PRACTICA 8

Objetivos

Objetivo General:

- Implementar un analizador sintáctico simple el cual pueda reconocer un conjunto de sentencias de tipo asignación de variables (PARSER_V1).

Objetivos Particulares:

- La implementación de PARSER_V1 debe realizarse en base a las Gramáticas Libres de Contexto - G_2 definidas en la clase del 07 de noviembre del 2018.
- PARSER_V1 podrá reconocer un programa definido en el lenguaje UAMI con un número indefinido de asignaciones simples, esto es; un Identificador, un símbolo de =, un número entero y un símbolo de;
- Implementar un método (parea) que permita comparar cada uno de los tokens que le regresa el analizador lexicográfico con los tokens definidos en cada una de las reglas de producción del PARSER_V1, para comprobar que los tokens estén definidos en orden.
- En caso de que estos tokens no estén definidos en un orden establecido, PARSER_V1 debe generar los errores sintácticos.
- Tomar las clases UAMI, Alex, Palabras Reservadas y Tabla_de_Simbolos que se desarrollaron en la Practica 6 – ALEX FINAL; e incorporarlos a la implementación del PARSER_V1

Documentos a entregar

- El proyecto desarrollado en equipo o de forma individual
- La revisión de la práctica se revisará en el laboratorio AT-105.

Plazo de entrega

Para la evaluación de esta práctica se llevará de la siguiente forma:

- La hora y fecha límite para la revisión de la práctica es el día 20 de noviembre antes de las 8:00 AM. Como el 20 de noviembre es día festivo la práctica será enviada por correo antes de las 7:59:59 AM. El correo para enviarla es opelo1209@gmail.com.

Nota. No se recibirá ninguna práctica fuera de ese horario, sin ninguna excepción.

Herramientas para el desarrollo de la Práctica

1. Se puede apoyar de la clase que se revisó en la clase del miércoles 07 de noviembre del 2018, para implementar el método que permita comparar los tokens regresados por el método Alexico de la clase Alex con los símbolos terminales de la G2 definida en el PARSER_V1.

2. Se tiene que modificar el método Alexico, de tal forma que ahora regrese solamente un entero correspondiente a la posición donde se encuentra el token en la tabla se símbolos.

   Para lograr esto, todos los elementos que reconozca el analizador lexicográfico se van a agregar en la tabla de símbolos a excepción de los **espacios en blanco y los tokens inválidos**.

   Es por esto que todas las gramáticas (**a excepción de las que anteriormente se mencionaron**) tendrán un funcionamiento similar a las que reconocen una cadena de caracteres y un identificador o palabra reservada.

   Ya que estas comprueban si un lexema está o no en la tabla de símbolos, y dependiendo de la respuesta se ingresa o no el lexema reconocido.

3. El funcionamiento de la clase Parser tiene los siguientes pasos:

   a. Se manda a llamar la función Alexico() para que en un atributo llamado preanalisis se guarde el token y el lexema actual con el cual se comience el análisis sintáctico.

      Dado que se van a guardar dos elementos de tipo String, se recomienda usar una lista de dos columnas de tipo String.

      De modo que en la posición 0 se guarde el lexema y en la posición 1 se guarde el token, ambos a través de la invocación de los Métodos Obtener_Token y Obtener Lexema de la Tabla de Símbolos, algo de la siguiente forma:

      ```
      pos = obj_UAMI.A1.ALexico(obj_UAMI)
      self.preanalisis[0] = obj_UAMI.T1.Obtener_Lexema(pos);
      self.preanalisis[1] = obj_UAMI.T1.Obtener_Token(pos);
      ```

      Donde T1 es un objeto de la clase Tabla_de_Simbolos enviado como uno de los parámetros de entrada desde el método inicia_Compilacion de la clase UAMI.

   b. Cada gramática que se implemente en esta práctica corresponde a un método, por lo cual se tendrán métodos con nombres como: Inicio(), Encabezado(), Secuencia() y Asignacion(). En cada uno de ellos se establecerá un orden en el que deben aparecer cada uno de los tokens presentes en el archivo fuente.

   c. Al momento que se lea un token, se deberá comparar con lo que se espera en cada G_2, si la comparación es verdadera se leerá otro token hasta formar una gramática dada.

   d. En el caso contrario se generará un error sintáctico, el cual se imprimirá en el archivo *.err

   e. El programa terminara su ejecución una vez que se encuentre el EOF del archivo *.fte.

