Universidad de San Carlos de Guatemala
Centro Universitario de Occidente División de Ciencias de la Ingeniería
Laboratorio de Lenguajes Formales y de Programación
Julio Fernando Ixcoy
Sección: A

Manual Técnico

Carlos Raúl Alberto López Peláez 202031871

Quetzaltenango, 14 de Octubre del 2023

Analizador Sintáctico

```
public class Sintactico {
    privat  List<List<Object>> listaToken;
    e

    public Sintactico() {
        listaToken = new ArrayList<>();
    }

    public List<List<Object>> valorSintactic  (String input){
                              o
        listaToken.clear();

        LexicoPrueb  lexico  = new LexicoPrueb (input);
        TokenPrueba tokenn;        a
        System.out.printl (input);
                  n
        do {
            tokenn = lexico .nextToken();
            if (tokenn.type != TipoEspacio.SPACE && tokenn.type !=
TipoComentari .COMENT) {
o               List<Object> listaT = Arrays.asList(tokenn.type,tokenn.value
,tokenn.fila,tokenn.column );
                listaToken.add(listaT);
                System.out.printl ("tokens agregados inicio :   + tokenn.
type);                    n       "
                System.out.printl ("tokens agregados inicio :   + listaToken
);                        n       "

}
        } while (tokenn.type != TipoEspacio.EOF);

        Stack pila = new Stack<>();
        System.out.printl (listaToken.size());
        int valor =listaToken.size();
        OperacionesAritmetica  nuvOp = new OperacionesAritmetica ();
        Expresiones nuvEx = new Expresiones();
        Asignacion nuvAsi  = new Asignacion();
        AsignacionOperado  nuvAsigComp = new AsignacionOperado ();
        OperacionesComparacio  nuvOpLog = newrOperacionesComparacio ();
        AsignacionComparacio  nuvOpComp = new AsignacionComparacio ();
        OperacionesLogico  nuvOpLogcom  = new OperacionesLogico ();
        AsignacionLogico nuvAsigOpLogcomp = new AsignacionLogico();
        OperacionesIdentida  nuvIdent= new OperacionesIdentida ();
        AsignacionIdentida  nuvAsigIden = new AsignacionIdentida ();
        OperacionesPertenenci  nuvPer = new OperacionesPertenenci ();
        AsignacionPertenenci  nuvAsigPert= new AsignacionPertenenci ();
        OperacionesBit   nuvBit= new OperacionesBit ();
        AsignacionBit   nuvAsigBit= new AsignacionBit ();
        OperadoresTernario   nuvOpTernario = new OperadoresTernario ();
        OperadoresEntradaSalid   nuvEntSal= new OperadoresEntradaSalid ();
        Condicion  nuvCondicio = new Condicion();
                  n
        for (int index = valor-1; index >= 0; index--) {
            System.out.printl (index);
            pila.push(listaToken.get(index));
        }

        System.out.printl (pila);
        System.out.printl ("**** Operaciones Aritmeticas ****  );
        while(!pila.empty()){
            System.out.printl (pila);
            //listaToken = nuvEx.Entrada(pil
            listaToken = nuvOp.Entrad (pila);
        }                       a

        valor = listaToken.size();

        System.out.printl ("valor devuelto :   + listaToken);
                  n       "

        for (int index = valor-1; index >= 0; index--) {
            System.out.printl (index);
            pila.push(listaToken.get(index));

        }
```

Class Sintáctico

La clase sintáctico se encarga de recibir todos los tokens y pasarlos a pilas , donde después son usados para por las reglas de producción, cada regla de producción recibe una pila con los tomentos ordenados para que sean analizados.

- Clase Sintactico: Esta clase representa el analizador sintáctico y contiene un campo listatoken, que es una lista de listas de objetos.

- Método valorSintactico: Este método toma una cadena de entrada input, realiza el análisis léxico de la entrada y construye una lista de tokens. Luego, utiliza una serie de clases de reglas de producción para realizar el análisis sintáctico de la entrada. Estas clases incluyen OperacionesAritmeticas, Expresiones, Asignacion, OperacionesComparacion, OperacionesIdentidad, OperacionesLogicos, Expresiones, OperacionesPertenencia, OperacionesBits, AsignacionLogico, AsignacionOperador, AsignacionIdentidad, AsignacionPertenencia, AsignacionBits, OperadoresTernarios, OperadoresEntradaSalida, y Condicion.

- Proceso de análisis: El método procesa la lista de tokens utilizando estas clases de reglas de producción en un orden específico. Cada paso de análisis se realiza mediante un bucle while que opera sobre una pila de tokens. El resultado de cada paso de análisis se almacena en la variable listaToken. Este proceso se repite para cada clase de reglas de producción en un orden específico.

- Resultado: Al final, el método devuelve la lista de tokens que ha sido procesada por todas las clases de reglas de producción. Cada clase de reglas de producción realiza un análisis sintáctico específico en la lista de tokens.

