INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO

Carrera: Ingeniería en Sistemas Computacionales

Materia: Compiladores

Profesor: Rafael Norman Saucedo Delgado

Práctica 3: Analizador Léxico

Grupo: 3CV5

Introducción:

Vimos que la primera fase del análisis es el análisis léxico. El principal objetivo del analizador léxico es leer el flujo de caracteres de entrada y transformarlo en una secuencia de componentes léxicos que utilizara el analizador sintáctico. Al tiempo que realiza esta función, el analizador léxico se ocupa de ciertas labores de "limpieza", entre ellas esta eliminar los blancos o los comentarios. También se ocupa de los problemas que pueden surgir por los distintos juegos de caracteres o si el lenguaje no distingue mayúsculas y minúsculas. Para reducir la complejidad, los posibles símbolos se agrupan en lo que llamaremos clases léxicas. Tendremos que especificar qué elementos componen estas categorías, para lo que emplearemos expresiones regulares. Así que se implementará un analizador léxico para el lenguaje C usando flex en su versión 2.6.4.

Desarrollo

1.Ejemplificar el lenguaje (ejemplos escritos en lenguaje C)

La sintaxis que se debe seguir para realizar un programa en lenguaje C es como se muestra a continuación en los ejemplos:

```c
#include<stdio.h>
int main(void){

    /*algoritmo de ordenamiento*/
    int conjunto = [1,5,10,4];
    int i, j;

    for(i = 0; i < n; i++) {
        for(j = 0; j < n-1; i++) {
            if (conjunto[j] > conjunto[j + 1]){
                tmp = conjunto[j];
                conjunto[j] = conjunto[j + 1];
                conjunto[j + 1] = tmp;
            }
        }
    }

    return 0;
}
```

Imagen 1. Ejemplo 1 de código C

```c
#include<stdio.h>

int main(void){

        printf("hola mundo");

        return 0;
}
```

imagen 2. Ejemplo 2 de código en C

2.Identificar las clases léxicas (constantes, int, strings, float, double, char, nombres de funciones, separadores, operadores)

Las clases léxicas que se identificaron son:

- **Tipos de dato**: en esta clase léxica se agrupan todos los tipos de datos que puede soportar el lenguaje C como **int**, **char**, **long**, **short**, **float**, **double**, **void**.
- **Secuencias de escape**: engloba los caracteres especiales para definir ciertos caracteres especiales dentro de una cadena de texto, como ejemplo
- **Palabras reservadas**: clase léxica que engloba las palabras reservadas como **if**, **else**, **do**, **break**, **continue**, **for**, **goto**, **return**, **sizeof**, **while**, **unsigned**, por mencionar algunas de las palabras reservadas que contiene el lenguaje C.
- **Números:** esta clase engloba tanto números enteros como números con punto decimal.

- **Cadenas:** clase léxica que valida las cadenas de texto
- **Delimitador:** englobamos caracteres especiales que nos sirven para separar trozos de código o distintos elementos en una lista.
- **Id:** clase léxica para identificar el nombre de una variable, por ejemplo: int contador; el resultado esperado de la clase será int <id>;
- **Directivas:** esta clase léxica identifica las directivas como, por ejemplo: **#define**, **#ifdef**
- **Comentarios:** esta clase abarca los dos tipos de comentarios que se pueden ocupar en lenguaje C como los comentarios en una línea **"//esto es un comentario de una línea**" o comentarios de múltiples líneas "**/*comentario de múltiples líneas*/**"

3.Escribir las expresiones para cada clase léxica. (leer el manual flex)

Revisando el manual de flex específicamente en la sección de "Patterns" podemos visualizar una especie de sintaxis para la representación de expresiones regulares en flex al igual que una pequeña descripción de lo que significa cada sentencia.

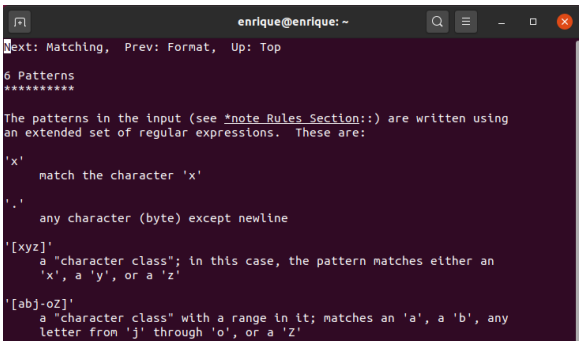

imagen 1. info flex

DIGITO: [0-9]

KEYWORDS:
"auto"|"break"|"case"|"const"|"continue"|"default"|"do"|"else"|"enum"|"extern"|"for"|"goto"|"if"|"register"|"return"|"signed"|"sizeof"|"static"|"typedef"|"union"|"unsigned"|"volatile"|"while"

