

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Carrera: Ingeniería en Sistemas Computacionales

Unidad de Aprendizaje: Compiladores

Profesor: Delgado Saucedo Norman

Práctica 4. Analizador Léxico

Alumno: Aguilera Rosas Landa Enrique

Boleta: 2014630005

Grupo: 3CV6

Fecha de Entrega: 12-11-2020

Índice:

1.Introducción:

1.1.Lenguaje C:

1.2.Lenguaje Ensamblador:

2.Desarrollo de la práctica:

2.1.Ejemplificar el Lenguaje:

2.2.Definición de clases léxicas:

2.3.Expresiones de cada clase:

2.4.Codificación en Flex:

2.6.Pruebas:

3.Conclusión:

4.Referencias:

1.Introducción:

El tipo de proyecto a realizar es un compilador cuyo lenguaje de entrada será lenguaje C y su salida será ensamblador. Aunque ya hay opciones por defecto que generan código de lenguaje c a ensamblador, me gustaría comprender el modo o mecanismo que hace funcionar este tipo de compilador.

1.1.Lenguaje C:

El lenguaje de programación C fue creado por Brian Kernighan y Dennis Ritchie a mediados de los años 70. La primera implementación del mismo la realizó Dennis Ritchie sobre un computador DEC PDP-11 con sistema operativo UNIX. C es el resultado de un proceso de desarrollo que comenzó con un lenguaje anterior, el BCPL, el cual influyó en el desarrollo por parte de Ken Thompson de un lenguaje llamado B, el cual es el antecedente directo del lenguaje C.

El lenguaje C es un lenguaje para programadores en el sentido de que proporciona una gran flexibilidad de programación y una muy baja comprobación de incorrecciones, de forma que el lenguaje deja bajo la responsabilidad del programador acciones que otros lenguajes realizan por sí mismos.

El lenguaje C es un lenguaje estructurado, en el mismo sentido que lo son otros lenguajes de programación tales como el lenguaje Pascal, el Ada o el Modula-2, pero no es estructurado por bloques, o sea, no es posible declarar subrutinas (pequeños trozos de programa) dentro de otras subrutinas, a diferencia de como sucede con otros lenguajes estructurados tales como el Pascal. Además, el lenguaje C no es rígido en la comprobación de tipos de datos, permitiendo fácilmente la conversión entre diferentes tipos de datos y la asignación entre tipos de datos diferentes[1].

1.2.Lenguaje Ensamblador:

El lenguaje ensamblador, o assembler (en inglés assembly language y la abreviación asm), es un lenguaje de programación de bajo nivel. Consiste en un conjunto de mnemónicos que representan instrucciones básicas para los computadores, microprocesadores, microcontroladores y otros circuitos integrados programables.

Fue usado principalmente en los inicios del desarrollo de software, cuando aún no se contaba con potentes lenguajes de alto nivel y los recursos eran limitados. Actualmente se utiliza con frecuencia en ambientes académicos y de investigación, especialmente cuando se requiere la manipulación directa de hardware, alto rendimiento, o un uso de recursos controlado y reducido.

Implementa una representación simbólica de los códigos de máquina binarios y otras constantes necesarias para programar una arquitectura de procesador y constituye la

representación más directa del código máquina específico para cada arquitectura legible por un programador.

Cada arquitectura de procesador tiene su propio lenguaje ensamblador que usualmente es definido por el fabricante de hardware, y está basado en los mnemónicos que simbolizan los pasos de procesamiento (las instrucciones), los registros del procesador, las posiciones de memoria y otras características del lenguaje.

Un lenguaje ensamblador es por lo tanto específico de cierta arquitectura de computador física (o virtual). Esto está en contraste con la mayoría de los lenguajes de programación de alto nivel, que idealmente son portátiles [2].

2.Desarrollo de la práctica:

Para el desarrollo de esta práctica se utilizará la herramienta “Flex” [3] Versión: 2.6.1.

Flex es una herramienta para la generación de programas que realizan concordancia de patrones en texto.

