



Instituto Politécnico Nacional
Escuela Superior de Cómputo



Compiladores

Practica 4: Flex

Nombre: Mauro Sampayo Hernández

Grupo: 3CV17

Profesor: Hernández Olvera Luis Enrique

Fecha de entrega: 20 de octubre del 2021

1. Introducción:

1.1 Flex

Flex, también conocida como Lex, es una herramienta para generar escáneres, es decir programas que reconocen patrones léxicos en un texto a partir de la especificación de expresiones regulares para describir dichos patrones.

El compilador Lex transforma los patrones de entrada en un diagrama de transición y genera código, en un fichero fuente en C llamado "lex.yy.c", que simula este diagrama de transición. 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, y siempre que encuentra uno, ejecuta el código C correspondiente.

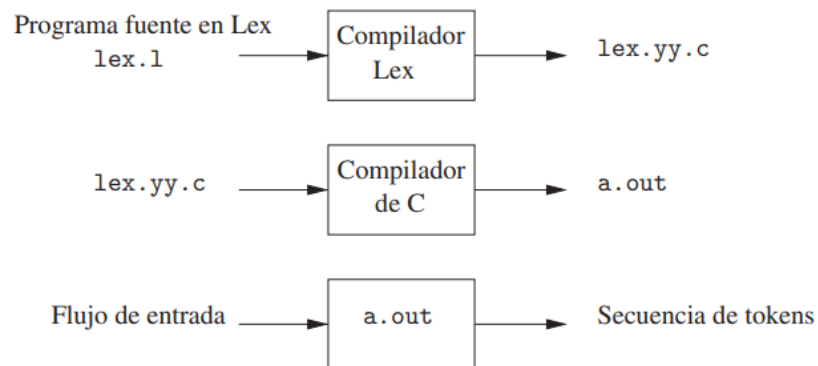


Figura 1. Creación de un analizador léxico en Lex

1.2 Estructura de los programas en Lex

Un programa en Lex tiene la siguiente forma:

Definiciones

%%

Reglas de traducción

%%

Código de usuario

1.2.1 Definiciones

La sección de Definiciones incluye las declaraciones de variables, constantes de manifiesto (identificadores que se declaran para representar a una constante; por ejemplo, el nombre de un token) y definiciones regulares. Las definiciones de nombre tienen la forma:

nombre definición

El **nombre** es una palabra que comienza con una letra o un subrayado “_” seguido por cero o más letras, dígitos, subrayados “_” o guiones “-”. La **definición** se considera que comienza en el primer carácter que no sea un espacio en blanco siguiendo al nombre y continuando hasta el final de la línea. Posteriormente se puede hacer referencia a la definición utilizando “{**nombre**}”, que se expandirá a “({**definición**})”. Por ejemplo,

DIGITO [0-9]
*ID [a-z][a-z0-9]**

define “DIGITO” como una expresión regular que empareja un dígito sencillo, e “ID” como una expresión regular que empareja una letra seguida por cero o más letras o dígitos. Una referencia posterior a

*{DIGITO}+.”.{DIGITO}**

es idéntica a

*([0-9])+.”.([0-9])**

y empareja uno o más dígitos seguido por un punto “.” seguido por cero o más dígitos.

1.2.2 Reglas de Traducción

La sección de Reglas de Traducción tiene una serie de reglas con la siguiente forma:

Patrón {Acción}

Donde el **patrón** es una expresión regular, la cual puede usar las definiciones regulares de la sección de declaraciones, y debe ir sin sangrar. Las **acciones** son fragmentos de código, por lo general escritos en C, que definen las acciones a ejecutar en caso de que se encuentre alguna concordancia con el patrón de texto de entrada, y que deben comenzar en la misma línea que el **patrón**.

1.2.3 Código de Usuario

Finalmente, la sección de Código de Usuario contiene las funciones adicionales que se pueden utilizar en las acciones y que se copia a "lex.yy.c" literalmente. La presencia de esta sección es opcional. Si se omite, el segundo "%%" en el fichero de entrada se podrá omitir también.

En las secciones de definiciones y reglas, cualquier texto sangrado o encerrado entre "%{' y `%}" se copia íntegramente a la salida (sin los %{'s). Los %{'s deben aparecer sin sangrar en líneas ocupadas únicamente por estos.

