UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

Asignatura: Compiladores (0434)

Proyecto: Analizador Sintáctico Ascendente

Profesor: Ing. Manuel Castañeda Castañeda

Integrantes del equipo:

- *Bautista Pérez Brian Jassiel*
- *González Ramírez David*
- *Guillén Castro Jorge Luis*
- *Meneses Gómez Iassiel Navih*

Grupo: 05
Semestre: 2022-2
Fecha de entrega: 25 de mayo de 2022

Índice

1. Introducción

El compilador es una herramienta que permite el traducir un lenguaje de programación a otro para su futura interpretación, el cuál maneja un análisis del código origen, para efectos de este proyecto se hizo enfásis en el análizador sintáctico.
El analizador sintáctico, también conocido como parser, representa la segunda fase de compilación, se encarga de realizar un análisis mediante las gramáticas para verificar si la sintaxis de la instrucción es adecuada para el programa, una vez realizado esto se genera un árbol sintáctico, el cual puede ser ascendente o descendente.
Para este proyecto, se implementó un árbol sintáctico ascendente, la construcción se realiza desde las hojas más externas del árbol y van poco a poco subiendo a sus raíces, esto se puede entender como una entrada, la cual va generando pequeñas derivaciones hasta llegar al axioma inicial.
Para la correcta realización del proyecto se va a implementar un analizador sintáctico ascendente de acuerdo a una gramática dada, donde el analizador léxico tendra que reconocer los números reales, identificadores, operadores y paréntesis, por lo cual se podrá indicar si la cadena es válida o no.

2. Desarrollo

Se debe de elaborar un analizador sintáctico ascendente que revise las sentencias de acuerdo a la siguiente gramática:
$E \rightarrow T$
$E \rightarrow E + T$
$E \rightarrow E - T$
$T \rightarrow i$
$E \rightarrow (E)$
i - Será cualquier número real o identificador.
Se utilizó el lenguaje Java para la implementación del código de nuestro analizador sintáctico.
Analizador sintáctico ascendente
A lo largo del curso se vio que un analizador léxico es aquel que conforma la primera fase de la compilación y está destinado a leer caracteres del archivo de entrada, donde se encuentra la cadena a analizar, reconocer subcadenas que correspondan a símbolos del lenguaje y retomar los tokens correspondientes y sus atributos. Además procesa los lexemas que le va a suministrar el analizador sintáctico y verificará que estén bien ordenados, si no lo están se generarán errores.
A continuación, el código:

```
1
2
3
4
5  import java.io.*;
6
7
8  public class bottomUpParser{
9    static int contador=0;
10   static
11   int flag=0;
12
13   public static void main(String[] args) throws IOException, Exception{
14
15     analizador();
16   }
17
18   //Definicion de la gramatica
19   public static void regla1(String cadena){
20     if(cadena.length()==2){
21       System.out.println("T -> i");
22       String cambio = cadena.replaceFirst("i","T");
23       System.out.println("Cadena: "+cambio);
24       regla5(cambio);
```

```
    }
    else if(cadena.length()>=3){
      System.out.println("T-> i");
      String cambio = cadena.replace("i+i","i+T");
      System.out.println("Cadena: "+cambio);
      if(cambio.equals(cadena)){
        System.out.println("\n******La cadena si fue aceptada******\n");
      }
      else{
        switch(cambio){
            case "i-i+T;" -> regla7(cambio);
            case "i-i-T;" -> regla6(cambio);
          case "E+T;" -> regla4(cambio);
          case "E-T;" -> regla3(cambio);
          case "i-(i+T);" -> regla7(cambio);
            case "i-(i-T);" -> regla6(cambio);
          case "(E+T);" -> regla4(cambio);
          case "(E-T);" -> regla3(cambio);
          default -> regla5(cambio);
        }
      }
    }
  }

  //Regla 2
  public static void regla2(String cadena){
    System.out.println("T -> E");
    String cambio = cadena.replace("(E)", "T");
    System.out.println("Cadena: "+cambio);
    regla6(cambio);
  }

  //Regla 3
  public static void regla3(String cadena){
    System.out.println("E -> E-T");
    String cambio = cadena.replace("E-T","E");
    System.out.println("Cadena: "+cambio);
    if(cambio.length()!=2){
      regla2(cambio);
    }else if(cambio.length()==2){
      regla11(cambio);
    }
  }

  //Regla4
  public static void regla4(String cadena){
    System.out.println("E -> E+T");
    String cambio = cadena.replace("E+T", "E");
    System.out.println("cadena: "+cambio);
    if(cambio.length()!=2){
      regla2(cambio);
    }
  }

  //Regla5
  public static void regla5(String cadena){
    if(cadena.length() == 2){
      System.out.println("E-> T");
      String cambio = cadena.replaceFirst("T", "E");
      System.out.println("Cadena: "+cambio);
    }
    else if(cadena.length() >= 3){
      contador++;
      System.out.println("E -> T");
      String cambio = cadena.replaceFirst("T","E");
      System.out.println("Cadena: "+cambio);
```

```
91      if(cambio.contains("(") && flag != 0){
92        regla2(cambio);
93      }
94      else{
95        regla10(cambio);
96      }
97    }
98  }
99
100 //Regla6
101 public static void regla6(String cadena){
102   System.out.println("T-> i");
103   String cambio = cadena.replace("i-T","T-T");
104   System.out.println("cadena: "+cambio);
105   if(cambio.length()!=2){
106     regla8(cambio);
107   }
108 }
109
110
111 //Regla7
112 public static void regla7(String cadena){
113   System.out.println("T-> i");
114   String cambio = cadena.replace("i+T","T+T");
115   System.out.println("cadena: "+cambio);
116   if(cambio.length()!=2){
117     regla9(cambio);
118   }
119 }
120
121
122 //Regla8
123 public static void regla8(String cadena){
124   System.out.println("E-> T");
125   String cambio = cadena.replace("T-T","E-T");
126   System.out.println("cadena: "+cambio);
127   if(cambio.length()!=2){
128     regla3(cambio);
129   }
130 }
131
132 //Regla9
133 public static void regla9(String cadena){
134   System.out.println("E-> T");
135   String cambio = cadena.replace("T+T","E+T");
136   System.out.println("cadena: "+cambio);
137   if(cambio.length()!=2){
138     regla4(cambio);
139   }
140 }
141
142 //Regla10
143 public static void regla10(String cadena){
144   System.out.println("T-> i");
145   String cambio = cadena.replace("i","T");
146   System.out.println("cadena: "+cambio);
147   if(cambio.length()!=2){
148     regla8(cambio);
149   }
150 }
151
152 //Regla11
153 public static void regla11(String cadena){
154   System.out.println("\n******La cadena si fue aceptada******\n");
155 }
156
```

```
157
158
159   public static void analizador() throws NoSuchFieldException, IOException{
160     String cadena;
161     int linea = 0, parentesis;
162     FileReader prueba = new FileReader("./prueba.txt");
163     BufferedReader b = new BufferedReader(prueba);
164     while((cadena = b.readLine()) !=null){
165       linea++;
166       //esta linea de codigo delimita la expresion regular del ejemplos solicitado
167       if(cadena.matches("i[+|-]?[(]*i[+|-]i[)]+[;]")){ //i-(i+i)
168       //aqui mostrara la cadena solicitada
169         System.out.println("\nEjemplo: "+cadena);
170         if(cadena.contains("()")){ //buscara la presencia de parentesis
171           parentesis = CP(cadena);
172           if(parentesis != 0){
173               System.out.print("Error en la linea: "+linea+". La cadena es: "+cadena+"\n");
174           }
175           else{
176               flag = 0;
177               regla1(cadena);
178           }
179         }
180         else{
181           regla1(cadena);
182         }
183       }
184       else{
185         System.out.println("Error en la lina: "+linea+". La cadena es: "+cadena+"\n");
186       }
187     }
188     b.close();
189   }
190
191   //Estado para los parentesis. Verifica que los parentesis
192   //esten balanceados
193   public static int CP(String cadena){
194     int i = 0, f = 0;
195     while(cadena.length()-1 != f){
196       switch (cadena.charAt(f)){
197         case '(' -> {
198           i++;
199           f++;
200         }
201         case ')' -> {
202           i--;
203           f++;
204         }
205         default -> f++;
206       }
207     }
208     return i;
209   }
210 }
```

