I.T.C.R. Compiladores & Intérpretes

Escuela de Ingeniería en Computación — Scanner
Prof: Ing. Erika Marín Schumann. — I Semestre del 2020

Motivación

Muchas de las aplicaciones que se desarrollan actualmente requieren validar los datos de entrada, ya sea de interfaz o de archivos. Para esto, se necesita los mismos conocimientos básicos que son utilizados en la construcción de la etapa de análisis léxico de un compilador, de ahí que sea de suma importancia tener experiencia al respecto.

Objetivos del proyecto

- Aprender a desarrollar un scanner para un lenguaje de programación
- Utilizar herramientas ya diseñadas para facilitar el diseño del scanner (JFLEX)
- Diseñar la primera etapa del compilador del curso.

Definición general

Esta primera etapa del proyecto conocida formalmente como Scanner o Análisis Léxico es de vital importancia que sea desarrollada con cuidado y visión hacía el futuro, ya que el resto del proyecto se basará en una buena definición del lenguaje a compilar.

Para esta primera etapa del proyecto se debe entregar un programa que reciba un código fuente escrito en el lenguaje de alto nivel llamado SOLIDITY y realice el análisis léxico respectivo. Para esto se debe definir el lenguaje aceptado utilizando la herramienta JFLex. Por lo tanto, la tarea debe ser escrita en Java.

Al finalizar el scaneo el programa deberá desplegarle al usuario el resultado del Análisis Léxico que efectuó. Se esperan dos aspectos del resultado.

1. **Listado de errores léxicos encontrados:** El programa debe desplegar una lista de todos los errores léxicos que se encontraron en el código fuente. Debe desplegar la línea en la que se encontró el error. Es importante que el programa debe poder recuperarse del error y no desplegar los errores en cascada ni terminar de hacer el scaneo al encontrar el primer error.
2. **Listado de los tokens encontrados:** Durante la ejecución del análisis léxico se debe llevar el control de todos los tokens o palabras aceptadas que se encuentren en el código fuente. Al finalizar el análisis, el programa debe ser capaz de desplegar una lista con cada uno de estos tokens, el tipo de token que son y las líneas del código fuente donde se presentan y la cantidad de ocurrencias del token en cada línea. Los tokens que presentaron errores no deben ser listados. Se espera que esta lista sea lo más ordenada posible y que sea fácil de leer.

Ej:

Token	Tipo de Token	Línea
Hola	IDENTIFICADOR	5, 7, 50(2)
+	OPERADOR	6,8,15(3),36,45

Descripción Detallada

Se les sugiere tomar en cuenta los siguientes aspectos para asegurar la completitud del programa.

- El detalle de los tipos de Token será asignado por ustedes mismos. El tipo de Token no se refiere a los tokens que debe aceptar el programa sino que tipo de cada uno de estos puede diferir en cada tarea. Por ejemplo el token “+” puede ser de tipo “OPERADOR” o “OPERADOR ARITMETICO”. Sin embargo, deben haber al menos 6 grandes grupos de Tokens:

    - IDENTIFICADORES
    - OPERADORES
    - PALABRAS RESERVADAS
    - TRANSAC
    - UNIDADES
    - LITERALES

- El programa debe identificar los **comentarios** y omitir todos los tokens que se encuentran dentro de ellos. Se cuenta con dos tipos de comentarios. Los comentarios de línea inician con la sequencia ‘//‘

Los comentarios de bloque son de la siguiente forma: comienzan con /** y terminan con */ y en cada línea empieza con *. Son como los comentarios de JAVADOC. Los comentarios no deben venir en el listado de tokens que presentará el sistema.

- Los **identificadores** son palabras que representan constantes, variables, tipos de datos, funciones y algunos otros datos. Un identificador es una secuencia de caracteres, que inicia con una letra, no tienen espacios ni símbolos: &, !, *, etc. No tiene tildes y no es alguna palabra reservada. El lenguaje es CaseSensitive

- Las **palabras reservadas** a aceptar son las siguientes:

  address as bool break byte bytes constructor continue contract delete do else enum false for from function hex if import int internal mapping modifier payable Pragma private public return returns solidity string struct this true ufixed uint var view while

  NOTA IMPORTANTE: El lenguaje permite definir los tamaños de los enteros y bytes por lo que las palabras: uint8, uint32, int16, bytes4, bytes8, exc. También se toman en cuenta como palabras reservadas.

