Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

CUCEI

División De Electrónica y Computación.

Seminario de Solución de Problemas de Traductores de Lenguajes II

Profesor: Dr. López Franco Michel Emanuel.

Alumno: López Pacheco Luis Fernando.

Código: 216041378.

Carrera: Ingeniería en Computación (INCO).

Sección: D02.

Calendario: 2021-B.

Actividad: Mostrar árbol sintáctico.

11/10/21

Introducción:

Para la creación de este programa reutilice el código generado en la actividad anterior, pero ahora realice algunas modificaciones en los nodos para poder identificar su tipo, además de ahora mostrar algunas reglas mientras se hace el análisis sintáctico, también fue necesario crear nuevos nodos para mostrarlas en ejecución y además realizar sus respectivas reducciones en la pila dependiendo del tipo de regla que sea utilizada. También en esta práctica será necesario la creación del árbol del siguiente código.

```
int a;
int suma(int a, int b){
return a+b;
}

int main(){
float a;
int b;
int c;
c = a+b;
c = suma(8,9);
}
```

Analizador Sintáctico.

Un analizador sintáctico (o parser) es un programa que es parte de un compilador. El compilador se asegura de que el código se traduce correctamente a un lenguaje ejecutable. La tarea del analizador es la descomposición y transformación de las entradas en un formato utilizable para su posterior procesamiento. Se analiza una cadena de instrucciones en un lenguaje de programación y luego se descompone en sus componentes individuales.

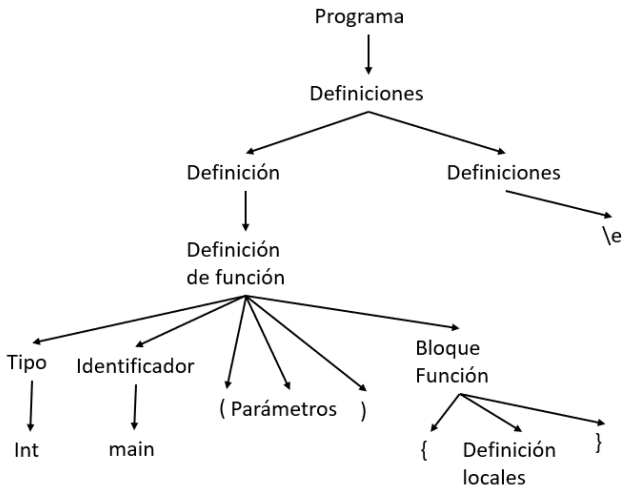
La principal tarea de un analizador no es comprobar que la sintaxis del programa fuente sea correcta, sino construir una representación interna de ese programa y, en el caso que sea un programa incorrecto, dar un mensaje de error. El analizador sintáctico consiste en la segunda fase de un compilador.

Funcionamiento de la gramática.

Para que sea posible ingresar el código mencionado anteriormente, es importante conocer el funcionamiento de la tabla que contiene la gramática, la cual hace posible poder realizar el análisis sintáctico. Para que sea posible esto cree el método “matriz” que se encarga de leer y generar la tabla que contiene la gramática. Y almacena en un arreglo la tabla de la gramática y en otro lo que realiza cada regla, esto lo logra con una lectura por renglón de cada elemento en el archivo.

Ejemplo utilizando la gramática.

Pila	Entrada	Salida
$0	int main(){}$	d5
$0int5	main(){}$	d8
$0int5main8	(){}$	d11
$0int5main8(11	){}$	R10 <Parametros> ::= \e
$0int5main8(11<Parametros> 14	){}$	17
$0int5main8(11<Parametros> 14)17	{}$	d20
$0int5main8(11<Parametros> 14)17{20	}$	R15 <DefLocales> ::= \e
$0int5main8(11<Parametros> 14)17{20<DefLocales>23	}$	d33
$0int5main8(11<Parametros> 14)17{20<DefLocales>23}33	$	R14 <BloqFunc> ::= { <DefLocales> }
$0int5main8(11<Parametros> 14)17	$	19
$0int5main8(11<Parametros> 14)17<BloqFunc> 19	$	R9 <DefFunc> ::= tipo identificador (<Parametros>) <BloqFunc>
$0	$	6
$0<DefFunc>6	$	R5 <Definicion> ::= <DefFunc>
$0	$	3
$0<Definicion>3	$	R2 <Definiciones> ::= \e
$0<Definicion>3 <Definiciones>7	$	R3 <Definiciones> ::= <Definicion> <Definiciones>
$0	$	2
$0<Definiciones>2	$	R1 <programa> ::= <Definiciones>
$0	$	1
$0<programa> 1	$	R0 ::= Aceptación



Implementación:

El método que se encarga de hacer el análisis sintáctico es “analyzer1” en esta ocasión realice varios cambios, el más evidente es la utilización de una lista ligada ya que con esta estructura me fue más fácil el manejo de los nodos al crear el árbol, para esto realice una copia de los tokens anteriormente analizados por el analizador léxico, que genera una lista con los token detectados de la cadena que ingresa el usuario.