Especificaciones de la Práctica

Para esta práctica se tiene que implementar las siguientes clases:

Clase PARSER_V1.py (Aplicación Grafica):

La ejecución de un programa para la implementación de PARSER_V1 podría verse de la siguiente forma:

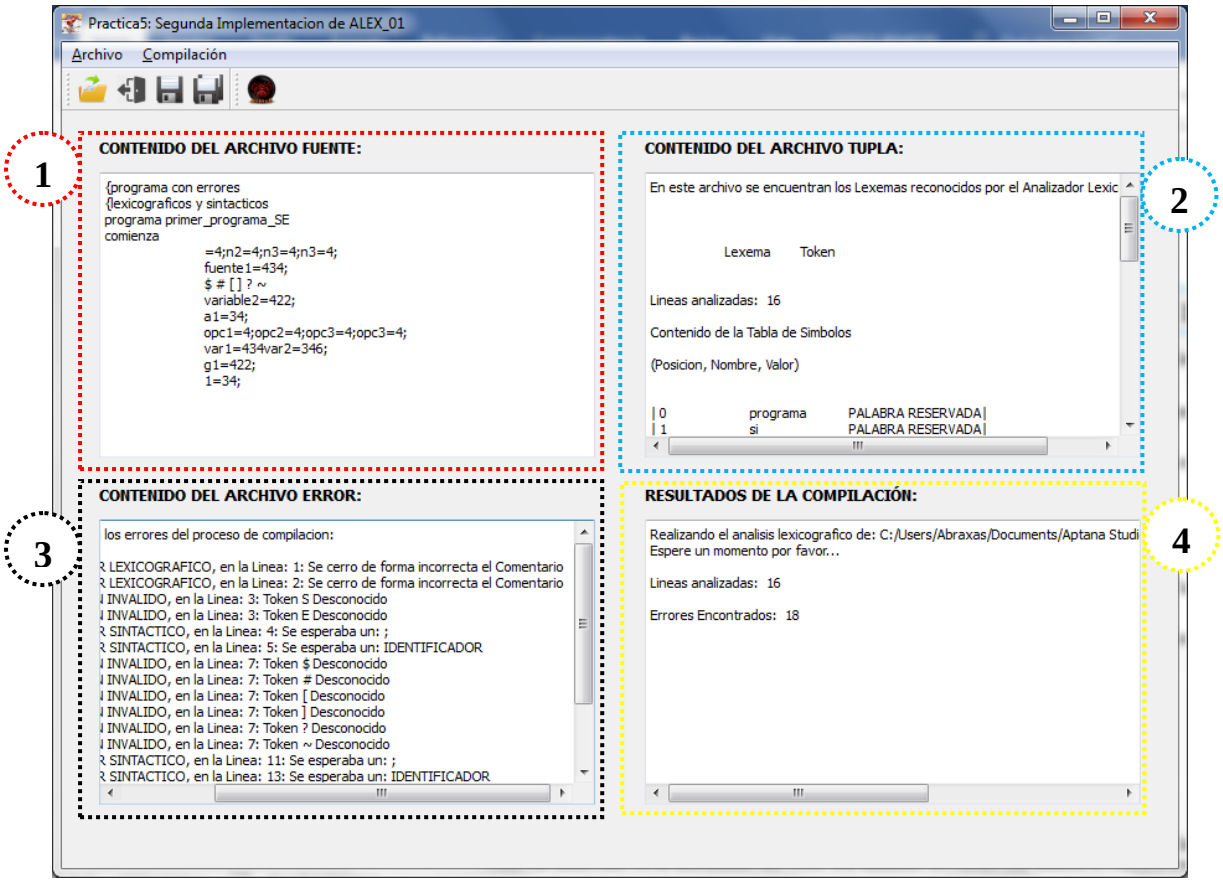

Figura 1. Ejecución de PARSER_V1

De la Figura 1 podemos señalar lo siguiente:

(1). Esta área de texto va seguir desplegando el contenido del archivo fuente (*.fte).

(2). Ya no es necesario mostrar los tokens y lexemas que se reconozcan en esta área de texto, solo se mostrara el contenido final de la Tabla de Símbolos.

(3). En el área de texto que muestra el contenido del archivo de errores (*.err) se mostraran los errores lexicográficos y sintácticos.

En el caso de los errores lexicográficos solo se podrán presentar casos en los que se tenga un token inválido y los diferentes errores lexicográficos de la Practica 6.

