Manual Tecnico

Jose Horacio Lopez Orellana - 1290-21-3372

Julio Alejandro Ponce Tobias - 1290-21-19426

Josué Daniel Oliva Barillas - 1290-21-2403

César Flores - 1290-21-292

Índice

lexico.l

(FLEX)

Flex es una herramienta que se utiliza para generar analizadores léxicos o escáneres en aplicaciones de procesamiento de lenguajes. Su nombre proviene de "Fast Lexical Analyzer Generator". Este software es una implementación de la herramienta Lex, que es parte del conjunto de herramientas de análisis sintáctico desarrolladas originalmente para sistemas operativos Unix. Flex simplifica la creación de analizadores léxicos al generar automáticamente código C que reconoce patrones en el texto de entrada. Este código es un archivo de definición de análisis léxico escrito en flex, que se utiliza en la construcción de compiladores y analizadores léxicos. El archivo se utiliza para reconocer tokens en un lenguaje de programación o un lenguaje específico, para compilar esto es necesario descargar flex. ya descargado flex y para verificar que este instalado, al haber terminado el programa se ejecuta el comando “flex lexico.l”

Estructura

%{ y %}: En Flex, estas marcas delimitan la sección donde se pueden incluir definiciones y declaraciones en C que serán copiadas directamente en el archivo de salida.

#include parser.tab.h y #include "token.h": Estas líneas incluyen archivos de encabezado que contienen definiciones y declaraciones utilizadas en

este archivo, posiblemente para permitir la comunicación con otros componentes del compilador o analizador.

extern int flag_err_type; y extern void yyerror(char *message);: Estas líneas declaran variables y funciones que se esperan que estén definidas en otro lugar del código. La declaración extern indica que estas variables y funciones se definirán en otro archivo y se enlazará más tarde en el proceso de compilación.

int lineno=1; y int line_init=-1; : Estas variables se utilizan para realizar un seguimiento del número de líneas en el código fuente actual y para marcar la línea de inicio.

char str_buf[256]; y char* str_buf_ptr; : Estas variables se utilizan para almacenar temporalmente el contenido de los comentarios en el código fuente.

FILE *vitacora_tokens_file = NULL; : Esta variable se utiliza para manejar un archivo donde se registran los tokens encontrados durante el análisis léxico.

Sección de opciones

%option noyywrap: Esta opción desactiva la generación de la función yywrap en el código de salida. La función yywrap se utiliza en ciertos casos para manejar la finalización del análisis.

%option case-sensitive: Esta opción hace que el analizador léxico distinga entre mayúsculas y minúsculas al reconocer patrones, lo que significa que considera las diferencias de mayúsculas y minúsculas al reconocer las palabras clave.

%option yylineno: Esta opción habilita la variable yylineno que realiza un seguimiento del número de líneas durante el análisis.

Definiciones de patrones

Se definen varios patrones usando expresiones regulares como LETTER, LETTERS, DIGIT, NZDIGIT, ALPHANUM, ALPHANUM_ y ASCII. Estas definiciones de patrones facilitan la escritura de reglas para reconocer tokens específicos más adelante en el código.

Reglas de reconocimiento de tokens

El código define varias reglas de reconocimiento de tokens utilizando expresiones regulares, se abre con dos signos de porcentaje y dentro de ella van las expresiones regulares, se cierra con otros dos signos de porcentaje. Cada regla consta de un patrón y un bloque de código C que se ejecuta cuando el analizador léxico encuentra una coincidencia con el patrón, para cada regla de reconocimiento de tokens, se realiza una acción específica. Por ejemplo, cuando se detecta un token como

<AUTOMATA_AFN>, </AUTOMATA_AFN>, <ALFABETO>, </ALFABETO>, etc., se imprime un mensaje correspondiente y se registra la información en un archivo llamado "vitacora_tokens.html" utilizando las funciones token_print y close_vitacora_file.

Además de las palabras clave específicas, también se reconocen y manejan otros elementos como símbolos de puntuación, números enteros y cadenas, se capturan espacios en blanco y saltos de línea para realizar un seguimiento del número de líneas utilizando la variable lineno, el código también maneja los comentarios HTML (<!-- y -->) mediante el uso de estados de inicio y fin para reconocer y procesar los comentarios en el código fuente.