Es importante mencionar que lo primero que realiza el método “analyzer1” es la generación de la matriz con este mismo método (matriz()). Posteriormente se añade el token terminal que es el símbolo “$”. También se crean 2 nodos extras, para el manejo de los nodos y los no terminales, y antes de comenzar la iteración preprueba “do while”, que funciona de la misma manera como en las actividades anteriores, solo que ahora se hace un desplazamiento diferente ya que utilice como mencione anteriormente una lista ligada.

Además de utilizar 2 pilas una de enteros y otra pila de objetos que almacena los nodos generados. El método “creacionArbol” por medio de recibir por parámetro una pila

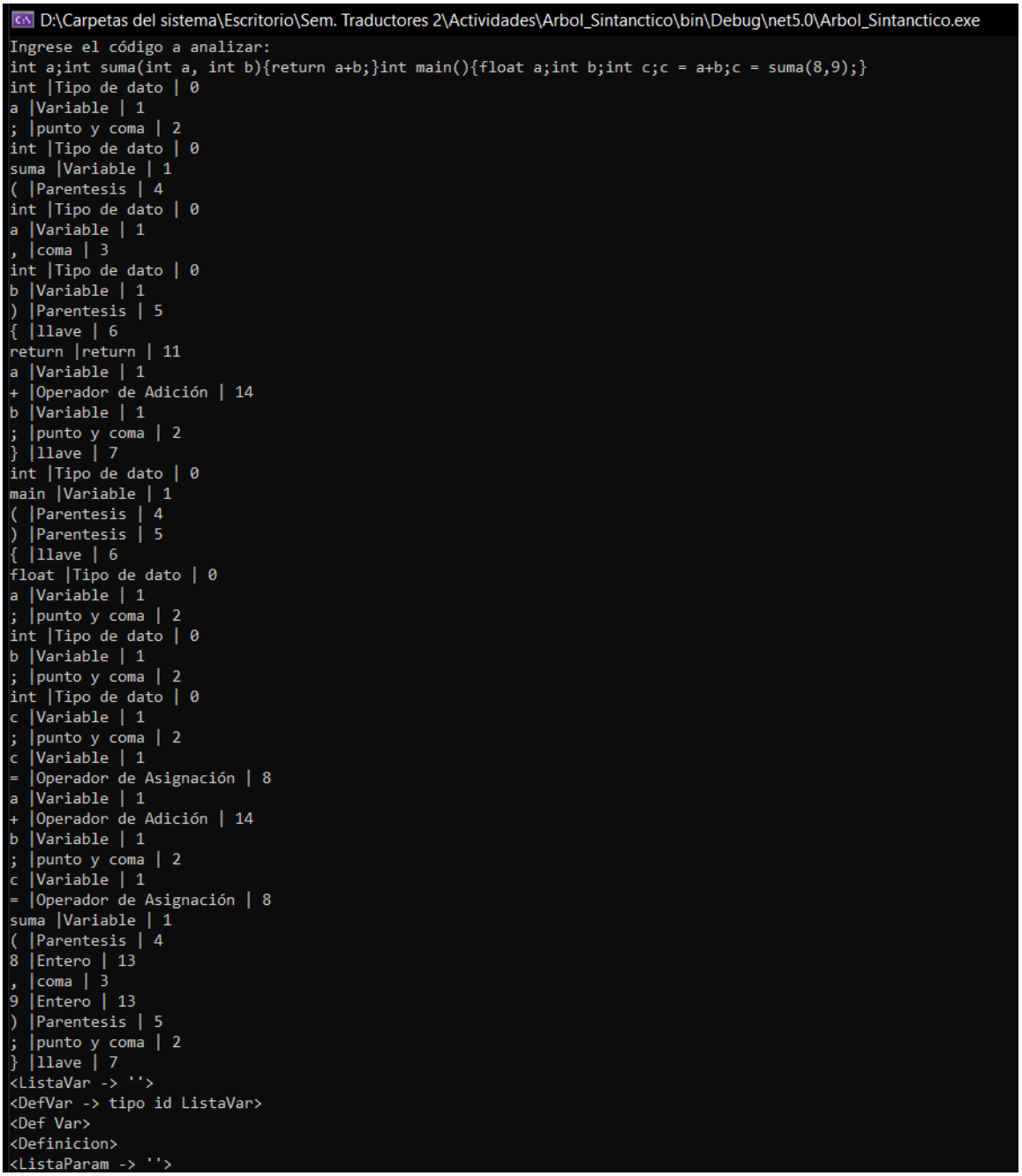
y una regla, que determina cuantas eliminaciones en la pila de enteros se realizarán, debido a que hay varias reglas que hacen el mismo numero de eliminaciones se agrupan con un “switch” que dependiendo de la regla, hará una serie de eliminaciones, y creación de sus respectivos nodos. El método “creaciónArbol” ahora también muestra al hacer una reducción la regla utilizada.

