Instituto Tecnológico de Costa Rica

Escuela de Computación

IC-5701

Compiladores e Intérpretes

I Proyecto Programado
Scanner

Estudiantes

Melisa Cordero Arias 2016126133
Yanil Gómez Navarro 201281509

Profesora

Erika Marín Schumann

II Semestre 2018

Índice

Introducción

El propósito de este proyecto es crear la fase del Análisis Léxico de un compilador que lea código ABC. Para facilitar la creación de este “scanner” se utiliza la herramienta JFLEX, esta es una biblioteca del lenguaje de programación Java que a través de expresiones regulares reconoce los distintos tokens que se reciben. Un beneficio de la utilización de esta biblioteca es que se aplica el conocimiento adquirido sobre lenguajes regulares y expresiones regulares.

Este proyecto es la primera etapa de todos los proyectos que se van a realizar en el curso de Compiladores e Intérpretes en el segundo semestre del año 2018. Por esta razón es un proyecto de gran importancia debido a que cualquier error o alguna funcionalidad faltante puede ocasionar repercusiones en los siguientes proyectos, los cuales son las siguientes fases del compilador. Con esto en mente se intenta realizar el scanner de una manera suficientemente específica para facilitar el trabajo de la siguiente fase, el análisis sintáctico o “parser”.

Estrategia de solución

La primera actividad que se realizó para la solución del proyecto fue leer la especificación y detectar algunos puntos clave para crear el analizador léxico. Uno de los puntos clave, incluso se podría decir que el más importante, fue que era requerida la utilización de la biblioteca jflex en Java. Por esa razón la siguiente actividad necesaria fue la investigación sobre esta herramienta. Durante esa investigación se encontró material muy diverso, dentro de este algunos ejemplos. Para entender mejor la manera en que trabajaba la herramienta, se escogió el ejemplo más similar al proyecto programado y se siguió el tutorial de dicho ejemplo.

Luego de haber programado el ejemplo de la herramienta jflex (el cual también era un scanner de algunas palabras) se prosiguió por cambiar los tokens que reconocía este “scanner” por algunas palabras de la especificación del proyecto como las palabras reservadas. Lo siguiente por definir era de qué manera se iban a guardar y mostrar las dos salidas necesarias. Se decidió solo guardar la tabla de símbolos e imprimir los errores en el momento en el que sucedían debido a que estos no eran necesarios al final de la ejecución. Para la tabla de símbolos se utilizó un Hashmap debido a que la estructura de llave valor facilitaba las búsquedas, también se creó un objeto Token para guardar los distintos tokens encontrados en el programa.

Cuando la estructura del programa fue definida y acordada entre los dos miembros del equipo, se siguió la programación normal del scanner agregando los distintos tokens que hacían falta. Finalmente, cuando el scanner logró reconocer cualquier programa léxicamente correcto se prosiguió por la detección de los distintos errores que podían existir.

Análisis de Resultados

Se detalla la clasificación utilizada para los tokens

- ➢ Palabras Reservadas
- ➢ Identificadores
- ➢ Operadores
- ➢ Literales:
  - Enteros
  - Flotantes
  - Strings
  - Caracteres

Funcionalidad Detallada

Las funcionalidades con las que debía cumplir el analizador léxico y su porcentaje de logro se resumen de la siguiente manera:

Función	Porcentaje de logro
1. Impresión de errores encontrados.	100%
2. Listado de tokens encontrados con todas sus ocurrencias.	100%
3. No distinguir entre minúsculas y mayúsculas.	100%
4. Identificar comentarios.	100%
5. Identificar identificadores.	100%
6. Identificar palabras reservadas.	100%

7. Identificar literales.	100%
8. Identificar operadores.	100%
9. Detectar errores.	100%

Casos de pruebas

➔ Prueba de palabras reservadas

Las palabras reservadas que se espera que acepte son:

```
AND, ARRAY, BEGIN, BOOLEAN, BYTE, CASE, CHAR, CONST, DIV, DO,
DOWNTO, ELSE, END, FALSE, FILE, FOR, FORWARD, FUNCTION, GOTO,
IF, IN, INLINE, INT, LABEL, LONGINT, MOD, NIL, NOT, OF, OR,
PACKED, PROCEDURE, PROGRAM, READ, REAL, RECORD, REPEAT, SET,
SHORTINT, STRING, THEN, TO, TRUE, TYPE, UNTIL, VAR, WHILE, WITH,
WRITE y XOR.
```

Todas estas escritas con letras mayúsculas o minúsculas son reconocidas correctamente por el scanner.

