INSTITUTO POLITÉCNICO NACIONAL

ESCOM

Ingeniería en sistemas computacionales

Compiladores

Práctica 4

Camacho Pérez Karen Fernanda

Grupo: 3CV

Profesor: M., en C. Rafael Norman Saucedo Delgado

Introducción

Descripción del proyecto

El proyecto que se ha elegido es el compilador del lenguaje C. Por lo tanto, en esta práctica se realizará un analizador léxico para este lenguaje.

C es un lenguaje de programación de propósito general originalmente desarrollado por Dennis Ritchie entre 1969 y 1972 en los Laboratorios Bell, como evolución del anterior lenguaje B, a su vez basado en BCPL.Al igual que B, es un lenguaje orientado a la implementación de sistemas operativos, concretamente Unix [1].

Para realizar este analizador léxico se utilizará la versión de flex 2.5.35

Desarrollo

1. Ejemplificar el lenguaje

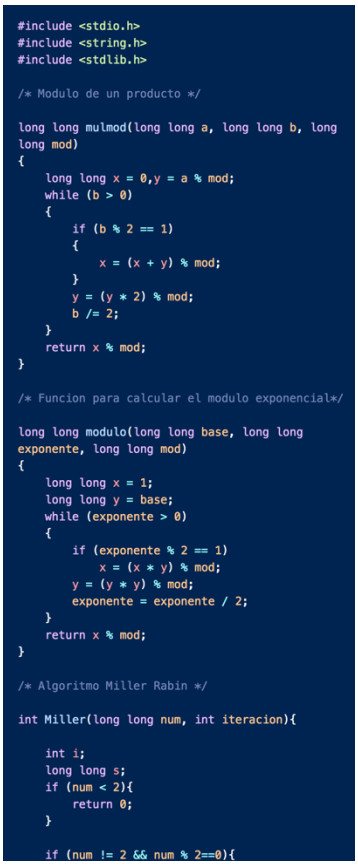
   Ejemplo de un programa de hilos en C

```c
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<unistd.h>
#include<string.h>
typedefstructdato {
inta;
charc[50];
} dato;
/* compilar con: gcc -pthread -o hilos hilos.c
debe especificarse las librerias de pthread */
void*hola_desde_hilo(void*arg)
{
    dato*a= (dato*)arg;
    printf("%s - %d\n", a->c, a->a);
    sleep(2);
    pthread_exit(NULL);`
}

intmain(intargc, char*argv[])
{
    intnum_hilos=atoi(argv[1]);
    intid[num_hilos];
    pthread_thilo[num_hilos];
    datodatos[num_hilos];
    for (inti=0; i<num_hilos; i++)
```

Ilustración 1 Programa hilos.c

Ejemplo de programa para determinar si un número es primo.



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* Modulo de un producto */

long long mulmod(long long a, long long b, long
long mod)
{
    long long x = 0,y = a % mod;
    while (b > 0)
    {
        if (b % 2 == 1)
        {
            x = (x + y) % mod;
        }
        y = (y * 2) % mod;
        b /= 2;
    }
    return x % mod;
}

/* Funcion para calcular el modulo exponencial*/

long long modulo(long long base, long long
exponente, long long mod)
{
    long long x = 1;
    long long y = base;
    while (exponente > 0)
    {
        if (exponente % 2 == 1)
            x = (x * y) % mod;
        y = (y * y) % mod;
        exponente = exponente / 2;
    }
    return x % mod;
}

/* Algoritmo Miller Rabin */

int Miller(long long num, int iteracion){

    int i;
    long long s;
    if (num < 2){
        return 0;
    }

    if (num != 2 && num % 2==0){
```

Ilustración 2 Programa primo.c

2. Identificar las clases léxicas

Clases léxicas en C: [2]

- Palabras reservadas
- Identificadores
- Literales
- Operadores
- Delimitadores
- Comentarios

3. Escribir las expresiones para cada clase léxica

- Palabras reservadas

