Práctica 4

IPN Escuela Superior de Cómputo
Ing. En Sis. Computacionales

Compiladores
M.C Saucedo Delgado Rafael Norman

2014090186
Elizalde Díaz Roberto Carlos

Introducción

En este documento se encontrará un descripción de los elementos utilizados para la elaboración de un programa con la función de analizador léxico utilizando la herramienta Flex y la especificación del lenguaje C en lex de 1985.

La herramienta Flex utilizada en esta práctica corre bajo la versión 2.6.4 en el sistema operativo Ubuntu version 18.04 LTS y VS Code como editor de texto y la versión 7.5.0 del compilador gcc para compilar los archivos generados y posteriormente realizar las pruebas correspondientes de los ejemplos descritos a continuación, todo mediante la elaboración y ejecución de un archivo Makefile.

Desarrollo

Ejemplos del lenguaje
Los siguientes ejemplos representan códigos fuente de tres programas básicos escritos en el lenguaje C que serán utilizados para probar el correcto funcionamiento del analizador léxico, producto fina de esta práctica.

Ejemplo 1

```c
#include <stdio.h>
int main() {
   // printf() muestra en pantalla la cadena entre comillas
   printf("Hello, World!");
   return 0;
}
```

Ejemplo 1. helloworld.c

Ejemplo 2

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

    int number1, number2, sum;

    printf("Introduzca dos enteros: ");
    scanf("%d %d", &number1, &number2);

    // calculando sum
    sum = number1 + number2;

    printf("%d + %d = %d", number1, number2, sum);
    return 0;
}
```

Ejemplo 2. sum.c

Ejemplo 3

```c
#include <stdio.h>
int main() {
    int num;
    printf("Enter an integer: ");
    scanf("%d", &num);

    // Verdadero si es divisble entre 2
    if(num % 2 == 0)
        printf("%d is even.", num);
    else
        printf("%d is odd.", num);

    return 0;
}
```

Ejemplo 3. evenorodd.c

Clases léxicas y expresiones regulares

```
"/*"                        { printf("<BLOQUE_COMENTADO_INICIO>") }

"auto"                  { printf("<PALABRARESERVADA_AUTO>");}
"break"                 { printf("<PALABRARESERVADA_BREAK>"); }
"case"                  { printf("<PALABRARESERVADA_CASE>"); }
"char"                  { printf("<PALABRARESERVADA_CHAR>"); }
"const"                 { printf("<PALABRARESERVADA_CONST>"); }
"continue"              { printf("<PALABRARESERVADA_CONTINUE>"); }
"default"               { printf("<PALABRARESERVADA_DEFAULT>"); }
"do"                        { printf("<PALABRARESERVADA_DO>"); }
"double"                { printf("<PALABRARESERVADA_DOUBLE>"); }
"else"                  { printf("<PALABRARESERVADA_ELSE>"); }
"enum"                  { printf("<PALABRARESERVADA_ENUM>"); }
"extern"                { printf("<PALABRARESERVADA_EXTERN>"); }
"float"                 { printf("<PALABRARESERVADA_FLOAT>"); }
"for"                       { printf("<PALABRARESERVADA_FOR>"); }
"goto"                  { printf("<PALABRARESERVADA_GOTO>"); }
"if"                        { printf("<PALABRARESERVADA_IF>"); }
"int"                       { printf("<PALABRARESERVADA_INT>"); }
"long"                  { printf("<PALABRARESERVADA_LONG>"); }
"register"              { printf("<PALABRARESERVADA_REGISTER>"); }
"return"                { printf("<PALABRARESERVADA_RETURN>"); }
"short"                 { printf("<PALABRARESERVADA_SHORT>"); }
"signed"                { printf("<PALABRARESERVADA_SIGNED>"); }
"sizeof"                { printf("<PALABRARESERVADA_SIZEOF>"); }
"static"                { printf("<PALABRARESERVADA_STATIC>"); }
"struct"                { printf("<PALABRARESERVADA_STRUCT>"); }
"switch"                { printf("<PALABRARESERVADA_SWITCH>"); }
"typedef"               { printf("<PALABRARESERVADA_TYPEDEF>"); }
"union"                 { printf("<PALABRARESERVADA_UNION>"); }
"unsigned"              { printf("<PALABRARESERVADA_UNSIGNED>"); }
"void"                  { printf("<PALABRARESERVADA_VOID>"); }
"volatile"              { printf("<PALABRARESERVADA_VOLATILE>"); }
"while"                 { printf("<PALABRARESERVADA_WHILE>"); }


