INSTITUTO POLITÉCNICO NACIONAL

ESCOM

INGENIERÍA EN SISTEMAS COMPUTACIONALES

COMPILADORES

“Compilador de C”

PROFESOR:

SAUCEDO DELGADO RAFAEL NORMAN

ALUMNO:

OSMAR PÉREZ BAUTISTA

13/Noviembre/2020

I. Introducción

El motivo del siguiente proyecto es implementar un compilador entre los lenguajes C a Ensamblador . La primera parte de este proyecto consiste en el análisis de las clases léxicas y su implementación en el analizador léxico Flex.

El compilador será desarrollado con el siguiente entorno:

- Flex == 2.6.4

La gramatica de C esta definida en https://www.lysator.liu.se/c/ANSI-C-grammar-l.html

II. Metodología

II.I Ejemplos del lenguaje

Un ejemplo clásico sería Bubble Sort

```
void bubbleSort(int arr[], int n)

{

   int i, j;

   for (i = 0; i < n-1; i++)



       /* Last i elements are already in place    */

       for (j = 0; j < n-i-1; j++)

           if (arr[j] > arr[j+1])

              swap(&arr[j], &arr[j+1]);

}
```

II.I Clases léxicas

Token: Componente léxico de cualquier lenguaje de programación.

Palabras reservadas: Palabras que tienen una definición dentro del lenguaje.

auto break case char const continue default do double else enum extern float for goto if int long register return shortsigned sizeof static struct switch typedef union unsigned void volatile while

Identificador: Palabras que podemos usar para denotar algo en el lenguaje

- El primer simbolo sera un caracter alfabetico después podemos poner caracteres alfanuméricos [a-z] o [0-9] y también admite el simbolo '_'
- Las mayúsculas y minúsculas son diferentes
- Los identificadores no pueden ser palabras reservadas

Literal: Especificación de valores de un tipo de dato

- Numeros
  - Sufijos: u|U (unsigned) l|L (long) f|F (float)
  - Prefijos: 0 (octal) 0x(Hexadecimal)

- Caracteres
  - 'a' 'b' 'c' ….. '0' '1'....
  - Secuencias de escape:
    - \n Salto de línea
    - \b Retroceso
    - \t Tabulación horizontal
    - \v Tabulación vertical
    - \\ Contrabarra
    - \f Salto de página
    - \' Apóstrofe
    - \" Comillas dobles
    - \0 Fin de una cadena de caracteres

- Cadena de caracteres

  "Conjunto de caracteres"

Operador: Realiza una operación matemática.

Símbolo	NOMBRE
* / %	Operadores de multiplicación
+ -	Operadores aditivos
<< >>	Operadores de desplazamiento
< > <= >= == !=	Operadores relacionales
& | ^	Operadores bit a bit
&& ||	Operadores lógicos
,	Operador de evaluación secuencial

Símbolo	NOMBRE
- ~ !	Operadores de negación y complemento
* &	Operadores de direccionamiento indirecto y address-of
`sizeof`	Operador de tamaño
+	Operador unario más
++ --	Operadores unarios de incremento y decremento

Delimitador Símbolos usados como separadores de las distintas construcciones de un lenguaje de programación

[ ] { } ( ) , ; : ... * = #

Comentarios: Cualquier secuencia de caracteres que empieze por /* y termine con */

No se pueden introducir comentarios dentro de otros.