```
19   "auto"          { printf("<Palabra Reservada>"); }
20   "break"         { printf("<Palabra Reservada>"); }
21   "case"          { printf("<Palabra Reservada>"); }
22   "char"          { printf("<Palabra Reservada>"); }
23   "const"         { printf("<Palabra Reservada>"); }
24   "continue"      { printf("<Palabra Reservada>"); }
25   "default"       { printf("<Palabra Reservada>"); }
26   "do"            { printf("<Palabra Reservada>"); }
27   "double"        { printf("<Palabra Reservada>"); }
28   "else"          { printf("<Palabra Reservada>"); }
29   "enum"          { printf("<Palabra Reservada>"); }
30   "extern"        { printf("<Palabra Reservada>"); }
31   "float"         { printf("<Palabra Reservada>"); }
32   "for"           { printf("<Palabra Reservada>"); }
33   "goto"          { printf("<Palabra Reservada>"); }
34   "if"            { printf("<Palabra Reservada>"); }
35   "int"           { printf("<Palabra Reservada>"); }
36   "long"          { printf("<Palabra Reservada>"); }
37   "register"      { printf("<Palabra Reservada>"); }
38   "return"        { printf("<Palabra Reservada>"); }
39   "short"         { printf("<Palabra Reservada>"); }
40   "signed"        { printf("<Palabra Reservada>"); }
41   "sizeof"        { printf("<Palabra Reservada>"); }
42   "static"        { printf("<Palabra Reservada>"); }
43   "struct"        { printf("<Palabra Reservada>"); }
44   "switch"        { printf("<Palabra Reservada>"); }
45   "typedef"       { printf("<Palabra Reservada>"); }
46   "union"         { printf("<Palabra Reservada>"); }
47   "unsigned"      { printf("<Palabra Reservada>"); }
48   "void"          { printf("<Palabra Reservada>"); }
49   "volatile"      { printf("<Palabra Reservada>"); }
50   "while"         { printf("<Palabra Reservada>"); }
```

Ilustración 3 Expresiones regulares, palabras reservadas C

- Identificadores

```
52   {L}({L}|{D})*        { printf("<Identificador>"); }
53
54   0[xX]{H}+{IS}?       { printf("<Constante Entera>"); }
55   0{D}+{IS}?       { printf("<Constante Entera Octal>"); }
56   {D}+{IS}?        { printf("<Constante Entera Decimal>"); }
57   L?'(\\.|[^\\'])+'    { printf("<Constante Cadena>"); }
58
59   {D}+{E}{FS}?         { printf("<Constante Flotante>"); }
60   {D}*"."{D}+({E})?{FS}?   { printf("<Constante Flotante>"); }
61   {D}+"."{D}*({E})?{FS}?   { printf("<Constante Flotante>"); }
62
```

Ilustración 4 Expresiones regulares, identificadores en C

- Literales

```
62
63    L?\"(\\.|[^\\"])*\" { printf("<Literal>"); }
64
```

Ilustración 5 Expresión regular, literales en C

- Operadores

```
66    ">>="           { printf("<Operador>"); }
67    "<<="           { printf("<Operador>"); }
68    "+="            { printf("<Operador>"); }
69    "-="            { printf("<Operador>"); }
70    "*="            { printf("<Operador>"); }
71    "/="            { printf("<Operador>"); }
72    "%="            { printf("<Operador>"); }
73    "&="            { printf("<Operador>"); }
74    "^="            { printf("<Operador>"); }
75    "|="            { printf("<Operador>"); }
76    ">>"            { printf("<Operador>"); }
77    "<<"            { printf("<Operador>"); }
78    "++"            { printf("<Operador>"); }
79    "--"            { printf("<Operador>"); }
80    "->"            { printf("<Operador>"); }
81    "&&"            { printf("<Operador>"); }
82    "||"            { printf("<Operador>"); }
83    "<="            { printf("<Operador>"); }
84    ">="            { printf("<Operador>"); }
85    "=="            { printf("<Operador>"); }
86    "!="            { printf("<Operador>"); }
```

Ilustración 6 Expresiones regulares, operadores en C

- Delimitadores

```
88    ("{"|"<%")      { printf("<Delimitador>"); }
89    ("}"|"%>")      { printf("<Delimitador>"); }
90    ","           { printf("<Delimitador>"); }
91    ":"           {printf("<Delimitador>"); }
92    "="           { printf("<Delimitador>"); }
93    "("           { printf("<Delimitador>"); }
94    ")"           { printf("<Delimitador>"); }
95    ("["|"<:")      { printf("<Delimitador>"); }
96    ("]"|":>")      { printf("<Delimitador>"); }
```

Ilustración 7 Expresiones regulares, delimitadores en C

- Comentarios

