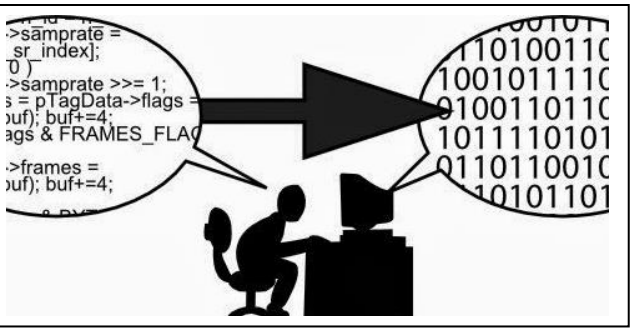
Universidad de Guadalajara
CUCEI – Centro Universitario de Ciencias Exactas e Ingenierías
Seminario de Solución de Problemas de Traductores de Lenguajes II
División de Ciencias Computacionales
Mtro. Michel Emanuel López Franco

Programa 3



Velasco Hernandez Victor Manuel

Código: 216598879

| D02 | 2022A |

08/Febrero/2022

Lunes y miércoles:

|13:00-15:00|

Mini analizador sintáctico

	Universidad de Guadalajara **CUCEI** **Seminario de Solución de Problemas de Traductores de Lenguajes II**	**Victor Manuel Velasco Hernández** **|Código: 216598879 | | 2022A |** **08/ Febrero /2022**

Actividad 3 – Reporte de mini analizador sintáctico

Introducción:

El analizador sintáctico es una parte fundamental de los compiladores ya que es la segunda etapa de este mismo, en el cuál se complementa con el analizador léxico el cuál genera tokens, a estos se les asignara una estructura con la implementación de un segundo nivel el cuál detectará las estructuras definidas por el usuario.

Desarrollo:

Esta actividad corresponde a un analizador sintáctico sencillo, el cuál identifica procesos y asimila con sus respectivas estructuras, las cuales son contenidas en las tablas R1 y R2, conteniendo las reglas que les corresponden a cada uno de los ejemplos.

Primero se declaran las siguientes tablas, las cuales corresponden a las reglas de las expresiones regulares del análisis sintáctico:

```
9   //Corresponde a la expresión regular del ejemplo 1
10  int tablaR1[5][4] = {
11      2,0,0,1,
12      0,0,-1,0,
13      0,3,0,0,
14      4,0,0,0,
15      0,0,-2,0
16  };
17
18  //Tabla aluciva a las derivaciones de la expresión regular
19  int tablaR2[5][4] = {
20      2,0,0,1,
21      0,0,-1,0,
22      0,3,-3,0,
23      2,0,0,4,
24      0,0,-2,0
25  };
```

Para lo cuál, usaremos nuestro “main” para ejecutar los 2 ejercicios solicitados:

```
40 v int main(int argc, char* argv[]) {
41
42     //Responde a los dos ejercicios
43     r1(entrada1);
44     r2(entrada2);
45
46
47       return 0;
48 }
```

Para después ejecutar las funciones “r1” y “r2”, la primera tiene comentarios respecto a su funcionamiento:

```
50  //Análisis léxico del primer ejercicio
51 v void r1(string c) {
52       int contadorinterno = 0;
53     //Se inicia la pila que servirá de buffer
54       Pila pila;
55     //Se introduce la entrada correspondiente
56       lexico.entrada(entrada1);
57     //Se aplica el $0
58       pila.push(TipoSimbolo::PESOS);
59       pila.push(0);
60     //Se empieza a recorrer la entrada
61       lexico.sigSimbolo();
62
63      //Se lee en la fila el dato que está en la pila
64       fila=pila.top();
65       columna=lexico.tipo;
66     //Y se define una acción en base a la tabla que contiene las
   reglas de cada ejemplo
67       accion=tablaR1[fila][columna];
68       //Se muestan los datos de la pila
69       pila.muestra();
70       cout<<"entrada: "<<lexico.simbolo<<endl;
71
```