Funciones de utilidad

token_print: Esta función se utiliza para imprimir y registrar tokens encontrados durante el análisis léxico. Además, mantiene un seguimiento de las líneas actuales y registra la información en un archivo de registro si se cumplen ciertas condiciones.

close_vitacora_file: Esta función se utiliza para cerrar el archivo de registro de tokens ("vitacora_tokens.html") después de que se completa el análisis léxico.

comment_print: Esta función se utiliza para imprimir y registrar comentarios encontrados en el código fuente, siempre y cuando la opción de depuración y visualización de comentarios está habilitada.

parser.Y

(BISON)

Bison es un generador de analizadores sintácticos o parsers. Es una herramienta que se utiliza para generar analizadores sintácticos en C o en otros lenguajes de programación. Su nombre es un acrónimo recursivo que significa "Bison is Yacc", en referencia a Yacc (Yet Another Compiler Compiler), un generador de analizadores sintácticos ampliamente utilizado en sistemas Unix. Este código se trata de un programa en C que parece ser un analizador sintáctico para un lenguaje específico relacionado con autómatas finitos no deterministas (AFN), para compilar esto es necesario descargar bison. Ya descargado bison y para verificar que este instalado, al haber terminado el programa se ejecuta el comando “bison -d parser.y”

Estructura de Bison

%{ y %}: Estos símbolos se utilizan para delimitar la sección de código C en el archivo de definición de Bison. Todo el código C ubicado entre estos símbolos se incluirá directamente en el archivo generado por Bison.

%union: Esta directiva se utiliza para especificar los tipos posibles de valores que pueden estar asociados con los tokens. En el código proporcionado, se definen varios tipos como int, float, char, y char*, lo que permite al analizador sintáctico manejar diferentes tipos de valores.

%token: Estos símbolos se utilizan para definir tokens que se utilizarán durante el análisis léxico. Se les asigna un tipo y un identificador. Por

ejemplo, en el código se definen tokens como T_ALFABETO_OP, T_ESTADO_OP, T_FINAL_OP, T_TRANSICIONES_OP, entre otros.

%type: Esta directiva se utiliza para especificar el tipo de retorno de ciertas reglas de producción. En el código proporcionado, se utiliza para especificar el tipo de retorno de ciertas reglas como alfabeto, estado, inicial, final, atributofin, y transiciones.

%start: Este símbolo se utiliza para especificar el símbolo inicial de la gramática. En el código, se define como programa, lo que indica que la regla de producción llamada programa es el punto de inicio del análisis.

%%: Este símbolo se utiliza para separar las secciones de definición de la gramática de las secciones de código C.

Reglas de gramática: Estas reglas describen la estructura sintáctica permitida en el lenguaje que se está analizando. Se definen utilizando la notación BNF (Backus-Naur Form) y EBNF (Extended Backus-Naur Form). Por ejemplo, se definen reglas como alfabeto, estado, inicial, final, atributofin, y transiciones.

yyparse(): Esta función es generada por Bison y se utiliza para iniciar el análisis sintáctico. Es responsable de ejecutar el análisis basado en las reglas de la gramática definidas en el código de Bison.

yyerror(const char *message): Esta función se utiliza para manejar errores durante el análisis sintáctico. En el código proporcionado, se registra cada error en un archivo de registro y también se muestra en la salida estándar.

Bibliotecas

La sección inicial del código incluye las bibliotecas necesarias para el funcionamiento del programa, como stdio.h, stdlib.h, string.h y stdbool.h, además incluye el archivo de encabezado token.h fue hecho manualmente y que contiene declaraciones, definiciones relacionadas con el manejo de tokens, también se declaran ciertas variables y arreglos externos que se utilizan en el programa para almacenar información y datos relacionados con el análisis.

Definición de tokens

En esta sección, se definen varios tokens utilizando la notación %token seguida de su tipo y un identificador. Los tokens representan elementos específicos del lenguaje que el analizador sintáctico reconocerá durante el análisis del código fuente. La sección %union especifica los tipos de valores que pueden estar asociados con los tokens. En este caso, se definen tipos como int, float, char y char para manejar diferentes tipos de valores.

