Instituto Pólitecnico Nacional

Soy pólitecnico porque aspiro a ser todo un hombre

Escuela Superior De Cómputo

Ingeniería en Sistemas Computacionales

Analizador léxico con flex

Profesor: M.C. Saucedo Delgado Rafael Norman

Alumno: Ricardo Erick Torres Rosas

Materia: Compiladores

10-Octubre-2020

Índice

Índice de figuras

1. Introducción

1.1. Analizador léxico: flex

El analizador léxico, Flex, es open source que surge después de Lex, el cual es un analizador léxico de AT&T. La primera vez que conocí flex fue en el libro "El dragon morado" y de ahí he buscado información páginas web.

1.1.1. Descripción de flex

De acuerdo a [1] "Flex es una herramienta para generar escáneres: programas que reconocen patrones léxicos en un texto. flex lee los ficheros de entrada dados, o la entrada estándar si no se le ha indicado ningún nombre de fichero, con la descripción de un escáner a generar. La descripción se encuentra en forma de parejas de expresiones regulares y código C, denominadas reglas. flex genera como salida un fichero fuente en C, lex.yy.c, que define una rutina yylex(). Este fichero se compila y se enlaza con la librería -lfl para producir un ejecutable. Cuando se arranca el fichero ejecutable, este analiza su entrada en busca de casos de las expresiones regulares. Siempre que encuentra uno, ejecuta el código C correspondiente."

1.1.2. Versión de Flex y Ubuntu

La versión de flex se ocupará es flex 2.6.4, sobre el sistema operativo Ubunto 20.04.

- Instalación de flex:

```
sudo apt-get install flex
```

- Versión de flex

```
flex --version
```

- Versión de Ubuntu

```
lsb_release -a
```

1.2. Descripción del proyecto

El lenguaje de programación que se ocupará para el desarrollo de está práctica es el lenguaje TRRE, un lenguaje desarrollado por mi. La idea de este lenguaje surge por el gusto a la programación y a la fotografía.

La intención de este lenguaje es poder trabajar con operaciones básicas que se le pueden aplicar a una imagen, como sacar su negativo, rotar, escala de grises, etc.

El lenguaje TRRE es compilado para dar como salida un archivo .py, el cual es el que realizara las funciones que trabajaran con las imágenes.

Una ventaja de este lenguaje es la salida, la cual sera una imagen diferente a la original y esto lo conseguiremos sin la necesidad de contar con tantos recursos.

2. Desarrollo

2.1. Ejemplificar lenguaje

- Ejemplo 1.

```
impotar ``c:/user/erick/Desktop/imagen.jpg''
etiqueta = imagen1
imagen1.escalagrises
imagen1.rotar
exportar (``c:/user/erick/Desktop/imagenSalida.jpg'', imagen1)
```

- Ejemplo 2.

```
impotar ``c:/user/erick/Desktop/imagen.jpg''
etiqueta = imagen
impotar ``c:/user/erick/Desktop/imagen1.jpg''
etiqueta = imagen1
imagen1.negativo
imagen2 = imagen + imagen1
exportar  (``c:/user/erick/Desktop/imagenSalida.jpg'', imagen 2)
```

2.2. Identificar clases léxicas

```
palabraReservada        ``importar''|``etiqueta''|``exportar''
funciones               ``rotar''|``negativo''|``escalagrises''
letra                   [a-zA-Z:]
cadena                  {letra}({letra})*
jpg                     {letra}({letra})*{punto}{letra}({letra})*
digito                  [0-9]
numero                  {digito}({digito})*
espacio                 [ \t]
igual                   [=]
comillas                ["]
diagonal                [/]
punto                   [.]
abreParentesis          [(]
cieraParentesis         [)]
sumar                   [+]
```

2.3. Expresiones regulares

```
{jpg}                   {printf(``>>es una imagen jpg\n'');}
{palabraReservada}      {printf(``>>es una palabra reservada\n'');}
{funciones}             {printf(``>>es una funcion\n'');}
{letra}                 {printf(``>>es una letra\n'');}
{cadena}({numero}{cadena}|{cadena}{numero})*
                        {printf(``>>es una cadena\n'');}

{numero}                {printf(``>>es un digito\n'');}
{digito}                {printf(``>>es un digito\n'');}
{espacio}               {printf(``>>es un espacio\n'');}
{igual}                 {printf(``>>es un igual\n'');}
{comillas}              {printf(``>>comillas\n'');}
{diagonal}              {printf(``>>es una diagonal\n'');}
{punto}                 {printf(``>>es un punto\n'');}
{abreParentesis}        {printf(``>>abre parentesis\n'');}
{cieraParentesis}       {printf(``>>cierra parentesis\n'');}
{sumar}                 {printf(``>>es una suma\n'');}
{palabraReservada}({espacio})*{igual}({espacio})*{cadena}
                        {printf(">>es una asignacion de etiqueta\n'');}

{palabraReservada}({espacio})*{comillas}({cadena}{diagonal})*{jpg}{comillas}
                        {printf(">>importar imagen\n'');}
```

2.4. Códigos

2.4.1. 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.out
```

2.4.2. lexico.l