TIPOSDATO: "char"|"int"|"long"|"short"|"float"|"double"|"void"

DIRECTIVAS: "#"[a-za-z]+

LIBS: "<"[a-z]+("."[a-za-z]+|"")">"

OPERADORES_ARITMETICOS:

("+"|"-"|"*"|"/"|"%")|"="|("+"|"-"|"*"|"/"|"%")"="|"++"|"--"

OPERADORES_COMPARACION: "<"|">"|">="|"<="|"!="|"=="|"!"|"&&"|"||"

OPERADORES_BIT: ">>"|">>="|"<<"|"<<="|"~"|"&"|"&="|"|"|"|="|"^"|"^="

OPERADORES: {operadores_aritmeticos}|{operadores_comparacion}|{operadores_bit}

DELIMITADOR: "{"|"}"|"["|"]"|"("|")"|";"|","

ID: ([a-za-z]|"_")([a-za-z0-9]|"_")*

NUMEROS: {digito}+"."{digito}+|{digito}+

SECUENCIAS_ESCAPE: "\\""\""|"\\""n"

CARACTERES_ESPECIALES: "."|" "|"'"|"#"|"?"|":"|"\\"|"@"

CHARS: "'"({secuencias_escape}|[a-za-z0 9]|{caracteres_especiales}| {delimitador}| {operadores}|{id}|{keywords}|{tiposdato}|{libs}|{directivas}|{numeros})*"'"

STRINGS: "\""({secuencias_escape}|[a-za-z0-9]|{caracteres_especiales}|{delimitador}|

{operadores}|{id}|{keywords}|{tiposdato}|{libs}|{directivas}|{numeros})*"\""

CADENAS: {strings}|{chars}

COMENTARIOS_MULTIPLES: "/""*"+([a-za-z0-9]|{caracteres_especiales}|

{delimitador}|{operadores}|{id}|{keywords}|{tiposdato}|{cadenas}|{libs}|{directivas}|{numeros}|{secuencias_escape})*"*"+"/"

COMENTARIOS_UNALINEA: ("/"|"")"*"([a-za-z0-9]|{caracteres_especiales}|

{delimitador}|{operadores}|{id}|{keywords}|{tiposdato}|{cadenas}|{libs}|{directivas}|{numeros}|{secuencias_escape})*

COMENTARIOS: {comentarios_unalinea}|{comentarios_continuacion}

4.Códificar en lex, lexico.l

Sección 1 (Definiciones): en esta parte se puede ingresar código tanto de lenguaje C o C++, en este caso solo necesitamos la cabecera estándar ***#include<stdio.h>*** para realizar algunas impresiones.

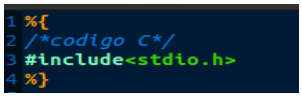

imagen 2. Definiciones

Sección 2 (Definiciones de expresiones regulares): en esta sección se implementan las expresiones regulares, tenemos por ejemplo la definición de la expresión regular de KEYWORDS para identificar las palabras como if, else, while, for que son palabreas reservadas del lenguaje C. Usando el operador or (|) se puede hacer lo siguiente en flex.

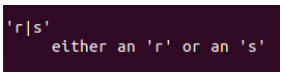

imagen 3. Expresión regular con el operador OR (|)

imagen 4. Palabras reservadas

Para las expresiones DIRECTIVAS y LIBS hice uso de un rango de caracteres #[A-Za-z]+ esto nos indica que encontrara coincidencias cuando se escriban palabras como #define ya que primero debe ir el símbolo "#" y seguida de una o más letras.

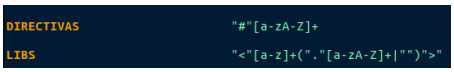

imagen 5 Directivas y Libs

Para la definición de la expresión regular que verifica si son espacios ya sea de una línea o múltiples líneas se concatenaron los caracteres "/" y "*", después pueden ir tanto caracteres especiales, delimitadores, nombres de variables, palabras reservadas, cadenas, bibliotecas, números, secuencias de escape y al ultimo se concatena con "*" y "/" ya que un comentario puede ir de la siguiente manera:

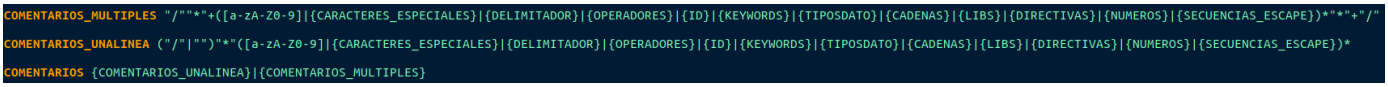

imagen 6. Comentarios múltiples en lenguaje C

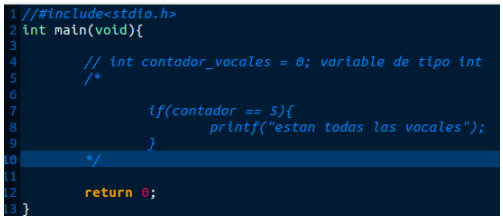

imagen 7. Ejemplo de comentarios en lenguaje C

Sección 3 (Código de usuario):

En esta sección se puede implementar código C que se copiara tal cual en el fichero de salida.