En la clase nodo fue necesario crear otros objetos que heredan de la clase Nodo para hacer el correcto manejo del árbol pila, ya que tuve que implementar como clases algunas palabras reservadas y en su constructor todos las eliminaciones necesarias que requiere la expresión o el token que se haya ingresado.

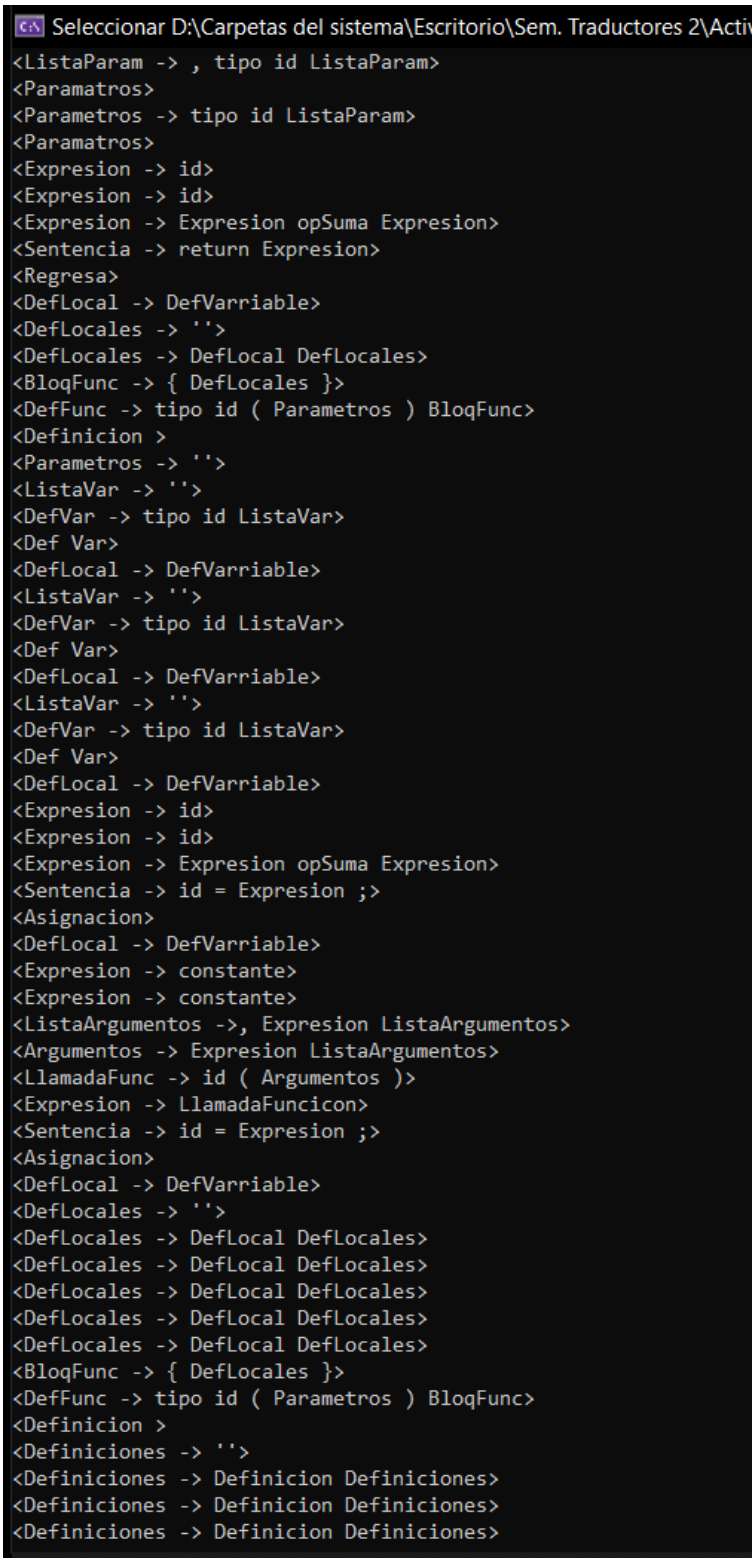
También fueron necesarias algunas modificaciones en la clase léxico y token para que funcionará correctamente con la gramática, principalmente realice cambios en los id de cada símbolo, y también que fueran detectadas más palabras reversadas y produjeran su respectivo id, para ser posteriormente analizadas con el método analyzer (analizador sintáctico). Solamente como la implementación la realice en consola y no con interfaz gráfica todos los elementos que sean ingresados por el usuario tienen que ser de forma líneal (sin saltos de línea), por ejemplo: int main() {int a,b,sum; while(a<b){sum = a+b;}} <- es una entrada valida.

