Universidad Católica San Pablo

Compiladores

Carlos Eduardo Atencio Torres

Semestre 2020-II

Práctica 1 – Regex e introducción a FLEX

Objetivo:

Construir un analizador léxico utilizando la herramienta FLEX

Herramientas:

FLEX: Es una herramienta que nos permite generar analizadores léxicos. Tiene su propia sintaxis y pide como requisito tener un conocimiento básico sobre manipulación de expresiones regulares.

Para instalar en linux:

sudo apt-get install flex

Para instalar en windows:

http://gnuwin32.sourceforge.net/packages/flex.htm

1. EXPRESIONES REGULARES

También llamadas REGEX, se trata de una secuencia de caracteres que conforman un patrón de búsqueda. Usualmente usadas para buscar o buscar y reemplazar cadenas.

Wikipedia

Símbolos más comunes

?	Ninguna o una ocurrencia
*****	Ninguna o varias ocurrencias
+	Mínimo una ocurrencias
{n}	Exactamente n ocurrencias
|	“o”

[ ]	Opciones
.	El punto significa “cualquier cosa”
$	Fin de línea

Símbolos comodines (*):

\d	dígito
\s	cadena
( )	agrupación

(*) Los símbolos comodines pueden o no estar presentes en el lenguaje de programación escogido.

Caracteres especiales

\n	Salto de línea
\t	Tabulación

Si queremos por ejemplo reconocer “.” como punto, debemos de utilizar el caracter “\”. Por ejemplo: \. \? \+

OBS: Tener cuidado al colocarlo en el patrón de un lenguaje de programación pues el “\” debe colocarse con “\\” (doble), así: “\\.” “\\?” “\\+”

Las expresiones regulares pueden ser programadas en diferentes lenguajes de programación. Por ejemplo para C++ a partir de la std 11 ya tiene soporte para regex. En python existe la biblioteca **re**:

1. Verificar si una cadena está conformada por números:

```
import re

p = re.compile('[0-9]+$')

if re.match(p):
    print("SI ES UN NUMERO")
else:
    print("NO ES UN NUMERO")
```

2. Obtener todas las palabras de un texto:

```
import re

regstr = '[a-z]+'
text = 'abc cde 777ghi  jkl999mno'
groups = re.findall(regstr, text)

for group in groups:
      print( group )
```

Ejercicios:

1. Reconocer una dirección ip.
   - Que acepte valores correctos desde 0.0.0.0 hasta 299.299.299.299
   - Importante: A pesar que los valores van hasta 255 en un IP válido, suponga que puede aceptar hasta 299.
2. Reconocer el nombre de una variable (que no comience por número)
3. Identificar en solicitudes, el tema de solicitud. Por ejemplo:

Estimado profesor, recientemente he estado preocupado por muchos aspectos de su curso y eso ha ocasionado que no me concentre correctamente en sus clases. Si es que no apruebo su curso tendré problemas en casa porque solo tengo este curso y no trabajo. Por ese motivo, me gustaría pedirle más información sobre los temas recientemente tocados. Estaría muy agradecido y en verdad le deseo una bonita tarde. Hasta luego.

Proponga al menos 3 formas de reconocer una solicitud y construya sus regex.

2. FLEX

1) Crearemos un ejemplo que será un programa para borrar todos los espacios en blanco y tabuladores de los extremos de las líneas.

- En un archivo llamado test.l (test punto ele), colocaremos la siguiente sintaxis:

```
1. %%
2. [\*]+$  ;
```

(desconsidere los números)

Entonces la línea 1, %% significa que el programa comienza allí.

2) Luego ejecutaremos en consola:

flex test.l

si no hubiera errores aparecerán el archivo **lex.yy.c**

3) Compilares el archivo lex como una librería:

gcc lex.yy.c -L/lib -lfl

4) Esto nos creará un archivo ejecutable (**a.out** en linux o **a.exe** en windows)

5) Creamos un archivo de prueba.txt:

```
Este es un *******_ _ _
programa con  ***
varios asteriscos finales **
al final *
```

_ Los subguiones apenas para representar espacios en blanco

6) Ejecutar:

./a.out < prueba.txt > salida.txt

7). Verificamos que el archivo de salida no tenga asteriscos al final de la fila, excepto por la primera línea.

Ejercicios:

1. En un ejercicio similar, quite los números al final de la línea