mEmoria practicas teoria de la computacion

Pablo Villa Camañes 874773

\*\*Práctica 2: Desarrollo del Analizador Léxico\*\*

En esta práctica, nos enfocaremos en desarrollar el analizador léxico para un lenguaje específico. Las siguientes etapas del análisis, sintáctico y semántico, se abordarán en prácticas posteriores. El objetivo actual es identificar todos los tokens presentes en el lenguaje y los lexemas asociados a cada token. El funcionamiento del analizador consistirá en mostrar por pantalla los lexemas y los tokens correspondientes. Cualquier cadena que no pueda identificarse como un token generará un mensaje de error, indicando el texto, la línea y la columna donde se encontró.

//Tokens Clave//

Dentro de nuestro archivo de texto, nos encontraremos con términos cruciales y operadores fundamentales, tales como:

1. lista espacios 🡪 Token: LISTA ESPACIOS
2. lista tipos 🡪 Token: LISTA TIPOS
3. lista eventos 🡪 Token: LISTA EVENTOS
4. == 🡪 Token: IGUAL
5. , 🡪 Token: COMA
6. ; 🡪 Token: PUNTO COMA
7. { 🡪 Token: LLAVE ABRIR
8. } 🡪 Token: LLAVE CERRAR

Cada uno de estos tokens representa un token diferente, pero son nuestros tokens claves o nuestros operadores claves.

//Tokens Específicos://

1. Nombre Espacios:

- Comienza con una mayúscula, seguida de minúsculas y posiblemente acentuación.

- Puede contener números (0-9) seguidos de "\_".

- Posibilidad de concatenación con otra cadena de letras o números a través de la clausura.

2. Nombre Tipos:

- Comienza con mayúscula y posible acentuación.

- Seguido de letras minúsculas, mayúsculas o "\_".

- Cerrando con una clausura para leer múltiples caracteres.

3. Nombre Eventos:

- Comienza con mayúscula y posible acentuación.

- Seguido de letras minúsculas o mayúsculas.

- Cerrando con una clausura para leer múltiples caracteres.

4. Fecha:

- Día: Puede ser 01-31. Siendo válido (0 seguido de 1-9, 1 seguido de 0-9,

2 seguido de 0-9, 3 seguido de 0-1).

- Mes: Puede ser 01-12. Siendo válido (0 seguido de 1-9, 1 seguido de 0-2)

- Año: Números del 0 al 9 con clausura para leer los 3 siguientes dígitos.

- Separados por "/".

5. Hora:

* Minutos: (00:59) Siendo válido (0 seguido 0-9, 1 seguido 0-9, 2 seguido 0-9 3 seguido 0-9, 4 seguido 0-9, 5 seguido 0-9)
* Hora: (00:23) Siendo válido (0 seguido 0-9, 1 seguido 0-9, seguido 2 0-3)

6. Errores

\r|\n|\s {} sirve para que reconozca los saltos de línea y los espacios y no los de como error

. sirve para lanzar un mensaje de error mostrando el carácter no valido que se ha introducido que no se ha declarado como un token especifico

\*\* Practica 4 Desarrollo Analizador Sintáctico \*\*

Dado el analizador léxico creado en la práctica dos necesitamos crear un analizador sintáctico con la herramienta Cup que reconozca el fichero de la practica 2.

Para el fichero cup podemos hacernos referencia de nuestro fichero cup de la practica 3 de la calculadora el cup se basa en crear una gramática con unas reglas básicas que reconozca el fichero de prueba.

Tendremos que inicializar los tokens terminales como la coma punto y coma las llaves los nombres de los espacios de tipos de eventos.

Y los no terminales que serán los que en la gramática están a la izquierda como el símbolo inicial s los espacios, tipos, eventos que serán una lista con comas de los diferentes espacios tipos y eventos.

El símbolo inicial s tendrá las llaves de apertura con las lista espacios la llave de cerrar la llave de apertura con la lista tipos la llave de cerrar y la lista eventos.

Después tendremos que compilarlo con el cup sacando nuestros ficheros sym y parser

El. Flex se modificará y cambiaremos el reconocimiento de tokens por sym. Y el nombre

del token correspondiente y compilarlo en nuestro Jflex para sacar nuestro .java

El java sym y parser se pasará a nuestro proyecto de NetBeans con nuestra clase principal sacada de Moodle que se compilará adjuntando nuestro texto para que sea reconocido por nuestro Flex y cup.

\*\* Practica 5 Análisis Sintáctico y Semántico Avanzado\*\*

El Análisis sintáctico y semántico es valido pero necesitamos que nuestro texto tenga coherencia como por ejemplo que las fechas, horas sean validas es decir que sea como máximo 31 días 12 meses y las horas mayores que 00 y menor que 23 Y los minutos a partir de 00 y menos que 59.

Para captar errores se ha usado la clausula error en la línea de S si no capta lo dado se captará un error.

Los espacios y los tipos no pueden estar repetidas se almacenarán en un array de strings de espacios y tipos cada vez q se introduzca un espacio o tipo se comprobará con el array de su tipo

Los terminales espacios tipos eventos fecha y hora deberán ser de tipo string

Además, Para los eventos la fecha de fin no podrá ser anterior a la de inicio por lo que con un try and catch comprobaremos las dos fechas para que sea valida

Por último, Comprobaremos con una función booleana que los eventos no se solapan

BNF Gramática

<S >: = LIST\_ESPACIO LLAVE\_APR ESPACIOS LLAVE\_FIN | LIST\_TIPOS LLAVE\_APR TIPOS LLAVE\_FIN | LIST\_EVENTOS LLAVE\_APR EVENTOS LLAVE\_FIN|error;

<ESPACIOS>:: = NOM\_ESPACIOS: n\_espacio

{: insertar\_espacio(n\_espacio):}

ESPACIOS | NOM\_ESPACIOS: n\_espacio

{: insertar\_espacio(n\_espacio) :}

COMA ESPACIOS | ε;

<TIPOS> ::= NOM\_TIPOS: n\_tipo

{: insertar\_tipo(n\_tipo) :}

TIPOS | NOM\_TIPOS COMA TIPOS | ε ;

<EVENTOS> ::= NOM\_EVENTOS IGUAL NOM\_TIPOS: n\_tipo

{:

if(!L\_tipos.contains(n\_tipo)){

System.out.println("error nombre no reconocido");

}

:}

IGUAL NOM\_ESPACIOS:n\_espacios

{:

if(!L\_espacios.contains(n\_espacios)){

System.out.println("error nombre no reconocido");

}

:}

IGUAL FECHA: idInicio HORA : idHoraInicio IGUAL FECHA:idFin HORA:idHoraFin PUNTOCOMA EVENTOS | ;

{:

try {

SimpleDateFormat fecha = new SimpleDateFormat("dd/mm/aaaa hh:nn");

Date inicio = fecha.parse(idInicio + " " + idHoraInicio);

Date fin = fecha.parse(idFin + " " + idHoraFin);

if(fin.before(inicio)){

System.out.println("error fecha no valida);

} else{

System.out.println("fecha valida");

}

if(reservas\_solapadas(n\_espacios,inicio,fin){

system.out.println("Error las reservas estan solapadas");

}

else{

Evento v1 = new Evento(n\_espacios,inicio,fin);

Eventos.add(v1);

}

} catch(ParseException e) {

System.out.println("Error mal ");

}

:}