```
%{
#include<stdio.h>
%}
palabraReservada        "importar"|"etiqueta"|"exportar"
funciones               "rotar"|"negativo"|"escalagrises"
letra                   [a-zA-Z:]
cadena                  {letra}({letra})*
jpg                     {letra}({letra})*{punto}{letra}({letra})*
digito                  [0-9]
numero                  {digito}({digito})*
espacio                 [ \t]
igual                   [=]
comillas                ["]
diagonal                [/]
punto                   [.]
abreParentesis          [(]
cieraParentesis         [)]
sumar                   [+]

%%
{jpg}                   {printf(">>es una imagen jpg\n");}
{palabraReservada}      {printf(">>es una palabra reservada\n");}
{funciones}             {printf(">>es una funcion\n");}
{letra}                 {printf(">>es una letra\n");}
{cadena}({numero}{cadena}|{cadena}{numero})*            {printf(">>es una cadena\n");}
{numero}                {printf(">>es un digito\n");}
{digito}                {printf(">>es un digito\n");}
{espacio}              {printf(">>es un espacio\n");}
{igual}                 {printf(">>es un igual\n");}
{comillas}              {printf(">>comillas\n");}
{diagonal}              {printf(">>es una diagonal\n");}
{punto}                 {printf(">>es un punto\n");}
{abreParentesis}        {printf(">>abre parentesis\n");}
{cieraParentesis}       {printf(">>cierra parentesis\n");}
{sumar}                 {printf(">>es una suma\n");}
{palabraReservada}({espacio})*{igual}({espacio})*{cadena} {printf(">>es una asignacion de etiqueta\n");}
{palabraReservada}({espacio})*{comillas}({cadena}{diagonal})*{jpg}{comillas} {printf(">>importar imagen\n");}
%%
```

2.4.3. main.c

```
int main(void){

yylex();

return 0;
}
```

2.5. Pruebas

2.5.1. pasos para el .out

ls para ver los datos en la carpeta actual, esta sentencia no es necesaria, pero nos ayuda a ver los archivos que tenemos.

```
erick@unknown:~/compiladores$ ls
lexico.l  main.c  Makefile
```

Figura 1: ls

make run, ejecuta todas las sentencias escritas en el codigo.

```
erick@unknown:~/compiladores$ make run
gcc -c main.c
main.c: In function 'main':
main.c:4:2: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
    4 |  yylex(); /* invoca/inicia el autómata del analizador léxico */
      |  ^~~~~
flex lexico.l
lexico.l:41: warning, la regla no se puede aplicar
gcc -c lex.yy.c
gcc main.o lex.yy.o -lfl
./a.out
```

Figura 2: make run

después de ejecutar make run, volvemos a usar ls, para ver el contenido de la carpeta

```
erick@unknown:~/compiladores$ ls
a.out  lexico.l  lex.yy.c  lex.yy.o  main.c  main.o  Makefile
```

Figura 3: ls

Limpiamos archivos que ya no ocuparemos con make clean

```
erick@unknown:~/compiladores$ make clean
rm -f a.out main.o lex.yy.o lex.yy.c
erick@unknown:~/compiladores$ ls
lexico.l  main.c  Makefile
```

Figura 4: make clean

2.5.2. Pruebas del a.out

ejecutamos con ./a.out y a probar.

```
erick@unknown:~/compiladores$ ./a.out
h
>>es una letra

hola
>>es una cadena

importar
>>es una palabra reservada

=
>>es un igual

+
>>es una suma

("hola")
>>abre parentesis
>>comillas
>>es una cadena
>>comillas
>>cierra parentesis
```

```
imagen2 = imagen1 + imagen
>>es una cadena
>>es un espacio
>>es un igual
>>es un espacio
>>es una cadena
>>es un espacio
>>es una suma
>>es un espacio
>>es una cadena
```

Figura 5: prueba 1 y 2

```
"c:/user/erick/desktop/imagen.jpg"
>>comillas
>>es una cadena
>>es una diagonal
>>es una cadena
>>es una diagonal
>>es una cadena
>>es una diagonal
>>es una cadena
>>es una diagonal
>>es una imagen jpg
>>comillas
```

```
importar "hola.jpg"
>>importar imagen

importar "c:/user/erick/desktop/hola.jpg"
>>importar imagen
```

Figura 6: prueba 3 y 4

3. Conclusiones

En el proceso de la práctica fui adquiriendo conocimiento del manejo del archivo lexico.l, la práctica me gustó pues aunque aun no logro ver todo el trasfondo del poder de flex, he estado leyendo mas sobre flex, este te retorna tokens los cuales puedes manejar para procesar tu lenguaje. Referente al formato de esta práctica es el que me gustaría entregar en todas las prácticas aunque no me ha sido posible por el manejo de mis tiempos, tengo mucho por aprender, pero cada día avanzo un poco más. El proyecto es algo que me motiva, porque siento que puedo aprender a la par dos cosas que me gustan, tanto a realizar un compilador, como el proceso de imágenes, lamentablemente aun no logró saber de manera cuantificable a que alcance llegaré.

4. Referencias

[1]. Ubuntu Manpage: flex - generador de analizadores léxicos rápidos [Internet]. Manpages.ubuntu.com. 2020 [cited 13 November 2020]. Available from: http://manpages.ubuntu.com/manpages/bionic/es/man1/flex.1.html

[2]. Aho A. Compiladores. Pearson Addison Wesley, SA de CV; 2008.