**PROYECTO PRÁCTICO FASE I**

**GENERADOR DE SCANNER**

**Objetivo**

Dentro del análisis de los lenguajes Formales es importante conocer las fases del proceso de compilación, la finalidad del proyecto es que el estudiante comprenda la función de los analizadores léxico y sintáctico de un compilador a través de la generación de un programa que sea capaz de reconocer un lenguaje y finalmente evaluar si las palabras utilizadas están bien formadas de acuerdo con una gramática.

El proyecto consta de 3 fases, las cuales son dependientes, es decir, que para poder terminar la fase III, las anteriores deben estar completas, de lo contrario no se podrá entregar el funcionamiento completo.

**Explicación de la gramática**

En la primera fase del proyecto es necesario la lectura de un archivo de texto llamado: GRAMATICA.txt el cual contiene la definición de la gramática.



Dicho archivo está compuesto de las siguientes partes:

1. SETS: Contiene la definición abreviada de un conjunto de símbolos terminales, esta parte puede o no venir dentro del archivo, no es necesario que aparezca pero si aparece, debe poseer al menos un SET.
   1. Ejemplo

SETS

LETRA = 'A'..'Z'+'a'..'z'+'\_'

DIGITO = '0'..'9'

CHARSET = CHR(32)..CHR(254)

* 1. Tomar en cuenta las siguientes características:
     1. La palabra SETS debe estar en mayúscula.
     2. Los sets pueden estar concatenados a través del signo “+”, como muestra el set: LETRA.
     3. Se puede utilizar la función CHR como lo muestra el set: CHARSET.
     4. Puede haber muchos espacios en blanco entre el identificador, el símbolo “=” y la definición.
     5. Puede haber varios saltos de línea (Enters) entre un SET y otro.

1. TOKENS: Los tokens representan los símbolos terminales y no terminales de la gramática, en esta fase no nos importa si un identificador ha sido declarado o no en los SETS,
   1. Ejemplo

TOKENS

TOKEN 1= DIGITO DIGITO \*

TOKEN 2='"' CHARSET '"'|''' CHARSET '''

TOKEN 4 = '='

TOKEN 5 = '<''>'

TOKEN 6 = '<'

TOKEN 7 = '>'

TOKEN 8 = '>''='

TOKEN 9 = '<''='

TOKEN 10 = '+'

TOKEN 11 = '-'

TOKEN 12 = 'O''R'

TOKEN 13 = '\*'

TOKEN 14 = 'A''N''D'

TOKEN 15 = 'M''O''D'

TOKEN 16 = 'D''I''V'

TOKEN 17 = 'N''O''T'

TOKEN 40 = '(''\*'

TOKEN 41 = '\*'')'

TOKEN 42 = ';'

TOKEN 43 = '.'

TOKEN 44 = '{'

TOKEN 45 = '}'

TOKEN 46 = '('

TOKEN 47 = ')'

TOKEN 48 = '['

TOKEN 49 = ']'

TOKEN 50 = '.''.'

TOKEN 51 = ':'

TOKEN 52 = ','

TOKEN 53 = ':''='

TOKEN 3= LETRA ( LETRA | DIGITO )\* { RESERVADAS() }

* 1. LA PALABRA TOKENS debe existir y estar en mayúscula
  2. Esta sección debe existir
  3. Cada token debe poseer la palabra: TOKEN y un número, seguido del signo igual “=”.
  4. Después del signo igual debe venir una expresión regular, que puede ser uno o varios caracteres (Encerrados en apóstrofes).
  5. Los signos utilizados para las operaciones de las expresiones regulares son los únicos que no necesitan estar entre comillas, a menos que se quiera denotar su uso como signo terminal.
     1. Los signos de operaciones para las expresiones regulares son: + \* ? ( ) |

1. ACTIONS: La palabra ACTIONS contiene definición de funciones, en este caso específico las palabras reservadas del lenguaje, es importante que la función: Reservadas() siempre debe existir y puede haber otras funciones.
   1. Ejemplo

ACTIONS

RESERVADAS()

{

18 = 'PROGRAM'

19 = 'INCLUDE'

20 = 'CONST'

21 = 'TYPE'

22 = 'VAR'

23 = 'RECORD'

24 = 'ARRAY'

25 = 'OF'

26 = 'PROCEDURE'

27 = 'FUNCTION'

28 = 'IF'

