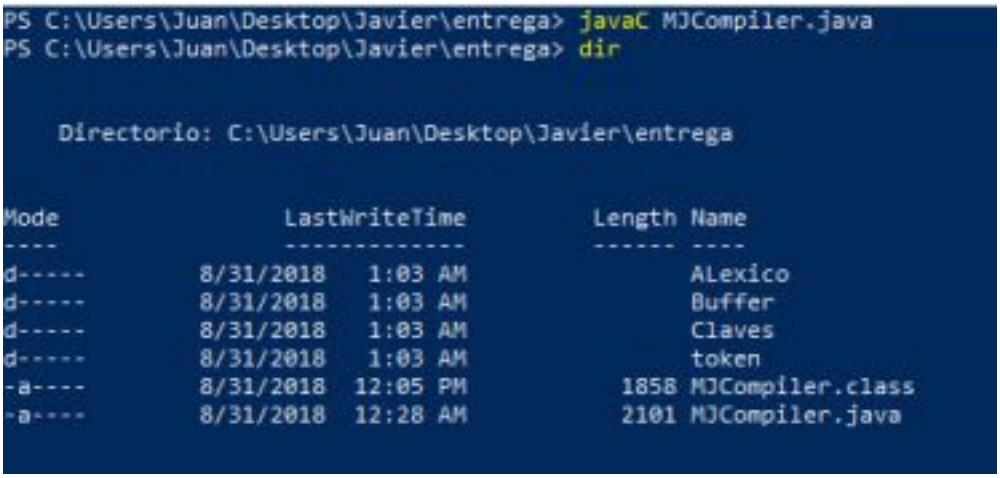
Proyecto : Compiladores e intérpretes

Etapa 1 : Analizador Léxico.
Alumno : Javier Amorosi.
LU: 94528.

Índice :

Compilación del código fuente : La compilación del código fuente debe ser realizada en la carpeta donde se encuentra el archivo con la clase Main de la siguiente manera.



Como podemos ver la invocación del comando javac dio como resultado la conversión de los archivos .java en archivos .class que pueden ser ejecutados por consola con el comando java.

*aclaración: La main Class no es MJCompiler sino MiniJavaCompiler quedando el comando de la siguiente manera a la hora de compilar : javaC MiniJavaCompiler.java

Alfabeto de entrada MiniJava :

El alfabeto de entrada es todo el abacedario ascii. Básicamente hay que tener en cuenta que incluso hasta los simbólos que en un determinado contexto son errores léxicos, como $, forman parte del alfabeto de entrada ya que pueden ser usados n una cadena o formar parte de los comentarios. **Tokens reconocidos por el Analizador Léxico:**

A continuación listamos los Tokens Reconocidos por el análizador Lexico y sus respectivas E/R:

Enteros:

alfabeto:[0..9] todos los dígitos de la base 10

Entero : c+ donde + se interpreta como 1 o más repeticiones de c, y donde c corresponde a un carácter del alfabeto.

El Token Leído es de tipo NUM donde en el lexema guarda el entero leído.

Caracteres:

carácter : '|(a|b|...|z|..%)|(\(a|b...|z))'.

Aclaración : la cadena de caracteras primera no incluye '.

El Token Leído es de tipo Char y el lexema guarda el valor del carácter leído. Excepción: el carácter '\\' tendra por lexema \.

Identificadores:

idClase: A|B|..|Z(A|B...|Z|a|b..|z|0..|9|_)*

La ER se interpreta como un Caracter en Mayúscula seguido de un carácter dígito o underscores.

EL Token reconocido es de tipo idClase y el lexema guarda los valores de los caracteres leídos.

idMetVar: a|b|..|z(A|B...|Z|a|b..|z|0..|9|_)*

La ER se interpreta como un Caracter en minúscula seguido de un carácter dígito o underscores.

EL Token reconocido es de tipo idMetVar y el lexema guarda los valores de los caracteres leídos.

Para las palabras reservadas se usan las mismas ER que los identificadores de metodos y variables, al final son mapeadas y si corresponden a una clase en particular el tipo es modificado por el tipo correspondiente. Incluye a booleanos como true y false, null,etc.

Strings:

string: “((a|b....|%|&|...0...|9)Char)”

Char: (a|b....|%|&|...0...|9)Char|ϵ

comillas dobles seguidas de una secuencia de carácteres o dígitos y finaliza con comillas dobles.

El Token reconocido es de tipo String y el lexema será la secuencia de caracteres dentro de las comillas dobles.

Operadores:

Mayor,Mayor e Igual: >|>=

El token reconocido es de tipo MAY o MAYIGUAL y sus respectivos lexemas son las cadenas que forma la ER.

Menor,Menor e Igual: >|>=

El token reconocido es de tipo MEN o MENIGUAL y sus respectivos lexemas son las cadenas que forma la ER.

Igual,Igual e Igual: =|==

El token reconocido es de tipo IGUAL o igualGUAL y sus respectivos lexemas son las cadenas que forma la ER.

NOT,DISTINTO: !|!=

El token reconocido es de tipo MEN o MENIGUAL y sus respectivos lexemas son las cadenas que forma la ER.

Para los operadores &&,/,*,+,-,||, Las expresiones regulares están compuestas de esas cadenas en sí y representan un caso trivial. El Token reconocido por la ER : && es de tipo AND y tiene por lexema &&. Esto es ánalogo también para las ER que forman la puntuación.

Errores Léxicos detectados :

1. Comentarios sin cerrar : Ejemplo : /* este es un comentario sin cerrar
2. caracteres invalidos : ejemplo $.
3. enteros mal formados : ejemplo 101hola.
4. Caracteres sin cerrar ‘askdjaslfkdj…
5. Caracteres con más de un dígito ‘hola’.
6. Cadenas sin cerrar.

Logros a alcanzar:

Los logros a alcanzar esperados para esta etapa son: Multi-detección de errores léxicos.

Casos de prueba : Los casos de prueba se encuentran en la carpeta casos de prueba, contiene los correspondientes casos de prueba con los resultados esperados.

Decisiones de Diseño :

Durante el transcurso del desarrollo del analizador léxico surgieron varios puntos donde se tuvieron que tomar decisiones de diseño. La primer decisión de diseño importante fue en cuanto a la arquitectura del sistema, donde por modularidad se opto por dividir la clase que consume del buffer, de la clase que efectúa el análisis léxico. Como así también contar con una clase que se encargara de proveer los mapeos y otros servicios a la clase analizador lexico, con la idea de hacer esté código más legible. Otra decisión muy importante fue la de implementar los autómatas como un switch, para hacer el software más óptimo, bajo esta misma lógica se implementaron mapeos con tablas de hash para los estados validos, etc. También se decidió reconocer los enteros mal formados como error léxico, ya que su detección en etapas posteriores sería más costosa. Los tipos de los Token se representan no como enteros sino como enumerados, ya que esto permite una comparación más ágil.

Clases utilizadas: Se procede a especificar los paquetes y dentro de cada paquete las clases utilizadas.

- claves : Dentro de este paquete se usaron las clases de ClavesServices que mantiene un mapeo con las palabras claves y sus tokens correspondientes; también se implemento notFoundClaveException en el caso de que una clave no esté en el mapeo.
- Buffer : En este paquete se implemento el buffer, encargado de leer en el archivo especificado por el usuario en la consola.
- Lexico : Se implementó la clase Alexico que es la encargada del analisis léxico.
- Token : en este paquete se implementa la clase Token y la excepción TokenException.
- MiniJavaCompiler: clase con el método main que implementa el compilador.