Reglas de Producción

En general todas las reglas de producción cuentan con una lógica muy parecida

- **OperacionesAritmeticas**: Esta clase se encarga de analizar expresiones aritméticas, incluyendo sumas, restas, multiplicaciones, divisiones, exponentes y módulos. Los tokens se almacenan en una lista, y las expresiones completas se almacenan en listaTotal.

- **Expresiones**: Similar a OperacionesAritmeticas, esta clase se encarga de analizar expresiones más generales, incluyendo operaciones aritméticas y cadenas. También almacena los resultados en listaTotal.

- **OperacionesComparacion**: Esta clase se enfoca en el análisis de operadores de comparación, como igual a, diferente de, mayor que, menor que, etc. Las expresiones completas se almacenan en listaTotal.

- **OperacionesIdentidad**: Analiza operadores de identidad (por ejemplo, ==, !==). Las expresiones completas se almacenan en listaTotal.

- **OperacionesLogicos**: Esta clase se encarga de operadores lógicos, como & (AND) y |(OR). Los resultados se almacenan en listaTotal.

- **OperacionesPertenencia**: Analiza operadores de pertenencia. Almacena las expresiones completas en listaTotal.

- **OperacionesBits**: Se encarga de operaciones a nivel de bits, como &, |, ^, y desplazamientos. Los resultados se almacenan en listaTotal.

- **OperadoresTernarios**: Analiza operadores ternarios condicionales. Las expresiones completas se almacenan en listaTotal.

- **OperadoresEntradaSalida**: Analiza operadores de entrada/salida, como <<, >> . Almacena los resultados en listaTotal.

- **Condicion**: Esta clase se centra en el análisis de condiciones, como las que se utilizan en estructuras condicionales como if, else if, y switch. Las expresiones completas se almacenan en listaTotal.

- En general, todas estas clases de reglas de producción están diseñadas para analizar diferentes aspectos del código fuente, como expresiones aritméticas, operadores lógicos, asignaciones, y condiciones, y luego almacenar los resultados en una lista llamada listaTotal que representa la estructura sintáctica del código. Cada clase se encarga de un aspecto específico del análisis sintáctico y utiliza una estructura similar para procesar los tokens y construir expresiones completas.

Asignaciones

- **Clase Asignacion**:Esta clase se encarga de manejar asignaciones simples, donde se asigna un valor a una variable. Utiliza las clases TipoIdentificador y TipoAsignacion para determinar si se trata de una asignación válida. Almacena la asignación en la lista listaToken.

- **Clase AsignacionOperador**: Esta clase se encarga de asignaciones que involucran operadores, como +=, -=, *=, etc. Utiliza las clases TipoIdentificador, TipoAsignacion, y TipoOperador para reconocer este tipo de asignaciones. Contiene métodos como Entrada, Asignador, y Valor para procesar las diferentes partes de la asignación. Almacena la asignación en la lista listaToken.

- **Clase AsignacionComparacion**: Esta clase se encarga de asignaciones que involucran comparaciones y operadores relacionales, como ==, !=, <=, >=, etc. Utiliza las clases TipoIdentificador, TipoAsignacion, TipoOperador, y TipoComparacion para identificar este tipo de asignaciones. Contiene métodos como Entrada, Asignador, y Valor para procesar las diferentes partes de la asignación. Almacena la asignación en la lista listaToken.

- **Clase AsignacionLogico**: Esta clase se encarga de asignaciones que involucran operadores lógicos, como &, |. Utiliza las clases TipoIdentificador, TipoAsignacion, TipoOperador, y TipoLogico para reconocer este tipo de asignaciones.Contiene métodos como Entrada, Asignador, y Valor para procesar las diferentes partes de la asignación. Almacena la asignación en la lista listaToken.

- **Clase AsignacionIdentidad**: Esta clase se encarga de asignaciones que involucran operadores de identidad, como ==, !=. Utiliza las clases TipoIdentificador, TipoAsignacion, TipoOperador, y TipoIdentidad para identificar este tipo de asignaciones. Contiene métodos como Entrada, Asignador, y Valor para procesar las diferentes partes de la asignación. Almacena la asignación en la lista listaToken.

- **Clase AsignacionPertenencia**: Esta clase se ocupa de asignaciones que involucran operadores de pertenencia, como in. Utiliza las clases TipoIdentificador, TipoAsignacion, TipoOperador, y TipoPertenencia para reconocer este tipo de asignaciones.Contiene métodos como Entrada, Asignador, y Valor para procesar las diferentes partes de la asignación. Almacena la asignación en la lista listaToken.

- **Clase AsignacionBits:** Esta clase se encarga de asignaciones que involucran operadores a nivel de bits, como ^, <<, >>. Utiliza las clases TipoIdentificador, TipoAsignacion, TipoOperador, y TipoBits para identificar este tipo de asignaciones.Contiene métodos como Entrada, Asignador, y Valor para procesar las diferentes partes de la asignación. Almacena la asignación en la lista listaToken.