Tokens Terminales:

1. T_AUTOMATA_AFN_OP

2. T_AUTOMATA_AFN_END

3. T_ALFABETO_OP

4. T_ALFABETO_END

5. T_ESTADO_OP

6. T_ESTADO_END

7. T_INICIAL_OP

8. T_INICIAL_END

9. T_FINAL_OP

10. T_FINAL_END

11. T_TRANSICIONES_OP

12. T_TRANSICIONES_END

13. T_EPSILON

14. T_COMMA

15. T_INT

16. T_STRING

17. T_EOF

Tokens No terminales:

1. programa

2. alfabeto

3. alfabetoatr

4. estado

5. estadoatr

6. inicial

7. final

8. atributofin

9. transiciones

Definición de reglas de gramática

Aquí se definen las reglas de la gramática del lenguaje que se va a analizar. Se utilizan las notaciones BNF y EBNF para describir la estructura sintáctica permitida en el lenguaje. Estas reglas definen cómo deben estructurarse los distintos componentes del lenguaje, como el alfabeto, los estados, las transiciones y otras partes del autómata finito no determinista (AFN).

Regla de inicio - programa: La regla programa es la regla de inicio del análisis sintáctico. Define la estructura general del programa y especifica

cómo deben organizarse y relacionarse las diferentes secciones del código relacionado con el autómata finito no determinista (AFN).

Reglas relacionadas con el alfabeto: La regla alfabeto define la estructura del alfabeto en el lenguaje. La regla alfabetoatr especifica cómo deben organizarse los atributos relacionados con el alfabeto, como las cadenas de caracteres utilizadas en el mismo.

Reglas relacionadas con los estados: La regla estado define la estructura de los estados en el lenguaje. La regla estadoatr describe cómo se definen los atributos asociados con los estados, como los valores enteros que identifican a los estados.

Reglas para el estado inicial: La regla inicial especifica la estructura del estado inicial en el autómata finito no determinista. Define cómo se declara y se asigna el estado inicial en el programa.

Reglas para los estados finales : La regla final describe la estructura de los estados finales en el lenguaje. La regla atributofin define los atributos asociados con los estados finales, como los valores enteros que identifican a estos estados.

Reglas para las transiciones : La regla transiciones define la estructura de las transiciones en el lenguaje. La regla transatr describe cómo se definen los atributos asociados con las transiciones, como los valores enteros y las cadenas de caracteres que representan las transiciones entre estados.

Funciones y lógica del analizador

La función main es la función de inicio del programa, en ella se manejan los argumentos de línea de comandos y se inicia el proceso de análisis invocando yyparse(), que es el analizador sintáctico generado por Yacc. La función yyerror se utiliza para manejar y registrar errores durante el análisis. Esta función es llamada en caso de que se produzca algún error durante el análisis del código fuente. Registra los errores en un archivo de registro llamado "vitacora_errores.html" y también muestra los mensajes de error en la salida estándar.

Compilacion.bat

Compila flex y bison en un batch file, ejecuta el parser.

rem Flex command: El comando flex lexico.l indica que Flex debe tomar como entrada el archivo lexico.l y generar un archivo de salida lex.yy.c, que contiene el código del analizador léxico.

rem Bison command: El comando bison -d parser.y indica que Bison debe tomar como entrada el archivo parser.y y generar archivos de salida Parser.tab.c y Parser.tab.h, que contienen el código del analizador sintáctico.

rem GCC commands: Estas líneas ejecutan comandos del compilador GCC (GNU Compiler Collection). Los comandos gcc -c lex.yy.c -o lex.yy.o y gcc -c Parser.tab.c -o Parser.tab.o compilan los archivos lex.yy.c y Parser.tab.c respectivamente, generando archivos objeto lex.yy.o y Parser.tab.o.

rem Linking: Esta línea realiza el enlazado de los archivos objeto generados anteriormente. El comando gcc lex.yy.o Parser.tab.o -o myParser.exe enlaza los archivos objeto lex.yy.o y Parser.tab.o para generar un ejecutable llamado myParser.exe.

rem Solicitar al usuario la ruta del archivo XML a ejecutar: Esta línea está comentada y parece ser un intento de solicitar al usuario que ingrese la ruta del archivo XML que se va a analizar. Sin embargo, al estar comentada, no tiene efecto en el script.

rem Ejecutar el parser con el archivo XML proporcionado por el usuario: Esta línea está comentada y parece ser un intento de ejecutar el analizador con el archivo XML proporcionado por el usuario. Sin embargo, al estar comentada, no tiene efecto en el script.

Main.cpp

Este código es un programa en C++ que realiza varias operaciones relacionadas con el manejo de archivos y la manipulación de autómatas finitos

Librerías Incluidas

El programa comienza con la inclusión de varias librerías estándar de C++, que son necesarias para realizar diversas operaciones. Estas incluyen librerías para manejo de entrada/salida, manejo de strings, manejo de archivos, manipulación de datos, operaciones en el sistema operativo, entre otros.

