	Carátula para entrega de prácticas	
Facultad de Ingeniería		Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): René Adrián Dávila Pérez

Asignatura: Compiladores

Grupo: 4

Integrante(s): 315118894

317099681

315081143

313044270

Semestre: 2024-1

Fecha de entrega: 05/10/23

Observaciones:

CALIFICACIÓN: _____

Índice

Contenido

Introducción

Un analizador léxico o scanner, es un programa que analiza el texto fuente de un programa informático y lo divide en unidades más pequeñas, llamadas tokens. Estas unidades son los elementos básicos de un lenguaje de programación, como palabras reservadas, identificadores, operadores, literales, etc.

El analizador léxico es la primera etapa en el proceso de compilación de un programa informático. Su función es identificar los tokens que forman el código fuente y generar un flujo de tokens que será utilizado por el siguiente paso en el proceso de compilación, el analizador sintáctico.

Un scanner se puede desarrollar con distintos métodos:

- Con un autómata finito
- Con un programa a medida
- Utilizando una herramienta específica como Flex

En este trabajo se expondrá un analizador léxico (scanner) que procesa un lenguaje llamado “Tu_P_L”, un lenguaje en español. Para este proyecto se hará uso de la herramienta “FLEX”, ya que hacer un autómata finito es una tarea increíblemente laboriosa y FLEX proporciona las herramientas necesarias para construir de forma sencilla el analizador léxico.

Hipótesis

La implementación de un analizador léxico para el lenguaje específico propuesto permitirá analizar y dividir de manera efectiva y precisa el código fuente escrito en este lenguaje en una secuencia de tokens, que incluirán palabras reservadas, operadores aritméticos y relacionales, asignaciones, delimitadores, espacios, variables, cadenas, números y funciones, según las reglas de la gramática definida para este lenguaje.

Objetivos y Expectativas del Analizador Léxico:

1. **Reconocimiento de Palabras Reservadas:** El analizador léxico deberá identificar y clasificar correctamente las palabras reservadas, como "entero," "buleano," "flotante," "SI," "EntonCes," "Para," y "MiEntras." Estas palabras reservadas serán tokens fundamentales para definir la estructura y el flujo del programa.
2. **Identificación de Operadores Aritméticos y Relacionales:** El analizador léxico deberá detectar los operadores aritméticos ("+", "*", "-", "/") y los operadores relacionales ("<", ">", "=3", "E=)", "W!") cuando aparezcan en el código fuente.
3. **Manejo de Asignaciones:** El analizador léxico deberá ser capaz de reconocer y clasificar correctamente las asignaciones representadas por "<=".
4. **Delimitadores y Espacios:** Deberá identificar y gestionar adecuadamente los delimitadores ("UwU," "(", ")") y los espacios en blanco ("\t," "\n," " ") que puedan aparecer en el código
5. **Identificación de Variables:** El analizador léxico deberá reconocer las variables representadas por el patrón ".|." y asignarles un token correspondiente.
6. **Reconocimiento de Cadenas:** Deberá identificar y clasificar las cadenas de caracteres, marcadas por "CaD," como tokens válidos.
7. **Detección de Números:** El analizador léxico deberá identificar secuencias de dígitos ([0-9]+) y considerarlas como tokens de números válidos.
8. **Identificación de Funciones:** Deberá reconocer las funciones definidas por ":-*" y generar tokens correspondientes para representarlas en el análisis posterior.

Se espera que el analizador léxico funcione de manera eficiente incluso para programas largos y complejos, procesando el código fuente de manera rápida y sin consumo excesivo de recursos. El analizador léxico propuesto tiene como objetivo fundamental dividir el código fuente escrito en el lenguaje específico en una secuencia de tokens válidos. Su éxito se medirá por su capacidad para reconocer y clasificar los elementos léxicos del lenguaje según las reglas definidas

Desarrollo

Para la construcción del lenguaje "Tu_P_L" primero se tiene que definir cómo será el lenguaje, este lenguaje se escogió por parte de los integrantes del equipo y se propuso lo siguiente

Lenguaje

Primero se definió los tokens o elementos léxicos del lenguaje. Esto incluye palabras reservadas, operadores, delimitadores, tipos de datos, identificadores, números y cualquier otro elemento que sea significativo en el lenguaje. Cada token se define utilizando expresiones regulares o patrones que describen cómo se ven y cómo se deben reconocer.

Palabras reservadas:

- "entero"
- "buleano"
- "flotante"

- "SI"
- "EntonCes"
- "Para"
- "MiEntras"

Estas son palabras propias del lenguaje que serán utilizadas para reconocer sentencias y ciertas estructuras del lenguaje y no con otros fines, por ejemplo como identificadores.

Operadores aritméticos:

- "+"
- "*"
- "-"
- "/"

Tu_P_Lenguaje permitirá realizar las operaciones aritméticas básicas. Estos son los tokens que se utilizarán para poder formar dichas expresiones.

Operadores relacionales:

- "<" (menor que)
- ">" (mayor que)
- "(=3" (menor o igual que)
- "E=)" (mayor o igual que)
- "W!" (diferente de)

Estos tokens definen las distintas comparaciones que se pueden hacer entre los operandos.

Asignación

- "<=" (igual a)

Este token es un operador que se utilizará para asignar valores a las variables.

Delimitadores:

- "UwU"
- "("
- ")"

Sirven para agrupar elementos o delimitar el fin de alguna estructura.