En la sección de reglas, cualquier texto o %{' sangrado que aparezca antes de la primera regla podrá utilizarse para declarar variables que son locales a la rutina de análisis y (después de las declaraciones) al código que debe ejecutarse siempre que se entra a la rutina de análisis. Cualquier otro texto sangrado o %{' en la sección de reglas sigue copiándose a la salida, pero su significado no está bien definido y bien podría causar errores en tiempo de compilación.

En la sección de definiciones (pero no en la sección de reglas), un comentario sin sangría (es decir, una línea comenzando con "/*") también se copia literalmente a la salida hasta el próximo "*/".

1.3 Patrones

Los patrones en la entrada se escriben utilizando un conjunto extendido de expresiones regulares. Estas son:

x → Empareja el carácter "x"

. → Cualquier carácter (byte) excepto una línea nueva

[xyz] → Una "clase de caracteres"; en este caso, el patrón empareja una "x", una "y", o una "z".

[abj-oZ] → Una "clase de caracteres" con un rango; en este caso empareja una "a", una "b", cualquier letra desde la "j" hasta la "o", o una "Z".

[^A-Z] → Una "clase de caracteres negada", es decir, cualquier carácter menos los que aparecen en la clase. En este caso, cualquier carácter EXCEPTO una letra mayúscula.

[^A-Z\n] → Cualquier carácter EXCEPTO una letra mayúscula o una línea nueva.

r* → Cero o más r's, donde r es cualquier expresión regular.

r+ → Una o más r's.

r? → Cero o una r (es decir, "una r opcional").

r{2,5} → De dos a cinco r's.

r{2,} → Dos o más r's.

r{4} → Exactamente 4 r's.

{nombre} → La expansión de la definición de "nombre".

"[xyz]"foo → La cadena literal: [xyz]"foo.

\x → Si x es una "a", "b", "f", "n", "r", "t", o "v", entonces la interpretación ANSI-C de \x. En otro caso, un literal "x" (usado para indicar operadores tales como "**")

\0 → Un caracter NUL (código ASCII 0).

\123 → El caracter con valor octal 123.

\x2a → El caracter con valor hexadecimal 2a.

(r) → Empareja una r; los paréntesis se utilizan para anular la precedencia (ver más abajo).

rs → La expresión regular r seguida por la expresión regular s; se denomina "concatenación".

r|s → Bien una r o una s.

r/s → Una r pero sólo si va seguida por una s.

^r → Una r, pero sólo al comienzo de una línea (es decir, justo al comienzo del análisis, o a la derecha después de que se haya analizado una línea nueva).

r\$ → Una r, pero sólo al final de una línea (es decir, justo antes de una línea nueva).

<s>r → Una r, pero sólo en la condición de arranque s.

<s1,s2,s3>r → Una r, pero en cualquiera de las condiciones de arranque s1, s2, o s3.

<*>r → Una r en cualquier condición de arranque, incluso una exclusiva.

<<EOF>> → Un fin-de-fichero.

<s1,s2><<EOF>> → Un fin-de-fichero en una condición de arranque s1 o s2.

2. Problemas enfrentados al programar la práctica:

Los problemas enfrentados durante la realización de esta práctica fueron principalmente los relacionados a la creación de las expresiones regulares, pues algunas expresiones tales como las de las operaciones aritméticas chocaban con las relacionadas a la de los números decimales y enteros con signo durante la evaluación de las cadenas de texto debido a la presencia de los símbolos de suma “+” y resta “-” precediendo un número en ambos patrones.

Para solucionar dicho problema se optó por realizar una nueva definición llamada “REAL” que tiene definida una expresión regular que considera la aparición tanto de números enteros como de números decimales, los cuáles pueden o no contar con signo para que de esta manera no hubiese errores durante la evaluación de las sumas y restas con números reales con signo.

Finalmente se debe mencionar que en general para todas las expresiones regulares definidas para cada uno de los puntos solicitados por el profesor, se debieron afrontar algunos retos pequeños, tales como el de investigar la estructura que conforman los textos aceptados en cada caso (principalmente este problema se presentó al momento de generar la expresión regular del nombre de las variables, al ser que en ANSI C se deben cumplir ciertas reglas para que el nombre de una variable pueda ser aceptado y validado por el compilador).