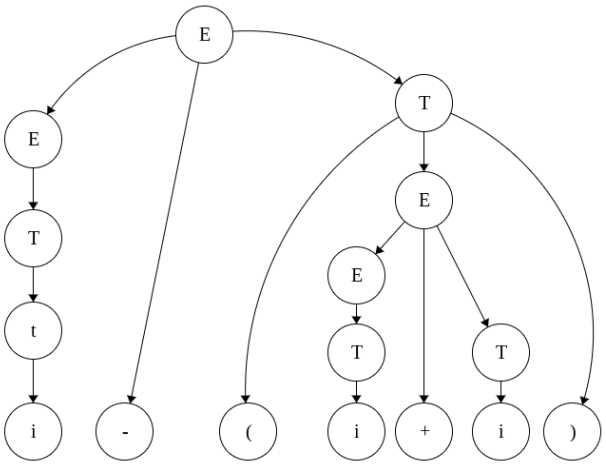
3. Análisis de resultados

A continuación el árbol sintáctico hecho de manera manual junto con su análisis:
Gramática.
1)$E \to T$
2)$E \to E + T$
3)$E \to E - T$

4)$T \to i$
5)$E \to (E)$

Cadena de Entrada. $i - (i + 3{,}4)$
4)$T \to i - (i + T)$
4)$T \to i - (T + T)$
1)$E \to i - (E + T)$
2)$E \to i - (E)$
5)$T \to i - i + T$
4)$T \to T - T$
1)$E \to E - T$
3)$E \to E$

Cadena transformada con la gramática para realizar el analizador sintáctico ascendente: $i - (i + i)$
3)$E \to E - T$
1)$E \to T - T$
4)$T \to i - T$
5)$T \to i - (E)$
2)$E \to i - (E + T)$
1)$E \to i - (T + T)$
4)$T \to i - (i + T)$
4)$T \to i - (i + i)$



Árbol Sintáctico Ascendente.

4. Conclusiones

En este proyecto se pusieron a prueba los conceptos vistos en clase y se pudo comprender de una manera práctica las fases que lleva a cabo el compilador para realizar la transición entre código fuente a código máquina, que es un proceso que a simple vista no se distingue pero que conociendo estos conceptos de analizador léxico, sintáctico y semántico podemos ver que es un proceso no tan sencillo.