➔ Prueba de comentarios

Se espera que se ignoren los comentarios de línea que se representan con //, y los comentarios de bloque que se delimitan con (* *) o { }.

- `// Comentarios`
- `(*Este es un comentario`
  `para probar*)`
- `{Este comentario`
  `de prueba}`

Todas las palabras escribas dentro de un comentario son ignoradas correctamente.

➔ Prueba de identificadores.

Los tokens que se consideran identificadores se espera que cumplan con las siguientes reglas:

1. No ser una palabra reservada o un operador.
2. Iniciar con una letra y solo contener letras o símbolos en los siguientes caracteres.
3. Tener una longitud menor a 127.

- `CASA  IDENTIFICADOR  3`
- `SUMA  IDENTIFICADOR  4`
- `Comida,Comida,ComIda,COMida,COMIDA`
  `IDENTIFICADOR  5(5)`
- `CA34SA  IDENTIFICADOR  6`

Todos los tokens que cumplen con esas reglas, son reconocidos correctamente como identificadores.

- `[Línea: 11]=>casa!:` *`Identificador erróneo: no se puede utilizar los caracter !&# en los identificadores.`*
- `[Línea:11]=>esteesunidentificadorextremadamentegrandequeocupatodoeltamannopermitidoporlaespecificaciondellenguajeABCyquedebeservalidadoporelescaneresteesunidentificadorextremadamentegrandequeocupatodoeltamannopermitidoporlaespecificaciondellenguajeABCyquedebeservalidadoporelescaner:` *`Identificador erróneo, no puede ser mayor a 127 caracteres`*

Estos tokens no cumplen con las reglas para identificadores, por lo que dan error.

➔ Prueba de operadores

Se espera que los siguientes tokens sean reconocidos como operadores:

```
","  ";"  "++" "--" ">=" ">"  "<=" "<"  "<>" "="  "+"  "-"  "*"
"/"  "("  ")"  "["  "]"  ":=" "."  ":"  "+=" "-=" "*=" "/=" ">>"
"<<" "<<="     ">>="     NOT  OR   AND  XOR  DIV  MOD
```

Todos los tokens que coinciden con esos caracteres son reconocidos correctamente como operadores.

➔ Prueba de literales

Se espera que se puedan reconocer los números enteros, números flotantes, cadenas de caracteres y caracteres.

- ➔ Los números enteros son cualquier secuencia de dígitos sin parte flotante.
- ➔ Los números flotantes son aquellos números con parte flotante. Dentro de este tipo también están los números escritos en notación científica.
- ➔ Las cadenas de caracteres son varios caracteres encerrados en “”.
- ➔ Los caracteres son representados por un # y el código ascii.

```
● 44              INTEGER   36
● 67              INTEGER   47
● 12345E+123      FLOAT     46
● 3423E123        FLOAT     42
● 12345123        INTEGER   38
● 1234567890      INTEGER   37
● "ARBOL"         STRING    26
● 32              INTEGER   16
● 123.4556        FLOAT     40
● 1234E-123       FLOAT     44
● "BUENO
  MALO"           STRING    29
● #67             CHAR      32
```

Todos estos tipos son reconocido correctamente, también se clasifican de acuerdo a la categoría a la que pertenecen.

- [Línea: 55]=>.5: Número erróneo: no se puede iniciar un número con punto
- [Línea: 56]=>5.: Número erróneo: no se puede finalizar un número con punto
- [Línea: 57]=>58yui: Número erróneo: no se puede ingresar letras dentro de un número.

Los errores de escritura de estos números como lo pueden ser “.5”, “2.”, “2E3”, son detectados correctamente.