Para que funcione el analizador sintáctico primero se ingresa una cadena generada por el usuario y después se realiza el análisis léxico, en está implementación posteriormente se muestra cada uno de los token generados y por último se muestra las reglas que fueron utilizadas y si la sintaxis fue valida o no.

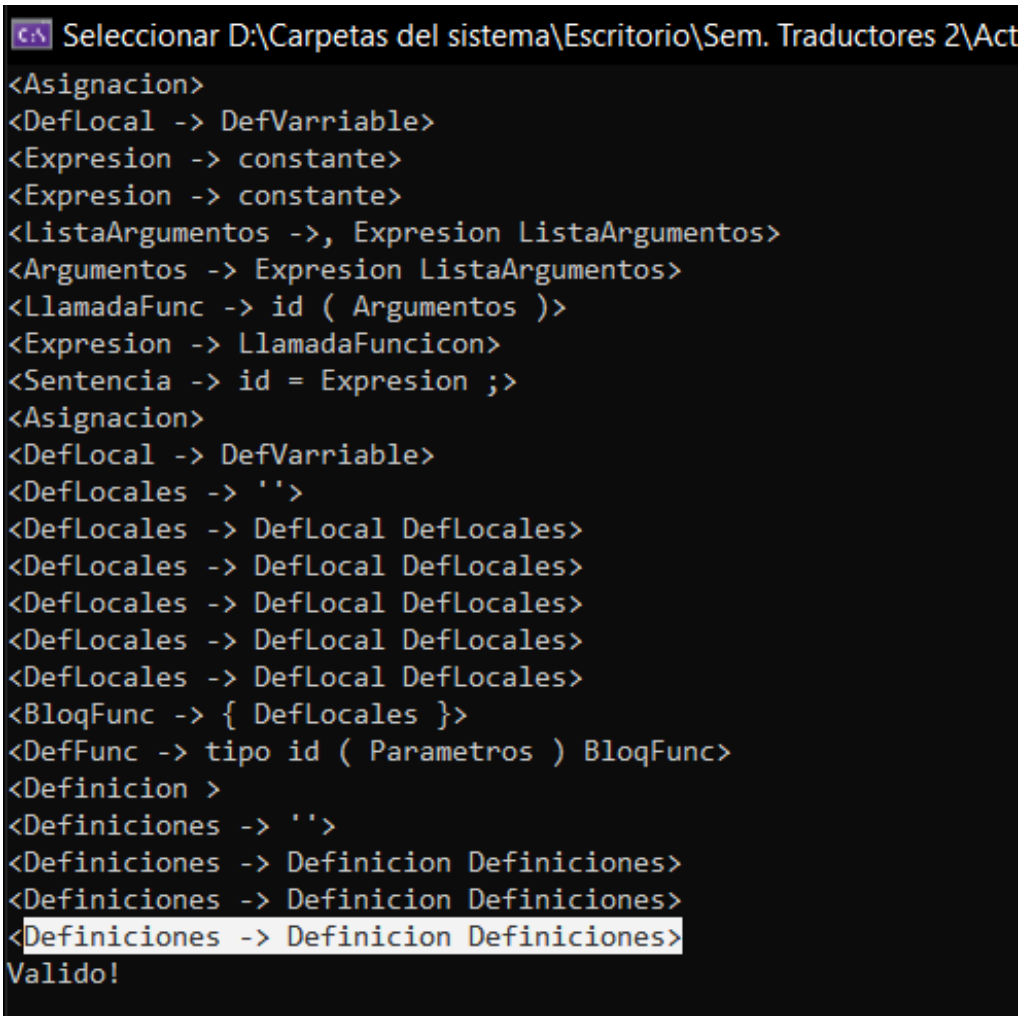
Demostración y Ejecución del programa.