3. Pruebas y Capturas de Pantalla:

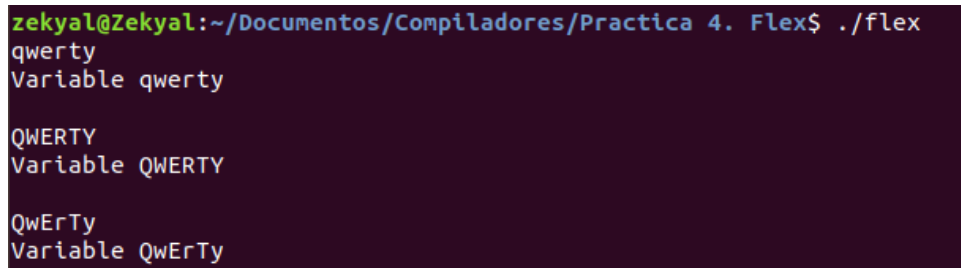
Se realizó el desarrollo de expresiones regulares en Flex que reconocen determinados patrones léxicos en ANSI C propuestos por el profesor, y los cuales se enlistarán a continuación junto a algunas pruebas realizadas con el objetivo de mostrar su correcto funcionamiento.

3.1 Nombres de Variables

Esta expresión regular reconoce cualquier nombre que se le pueda asignar a una variable en ANSI C. El nombre de una variable en ANSI C puede tener letras de la "A" a la "Z", ya sea en mayúsculas o minúsculas siempre y cuando pertenezcan al alfabeto inglés. También se puede hacer uso del símbolo **guion bajo** "_", y de números, aunque estos últimos no pueden aparecer como primer carácter del nombre de la variable.

A continuación, se muestran algunas pruebas realizadas con algunos casos que podrían presentarse al evaluar un nombre de variable en ANSI C con dicha expresión regular:

- Caso en el que el nombre de la variable se compone de puras letras mayúsculas y/o minúsculas:



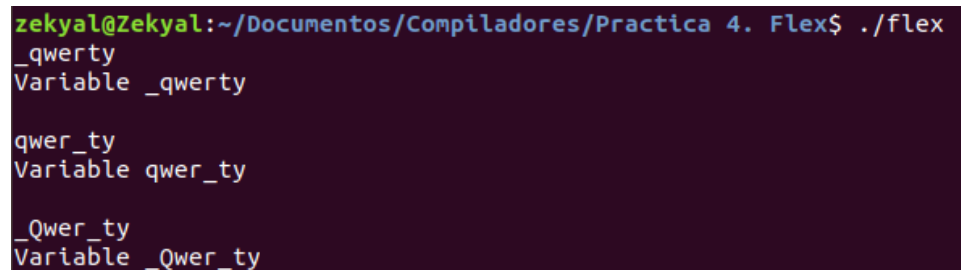
```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
qwerty
Variable qwerty

QWERTY
Variable QWERTY

QwErTy
Variable QwErTy
```

Figura 2. Nombres de variables con puras letras mayúsculas y minúsculas

- Caso en el que el nombre de la variable tiene uno o más guiones bajos "_". Cabe destacar que independientemente de donde se encuentre escrito el guion bajo "_" este será válido:



```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
_qwerty
Variable _qwerty

qwer_ty
Variable qwer_ty

_Qwer_ty
Variable _Qwer_ty
```

Figura 3. Nombres de variables con guión bajo "_"

- Caso en el que el nombre de la variable tenga números. Cabe destacar que cuando el nombre inicia con un número, Flex evaluará el número por separado y lo empatará como un “Número Entero”, para después evaluar el resto de la entrada como el nombre de una variable. Esto sucede debido a que los nombres de las variables en ANSI C no pueden iniciar con valores numéricos:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
qwerty24
Variable qwerty24

24qwer_ty
Numero Entero 24
Variable qwer_ty

2qwerty_2
Numero Entero 2
Variable qwerty_2
```

Figura 4. Nombres de variables con caracteres numericos

3.2 Números Enteros con y sin signo

Esta expresión regular se encarga de reconocer números enteros de cualquier longitud, independientemente de si estos cuentan con un signo (“+” ó “-“) o no.

A continuación, se muestran algunas pruebas realizadas con algunos casos que podrían presentarse al evaluar un número entero con dicha expresión regular:

- Caso en el que el número no tiene signo:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
234
Numero Entero 234

12
Numero Entero 12
```

