADO = 2^2;
4 B = 14;
5 A = 13;
6 FORMULA = 2B-2A;
```

Errores dentro del código

1) H0LA
2) [
3)]
4) :
5) EL3VADO
6) ^
7) 2B
8) 2A

[DEMOSTRACIÓN]

- **Uso del Visual Studio Code**

```
// Caso donde identifica algún simbolo que no es parte de la gramática

case -1:

// Modo pánico, señala el error pero continúa con el análisis

System.out.println("Error [Fase Lexica]: La linea " + numero_linea + " error, lexema no reconocido: " + codigoFuente.substring(posicionAnterior, posicionActual + 1));
posicionActual++;
posicionAnterior = posicionActual;
estado = 0;
break;
```

[DEMOSTRACIÓN]

```
// Caso donde un identificador tiene números o un número tiene letra
// Cuando esto sucede avanza hasta encontrar un simbolo conocido o un espacio o nueva línea

case -2:

// Modo pánico: avanzar hasta encontrar un símbolo que permita continuar

while (posicionActual < longitud) {

    caracter = codigoFuente.charAt(posicionActual);

    // Si encontramos un espacio en blanco, un salto de línea o un símbolo

    if (Character.isWhitespace(caracter)) {
        estado = 0;
        System.out.println("Error [Fase Lexica]: La línea " + numero_linea + " error, lexema no reconocido: " + codigoFuente.substring(posicionAnterior, posicionActual));
        break;

    // Si encontramos un símbolo permitido

    } else if (caracter == '=' || caracter == '+' || caracter == '-' ||
            caracter == '*' || caracter == '/' || caracter == '(' ||
            caracter == ')' || caracter == ';') {
        estado = 0;
        System.out.println("Error [Fase Lexica]: La línea " + numero_linea + " error, lexema no reconocido: " + codigoFuente.substring(posicionAnterior, posicionActual));
        break;
    }


    posicionActual++;
}
posicionAnterior = posicionActual;
break;
```

[DUDAS Y/O CONSULTAS]

- **Espacio para consultas**

[BIBLIOGRAFÍA]

- **Bibliografía**

Aho, A., Lam, M., Sethi, R., & Ullman, J. (2006). Compilers: Principles, techniques and tools (2nd. Edition ed.). Addison-Wesley.