Analizador Léxico

Compiladores

Hernández Gómez Ricardo

Profesora: Sandoval Montaño Laura

Grupo: 2

Fecha: 24/09/2018

Descripción

Analizador léxico hecho en Flex, diseñado para reconocer los componentes de un archivo de entrada. El lenguaje está definido por 8 clases de componentes léxicos distintos: palabras reservadas, identificadores, símbolos especiales, operador de asignación, operadores relacionales, operadores aritméticos, constantes cadena, constantes numéricas enteras, y constantes numéricas reales. Todos los elementos no reconocidos como de alguna de estas clases, se identifica como error.

Debido a que las palabras reservadas y los operadores relacionales están definidos por defecto, estos se encuentran ubicados en sus correspondientes archivos, desde los cuales se leen antes del análisis.

Cada vez que un componente léxico es reconocido se añade a un arreglo, la posición que pasa a ocupar correspondiendo al valor que tomará para la generación de tokens. Sin embargo, antes de añadirse, se comprueba, mediante una búsqueda lineal, que el componente no se encuentre presente ya.

Una vez leído todo el archivo se procede a imprimir el contenido de los arreglos en terminal para que el usuario compruebe el trabajo realizado, así como a generar archivos individuales para cada tabla.

Características de los componentes:

- *Palabras reservadas:*
  - Empiezan con una letra mayúscula.
  - Están definidas por defecto en un archivo.
- *Identificadores:*
  - Letra inicial minúscula.
  - Tamaño máximo de 8 letras.
  - Formados sólo por letras.
- *Símbolos especiales:*
  - Los símbolos especiales son: ( ) , ; [ ]
  - Se almacenan tal y como se reconocen en la tabla de tokens.
- *Operador de asignación:*
  - El operador es: :=
  - El valor con el que se guarda como token es: =
- *Operadores relacionales:*
  - Inician con un punto: .
  - Terminan con un punto: .
  - Letras intermedias mayúsculas.
  - Definidas por defecto desde un archivo.
- *Operadores aritméticos:*

- Los operadores son: + - * / %
- Se almacena tal y como se reconocen en la tabla de tokens.

❖ *Constante cadena:*
- Definidas por dobles comillas al inicio y al final: “ “
- Se almacenas con comillas.

❖ *Constantes numéricas enteras:*
- Números en base 10.
- Cualquier combinación de números del 0 al 9.
- Se almacenan en una tabla.

❖ *Constantes numéricas reales:*
- Números con decimales.
- Números en notación científica.
- Se almacenan en una tabla.
- La notación científica puede incluir signo o no.
- La notación científica está definida por una “e” que puede ser mayúscula o minúscula.

Todo símbolo no identificado como componente léxico se toma como error. Se considera como error todo lo marcado por la siguiente expresión:

```
[^ \t(\r)(\n)(\r\n)]
```

Esto con el fin de tomar en cuenta los finales de línea distintos existentes como producto de la edición de texto en los 3 principales tipos de sistemas operativos existentes: Linux, Windows y Mac.

Expresiones regulares:

<letMin> = a-z

<letMay> = A-Z

<dig> = 0-9

❖ Palabras reservadas:

<letMay><letMin>*

❖ Identificadores:

<letMin>(<letMin>|<letMay>)*

- Símbolos especiales:

(|) | , | ; | [|]

- Operador de asignación:

:=

- Operadores relacionales:

. <letMay><letMay>(<letMay>|ε) .

- Operadores aritméticos:

+ | * | / | % | \ | -

- Constante cadena:

“ (<letMay>|<letMin>|<dig>|)* “

- Constantes numéricas enteras:

<dig>+

- Constantes numéricas reales:

<dig>*(.<dig>*|(.| ε)<dig>*(E|e)(+|-| ε)<dig>+)

Análisis

- *Planificación:*

Actividad	Elabora
Expresiones regulares	Ricardo Hernández Gómez
Implementación de expresiones regulares en Flex	Ricardo Hernández Gómez
Programación en C	Ricardo Hernández Gómez
Pruebas	Ricardo Hernández Gómez

- *Diseño:*
    - Tablas de palabras reservadas y operadores relacionales:

Las palabras reservadas y lo operadores relacionales están definidos por defecto en archivos .txt ubicados en el mismo directorio que el programa. Las palabras están definidas con el siguiente formato.