```
16    %%
17    "/*"            { printf("<Comentario>"); }
```

Ilustración 8 Expresión regular, comentario en C

4. Codificar en Lex
   En esta sección se agraga parte del código de flex, dentro del archivo lexico.l

```
/* ANALIZADOR LEXICO C CAMACHO PEREZ KF */


D           [0-9]
L           [a-zA-Z_]
H           [a-fA-F0-9]
E           [Ee][+-]?{D}+
FS          (f|F|l|L)
IS          (u|U|l|L)*

%{
#include <stdio.h>

%}

%%
"/*"            { printf("<Comentario>"); }

"auto"          { printf("<Palabra Reservada>")
; }
"break"         { printf("<Palabra Reservada>")
; }
"case"          { printf("<Palabra Reservada>")
; }
"char"          { printf("<Palabra Reservada>")
; }
"const"         { printf("<Palabra Reservada>")
; }
"continue"      { printf("<Palabra Reservada>")
; }
"default"       { printf("<Palabra Reservada>")
; }
"do"            { printf("<Palabra Reservada>")
```

```
"goto"          { printf("<Palabra Reservada>")
; }
"if"            { printf("<Palabra Reservada>")
; }
"int"           { printf("<Palabra Reservada>")
; }
"long"          { printf("<Palabra Reservada>")
; }
"register"      { printf("<Palabra Reservada>")
; }
"return"        { printf("<Palabra Reservada>")
; }
"short"         { printf("<Palabra Reservada>")
; }
"signed"        { printf("<Palabra Reservada>")
; }
"sizeof"        { printf("<Palabra Reservada>")
; }
"static"        { printf("<Palabra Reservada>")
; }
"struct"        { printf("<Palabra Reservada>")
; }
"switch"        { printf("<Palabra Reservada>")
; }
"typedef"       { printf("<Palabra Reservada>")
; }
"union"         { printf("<Palabra Reservada>")
; }
"unsigned"      { printf("<Palabra Reservada>")
; }
"void"          { printf("<Palabra Reservada>")
; }
"volatile"      { printf("<Palabra Reservada>")
; }
"while"         { printf("<Palabra Reservada>")
; }

{L}({L}|{D})*       { printf("<Identificador>")
```

5. Pruebas

Las siguientes imágenes muestran la salida del analizador léxico con los programas ejemplificados anteriormente.

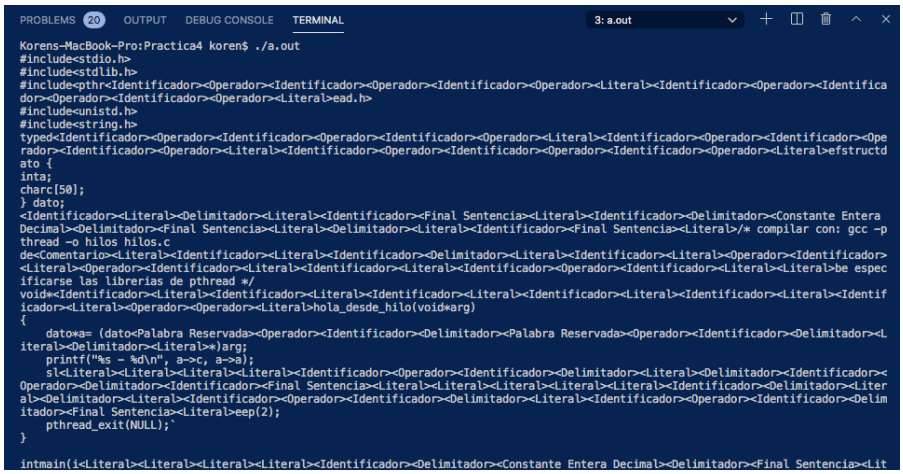


Ilustración 9 Salida Analizador Léxico programa hilos.c

Ilustración 10 Salida analizador léxico, programa primo.c

Conclusiones

Esta práctica fue relativamente rápida y considero que mis conocimientos, aún muy básicos, de Flex se reforzaron. Creo que ya pude entender mucho mejor cómo es que se codifican las expresiones regulares para los lenguajes de programación en Lex.

Referencias

[1] [Online]. Available: https://www.bell-labs.com/usr/dmr/www/chist.html.
[2] [Online]. Available: https://elvex.ugr.es/decsai/c/apuntes/tokens.pdf.