5.Pruebas (compilar, introducir programas)

Para compilar el archivo lexico.l lo hacemos con el comando flex lexico.l el cual nos va a generar un archivo llamado lex.yy.c.

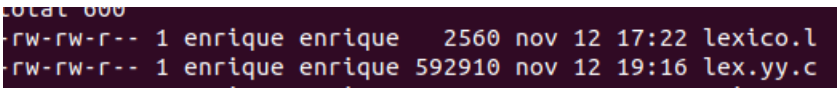

imagen 8. Compilación del archivo lexico.l

Para el archivo lex.yy.c simplemente lo compilamos con el comando gcc -c lex.yy.c ya que si lo ligamos nos dará un mensaje diciendo que hay problemas de referencia hacia funciones, entonces se nos va a generar un archivo llamado lex.yy.o

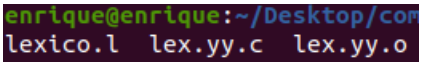

imagen 9. Compilación del archivo lex.yy.c

Ahora hacemos un archivo main.c esto para resolver los problemas de ligado, este archivo contendrá la llamada a la función yylex() que esta invoca al autómata del analizador léxico que se encuentra en el archivo lex.yy.c y que ya fue compilada en el lex.yy.o. Ahora hacemos el ligado del archivo main.o, lex.yy.o y las bibliotecas (-lfl) y asi nos generara un archivo a.out.

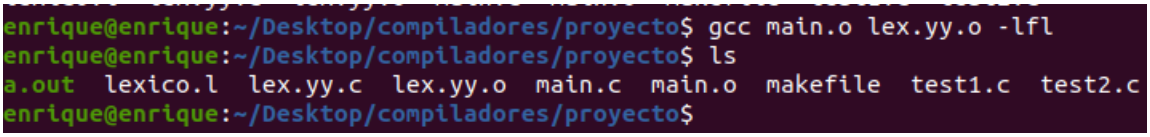

imagen 10. Ligado de los archivos main.o , lex.yy.o y las libs de flex

Pero para facilitarnos la compilación de estos archivos hacemos un makefile que este llevara la gestión de la compilación del programa. Los comandos que usaremos son los siguiente:

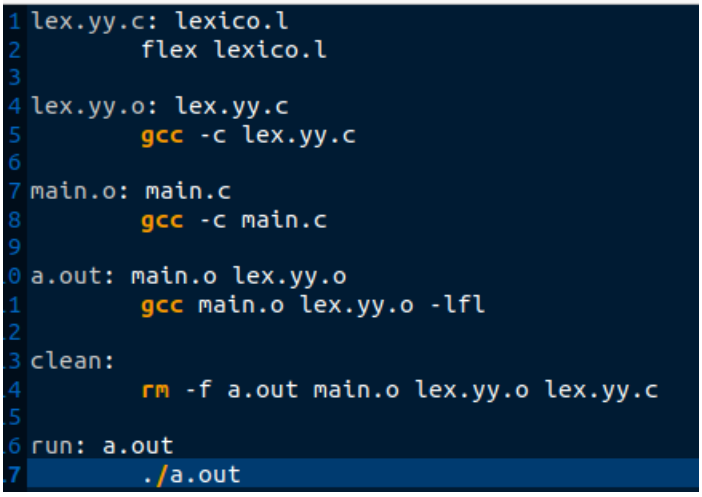

imagen 11. Archivo makefile

Para probar el analizador léxico se selecciono como prueba un algoritmo de ordenamiento el cual se muestra a continuación:

```c
#include<stdio.h>
int main(void){

    /*algoritmo de ordenamiento*/
    int conjunto = [1,5,10,4];
    int i, j;

    for(i = 0; i < n; i++) {
        for(j = 0; j < n-1; i++) {
            if (conjunto[j] > conjunto[j + 1]){
                tmp = conjunto[j];
                conjunto[j] = conjunto[j + 1];
                conjunto[j + 1] = tmp;
            }
        }
    }

    return 0;
}
```

imagen 12. Código de prueba para el analizador léxico

De acuerdo con las reglas que se establecieron anteriormente para cada clase léxica tenemos lo siguiente:

