

Seminario de Solución de Problemas de
Traductores de Lenguajes II
Centro Universitario de Ciencias Exactas e
Ingenierías



Actividad 2:
Analizador Sintáctico.

Alumno: Ortiz Macías Guillermo

Sección: D06

Profesor: Ramos Barajas, Armando.

Introducción

Esta tarea busca desarrollar un analizador sintáctico de un compilador, por medio de una tabla que representa una gramática. Dicha tabla se obtiene por medio de un archivo separado por comas.

Metodología

El analizador se realizó con ayuda del lenguaje de programación C++ y con el analizador léxico que se ya había creado en una práctica anterior. Al léxico le hice modificaciones para que regresara un número que identificara el token en lugar del nombre del mismo. Dentro del analizador sintáctico se llama al léxico para que éste regrese una lista con los números que representan los elementos del código.

Antes de empezar a parsear el código hice una función para obtener el número en una fila y columna del archivo donde está la tabla, y otra que regresa una tupla (regla, número) del archivo de reglas.

Para la función de parseo se tiene una pila donde se van a ir almacenando identificadores y una cola que almacena la entrada por el archivo que se obtuvo en el analizador léxico.

Teniendo estas dos funciones empecé la función para parsear el código. El cual es un ciclo while que termina cuando se encuentra un error o cuando se acepta el código. Lo primero que hago es determinar la fila y la columna que voy a evaluar, donde la fila es el último elemento de la pila de parseo y la columna es el primer elemento de la cola que retornó el analizador léxico. En seguida se obtiene el número en esa fila y esa columna dentro de la tabla.

Después de esto hay una serie de ifs:

- Si el valor de la columna es -1 quiere decir que hubo un error en el léxico.
- Si el valor dentro de la tabla es -1, el código se acepta.
- Si el valor dentro de la tabla es 0, hubo un error.
- Si el valor dentro de la tabla es positivo, se aplica un desplazamiento.
- Si el valor dentro de la tabla es negativo, se aplica una regla.

Cuando hay un error o se acepta el código, la respectiva bandera se activa y se termina la ejecución.

Cuando hay un desplazamiento, se apila la columna y el valor de la tabla, y se elimina el primer elemento de la cola que se obtuvo en el léxico.

Cuando hay una regla, se obtiene la regla con la función antes mencionada, se desapilan el doble de los elementos que los de la parte derecha de la regla, se apila el de la parte izquierda y se obtiene otro valor de la tabla que también se apila.

La gramática utilizada para este analizador es la siguiente:

0 Programa -> Definiciones

1 Definiciones -> "

2 Definiciones -> Definicion Definiciones

3 Definicion -> DefVar

4 Definicion -> DefFunc

5 DefVar -> tipo id ListaVar ;

6 ListaVar -> "

7 ListaVar -> , id ListaVar

8 DefFunc -> tipo id (Parametros) BloqFunc

9 Parametros -> "

10 Parametros -> tipo id ListaParam

11 ListaParam -> "

12 ListaParam -> , tipo id ListaParam

13 BloqFunc -> { DefLocales }

14 DefLocales -> "

15 DefLocales -> DefLocal DefLocales

16 DefLocal -> DefVar

17 DefLocal -> Sentencia

18 Sentencias -> "

19 Sentencias -> Sentencia Sentencias

20 Sentencia -> id = Expresion ;

21 Sentencia -> if (Expresion) SentenciaBloque Otro

22 Sentencia -> while (Expresion) Bloque
23 Sentencia -> return Expresion ;
24 Sentencia -> LlamadaFunc ;
25 Otro -> "
26 Otro -> else SentenciaBloque
27 Bloque -> { Sentencias }
28 Argumentos -> "
29 Argumentos -> Expresion ListaArgumentos
30 ListaArgumentos -> "
31 ListaArgumentos -> , Expresion ListaArgumentos
32 Atomo -> LlamadaFunc
33 Atomo -> id
34 Atomo -> constante
35 LlamadaFunc -> id (Argumentos)
36 SentenciaBloque -> Sentencia
37 SentenciaBloque -> Bloque
38 Expresion -> (Expresion)
39 Expresion -> opSuma Expresion
40 Expresion -> opNot Expresion
41 Expresion -> Expresion opMul Expresion
42 Expresion -> Expresion opSuma Expresion
43 Expresion -> Expresion opRelac Expresion
44 Expresion -> Expresion opIgualdad Expresion
45 Expresion -> Expresion opAnd Expresion
46 Expresion -> Expresion opOr Expresion
47 Expresion -> Atomo