Figura 5. Números enteros sin signo

- Caso en el que el número tiene signo:

```

zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
-234
Numero Entero -234

+12
Numero Entero +12

-23 +12 -654
Numero Entero -23
Numero Entero +12
Numero Entero -654

```

Figura 6. Números enteros con signo

3.3 Números en forma decimal con y sin signo

Esta expresión regular se encarga de reconocer números decimales de cualquier longitud ya sea en la parte entera o en la parte decimal del mismo; independientemente de si estos cuentan con un signo (“+” o “-”) o no.

A continuación, se muestran algunas pruebas realizadas con algunos casos que podrían presentarse al evaluar un número decimal con dicha expresión regular:

- Caso en el que el número no tiene signo:

```

zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
234.56
Numero Decimal 234.56

3.141592653589
Numero Decimal 3.141592653589

```

Figura 7. Números decimales sin signo

- Caso en el que el número tiene signo:

```

zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
-2.3
Numero Decimal -2.3

+12.454
Numero Decimal +12.454

-3.1416 +234234.32453 -0.1
Numero Decimal -3.1416
Numero Decimal +234234.32453
Numero Decimal -0.1

```

Figura 8. Números decimales con signo

3.4 Potencia de números reales (con o sin signo)

Esta expresión regular reconoce una operación de potenciación, la cual debe contar con una base y un exponente reales, así como también el símbolo de potencia “^” para su correcta validación.

A continuación, se muestran algunas pruebas realizadas con algunos casos que podrían presentarse al evaluar una Potencia con dicha expresión regular:

- Caso en el que los valores de la potencia no tengan símbolo:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
3^3
Potencia 3^3

4.55^3.33
Potencia 4.55^3.33
```

Figura 9. Potencias con valores numéricos sin signo

- Caso en el que los valores de la potencia tengan símbolo:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
-3^-4
Potencia -3^-4

+5.6^+6
Potencia +5.6^+6

-4.76583465^3.333333
Potencia -4.76583465^3.333333
```

Figura 10. Potencias con valores numéricos con signo

3.5 Operaciones Matemáticas

Para este caso no se cuenta con una sola expresión regular, sino que con varias, las cuáles evalúan diversas operaciones aritméticas y que se enlistan a continuación:

3.5.1 Suma

Esta expresión regular reconoce una operación de adición o suma, la cual debe contar con dos sumandos reales y el símbolo de suma “+” para su correcta validación.

A continuación, se muestran algunas pruebas realizadas con algunos casos que podrían presentarse al evaluar una Suma con dicha expresión regular:

- Caso en el que los valores de la suma no tengan símbolo:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
2+2
Suma 2+2

3.4+6.66
Suma 3.4+6.66
```

Figura 11. Sumas con valores numéricos sin signo

- Caso en el que los valores de la suma tengan símbolo:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
-2.3+-6.6
Suma -2.3+-6.6

+5++7
Suma +5++7

-3.34+6
Suma -3.34+6
```

Figura 12. Sumas con valores numéricos con signo

- Algunos casos adicionales:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
6+6+6
Suma 6+6
Numero Entero +6

6+7-2
Suma 6+7
Numero Entero -2
```

Figura 13. Casos especiales de la Suma

3.5.2 Resta

Esta expresión regular reconoce una operación de resta, la cuál debe contar con un minuendo y un sustraendo reales, así como también el símbolo de resta “-” para su correcta validación.

A continuación, se muestran algunas pruebas realizadas con algunos casos que podrían presentarse al evaluar una Resta con dicha expresión regular:

- Caso en el que los valores de la resta no tengan símbolo:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
2-2
Resta 2-2

6.6-8
Resta 6.6-8
```

Figura 14. Restas con valores numéricos sin signo

- Caso en el que los valores de la resta tengan símbolo:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
-6.66--4
Resta -6.66--4

+5-+6.7
Resta +5-+6.7

-6-5
Resta -6-5
```

Figura 15. Restas con valores numéricos con signo

- Algunos casos adicionales:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
6-9-8
Resta 6-9
Numero Entero -8

4-5+3.3
Resta 4-5
Numero Decimal +3.3
```

Figura 16. Casos especiales de la Resta

3.5.3 Multiplicación

Esta expresión regular reconoce una operación de resta, la cual debe contar con un multiplicando y un multiplicador real, así como también el símbolo de multiplicación “*” para su correcta validación.