{L}({L}|{D})*           { printf("<ID>"); }

0[xX]{H}+{IS}?          { printf("<CONSTANTE_ENTERO>"); }
0{D}+{IS}?              { printf("<CONSTANTE_ENTERO_OCTAL>"); }
{D}+{IS}?               { printf("<CONSTANTE_ENTERO_DECIMAL>"); }
L?'(\\.|[^\\'])+'       { printf("<CONSTANTE_CADENA>"); }

{D}+{E}{FS}?            { printf("<CONTANTE_FLOTANTE>"); }
{D}*"."{D}+({E})?{FS}?  { printf("<CONSTANTE_FLOTANTE>"); }
{D}+"."{D}*({E})?{FS}?  { printf("<CONSTANTE_FLOTANTE>"); }

L?\"(\\.|[^\\"])*\"     { printf("<CONSTANTE_CADENA>"); }
```

```
"..."                   { printf("<ELLIPSIS>"); }
">>="                   { printf("<ASIGNACION_CORRIMIENTO_D>"); }
"<<="                   { printf("<ASIGNACION_CORRIMIENTO_I>>"); }
"+="                    { printf("<ASIGNACION_SUM>"); }
"-="                    { printf("<ASIGNACION_RES>"); }
"*="                    { printf("<ASIGNACION_MUL>"); }
"/="                    { printf("<ASIGANCION_DIV>"); }
"%="                    { printf("<ASIGNACION_MOD>"); }
"&="                    { printf("<ASIGNACION_AND>"); }
"^="                    { printf("<ASIGNACION_XOR>"); }
"|="                    { printf("<ASIGNACION_OR>"); }
">>"                    { printf("<OPERADOR_CORRIMIENTO_D>"); }
"<<"                    { printf("<OPERADOR_CORRIMIENTO_I>"); }
"++"                    { printf("<OPERADOR_INCREMENTO>"); }
"--"                    { printf("<OPERADOR_DECREMENTO>"); }
"->"                    { printf("<OPERADOR_PTR>"); }
"&&"                    { printf("<OPERADOR_AND>"); }
"||"                    { printf("<OPERADOR_OR>"); }
"<="                    { printf("<OPERADOR_MENORIGUAL>"); }
">="                    { printf("<OPERADOR_MAYORIGUAL>"); }
"=="                    { printf("<OPERADOR_IGUA>"); }
"!="                    { printf("<OPERADOR_NOIGUAL>"); }
";"                     { printf("<;>"); }
("{"|"<%")          { printf("<{>"); }
("}"|"%>")          { printf("<}>"); }
","                     { printf("<,>"); }
":"                     { printf("<:>"); }
"="                     { printf("<=>"); }
"("                     { printf("<(>"); }
")"                     { printf("<)>"); }
("["|"<:")          { printf("<[>"); }
("]"|":>")          { printf("<]>"); }
"."                     { printf("<.>"); }
"&"                     { printf("<&>"); }
"!"                     { printf("<!>"); }
"~"                     { printf("<~>"); }
"-"                     { printf("<->"); }
"+"                     { printf("<+>"); }
"*"                     { printf("<*>"); }
"/"                     { printf("</>"); }
"%"                     { printf("<SIGNO_PORCIENTO>"); }
"<"                     { printf("<SIGNO_MENOR>"); }
">"                     { printf("<SIGNO_MAYOR>"); }
"^"                     { printf("<^>"); }
"|"                     { printf("<|>"); }
"?"                     { printf("<?>"); }