La metodología que se aplicará a esta practica sera Cascada iterativa y los pasos que se realizaron se mencionan a continuación:

1. Ejemplificar el Lenguaje.
2. Definir las Clases Léxicas.
3. Escribir las expresiones para cada clase.
4. Codificación en flex.
5. Pruebas.

2.1.Ejemplificar el Lenguaje:

Todo programa de C consta, básicamente, de un conjunto de funciones, y una función llamada `main`, la cual es la primera que se ejecuta al comenzar el programa, llamándose desde ella al resto de funciones que componen nuestro programa. A continuación se muestra un ejemplo de `main()`:

EJEMPLO 1

```
#include <stdio.h>
main()
{
    printf("Bienvenido a la Programacion
           en lenguaje C \n");
    return 0;
}
```

En C, toda variable, antes de poder ser usada, debe ser declarada, especificando con ello el tipo de dato que almacenará. Toda variable en C se declara de la forma:

<tipo de dato> <nombre de variable> [, nombre de variable];

Algunos ejemplos de variables de C serían:

EJEMPLO 2

```
#include <stdio.h>

main()

{
    int multiplicador;      /* defino multiplicador como un entero */
    int multiplicando;     /* defino multiplicando como un entero */
    int resultado;         /* defino resultado como un entero */
    multiplicador = 1000 ;  /* les asigno valores */
    multiplicando = 2 ;

    resultado = multiplicando * multiplicador ;
    printf("Resultado = %d\n", resultado);    /* muestro el resultado */
    return 0;
}
```

Existen, además, cuatro modificadores de tipo, los cuales se aplican sobre los tipos de datos anteriormente citados. Los modificadores de tipo permiten cambiar el tamaño, etc., de los tipos de datos anteriormente especificados. Estos modificadores, que sintácticamente anteceden a la declaración del tipo de dato, son: signed, unsigned, long y short. Es por ello que podemos declarar variables como:

- unsigned char a;
- long double b;
- short int i;

Además de los modificadores de tipo existen modificadores de acceso. Los modificadores de acceso limitan el uso que puede realizarse de las variables declaradas.

Los modificadores de acceso anteceden a la declaración del tipo de dato de la variable y son los siguientes:

- const
- volatile

La declaración de una variable como const permite asegurarse de que su valor no será modificado por el programa, excepto en el momento de su declaración, en el cual debe asignársele un valor inicial.

En C, las variables pueden ser declaradas en cuatro lugares del módulo del programa:

- Fuera de todas las funciones del programa, son las llamadas variables globales, accesibles desde cualquier parte del programa.
- Dentro de una función, son las llamadas variables locales, accesibles tan solo por la función en las que se declaran.
- Como parámetros a la función, accesibles de igual forma que si se declararán dentro de la función.
- Dentro de un bloque de código del programa, accesible tan solo dentro del bloque donde se declara. Esta forma de declaración puede interpretarse como una variable local del bloque donde se declara

EJEMPLO 3

```
#include <stdio.h>
int sum; /* Variable global, accesible desde cualquier parte */
        /* del programa */
void suma(int x) /* Variable local declarada como parámetro, */
                /* accesible solo por la función suma */
{
    sum=sum+x;
    return;
}
void intercambio(int *a,int *b)
{
    if (*a>*b)
    {
        int inter; /* Variable local, accesible solo dentro del */
                    /* bloque donde se declara */
        inter=*a;
        *a=*b;
        *b=inter;
    }
    return;
}
int main(void) /*Función principal del programa*/
{
    int contador,a=9,b=0; /*Variables locales, accesibles solo */
                        /* por main */
    sum=0;
    intercambio(&a,&b);
    for(contador=a;contador<=b;contador++) suma(contador);
    printf("%d\n",sum);
    return(0);
}
```

2.2.Definición de clases léxicas:

Al momento del desarrollo de esta práctica se identificaron y agruparon los elementos pertenecientes a las siguientes clases léxicas