29 = 'THEN'

30 = 'ELSE'

31 = 'FOR'

32 = 'TO'

33 = 'WHILE'

34 = 'DO'

35 = 'EXIT'

36 = 'END'

37 = 'CASE'

38 = 'BREAK'

39 = 'DOWNTO'

}

* 1. La palabra ACTIONS siempre debe venir acompañada de la función RESERVADAS ().
  2. Todas las funciones deben tener un identificador y unos paréntesis abierto y cerrado.
  3. Las funciones descritas en ACTIONS deben iniciar y finalizar con llaves {}.
  4. Los tokens dentro están conformados por: número, signo igual y luego el identificador entre apóstrofes

1. ERROR: La definición de errores debe venir al menos uno, el ERROR debe tener asignado un número, y el identificador debe tener como sufjio la palabra ERROR en mayúscula:
   1. Ejemplo:

ERROR = 54

* 1. Los identificadores solo deben poseer letras, y en la parte derecha del símbolo igual, solamente pueden haber números.

**SET DE PRUEBAS**

En esta sección se muestra un set de pruebas que se deben incluir y sus respectivos resultados:

|  |  |
| --- | --- |
| **Archivo de pruebas** | **Resultado esperado** |
|  | Formato correcto. |
|  | Error de formato encontrado cerca de línea 4 columna 32. |
|  | Formato correcto. |
|  | Error de formato encontrado cerca de línea 1 columna 6. |
|  | Error de formato encontrado cerca de línea 6 columna 32. |

**Entregables Fase I**

El programa debe codificarse en el lenguaje **C#** Utilizando Windows Forms, versión de IDE recomendado es Visual Studio 2019 aunque se puede utilizar cualquiera de sus versiones anteriores.

Entregables:

* Programa fuente, debidamente documentado internamente
* Programa ejecutable
* Set de casos de prueba utilizados con sus respectivos archivos y resultados esperados.
* Documentación de análisis que contenga las expresiones regulares y árboles de expresiones creados para parte del archivo.

FECHA DE ENTREGA:

Calificación presencial e individual el día: **miércoles 4 de marzo de 2020** en horario de clase, el código fuente a evaluar será el que esté subido en la sección de entrega del portal.

**Aspectos a evaluar**

* Validación de errores, uso de **procedimientos, funciones, clases, herencia, estructuras de datos** en todas las operaciones.
* Adecuada aplicación de los conocimientos vistos en clase
* Calidad de la documentación: ortografía, orden, limpieza y que esté completa.
* Calidad de la solución propuesta: que solucione el problema (que haga lo que requiere el enunciado) en forma eficaz.
* Funcionalidad del programa: debe cumplir a cabalidad con todos los requerimientos.
* Evidencia de la creación del programa y dominio de los conceptos utilizados.
* Uso de herramientas para versiones de código, específicamente TFS sobre Azure DevOps utilizando la cuenta de correo universitaria.
* Creatividad.

**Baremo de calificación**

|  |  |  |  |
| --- | --- | --- | --- |
| ITEM | Resultado | Ponderación | Observaciones |
| Solución entregada en el portal | Sí / No | -- | Si no está en el portal no se califican los demás ítems y se obtiene 0 puntos |
| Solución compila | Sí / No | -- | Si la solución no compila se obtiene 0 puntos |
| Lectura de archivos correcta | Sí / No | 10 puntos | Lee archivos de forma correcta con filtros y no se produce ninguna excepción durante la lectura de archivos. |
| Detecta correctamente los casos de prueba utilizados | Sí / No | 15 puntos por cada archivo de prueba, total: 75 puntos | Genera los resultados esperados de cada archivo de prueba (El catedrático pondrá los archivos de prueba) |
| Creación correcta de árboles de expresión para evaluar expresiones regulares | Sí / No | 10 puntos | No utiliza librerías de evaluación de expresiones regulares ni herramientas de análisis lexicográfico |
| Documentación técnica con expresiones regulares y árboles de expresión | Sí / No | 5 puntos | Presenta documento que muestra razonamiento utilizado. |
| Versionamiento de código (Opcional) | Sí / No | 5 puntos extra | Utiliza Azure DevOps o Git para versionar el código |
| Indica regla que no cumple el formato (opcional) | Sí / No | 5 puntos extra | Si encuentra un error, muestra una sugerencia de lo que esperaba. |