[ \t\v\n\f]         { printf("<CARACTER_ESCAPE>"); }
```

Pruebas (Compilación)

Compilación

flex lexico.l

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
luna@luna:~/Documentos/Desarrollo/ESCOM/Compiladores/Practica4$ flex lexico.l
luna@luna:~/Documentos/Desarrollo/ESCOM/Compiladores/Practica4$
```

gcc -c lex.yy.c

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
luna@luna:~/Documentos/Desarrollo/ESCOM/Compiladores/Practica4$ gcc -c lex.yy.c
lexico.l:115:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 yywrap()
 ^~~~~~
luna@luna:~/Documentos/Desarrollo/ESCOM/Compiladores/Practica4$
```

gcc -c main.c

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
luna@luna:~/Documentos/Desarrollo/ESCOM/Compiladores/Practica4$ gcc -c main.c
main.c: In function ‘main’:
main.c:2:5: warning: implicit declaration of function ‘yylex’ [-Wimplicit-function-declaration
]
     yylex();
     ^~~~~
luna@luna:~/Documentos/Desarrollo/ESCOM/Compiladores/Practica4$
```

gcc main.o lex.yy.o -lfl

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
luna@luna:~/Documentos/Desarrollo/ESCOM/Compiladores/Practica4$ gcc main.o lex.yy.o -lfl
luna@luna:~/Documentos/Desarrollo/ESCOM/Compiladores/Practica4$
```

Makefile

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
luna@luna:~/Documentos/Desarrollo/ESCOM/Compiladores/Practica4$ cat Makefile
lex.yy.c: lexico.l
	flex lexico.l
lex.yy.o: lex.yy.c
	gcc -c lex.yy.c
main.o: main.c
	gcc -c main.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.outluna@luna:~/Documentos/Desarrollo/ESCOM/Compiladores/Practica4$
```

Pruebas (Ejecución)

helloworld.c

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
luna@luna:~/Documentos/Desarrollo/ESCOM/Compiladores/Practica4$ ./a.out
#include <stdio.h>
int main() {
   // printf() muestra en pantalla la cadena entre comillas
   printf("Hello, World!");
   return 0;
}<ID>
```

Salida

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
}<ID>
<CARACTER_ESCAPE>
<|<|>
<ID>
<.>
<ID>
<|>|>
<CARACTER_ESCAPE>
<PALABRARESERVADA_INT>
<CARACTER_ESCAPE>
<ID>
<(>
<)>
<CARACTER_ESCAPE>
<{>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
</>
</>
<CARACTER_ESCAPE>
<ID>
<(>
<)>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
```

```
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
</>
</>
<CARACTER_ESCAPE>
<ID>
<(>
<)>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<ID>
<(>
<CONSTANTE_CADENA>
<)>
<;>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<PALABRARESERVADA_RETURN>
<CARACTER_ESCAPE>
<CONSTANTE_ENTERO_DECIMAL>
<;>
<CARACTER_ESCAPE>
```

Prueba sum.c

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
luna@luna:~/Documentos/Desarrollo/ESCOM/Compiladores/Practica4$ ./a.out
#include <stdio.h>
int main() {

    int number1, number2, sum;

    printf("Introduzca dos enteros: ");
    scanf("%d %d", &number1, &number2);

    // calculando sum
    sum = number1 + number2;

    printf("%d + %d = %d", number1, number2, sum);
    return 0;
}<ID>
```

Salida

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
}<ID>
<CARACTER_ESCAPE>
<|<|>
<ID>
<.>
<ID>
<|>|>
<CARACTER_ESCAPE>
<PALABRARESERVADA_INT>
<CARACTER_ESCAPE>
<ID>
<(>
<)>
<CARACTER_ESCAPE>
<{>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<PALABRARESERVADA_INT>
<CARACTER_ESCAPE>
<ID>
<,>
<CARACTER_ESCAPE>
<ID>
<,>
<CARACTER_ESCAPE>
<ID>
<;>
<CARACTER_ESCAPE>
```

```
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<ID>
<(>
<CONSTANTE_CADENA>
<)>
<;>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<ID>
<(>
<CONSTANTE_CADENA>
<,>
<CARACTER_ESCAPE>
<&>
<ID>
<,>
<CARACTER_ESCAPE>
<&>
<ID>
<)>
<;>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
</>
</>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
```

