Centro Universitario de Ciencias Exactas e Ingenierías

Programación de sistemas embebidos

Hands-on 2: Implementación de Analizadores Léxicos

Carrera: Ingeniería en computación.

Alumno: Fabián Joheshua Escalante Fernández

Materia: Compiladores

Calendario: 2025A

Fecha: 15/03/2024

Hands-on 2: Implementación de Analizadores Léxicos

Índice

Compiladores

Introducción

Para la compilación y el procesamiento de lenguajes el análisis léxico es el primer paso esencial para transformar un código fuente en una estructura entendible por el compilador. Vamos a ver en esta actividad una guía técnica para que estudiantes de informática y computación aprendan a utilizar herramientas como Lex o Flex.

Desarrollo

1. Requisitos y Herramientas Necesarias

Es fundamental contar con el entorno y las herramientas adecuadas. Dependiendo del sistema operativo, se recomienda:

Windows:

- Instalar WinFlex para una experiencia similar a Flex.
- Alternativamente, utilizar WSL (Windows Subsystem for Linux) o Cygwin para trabajar en un entorno Linux.

Linux:

- Instalar Flex y Bison mediante:
- sudo apt-get install flex bison

2. Ejercicios Propuestos

Ejercicio 1: Análisis Básico

Objetivo: Es crear un analizador léxico que reconozca palabras clave, identificadores y números.

Hands-on 2: Implementación de Analizadores Léxicos

Pasos:

- Diseño del archivo .l:
- Definir las secciones de código, reglas léxicas y las acciones a ejecutar.

Ejemplo de código Flex:

```
%{
#include <stdio.h>
%}

%%
"int"      { printf("Palabra clave: int\n"); }
"return"   { printf("Palabra clave: return\n"); }
[0-9]+     { printf("Número: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]* { printf("Identificador: %s\n", yytext); }
.          { /* Ignorar otros caracteres */ }
%%

int main(void) {
   yylex();
   return 0;
}
```

Compiladores

Compilación y Ejecución:

Comandos:

flex archivo.l

gcc lex.yy.c -o analizador -lfl

./analizador

Al procesar un archivo de texto que contenga, por ejemplo, int x = 10; return x;, se mostrará en consola la identificación de las palabras clave, números e identificadores.

Manejo de Comentarios y Cadenas de Texto

Objetivo: Extender el analizador anterior para que reconozca:

Pasos:

Actualización del archivo .l:

```
%{
#include <stdio.h>
%}

%%
"int"                { printf("Palabra clave: int\n"); }
"return"             { printf("Palabra clave: return\n"); }
[0-9]+               { printf("Número: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]* { printf("Identificador: %s\n", yytext); }
```

```
\"([^\\\n]|(\\.))*?\"   { printf("Cadena de texto: %s\n", yytext); }

"//".*                  { /* Ignorar comentario de una línea */ }

"/*"(.|\n)*?"*/"        { /* Ignorar comentario multilínea */ }

.                       { /* Otros caracteres */ }

%%


int main(void) {

    yylex();

    return 0;

}
```

Explicación:

La regla para cadenas de texto utiliza una expresión regular que permite reconocer contenido dentro de comillas dobles, incluyendo secuencias escapadas, para comentarios ignoran el contenido y no generan salida.

Compilación y Ejecución:

Igual que en el Ejercicio 1, siguiendo las instrucciones para cada sistema operativo.

Compiladores

Integración Avanzada

Objetivo: Integrar el analizador léxico en un programa que cuente la cantidad de palabras clave, identificadores, números, operadores y delimitadores en un archivo fuente.

Pasos:

Creación del archivo .l:

Incluir reglas para cada uno de los elementos a contar. Por ejemplo:

```
%{
#include <stdio.h>
int contadorPalabrasClave = 0, contadorIdentificadores = 0, contadorNumeros = 0;
// Variables para otros contadores
%}

%%
"int"      { contadorPalabrasClave++; }
"return"   { contadorPalabrasClave++; }
[0-9]+     { contadorNumeros++; }
[a-zA-Z_][a-zA-Z0-9_]* { contadorIdentificadores++; }
// Se pueden agregar reglas para operadores y delimitadores
.          { /* Ignorar */ }
%%
```

```
int main(void) {
    yylex();
    printf("Palabras clave: %d\n", contadorPalabrasClave);
    printf("Identificadores: %d\n", contadorIdentificadores);
    printf("Números: %d\n", contadorNumeros);
    // Mostrar los contadores de otros elementos
    return 0;
}
```

Integración en otros lenguajes:

C: Como se muestra en el ejemplo anterior, integrando el analizador directamente en el main().

Java: Utilizando ANTLR, se debe definir una gramática léxica y luego integrarla en una aplicación Java.

Python: Con PLY, se crean módulos que contienen definiciones de tokens y funciones de análisis, integrándolos en un script Python.

Compilación y Ejecución:

En C (Linux o Windows)

Utilizar los comandos mostrados en Ejercicio 1.

En Java y Python

Seguir las guías específicas de ANTLR y PLY, respectivamente, asegurándose de instalar las dependencias correspondientes y configurar el entorno.

Conclusión

La implementación de analizadores léxicos es un pilar fundamental en el desarrollo de compiladores y herramientas de procesamiento de lenguajes. En esta tarea hemos estructurado de manera progresiva para abordar desde conceptos básicos hasta la integración avanzada del analizador en programas en C, Java o Python, siguiendo las instrucciones aquí detalladas, los estudiantes podrán experimentar de manera práctica, comprender la estructura de un archivo .l.