Primeramente, en el recuadro color amarillo se tiene la declaración de la biblioteca estándar del lenguaje C por lo que el analizador léxico nos devolverá que se ha encontrado una <directiva> seguido de una <librería>, esto lo podemos verificar en las clases léxicas DIRECTIVAS Y LIBS, posteriormente en el recuadro color azul se tiene un <tipo de dato> más un <id> que sirve como identificador para una función o variable. Los paréntesis “()” son <delimitadores> al igual que las llaves “{}”, posteriormente en el recuadro morado se tiene un comentario y algunas declaraciones de tipos de datos con sus respectivos identificadores, un operador para la asignación y delimitadores eso se da por el código que se encuentra en el recuadro color rojo.

En el recuadro color verde se nos tiene que mostrar palabras reservadas como son el for e if, también delimitadores e identificadores, números y por último operadores.

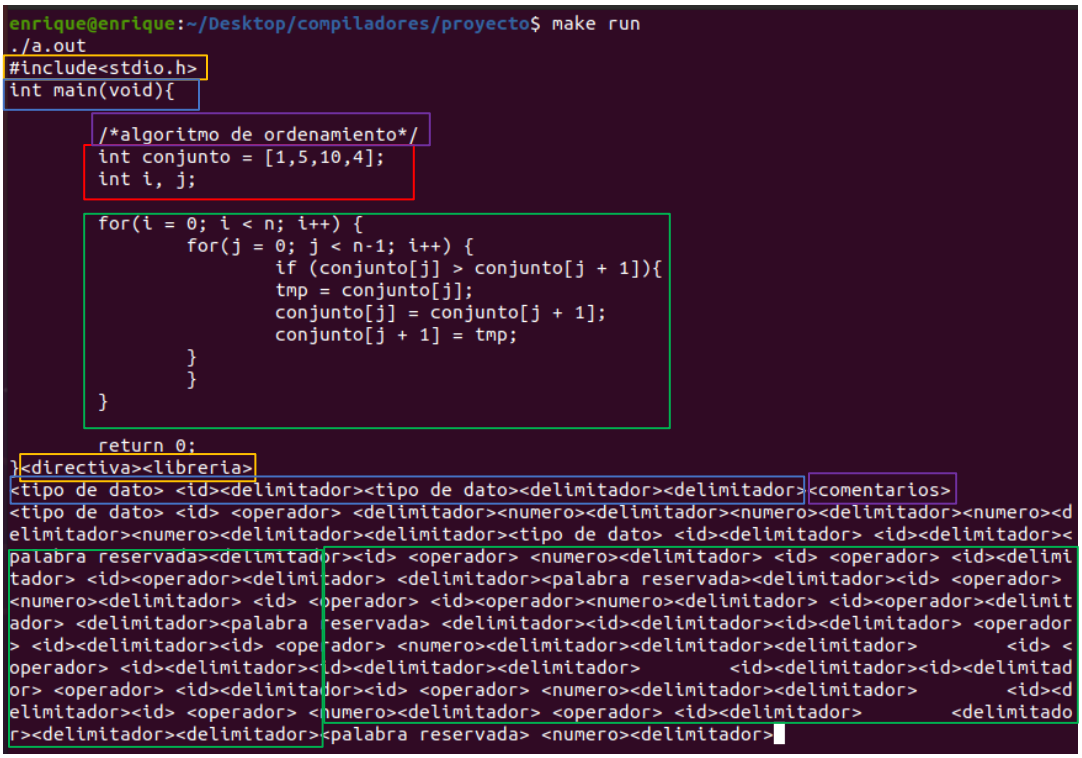

imagen 13. Resultados del analizador léxico

Conclusiones

Esta cuarta práctica sinceramente al principio no me llamo mucho la atención, pero mientras fui investigando un poco más a fondo como es que funcionaba un analizador léxico me termino interesando todo este tema. Y pues este trabajo servirá mucho en el momento de la creación de un compilador y me da curiosidad como es que se puede llegar a crear un lenguaje de programación.

Referencias

[1] B. W. K. Dennis M. Ritchie, THE C PROGRAMMING LANGUAGE, Perarson.

[2] U. M. Repository, «ubuntu manuals,» [En línea]. Available: http://manpages.ubuntu.com/manpages/bionic/man1/flex.1.html. [Último acceso: 12 11 2020].

[3] flex, «Releases westes/flex,» [En línea]. Available: https://github.com/westes/flex/releases. [Último acceso: 12 11 2020].