- Las palabras **TRANSAC** a aceptar son las siguientes:

  balance call callcode delegatecall send transfer

- Las palabras **UNITS** a aceptar son las siguientes:

  days ether finney hours minutes seconds szabo weeks wei years

- Los **literales** deben permitir número enteros, números flotantes, caracteres y strings.

  - Los enteros y de reales se representan con naturalidad: 5 67 58.9
  - Literales de fracciones decimales son formados por un `.` con al menos un número en un lado. Ejemplos incluyen `1.`, `.1` y `1.3`.
  - La notación científica está también soportada, donde la base puede tener fracciones, mientras que el exponente no puede. Ejemplos incluyen `2e10`, `-2e10`, `2e-10`, `2.5e1`.
  - Los Strings se representean entre comillas simples o dobles “ ” o ‘ ’.
  - Los strings literales soportan caracteres de escape, tales como `\n`, `\xNN` y `\uNNNN`. `\xNN`
  - Los literales hexadecimales son prefijos con la palabra clave `hex` y son cerrados por comillas simples o dobles (`hex"001122FF"`). Su contenido debe ser una cadena hexadecimal y no cualquier string
  - Literales octales no existen en Solidity y los ceros a la izquierda son inválidos.

- A continuación se detalla una lista con los **operadores** válidos en el lenguaje.

```
!       &&      ^       ==      !=      ||      <=      <       >=      >       &       |       ^
~       +       -       *       /       %       **      <<      >>      =       ,       ;       .
(       )       [       ]       ?       :       {       }       +=      -=      *=      /=
```

- Recuerden que esta es la primera etapa del proyecto del curso, por eso entre más detallado y funcional esté, más facilidad van a tener en el resto de etapas del proyecto

Documentación

Se espera que sea un documento donde especifique lo siguiente:

a. Portada, índice, introducción
b. Estrategia de Solución: Una explicación de como lograron solucionar el proyecto. Puede incluir consideraciones de diseño, lógica o cualquier aspecto que consideren importante para llegar a la solución.
c. Análisis de Resultados: Deberá elaborar un listado de todas y cada una de las actividades y tareas que deben cubrirse a nivel funcional, para cada una de ellas debe aportar el porcentaje de realización y en caso de no ser el 100% debe justificarse.
d. Lecciones aprendidas: Debe prepararse un listado de las lecciones aprendidas producto del desarrollo de la tarea programada. Las lecciones aprendidas pueden ser de carácter personal y/o técnico que involucre aspectos que han logrado un aprendizaje en temas de investigación, desarrollo de habilidades técnicas y habilidades blandas como trabajo en equipo, comunicación, forma de expresar ideas, entre otros.
e. Casos de pruebas: se espera que definan claramente cada prueba, cuáles son los resultados esperados y cuáles fueron los resultados obtenidos. No es necesario que sean grandes pero deben evaluar la funcionalidad completa del programa.
f. Manual de usuario: especificar como compilar y correr su tarea.
g. Bitácora de trabajo durante las semanas de trabajo, incluyendo verificaciones realizadas (si existieran) de consultas realizadas con el profesor o asistente.
h. Bibliografía y fuentes digitales utilizadas

Aspectos Administrativos

- El desarrollo de este programa debe de realizarse en grupos de exactamente tres personas salvo acuerdo con el profesor. Los grupos de trabajo deben permanecer iguales para los siguientes proyectos del curso.
- El trabajo se debe de entregar el día 01 de junio de 2020 antes de media noche, enviarlo por correo o subirlo al tec digital.
- Deben entregar el código fuente junto con el ejecutable y la documentación.
- Referencias:
  - http://www.jflex.de/index.html
  - https://solidity.readthedocs.io/en/v0.6.6/style-guide.html
  - https://solidity-es.readthedocs.io/es/latest/solidity-in-depth.html