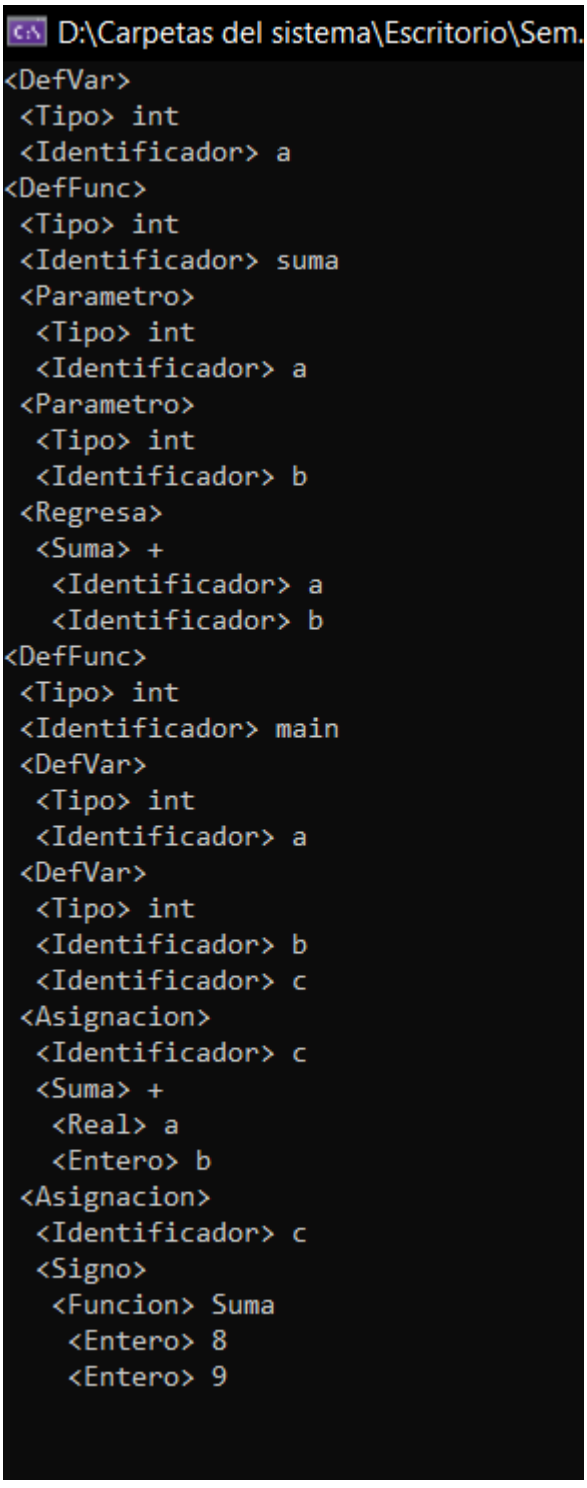
En esta imagen se puede visualizar el análisis léxico a cada uno de los caracteres ingresados y la creación de cada uno de los tokens, se muestra el tipo de cada token junto a su identificador.



En esta imagen se puede visualizar algunas producciones de las reglas utilizadas de la gramática al realizar el análisis sintáctico.



Al finalizar el análisis sintáctico se muestra un mensaje que indica si el código ingresado es válido o no.



Por últimos se muestra la impresión del árbol sintáctico generado del código ingresado.

Código.

- ***https://github.com/LuisLopezPacheco/TraductoresDeLenguajes2.git***

Conclusión.

Esta actividad se me complico bastante ya que fueron necesarios varias cambios con respecto a la actividad anterior, también tenia dudas de cómo se debía ser mostrado el árbol sintáctico, pero basándome en el material didáctico proporcionado por el profesor logre idear una forma de hacer la impresión del árbol sintáctico. También me fue de gran utilidad crear un ejemplo paso a paso para comprender mejor el funcionamiento de la gramática Considero que puede realizar más modificaciones para que al momento de mostrar las reglas solo sean visibles algunas y también serán necesarias más modificaciones para la implementación del analizador semántico.

Bibliografías.

- ***Material proporcionado durante la clase Seminario de Traductores de lenguajes 2 (profesor: López Franco Michel Emanuel) (2021).***
- ***Introducción al análisis sintáctico. (s.f.). Obtenido de: http://informatica.uv.es/docencia/iiguia/asignatu/2000/PL/2007/tema3.pdf***
- ***Peinado, F., & Sierra, J. L. (2010). Repaso. Lenguajes formales. Obtenido de: http://www.fdi.ucm.es/profesor/fpeinado/courses/compiling/repasolenguajesformales.pdf.***