- Sentencia de Control
- Especificador de Almacenamiento
- Tipo de Dato
- Bucle
- Operador
- Operador Relacional
- Operador Lógico
- Operador de desplazamiento
- Modificador de tipo de dato
- Modificador de acceso
- Operador de incremento
- Operador de decremento
- Salto de sección de código
- Asignación a derecha
- Asignación a izquierda
- Asignación de Multiplicación
- Asignación de División
- Asignación de Módulo
- Enumeración
- Sentencias
- Estructura
- Palabra Reservada
- Apuntador
- Módulo
- Constantes de barra invertida
- Ellipsis
- Constante
- Identificador

2.3. Expresiones de cada clase:

A continuación se muestran las clases mencionadas con anterioridad relacionadas con los “caracteres” o “comandos” correspondientes

```
"break"|"case"|"if"|"switch"|"else"|"continue"|"default"
"auto"|"extern"|"static"|"register"
"char"|"double"|"float"|"int"|"void"
"do"|"for"|"while"
"*"|"/"|"+"|"-"|"="|"?"
"<="|">="|"=="|"!="|"<"|>"
"&"|"!"|"|"|"^"|"~"|"&&"|"||"
">>"|"<<"
"signed"|"unsigned"|"long"|"short"
"const"|"volatile"
"++"|"+="
"--"|"--="
"goto"
">>="
"<<="
"*="
"/="
"%="
"enum"
"return"
"sizeof"
"struct"
"typedef"
"union"
"->"
"%"
"{"|"{"|"<%"
"}"|"}"|">%"
";"
":"
"("
")"
("["|"<:")
"]"|">:")
"."
[\\t\\v\\n\\f\\b\\r\\\"\\'\\0\\v\\a\\o\\x]
"...
{D}+{E}{FS}?
{L}({L}|{D})*
{D}*"."{D}+({E})?{FS}?
{D}+."{D}*({E})?{FS}?
L?\\(\\.|\\[\\^\\")*\\
0[xX]{H}{IS}?
0{D}+{IS}?
{D}+{IS}?
L?'(\\.|\\[\\^\\')+ '
%%

{printf(" Sentencia de Control ");}
{printf(" Especificador de Almacenamiento");}
{printf(" tipo de dato"); }
{printf(" Bucle");}
{printf(" Operador");}
{printf(" Operador Relacional");}
{printf(" Operador Logico");}
{printf(" Operador de desplazamiento"); }
{printf(" Modificador de tipo de dato");}
{printf(" Modificador de acceso");}
{printf(" Operador de Incremento");}
{printf(" Operador de Decremento");}
{printf(" Salto de seccion de codigo"); }
{printf(" Asignacion a Derecha"); }
{printf(" Asignacion a Izquierda"); }
{printf(" Asignacion de Multiplicacion"); }
{printf(" Asignacion de Division"); }
{printf(" Asignacion de Modulo"); }
{printf(" Enumeracion"); }
{printf(" Sentencia"); }
{printf(" Operador"); }
{printf(" Estructura"); }
{printf(" Palabra Reservada");}
{printf(" Palabra Reservada");}
{printf(" Operador Apuntador");}
{printf(" Modulo"); }
{printf(" "); }
{printf(" {"); }
{printf(" }");}
{printf(" ,");}
{printf(" :");}
{printf(" (");}

{printf(" )");}
{printf(" [");}
{printf(" ]");}
{printf(" .");}
{printf(" Constante de barra invertida");}
{printf(" Ellipsis"); }
{printf(" Constante"); }
{printf(" Identificador");}
{printf(" Flotante"); }
{printf(" Double"); }
{printf(" Cadena Constante"); }
{printf(" Hexadecimal"); }
{printf(" Identificador");}
{printf(" Long Int"); }
{printf(" Cadena Constante"); }
```


2.4.Codificación en Flex:

A continuación se incluye la codificación realizada en la herramienta Flex, el archivo tiene por nombre: "léxico" con formato ".l", la fuente desde la cual se obtuvo la gramática del lenguaje C se especifica a continuación y se puede encontrar el enlace en la sección de referencias. Gramática C "ANSI C grammar" [4]

```
D      [0-9]
L      [a-zA-Z_]
H      [a-zA-Z0-9]
E      [Ee][+-]?{D}+
FS     (f|F|l|L)
IS     (u|U|l|L)*

%{
#include <stdio.h>
}%

%%
"break"|"case"|"if"|"switch"|"else"|"continue"|"default"
"auto"|"extern"|"static"|"register"
"char"|"double"|"float"|"int"|"void"
"do"|"for"|"while"
"*"|"/"|"+"|"_"|"="|"?"
"<="|">="|"=="|"!="|"<"|">"
"&"|"!"|"|"|"^"|"~"|"&&"|"||"|"
">>"|"<<"
"signed"|"unsigned"|"long"|"short"
"const"|"volatile"
"++"|"+="
"--"|"-=
"goto"
">="
"<="
"*="
"/="
"%="
"enum"
"return"
"sizeof"
"struct"
"typedef"
"union"
"->"
"%"
";"
("{"|"<%" )
("}"|">%" )
","
":"
"("
")"
("["|"<:" )
("]"|">:" )
"."
[\\t\\v\\n\\f\\b\\r\\\"\\'\\0\\v\\a\\o\\x]
"..."
{D}+{E}{FS}?
{L}({L}|{D})*
{D}*"."{D}+({E})?{FS}?
{D}+ "."{D}*({E})?{FS}?
L?\\(\\.|\\[\\^\\])*\\
0[xX]{H}+{IS}?
0{D}+{IS}?
{D}+{IS}?
L?'(\\.|\\[\\^\\')+ '
%%

{printf(" Sentencia de Control ");}
{printf(" Especificador de Almacenamiento");}
{printf(" tipo de dato"); }
{printf(" Bucle");}
{printf(" Operador");}
{printf(" Operador Relacional");}
{printf(" Operador Logico");}
{printf(" Operador de desplazamiento"); }
{printf(" Modificador de tipo de dato");}
{printf(" Modificador de acceso");}
{printf(" Operador de Incremento");}
{printf(" Operador de Decremento");}
{printf(" Salto de seccion de codigo"); }
{printf(" Asignacion a Derecha"); }
{printf(" Asignacion a Izquierda"); }
{printf(" Asignacion de Multiplicacion"); }
{printf(" Asignacion de Division"); }
{printf(" Asignacion de Modulo"); }
{printf(" Enumeracion"); }
{printf(" Sentencia"); }
{printf(" Operador"); }
{printf(" Estructura"); }
{printf(" Palabra Reservada");}
{printf(" Palabra Reservada");}
{printf(" Operador Apuntador");}
{printf(" Modulo"); }
{printf(" ;"); }
{printf(" {"); }
{printf(" }"); }
{printf(" ,");}

{printf(" :");}
{printf(" (");}
{printf(" )");}
{printf(" [");}
{printf(" ]");}
{printf(" .");}
{printf(" Constante de barra invertida");}
{printf(" Ellipsis"); }
{printf(" Constante"); }
{printf(" Identificador");}
{printf(" Flotante"); }
{printf(" Double"); }
{printf(" Cadena Constante"); }
{printf(" Hexadecimal"); }
{printf(" Identificador");}
{printf(" Long Int"); }
{printf(" Cadena Constante"); }
```

A parte de la codificación en flex, se realizaron 2 códigos de manera aleatoria, un main.c y un archivo makefile. Los códigos se anexan a continuación