Y en el caso de los errores sintácticos se presentarán cuando un token no coincida con los elementos definidos en alguna regla de producción de la Gramática Libre de Contexto definida para esta práctica.

(4). Los resultados de la compilación serán similares a los que se presentaban en la Practica 6.

Esta aplicación grafica tendrá las mismas opciones que se definieron en la Practica 6, por ejemplo, abrir archivos, editar archivos, Inicia Compilación, salir del programa, etc.

Clase Parser (Funcionamiento del Analizador Sintáctico):

En esta clase se deben implementar cada una de las reglas de producción que se definieron en la clase del miércoles 07 de noviembre del 2018. Por ejemplo, el código que tendría el constructor de la clase sería siguiente:

```
def __init__(self):
    self.preanalisis = [0 for x in range(2)]
    self.preanalisis[0] = ""
     self.preanalisis[1] = ""
```

Teniendo esto el primer método de la clase Parser se podría declarar de la siguiente forma:

```
def Inicio(self, obj_UAMI):
    pos = obj_UAMI.A1.ALexico(obj_UAMI)
    self.preanalisis[0] = obj_UAMI.T1.Obtener_Lexema(pos);
    self.preanalisis[1] = obj_UAMI.T1.Obtener_Token(pos);
    self.Encabezado(obj_UAMI);
    self.Secuencia(obj_UAMI);
    self.Parea(obj_UAMI.PR.HECHO, obj_UAMI);
```

Después del llenado de la variable preanalisis mediante la invocación del método Alexico, se invocan dos reglas de producción que son Encabezado y Secuencia, siguiendo el orden de la derivación por la izquierda.

Finalmente se invoca al método Parea para comparar un símbolo terminal con el token o lexema actual. El código para el método Parea es la siguiente:

```
def Parea(self, se_espera, obj_UAMI):
    if self.preanalisis[0] == se_espera or self.preanalisis[1] == se_espera:
        pos = obj_UAMI.A1.ALexico(obj_UAMI)
        self.preanalisis[0] = obj_UAMI.T1.Obtener_Lexema(pos)
        self.preanalisis[1] = obj_UAMI.T1.Obtener_Token(pos)
         return True
    else:
        obj_UAMI.errores+=1
        obj_UAMI.f_err.write("Tipo de Error: " + obj_UAMI.PR.ERROR_S + ", en la Linea: " +
               str(obj_UAMI.lineas) + ": Se esperaba un: " + se_espera + "\n");
        return False
```

Clase UAMI:

Es el mismo funcionamiento de la Practica 6, **con la diferencia en que se elimina por completo el ciclo do-while para el recorrer el archivo fuente** y en su lugar solo abra una invocación al método Inicio a travez de un objeto de la clase Parser, algo de la siguiente forma:

```
…
P = Parser.Parser()
P.Inicio(obj_UAMI)

Resultados = Resultados + "\n\nLineas analizadas:   " + str(self.lineas)
self.f_tpl.write("\n\nLineas analizadas:   " + str(self.lineas))
```

```
        self.T1.Imprimir_Tabla(obj_UAMI)
        …
```

Clase Alex:

Es el mismo funcionamiento de la Practica 6, con la diferencia en que el método Alexico() regresa un entero correspondiente a la posición en la Tabla de Símbolos del token reconocido.

Clase Tabla_de_Simbolos:

Se mantiene el mismo funcionamiento de la Practica 6.

Clase Palabras Reservadas:

Se mantiene el mismo funcionamiento de la Practica 6.

Códigos de Prueba:

Para esta práctica se puede probar el correcto funcionamiento del Parser con los siguientes códigos.

Nombre del programa: prueba1.fte

```
{programa sin errores}
{lexicográficos y sintácticos}
programa primer_programa_se;
comienza
      n=4;n2=4;n3=4;n3=4;
      fuente1=434;
      variable2=422;
      a1=34;
      opc1=4;opc2=4;opc3=4;opc3=4;
      var1=434;var2=346;
      g1=422;
      f1=34;
```

Nombre del programa: prueba2.fte

```
{programa con errores
{lexicograficos y sintacticos
programa primer_programa_SE
comienza
      =4;n2=4;n3=4;n3=4;
      fuente1=434;
      $ # [ ] ? ~
      variable2=422;
      a1=34;
      opc1=4;opc2=4;opc3=4;opc3=4;
      var1=434var2=346;
      g1=422;
      1=34;
```