Victor Manuel Velasco Hernández
|Código: 216598879 | | 2022A |
08/ Febrero /2022

```
72    //Basados en la regla de la tabla, se hace el recorrido por
      posiciones que no tienen E
73        cout<<"accion: "<<accion<<endl;
74
75        while(contadorinterno!=3) {
76            pila.push(lexico.tipo);
77            pila.push(accion);
78            lexico.sigSimbolo();
79
80            fila=pila.top();
81            columna=lexico.tipo;
82            accion=tablaR1[fila][columna];
83
84            pila.muestra();
85    //Se imprimen las entradas analizadas
86            cout<<"entrada: "<<lexico.simbolo<<endl;
87            cout<<"accion: "<<accion<<endl;
88        contadorinterno++;
89        }
90      //}
```

```
91      //En base a las acciones efectuadas se empiezan a hacer
        eliminaciones para completar el proceso
92        if(accion<0){
93            for(int i=0;i<6;i++){
94                pila.pop();
95            }
96        }
97        if(accion==0)//Si presenta algún error, mandara un mensaje
98          cout << "Error"<<endl;
99        else{//Se revisa si la pila se encuntra a su estado final
100           pila.muestra();
101           fila=pila.top();
102           columna=3;
103           accion=tablaR1[fila][columna];
104
105           pila.push(3);
106           pila.push(accion);
107           pila.muestra();
108
109           cout<<"entrada: "<<lexico.simbolo<<endl;
110           cout<<"accion: "<<accion<<endl;
111
112           fila=pila.top();
113           columna=lexico.tipo;
114           accion=tablaR1[fila][columna];
```

```
116         pila.muestra();
117         cout<<"entrada: "<<lexico.simbolo<<endl;
118         cout<<"accion: "<<accion<<endl;
119       //Se manda el estado de aceptación
120         aceptacion=accion==-1;
121         if(aceptacion)
122             cout<<"Aceptacion"<<endl;
123     }
```

Al igual, la función del segundo ejercicio, en el código se cuentan con comentarios, los cuales son alusivos a su funcionamiento, muy similares a los de el ejercicio 1:

```
124   //Análisis sintáctico del ejercicio 2
125   void r2(string c){
126         contadorinterno= 0;
127         cout<<endl<<endl<<endl;
128       //Se introduce la entrada correspondiente al proceso
129         lexico.entrada("a + b + c + d + e + f");
130       //Se introduce el $0
131         pila.push(TipoSimbolo::PESOS);
132         pila.push(0);
133       //Se empieza a recorrer la entrada
134         lexico.sigSimbolo();
135         fila=pila.top();
136         columna=lexico.tipo;
137       //Se define la acción en base a lo que se encuentra contenido
      en la tabla
138         accion=tablaR2[fila][columna];
139
```

```
140     //Se muestra la información de la pila
141     pila.muestra();
142     cout<<"entrada: "<<lexico.simbolo<<endl;
143     cout<<"accion: "<<accion<<endl;
144     //Se empiezan a recorrer todas las posiciones de la entrada
        para inserta los tokens y acciones correspondientes dentro de
        la pila
145 v   while(contadorinterno!=16){
146 v   if(accion>0){
147         pila.push(lexico.tipo);
148         pila.push(accion);
149         lexico.sigSimbolo();
150         fila=pila.top();
151         columna=lexico.tipo;
152         accion=tablaR2[fila][columna];
153         pila.muestra();
154         cout<<"entrada: "<<lexico.simbolo<<endl;
155         cout<<"accion: "<<accion<<endl;
156     }
```

```
157     //En base a las acciones definidad se empizan a realizar
        operaciones específicas ya sea para limpiar la pila y llevar
        los procesos a un estado de aceptación
158 v   if(accion<0){
159         if(accion==-3)
160
161 v       {
162             pila.pop();
163             pila.pop();
164             fila=pila.top();
165             columna=3;
166
167             accion=tablaR2[fila][columna];
168             pila.push(3);
169             pila.push(accion);
170             pila.muestra();
171             cout<<"entrada: "<<lexico.simbolo<<endl;
172             cout<<"accion: "<<accion<<endl;
173
174         }
175 v       if(accion==4){
176 v           while(accion==4){
177 v             for(int y=0;y<6;y++){
178                 pila.pop();
```

```
194  //Entra en el estado final, cuando todos los procesos
     correspondientes fueron realizados
195      aceptacion=accion==-1;
196      if(aceptacion)
197          cout<<"Aceptacion"<<endl;
198
199  }
```

Conclusión:

Implementar en una escala pequeña un analizador sintáctico hace que nos demos cuenta de la importancia de cada uno de los elementos que lo conforma, ya que gracias a estos, es posible desarrollar la lógica que lo rige en todos los procesos relacionados a su análisis, esto basándose en definiciones establecidas, las cuales son acordes a las necesidades del programador y del compilador.