Instituto Politécnico Nacional

Escuela Superior de Computo

Ingeniería en Sistemas Computacionales(ISC)

Practica 4

Compiladores

Profesor:
M. en C. Saucedo Delgado Rafael Norman

Autor:
Barón Hernández Diego Ismael
2015630044

3CV5

Contents

1 Introducción

Para este proyecto se pretende hacer un compilador de lenguaje C a ensamblador con la ayuda de la herramienta flex para la parte del analizador lexico del lenguaje C, se desarrollara en el sistema operativo UNIX/Ubuntu 20.04 con la versión 2.6.4 de flex.

2 Desarrollo

2.1 Ejemplos del lenguaje

A continuación se mostraran ejemplos de programas escritos en lenguaje C, estos son sencillos y demostrativos.
El primer ejemplo que tenemos es el de main que es la función principal, dentro de esta se pone todo lo que se pretende ejecutar, es de tipo int ya que regresa un valor entero que hacer referencia al código de estado en el que termino la ejecución del programa y tiene un void en los parámetros para indicar que no es necesario que reciba algún dato para su funcionamiento.

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

Ahora si vemos dentro del ejemplo anterior podemos ver otro elemento importante como lo son las llamadas a funciones, en este caso estamos usando la llamada al analizador léxico generado por flex. Otro elemento importante son los ciclos y condicionales, a continuación se muestran ejemplos de estos:

```
int main(void){
        int i = 0;
        for(i = 1; i < 10; i++){
                if(i%2 == 0){
                        printf("i es par\n");
                }
        }
        return 0;
}
```

Otra cosa importante a destacar de este ejemplo es la declaración de variables, en este caso fue una variable de tipo int llamada i que utilizamos en el ciclo for, luego volvemos a usar en el if con el operador modulo para comprobar si el valor actual de i es par y si lo es usamos la función printf.

2.2 Clases Léxicas

Ahora bien al observar los ejemplos podemos empezar a darnos cuenta de algunas clases léxicas como son:

- Identificadores (funciones,nombres de variables, etc.)
- Secuencias de escape (return, break, continue, etc.)
- Constantes numéricas (enteros,flotantes)
- Constantes alfabéticas (char)
- Ciclos(for,while,do while)
- Control de flujos(if,if else)
- Operadores(lógicos y numéricos)

A grandes rasgos tenemos estas clases léxicas pero para un desarrollo posterior nos facilitara identificar los componentes de estas clases por separado, esto lo mostraremos mas adelante en el código LEX.

2.3 Expresiones regulares

Para la identificación de algunos elementos se utilizaran las siguientes expresiones regulares:

- 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)*

2.4 Código LEX

A continuación se muestra el código en flex para lo anterior:

```
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>
%}
/*Se expondran las clases lexicas del lenguaje c*/
%%
"/*"                    {printf("<comentario-abierto>"); }
"*/"                    {printf("<comentario-cerrado>"); }
"auto"                  {printf("<AUTO>"); }
"case"                  {printf("<CASE>"); }
"char"                  {printf("<CHAR>"); }
"const"                 {printf("<CONST>"); }
"continue"              {printf("<CONTINUE>"); }
"default"               {printf("<DEFAULT>"); }
"do"                    {printf("<DO>"); }
"double"                {printf("<DOUBLE>"); }
"else"                  {printf("<ELSE>"); }
"enum"                  {printf("<ENUM>"); }
"extern"                {printf("<EXTERN>"); }
"float"                 {printf("<FLOAT>"); }
"for"                   {printf("<FOR>"); }
"goto"                  {printf("<GOTO>"); }
"if"                    {printf("<IF>"); }
"int"                   {printf("<INT>"); }
"long"                  {printf("<LONG>"); }
"register"              {printf("<REGISTER>"); }
"return"                {printf("<RETURN>"); }
"short"                 {printf("<SHORT>"); }
"signed"                {printf("<SIGNED>"); }
"sizeof"                {printf("<SIZEOF>"); }
"static"                {printf("<STATIC>"); }
"struct"                {printf("<STRUCT>"); }
"switch"                {printf("<SWITCH>"); }
"typedef"               {printf("<TYPEDEF>"); }
"union"                 {printf("<UNION>"); }
"unsigned"              {printf("<UNSIGNED>"); }
"void"                  {printf("<VOID>"); }
"volatile"              {printf("<VOLATILE>"); }
"while"                 {printf("<WHILE>"); }

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

0[xX]{H}+{IS}?          {printf("<INTEGER-CONSTANT>"); }
0{D}+{IS}?              {printf("<OCTAL-CONSTANT>"); }
{D}+{IS}?               {printf("<DECIMAL-CONSTANT>"); }
L?'(\\.|[^\\'])+'       {printf("<STRING-CONSTANT>"); }

{D}+{E}{FS}?            {printf("<FLOAT-CONSTANT>"); }
```