A continuación, se muestran algunas pruebas realizadas con algunos casos que podrían presentarse al evaluar una Multiplicación con dicha expresión regular:

- Caso en el que los valores de la multiplicación no tengan símbolo:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
5*5
Multiplicacion 5*5

3.4*6.66
Multiplicacion 3.4*6.66
```

Figura 17. Multiplicaciones con valores numéricos sin signo

- Caso en el que los valores de la multiplicación tengan símbolo:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
-5*-5
Multiplicacion -5*-5

+6*+6.6
Multiplicacion +6*+6.6

-12*7
Multiplicacion -12*7
```

Figura 18. Multiplicaciones con valores numéricos con signo

3.5.4 División

Esta expresión regular reconoce una operación de resta, la cual debe contar con un dividendo y un divisor real, así como también el símbolo de división “/” para su correcta validación.

A continuación, se muestran algunas pruebas realizadas con algunos casos que podrían presentarse al evaluar una División con dicha expresión regular:

- Caso en el que los valores de la división no tengan símbolo:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
10/5
Division 10/5

12.5/5.66
Division 12.5/5.66
```

Figura 19. Divisiones con valores numéricos sin signo

- Caso en el que los valores de la división tengan símbolo:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
-5/-6
Division -5/-6

+5/+9
Division +5/+9

-4.555/8
Division -4.555/8
```

Figura 20 Divisiones con valores numéricos con signo

3.5.5 Módulo

Esta expresión regular reconoce una operación de Modulo, la cual debe contar con un dividendo y un divisor real, así como también el símbolo de Modulo “%” para su correcta validación.

A continuación, se muestran algunas pruebas realizadas con algunos casos que podrían presentarse al evaluar un módulo con dicha expresión regular:

- Caso en el que los valores del módulo no tengan símbolo:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
10%5
Modulo 10%5

13.56%4.44
Modulo 13.56%4.44
```

Figura 21. Modulos con valores numéricos sin signo

- Caso en el que los valores del módulo tengan símbolo:

```
zekyal@Zekyal:~/Documentos/Compiladores/Practica 4. Flex$ ./flex
-12%-4
Modulo -12%-4

+6%3.33
Modulo +6%3.33

-256%8
Modulo -256%8
```

Figura 22. Modulos con valores numéricos con signo

4. Conclusión:

A manera de conclusión puedo decir que la herramienta Flex, es muy útil al permitir el generar escáneres para el reconocimiento de patrones léxicos en textos a partir de expresiones regulares, lo que nos permite llevar a cabo la construcción y comprobación de expresiones que nos permitan evaluar determinados patrones léxicos utilizados para el desarrollo de compiladores tales como el de palabras reservadas, nombres de variables, operaciones tanto lógicas como aritméticas, definición de valores numéricos o cadenas de texto, etc.; para su posterior implementación dentro de un compilador que se esté desarrollando.

Adicionalmente se puede añadir que la herramienta de Flex brinda una estructura y un método de compilación que resultan bastante fáciles de entender y llevar a cabo para el desarrollo de expresiones regulares.

5. Referencias:

- Aho, A., 2011. Compiladores. 2nd ed. Pearson Educación de México, SA de CV, pp.140-144.
- Ajpdsoft.com. Introducción al lenguaje de programación ANSI C Imprimible Proyecto AjpdSoft. [online] Disponible en: <<https://www.ajpdsoft.com/modules.php?name=News&file=print&sid=432#variablesdeclaracion>> [Accedido el 19 de octubre de 2021].
- Paxson, V., 1995. Flex - un generador de analizadores léxicos. [online] Es.tldp.org. Disponible en: <<http://es.tldp.org/Manuales-LuCAS/FLEX/flex-es-2.5.html#SEC5>> [Accedido el 19 de octubre de 2021].