Y el parseo se realizó con el siguiente código:

```
void SintaxisAnalyzer::parse(){
    int row(0);
    bool error(false);
    bool accept(false);
    parserStack.push_back(row);
    while(!error && !accept && decodedCode.size() > 0){
        int column = decodedCode[0];
        int number = getNumberInTable(row, column);
        if(column == -1){
            result = "Error en análisis léxico.";
            cout << "Error en análisis léxico.";
            error = true;
        }else if(number == -1){
            accept = true;
            result = "Acepta.";
            cout << "Acepta.";
        }else if(number == 0){
            error = true;
            result = "Error.";
            cout << "Error.";
        }else if(number > 0){
            //cout << "desplazamiento" << endl;
            /// desplazamiento
            parserStack.push_back(column);
            parserStack.push_back(number);
            row = number;
            decodedCode.erase(decodedCode.begin());
        }else{
            //cout << "regla" << endl;
            vector<int> rule = getRule(number);
            int left = rule[0];
            int right = rule[1];
            //cout << "right " << right << endl;
            int i = 0;
            /// eliminar right * 2 elementos del stack
            //cout << "Antes de eliminar ";
```

```

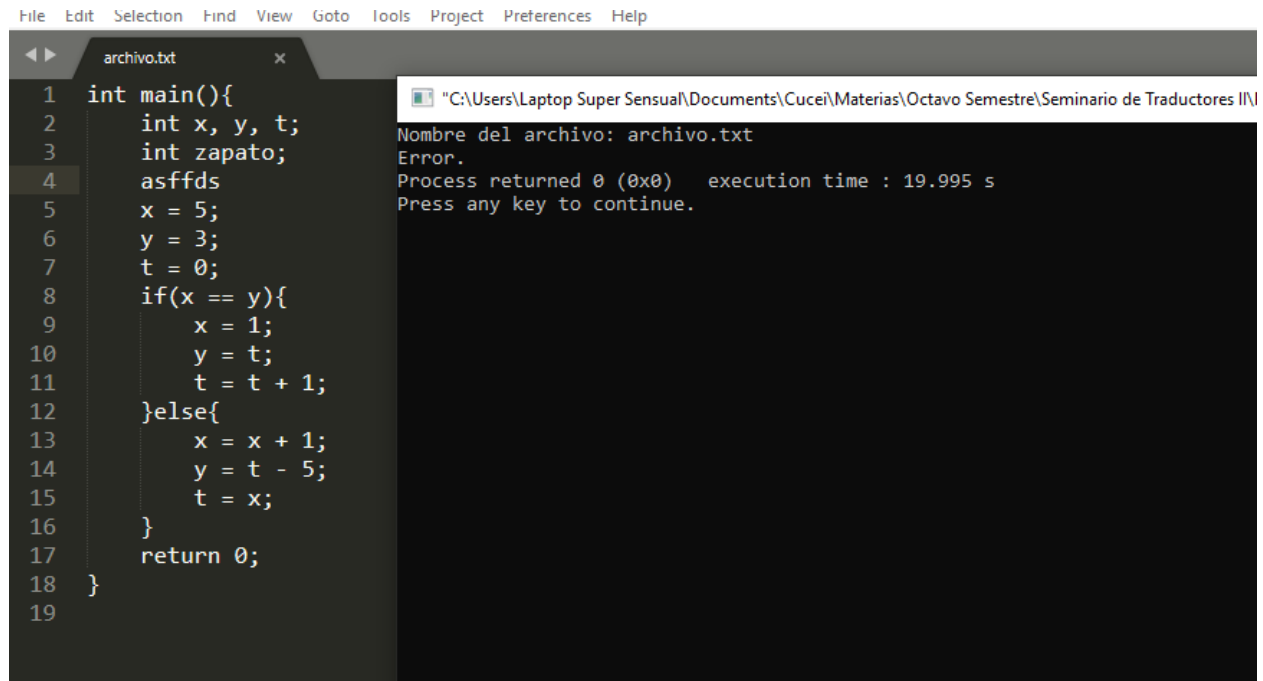
//printParserStack();
while(i < right * 2){
    parserStack.pop_back();
    i = i + 1;
}
//cout << "Despu+es de eliminar ";
//printParserStack();

/// agregar lado izquierdo al stack
parserStack.push_back(left);
/// obtener utlimos dos del stack en la tabla y agregarlo al stack
row = *(parserStack.rbegin()+1);
column = parserStack.back();
//printParserStack();
//cout << row << " " << column << endl;
number = getNumberInTable(row, column);
parserStack.push_back(number);
row = number;
}
//printParserStack();
}
}

```

Resultados obtenidos.

El programa pide al usuario un nombre de archivo y dice al usuario si el código pertenece o no a la gramática antes mencionada.



The screenshot shows a code editor with a menu bar (File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, Help) and a tab labeled 'archivo.txt'. The code in the editor is as follows:

```
1 int main(){
2     int x, y, t;
3     int zapato;
4     asffds
5     x = 5;
6     y = 3;
7     t = 0;
8     if(x == y){
9         x = 1;
10        y = t;
11        t = t + 1;
12    }else{
13        x = x + 1;
14        y = t - 5;
15        t = x;
16    }
17    return 0;
18 }
19
```

On the right side, there is a console window showing the output of the program:

```
"C:\Users\Laptop Super Sensual\Documents\Cucei\Materias\Octavo Semestre\Seminario de Traductores II\
Nombre del archivo: archivo.txt
Error.
Process returned 0 (0x0)   execution time : 19.995 s
Press any key to continue.
```

Conclusiones

El analizador sintáctico es una de las partes más importantes de un compilador ya que va a revisar que el código pertenezca al lenguaje que se produce en la gramática. Sin embargo no es toda la validación que se necesita ya que éste analizador no es capaz de detectar errores de variables que se necesita, pues no es capaz de detectar variables no declaradas, tipos de dato erróneos, uso de palabras reservadas, etc. Cosas que se validan en el analizador semántico.