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

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

"..."                     { printf("<ELLIPSIS>"); }
">>="                     { printf("<RIGHT_ASSIGN>"); }
"<<="                     { printf("<LEFT_ASSIGN>"); }
"+="                      { printf("<ADD_ASSIGN>"); }
"-="                      { printf("<SUB_ASSIGN>"); }
"*="                      { printf("<MUL_ASSIGN>"); }
"/="                      { printf("<DIV_ASSIGN>"); }
"%="                      { printf("<MOD_ASSIGN>"); }
"&="                      { printf("<AND_ASSIGN>"); }
"^="                      { printf("<XOR_ASSIGN>"); }
"|="                      { printf("<OR_ASSIGN>"); }
">>"                      { printf("<RIGHT_OP>"); }
"<<"                      { printf("<LEFT_OP>"); }
"++"                      { printf("<INC_OP>"); }
"--"                      { printf("<DEC_OP>"); }
"->"                      { printf("<PTR_OP>"); }
"&&"                      { printf("<AND_OP>"); }
"||"                      { printf("<OR_OP>"); }
"<="                      { printf("<LE_OP>"); }
">="                      { printf("<GE_OP>"); }
"=="                      { printf("<EQ_OP>"); }
"!="                      { printf("<NE_OP>"); }
";"                       { printf("<;>"); }
("{"|"<%")                { printf("<{>"); }
("}"|"%>")                { printf("<}>"); }
","                       { printf("<,>"); }
":"                       { printf("<:>"); }
"="                       { printf("<=>"); }
"("                       { printf("<(>"); }
")"                       { printf("<)>"); }
("["|"<:")                { printf("<[>"); }
("]"|":>")                { printf("<]>"); }
"."                       { printf("<.>"); }
"&"                       { printf("<&>"); }
"!"                       { printf("<!>"); }
"~"                       { printf("<~>"); }
"-"                       { printf("<->"); }
"+"                       { printf("<+>"); }
"*"                       { printf("<*>"); }
"/"                       { printf("</>"); }
"%"                       { printf("<MOD>"); }
```

```
"<"                     { printf("<GE_OP>"); }
">"                     { printf("<LE_OP>"); }
"^"                     { printf("<^>"); }
"|"                     { printf("< | >"); }
"?"                     { printf("<?>"); }

[ \t\v\n\f]             { printf(""); }
.                       { printf("<ignore bad characters>"); }

%%
```

2.5 Pruebas

Ahora veremos algunas pruebas del funcionamiento de este programa ya compilado, durante las pruebas se puso el ejemplo anterior en el que se verificaba si el valor de i es par a lo largo de un ciclo for, estos fueron los resultados:

```
int a = 0;
<INT><IDENTIFIER><=><CONSTANT><;>
for(int i = 0;i<10;i++)
<FOR><(><INT><IDENTIFIER><=><CONSTANT><;><IDENTIFIER><GE_OP><CONSTANT><;><IDENTIFIER><INC_OP><)>
if(i%2 == 0)
<IF><(><IDENTIFIER><MOD><CONSTANT><EQ_OP><CONSTANT><)>
printf("i es par");
<IDENTIFIER><(><STRING_LITERAL><)><;>
return 0;
<RETURN><CONSTANT><;>
```

Figure 1: Primera prueba del analizador léxico

Podemos observar que los resultados mostrados en consola son correctos con respecto a las entradas del programa, identifica correctamente los elementos de las sentencias, hay que señalar que los espacios en blanco y otros elementos de formato los omite.

3 Conclusiones

Obtuve un refresco de lo que es la programación en C lo cual me vino bastante bien ya que tiene mucho tiempo que no programo en este lenguaje, dentro de las dudas que me quedaron fueron dentro de las expresiones regulares las de E,FS,IS por mas que busque no logre encontrar a que se referían esas abreviaciones, también en la subclase IDENTIFIER en el original utilizaban una función llamada checktype esta todavía no estaba completa, por lo que intente completarla y me quedo la duda de como haría esta para identificar el tipo de la entrada.

4 Referencias

Kernighan, Brian; Ritchie, Dennis M. (Marzo de 1988). The C Programming Language (2nd edición). Englewood Cliffs, NJ: Prentice Hall. ISBN 0-13-110362-8.