Función isXML(const std::string& filename)

Esta función verifica si un archivo dado tiene la extensión ".xml" y si contiene etiquetas de apertura y cierre, lo que indica que se trata de un archivo XML válido. Para hacer esto, lee el contenido del archivo línea por línea y busca la presencia de < y >, que son indicativos de etiquetas de apertura y cierre respectivamente.

Función extractFileName(const std::string& filePath)

Esta función toma una ruta de archivo completa como entrada y extrae el nombre del archivo de esa ruta. Utiliza la función find_last_of para encontrar la última ocurrencia de un separador de ruta ('/' o '\\') y extraer el nombre del archivo.

Función AFNGraph

Esta función se encarga de generar un gráfico representando un diagrama de transición de estados. Para ello, utiliza un formato de archivo llamado DOT, que es un lenguaje simple para describir gráficos. Este archivo DOT se utiliza para representar gráficamente el diagrama de transición de estados, que se convierte en un archivo de imagen PNG utilizando la herramienta dot del sistema.

Función AFN

Esta función realiza varias operaciones importantes. Lee un archivo HTML llamado "vitacora_tokens.html" que contiene información sobre un Autómata Finito No Determinista (AFN). Lee detalles como el alfabeto, los estados, el estado inicial, los estados finales y las transiciones del archivo. Luego, genera un nuevo archivo HTML llamado "dibujo.html" que contiene

una tabla con la información del AFN y un gráfico de transición generado por la función AFNGraph.

Función AFD

Se abre un archivo de entrada "dibujo.html" y se verifica si se pudo abrir correctamente. Luego, se abre un archivo de salida "conversion.html" y se realizan comprobaciones similares. Se inicializan algunas variables de estado y se comienza a recorrer el archivo de entrada línea por línea. Durante este proceso, se busca información específica, como la tabla de transiciones, el alfabeto, el estado inicial, los estados finales y hace una conversión para generar de AFN a AFD. La información relevante se escribe en el archivo de salida "conversion.html".

Función AFDGraph

Se encarga de procesar el archivo "conversion.html" generado previamente. Se abre este archivo y se realiza la verificación similar a la función anterior. A continuación, se crea un archivo "AFD.dot" donde se escribe el código en el formato adecuado para representar el AFD en forma de grafo. Se identifican el estado inicial y los estados finales, y se crea el grafo correspondiente. Finalmente, se convierte el archivo .dot en un archivo de imagen .png utilizando el programa dot de Graphviz. Si el proceso es

exitoso, se muestra un mensaje de éxito; de lo contrario, se muestra un mensaje de error.

Función main

Esta función es el punto de entrada principal del programa. Muestra un menú al usuario con varias opciones, como seleccionar un archivo XML, crear un AFN, mostrar un AFD, pasar de AFN a un AFD, mostrar AFD y salir del programa. Dependiendo de la opción elegida por el usuario, se ejecutan diferentes funciones correspondientes a la opción seleccionada, el código también maneja casos de errores y muestra mensajes detallados de error en la consola si ocurre algún problema durante la ejecución del programa. Además, utiliza directivas de preprocesador para realizar operaciones específicas según el sistema operativo en el que se esté ejecutando el programa, como limpiar la pantalla de la consola y otras operaciones específicas del sistema.

Opciones del menú del main.cpp:

- **Opción 1 - Seleccionar Archivo XML y Parsearlo:** Esta opción llama a la función que ejecuta un script de shell denominado "build_parser.bat". El resultado de este script se evalúa y se muestra si la ejecución fue exitosa o si se produjo un error.

- **Opción 2 - Crear AFN:** Esta opción llama a la función AFN una función definida en el código o en un archivo de inclusión. La función AFN se encarga de crear un archivo HTML a partir de los datos analizados o ingresados por el usuario, para luego ser mostrados en un navegador.
- **Opción 3 - Mostrar AFN:** Esta opción llama a una función que ejecuta un comando del sistema para abrir el archivo "dibujo.html" en el navegador.
- **Opción 4 - Pasar de AFN a AFD:** Esta opción llama a dos funciones: AFD y AFDGraph. Estas funciones son responsables de realizar la conversión de un AFN a un AFD y de generar un gráfico correspondiente del AFD.
- **Opción 5 - Mostrar AFD:** Esta opción llama a una función que ejecuta un comando del sistema para abrir el archivo "conversion.html" en el navegador.
- **Opción 6 - Salir:** Esta opción simplemente muestra un mensaje de despedida y termina la ejecución del programa.