Lecciones Aprendidas

1. Siempre leer la especificación detenidamente, incluso las ultimas partes. En este proyecto yo (Melisa) no leí toda la especificación antes de empezar a investigar por lo que no vi la referencia de la herramienta JFLEX. Por esa razón tuve que ver muchos tutoriales para poder bajar y conectar la herramienta a netbeans.
2. Al iniciar un proyecto es importante pensar las distintas clases que se van a utilizar, sus atributos y sus métodos con los otros integrantes del grupo. Esto si lo realizamos pero fue muy útil después porque las dos sabíamos como funcionaba toda la estructura del programa.
3. Aunque algo ya funcione siempre hay que seguir probándolo. En muchas ocasiones durante el desarrollo del analizador léxico algunos tipos de tokens que ya se reconocían correctamente dejaron de funcionar. Esto debido a que otras expresiones regulares de otros tokens los identificaban y por el comportamiento del jflex se les asignaba tipos que no eran los correctos. Por esa razón debimos realizar muchas pruebas para estar seguras que el código nuevo no causaba problemas en el código viejo.

Manual de usuario

Los resultados de este proyecto se imprimen en consola, por lo que se requiere de un archivo *.txt* para incorporar los casos de prueba a ejecutar.
El archivo de prueba debe colocarse dentro de la carpeta de *CompiladorABC*; además se debe verificar que el nombre concuerde con la referencia del *.txt* en el archivo **CompiladorABC.java**, localizado en las carpetas: *CompiladorABC\src\compiladorabc.*

Además debe indicar la ubicación en donde se guardará el archivo *.flex*, dado que es necesario para generar el scanner. Dicho archivo se encuentra en las carpetas: *CompiladorABC//src//Scanner//Lexer.flex;* pero de igual manera es necesario completar con la ubicación que tiene en su computador.

Al tener todos los archivos correctamente especificados y ubicados, puede proceder a ejecutar el programa. Como resultado se mostrará una tabla con tres columnas, la primera “Token” se muestran los token encontrados en el archivo *.txt,* la segunda “Tipo token” indica el tipo de token de acuerdo al token encontrado y la tercera

“Repeticiones” contiene la línea en que fue encontrado el token, y las repeticiones si fue más de una vez.

TOKEN	Tipo del Token	Repeticiones
44	INTEGER	39
"B"	CHAR	2
DF	IDENTIFICADOR	16(2), 17
RT	IDENTIFICADOR	16
12345E+123	FLOAT	50
CA34SA	IDENTIFICADOR	7
3423E123	FLOAT	46

Por último al finalizar la tabla se muestran los errores encontrados en el archivo *.txt* , indicando la línea del error, seguida por la palabra y la indicación del porqué es un error.

```
[Línea: 11]=>esteesunidentificadorextremadamentegrandequeocupatodoeltamannopermitidoporlaespecificaciondeller
[Línea: 15]=>ca*23: Identificador erróneo: no se puede utilizar los caracter !&# en los identificadores.
[Línea: 54]=>.5: Número erróneo: no se puede iniciar un número con punto
[Línea: 55]=>5.: Número erróneo: no se puede finalizar un número con punto
[Línea: 57]=>58yui: Número erróneo: no se puede ingresar letras dentro de un número.
```

Bitácora de trabajo

Jueves 6 de setiembre
2pm - 5pm. Investigación sobre jflex. Programación de ejemplo base del scanner.

Viernes 7 de setiembre
6pm - 9pm. Investigación sobre la herramienta Jflex

Domingo 9 de setiembre
1pm - 3pm. Digitalización de operadores y Strings

Lunes 10 de setiembre
12md - 3pm. Programación de distintas clases y funciones.
6pm - 11pm. Manejo de Strings, caracteres, comentarios y creación de la clase error para mejor manejo.

Viernes 14 de setiembre

6am - 9am. Solución de errores de números
7pm - 9pm. Inicio de documentación.

Sábado 15 de setiembre
1pm - 4pm. Solución de errores encontrados

Domingo 16 de setiembre
4pm -6pm. Pruebas al scanner. Solución de errores. Escritura de casos de prueba de la documentación.
3pm-9pm Pruebas:Solución de errores. Documentación.

Martes 19 de setiembre
6pm - 9pm. Detalles finales en el código y la documentación.

Bibliografía

[HolaCodigo]. (2012, Octubre 2). Analizador Lexico con JFlex en Java (NetBeans). Recuperado de: https://www.youtube.com/watch?v=w-KfjJdRas8&t=29s

JFLEX. (s.f.). *JFLEX*, 1.6.1. Recuperado el 07 de Setiembre de 2018, de http://jflex.de/manual.html