Palabra1

Palabra2

Palabra3

.

.

.

La función “populateArray(char *fileName, int *numero)” se ejecuta para ambos, insertándose en sus respectivos arreglos conforme se va leyendo, línea por línea. Debido a esto es posible omitir el número de su valor del archivo, pues la posición en la que son insertados en el arreglo de palabras reservadas es este.

- Tabla de símbolos

Se genera una tabla de símbolos con el siguiente formato:

Posición Nombre Tipo

La columna Tipo se mantiene vacía.

- Otras tablas para comprobación del análisis:

Se generan tablas para las cadenas, enteros y reales encontrados durante el análisis y almacenados en arreglos. Cada uno se guarda con el siguiente formato:

"%d\t%s" valor, componente

- Técnica de búsqueda:

Antes de insertar cualquier elemento en su correspondiente arreglo de clase, se debe comprobar que este no existe dentro de este aún. Para esto se implementó una búsqueda lineal en la función "int busquedaLineal(char **tabla, char *objetivo, int tamanio)".

Se recorre el arreglo con un ciclo "for" y se realiza la comparación con la correspondiente cadena que se está buscando. Si se encuentra se regresa el índice en el arreglo, si no, se regresa el valor de -1 para indicarlo.

- Inserción en arreglos de cada clase:

Para optimizar el uso de memoria, cada arreglo va a reservando memoria conforme la va requiriendo gracias a "realloc". Debido a que se trata de arreglos de cadenas, se agrega espacio con "realloc" y se procede a asignar espacio para la cadena que se va a almacenar en el índice en cuestión con "malloc".

❖ *Implementación:*

La implementación se realizará utilizando Flex, así como el lenguaje de programación C. El código elaborado se puede revisar en el archivo: "analizadorLexicoHGR.l".

La implementación se realizó utilizando la distribución de Linux, Ubuntu, y la compilación se realizó utilizando gcc.

Instrucciones de ejecución

Para para poder ejecutar el programa es necesario generar el código en C y compilarlo desde Linux utilizando el compilador GCC.

Comenzamos por generar el código en archivo .C utilizando Flex:

```
$ flex analizadorLexicoHGR.l
```

Esto generará el archivo .C con el nombre “lex.yy.c”. Procedemos a compilarlo:

```
$ gcc lex.yy.c -lfl -o analizadorLexico
```

Una vez hecho esto podemos ejecutar el programa. Para su uso es necesario indicar el nombre del archivo a analizar inmediatamente después del programa.

```
$ ./analizadorLexico testfile.txt
```

El programa se ejecutará e imprimirá las tablas en pantalla, además de guardarlas como archivo de texto en la misma carpeta para su revisión.

Conclusiones

Un analizador léxico, como el primer tipo de análisis que se realiza en un proceso de compilación, tiene la responsabilidad de recorrer, carácter por carácter, todo el programa fuente de entrada, identificando cada componente léxico de acuerdo con las expresiones regulares definidas para cada uno, distribuyéndolos en tablas y guardándolos como tokens, indicando su clase y valor.

Con este proyecto de elaboración de un analizador léxico con Flex y C, pude reafirmar lo aprendido durante la clase de teoría, identificando claramente la necesidad de esta etapa de análisis, así como cada una de sus partes para su correcta implementación.

Después de extensas pruebas puede decir que el objetivo se cumplió, satisfaciéndose las necesidades de salida de la etapa para el analizador sintáctico que se necesitará implementar a continuación.