Espacios

- "\t"
- "\n"
- “ “

Para los espacios en blanco (tabuladores y espacios) y los saltos de línea. En Tu_P_Lenguaje estos caracteres no tienen significado especial para el lenguaje y deberá ignorarlos.

Variable

- ".|."[a-ZA-Z0-9]

Sirve para identificar las cadenas válidas que pueden ser utilizadas como identificadores de variables.

Cadena

- "CaD"[A-Za-z0-0-9]"UwU"

Se planea que el lenguaje tenga soporte de cadenas. Este patrón sirve para identificar las cadenas válidas.

Número

- "[0-9]+"

Este para reconocer número, especificamente, numeros enteros. Esta versión del lenguaje no tendrá soporte para números con valores decimales.

Función

- ":-*"

Token para reconocer funciones válidas del lenguaje. Sería como el equivalente a la palabra "def" que se utiliza en Python.

Uso de Herramientas

FLEX es un analizador léxico bajo licencia GPL. Cada vez que se encuentre uno de los patrones especificados en FLEX se puede ejecutar un conjunto de acciones asociadas. FLEX es el analizador de dominio público compatible con el analizador léxico más frecuentemente utilizado: LEX (bajo sistema UNIX). FLEX (y LEX) genera, dada una especificación correcta de patrones y acciones, un programa en lenguaje C que puede ser compilado para obtener un programa ejecutable. 1

El formato de un archivo de entrada para flex tiene la siguiente estructura

```
Definiciones
%%
Reglas
%%
Código del usuario
```

En la seccion de definiciones se colocan, entre otras cosas, declaraciones de nombres sencillos que tienen la finalidad de simplificar la especificación del analizador léxico.[2]

En la sección de reglas es donde se escriben los patrones que va a reconocer asi como las acciones que se van a realizar una vez que el analizador encuentre alguna cadena que coincida con dichos patrones. Las reglas que se definan aquí deben de tener la estructura

```
patrón { acción }
```

- El "patrón" es una expresión regular que describe cómo se ve un token específico.

- La "acción" es un código C que se ejecutará cuando se encuentre un token que coincida con el patrón.

Finalmente, en la sección de código de usuario se puede escribir cualquier código en C que necesite el usuario para su analizador.

los tipo de Tokens usados serán

```
RESERVADAS "entero"|"buleano"|"flotante"|"SI"|"EntoCes"|"Para"|"MiEntras"
OP_ARITMETICO "+"|"-"|"*"|"/"
OP_RELACIONAL "<"|">"|"(=3"|"E=)"|"!W"
ASIGNACION "<="
DELIMITADOR "UwU"|"("|")"|";"
ESPACIOBLANCO " "|"\t"
NUMERO [0-9]
```

Definimos las Reglas
En el archivo ".l", se escribieron las reglas que coinciden con los patrones de tus tokens definidos con anterioridad.

tomando un Fragmento de nuestro archivo flex

```
{RESERVADAS} { printf("Palabra reservada: %s\n",yytext); }
{OP_ARITMETICO}        { printf("Operador Aritmético: %s\n",yytext); }
{OP_RELACIONAL}      { printf("Operador Relacional: %s \n",yytext); }
```

Dentro de las acciones de las reglas de Flex, se debe especificar cómo se manejarán los tokens reconocidos. Esto puede incluir la devolución del token al programa principal para su procesamiento o almacenamiento en una tabla de símbolos.
En este caso las palabara RESERVADAS, OP_ARITMETCA y OP_RELACIONAL son los tokens que va a reconocer el lenguaje. La accion que va a realiza en esta primera etapa sera imprimir en pantalla el nombre del token junto con su valor.

En esta etapa, también puedes realizar acciones adicionales, como la conversión de números de cadena a valores numéricos, una excepción que utilizaremos en nuestro lenguaje sera el ignorar los espacion en blanco en la escritura del codigo.

```
{ESPACIOBLANCO} {/* Ignorar */}
```

Lo que se encuentra entre llaves es simplemente un comentario en C, entonces al leer un conjunto de espacios o tabuladores el analizador léxico lo va a consumir y no va a realizar ninguna accion (los va a "ignorar").

Conclusiones

El desarrollo de un analizador léxico utilizando la herramienta Flex es un paso fundamental en la construcción de un compilador, intérprete o procesador de lenguajes de programación. El analizador léxico se encarga de escanear y dividir el código fuente en una secuencia de tokens, lo que sienta las bases para una interpretación o compilación precisa.

Algunos de los aspectos clave a tener en cuenta en el desarrollo de un analizador léxico con Flex incluyen la definición de reglas para reconocer patrones léxicos, la asignación de valores y la devolución de tokens, así como la gestión de errores. Además, es importante que el analizador sea eficiente y capaz de manejar de manera adecuada el flujo de entrada.

En conclusión un analizador léxico bien desarrollado y probado es crucial para el éxito de proyectos de compilación o interpretación, ya que garantiza que el código fuente se interprete o

compile de manera correcta y eficiente. Además, una documentación adecuada del analizador léxico facilita su uso y mantenimiento en proyectos futuros.

Referencias

[1] García Fernández, Luis Amable, and María Gloria Martínez Vidal. "Primera práctica: Introducción al Analizador Léxico FLEX." [EN LINEA] Disponible en https://repositori.uji.es/xmlui/bitstream/handle/10234/5998/Primera_Practica_IS17_Curso_06_07.pdf?sequence=1 [Accedido: 1 Octubre 2023]

[2] Paxson Vern. "Flex, version 2.5". USA: THe Regents of the Univerity of California, 1990.