Main.c

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

Makefile

```
lex.yy.c: lexico.l
    flex lexico.l
main.o: main.c
    gcc -c main.c

lex.yy.o: lex.yy.c
    gcc -c lex.yy.c

a.out: main.o lex.yy.o
    gcc main.o lex.yy.o -lfl

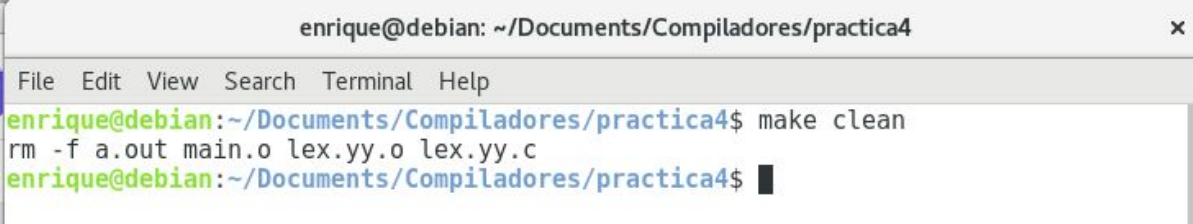
clean:
    rm -f a.out main.o lex.yy.o lex.yy.c

run: a.out
    ./a.out
```

2.6.Pruebas:

A continuación se muestra la pantalla de la terminal de consola en debian, que muestra el comportamiento de analizador léxico ante distintas entradas

Primero se utiliza el archivo Makefile para hacer la compilación y eliminar los archivos innecesarios. Para esto usamos make clean



```
enrique@debian: ~/Documents/Compiladores/practica4
File Edit View Search Terminal Help
enrique@debian:~/Documents/Compiladores/practica4$ make clean
rm -f a.out main.o lex.yy.o lex.yy.c
enrique@debian:~/Documents/Compiladores/practica4$
```

De manera seguida introducimos make a.out y ./a.out para realizar la ejecución del programa y poder iniciar con la captura del léxico que se desea evaluar

```
flex lexico.l
gcc -c lex.yy.c
gcc main.o lex.yy.o -lfl
enrique@debian:~/Documents/Compiladores/practica4$ make a.out
make: 'a.out' is up to date.
enrique@debian:~/Documents/Compiladores/practica4$ ./a.out
```

A continuación se muestran capturas de pantalla donde se introducen secciones de código C.

```
enrique@debian:~/Documents/Compiladores/practica4$ ./a.out
int main(){ int n = 5; }
tipo de dato Palabra Reservada ( ) { tipo de dato Identificador Operador Dígito ; } Salto de línea

enrique@debian:~/Documents/Compiladores/practica4$ ./a.out
int main(){ int n = 5; \n char="Hola"; }
# tipo de dato Palabra Reservada ( ) { tipo de dato Identificador Operador Dígito ; \ Identificador tipo de dato Operador Cadena Constante
; } Salto de línea

enrique@debian:~/Documents/Compiladores/practica4$ ./a.out
#include <stdio.h> void main(){int n=1; int n1=2; int suma=n+n1}
# Identificador Operador Relacional Identificador . Identificador Operador Relacional tipo de dato Palabra Reservada ( ) { tipo de dato Identifica
dor Operador Dígito ; tipo de dato Identificador Operador Dígito ; tipo de dato Identificador Operador Identificador Operador Identificador } Salt
o de línea
```

3.Conclusión:

El desarrollo de esta práctica resultó ser bastante interesante ya que sin duda después de conocer el funcionamiento de los AFD y AFN junto con las expresiones regulares, son conceptos que se deben de incluir al momento de realizar esta práctica. Un dato que me parece curioso es que dependiendo del programador, se pueden encontrar mas o menos clases léxicas y cada programador decide cómo debe agruparlas.

4.Referencias:

- [1] Bonet Esteban, E. V. (2000, Octubre 13). *Lenguaje C*. Informática UV. Retrieved 11 9, 2020, from [El lenguaje de programación C](#)
- [2] C. E. (2017, Mayo 9). *Lenguaje Ensamblador*. Algorítmica y Programación. Retrieved 11 9, 2020, from [Lenguaje ensamblador - Algorítmica y Programación](#)
- [3] Ubuntu. Manual de usuario de Flex “generador de analizadores léxicos”. Retrieved 11, 9, 2020 from rápidos[flex - generador de analizadores léxicos rápidos](#)
- [4] Degenner, J. (1995, - -). *ANSI C grammar, Lex specification*. ANSI C grammar, Lex specification. Retrieved 11 9, 2020, from [ANSI C grammar \(Lex\)](#)