Expresiones para cada clase léxica y programa en lex:

```
%%
(^\/\/.*?$|\/\*.*?\*\/)                    { printf("<comentario>"); }
"do"|"while"|"for"                         {printf("<ciclo>");}
"int"|"char"|"float"|"void"|"double"       {printf("<tipo de dato>");}
"if"|"else"|"switch"|"case"|"default"      {printf("<estructura_de_control>");}
"break"|"continue"|"goto"                  {printf("<sentencia_de_control>");}
"static"|"auto"|"extern"|"register"       {printf("<clase_de_almacenamiento>")}
[+-]?[0-9]+                                {printf("<entero>");}
[+-]?([0-9]*[.])?[0-9]+                    {printf("<decimal>");}
([a-zA-Z_][a-zA-Z0-9_]*)                   {printf("<identificador>");}
[a-zA-Z_]?\"(\\.|[^\\"])*\"                {printf("<cadena>");}
"const"          { printf("<constante>"); }
"enum"           { printf("<tipo de dato>"); }
"return"         { printf("<sentencia>"); }
"short"          { printf("<rango_de_valor>"); }
"long"           { printf("<rango_de_valor>"); }
"signed"         { printf("<rango_de_valor>"); }
"sizeof"         { printf("<operador_compilacion>"); }
"struct"         { printf("<estructura_de_datos>"); }
"typedef"        { printf("<palabra_reservada>"); }
"union"          { printf("<estructura_de_datos>"); }
"unsigned"       { printf("<rango_de_valor>"); }
"volatile"       { printf("<variable_volatil>"); }
"..."            { printf("<delimitador>"); }
">>="            { printf("<operador_asignacion>"); }
```

```
"<<="           { printf("<operador_asignacion>"); }

"+="            { printf("<operador_asignacion>"); }

"-="            { printf("<operador_asignacion>"); }

"*="            { printf("<operador_asignacion>"); }

"/="            { printf("<operador_asignacion>"); }

"%="            { printf("<operador_asignacion>"); }

"&="            { printf("<operador_asignacion>"); }

"^="            { printf("<operador_asignacion>"); }

"|="            { printf("<operador_asignacion>"); }

">>"            { printf("<operador_desplazamiento>"); }

"<<"            { printf("<operador_desplazamiento>"); }

"++"            { printf("<operador_unario_incremento_decremento>"); }

"--"            { printf("<operador_unario_incremento_decremento>"); }

"->"            { printf("<operador_acceso_instancia>"); }

"&&"            { printf("<operador_relacional>"); }

"||"            { printf("<operador_relacional>"); }

"<="            { printf("<operador_relacional>"); }

">="            { printf("<operador_relacional>"); }

"=="            { printf("<operador_relacional>"); }

"!="            { printf("<operador_relacional>"); }

";"             { printf("<delimitador>"); }

"#"             { printf("<delimitador>"); }

"{"             { printf("<delimitador>"); }

"}"             { printf("<delimitador>"); }

","             { printf("<delimitador>"); }

":"             { printf("<delimitador>"); }

"="             { printf("<asignacion>"); }
```

```
"("             { printf("<delimitador>"); }

")"             { printf("<delimitador>"); }

"["             { printf("<delimitador>"); }

"]"             { printf("<delimitador>"); }

"."             { printf("<operador_acceso_instancia"); }

"&"             { printf("<operadores_bit_a_bit>"); }

"!"             { printf("<operador_negacion_complemento>"); }

"~"             { printf("<operador_negacion_complemento>"); }

"-"             { printf("<operador_aditivo>"); }

"+"             { printf("<operador_aditivo>"); }

"*"             { printf("<operadores_multiplicacion>"); }

"/"             { printf("<operadores_multiplicacion>"); }

"%"             { printf("<operadores_multiplicacion>"); }

"<"             { printf("<operador_relacional>"); }

">"             { printf("<operador_relacional>"); }

"^"             { printf("<operadores_bit_a_bit>"); }

"|"             { printf("<operadores_bit_a_bit>"); }

"?"             { printf("?"); }

.               { printf("No reconocido");}

%%
```

III. Pruebas:

void bubblesort(int arr[], int n){}

<tipo de dato> <identificador><delimitador><tipo de dato>
<identificador><delimitador><delimitador><delimitador> <tipo de dato>
<identificador><delimitador><delimitador><delimitador>

int i, j;

for (i = 0; i < n-1; i++)

<tipo de dato> <identificador><delimitador> <identificador><delimitador>

<ciclo> <delimitador><identificador> <asignacion> <entero><delimitador> <identificador>
<operador_relacional> <identificador><entero><delimitador>
<identificador><operador_unario_incremento_decremento><delimitador>

/* Last i elements are already in place */

<comentario>

swap(&arr[j], &arr[j+1]);

<identificador><delimitador><operadores_bit_a_bit><identificador><delimitador><identificador>
<delimitador><delimitador>
<operadores_bit_a_bit><identificador><delimitador><identificador><entero><delimitador><delim
itador><delimitador>

print("hola mundo");

<identificador><delimitador><cadena><delimitador><delimitador>

IV. Conclusiones

La práctica fue algo entretenida al principio y aprendí a cómo realizar el análisis de código c, aunque después se hizo tediosa por la agrupación de cada uno de los elementos. Y en verdad creo que se facilitó muchísimo el análisis y el diseño de las expresiones por que teníamos como base la referencia gramatical del lenguaje ya implementado en lex.

V. Referencias

ANSI C grammar (Lex)

ANSI C grammar (Lex). (2020). Visitado el 13 de Noviembre 2020, desde https://www.lysator.liu.se/c/ANSI-C-grammar-l.html

The Flex Manual Page

The Flex Manual Page. (2020). Visitado el 13 de Noviembre 2020, desde http://dinosaur.compilertools.net/flex/manpage.html