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<=>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<+>
<CARACTER_ESCAPE>
<ID>
<;>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<ID>
<(>
<CONSTANTE_CADENA>
<,>
<CARACTER_ESCAPE>
<ID>
<,>
<CARACTER_ESCAPE>
<ID>
<,>
<CARACTER_ESCAPE>
<ID>
<)>
<;>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<PALABRARESERVADA_RETURN>
<CARACTER_ESCAPE>
<CONSTANTE_ENTERO_DECIMAL>
<;>
<CARACTER_ESCAPE>
```

evenorodd.c salida

```
<CARACTER_ESCAPE>
<|<|>
<ID>
<.>
<ID>
<|>|>
<CARACTER_ESCAPE>
<PALABRARESERVADA_INT>
<CARACTER_ESCAPE>
<ID>
<(>
<)>
<CARACTER_ESCAPE>
<{>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<PALABRARESERVADA_INT>
<CARACTER_ESCAPE>
<ID>
<;>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<ID>
<(>
<CONSTANTE_CADENA>
<)>
<;>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<ID>
<(>
<CONSTANTE_CADENA>
<,>
<CARACTER_ESCAPE>
<&>
<ID>
<)>
<;>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
</>
</>
<CARACTER_ESCAPE>
```

<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<ID>
<CARACTER_ESCAPE>
<CONSTANTE_ENTERO_DECIMAL>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<PALABRARESERVADA_IF>
<(>
<ID>
<CARACTER_ESCAPE>
<%>
<CARACTER_ESCAPE>
<CONSTANTE_ENTERO_DECIMAL>
<CARACTER_ESCAPE>
<OPERADOR_IGUA>
<CARACTER_ESCAPE>
<CONSTANTE_ENTERO_DECIMAL>
<)>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<ID>
<(>
<CONSTANTE_CADENA>
<,>
<CARACTER_ESCAPE>
<ID>
<)>
<;>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<PALABRARESERVADA_ELSE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>

<CARACTER_ESCAPE>
<ID>
<(>
<CONSTANTE_CADENA>
<,>
<CARACTER_ESCAPE>
<ID>
<)>
<;>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<CARACTER_ESCAPE>
<PALABRARESERVADA_RETURN>
<CARACTER_ESCAPE>
<CONSTANTE_ENTERO_DECIMAL>
<;>
<CARACTER_ESCAPE>

Conclusiones

El desarrollo de esta práctica me permitió ver de manera más clara y concisa el manejo de expresiones regulares y el como pueden ser usadas para generar resultados que nos permitan resolver problemas más específicos de manera más ordenada y estructurada. También reforzó los conocimientos ya algo oxidados sobre el uso del compilador gcc, tanto para la compilación de varios archivos para, como para la detección de errores en el código sin tener que generar un ejecutable.

Personalmente encontré muy útil la herramienta flex, no solo por las posibilidades que otorga, si no también por la facilidad de la elaboración del código que conforma un archivo ".l" o la buena documentación fácilmente accesible desde la terminal.

Y al final pero no menos importante, con la realización de esta práctica puede, por primera vez, aprender la elaboración y ejecución de archivos Makefile para una más sencilla compilación de este tipos de archivos y los que actualmente me encuentro desarrollando.

Referencias

Especificación Lex del lenguaje C https://www.lysator.liu.se/c/ANSI-C-grammar-l.html
Manual Flex http://dinosaur.compilertools.net/flex/manpage.html
Brian W.Kernighan Dennis M.Ritchie, *"The C Programing Lenguage"*, *Second Edition*