UNIVERSIDAD NACIONAL DE COLOMBIA

Primer conjunto de ejercicios - Examen final

Compiladores

Nombre: Laura Daniela Molina Villar

Ejercicios sección 3.3

Ejercicio 3.3.1

a) Lenguaje C

(i) Siempre está incluido ASCII, pero dependiendo del compilador podría codificar Unicode o UTF-8.

(ii) *Enteros*: una secuencia de dígitos que puede contar con ciertos prefijos y sufijos. Se considera como un octal si la secuencia comienza con 0, y solo puede estar conformado por dígitos del 0 al 7. Si la secuencia de dígitos está precedida por 0x o 0X se considera un numero hexadecimal. Los dígitos hexadecimales incluyen [aA] a [fF], que tienen valores de 10 a 15. Si la secuencia termina con los sufijos [lL] tradicionalmente indican constantes enteras de tipo long.

Flotantes: consta de una parte entera, un punto decimal, una parte fraccionaria, un [eE] y un exponente entero opcionalmente con signo. Las partes de números enteros y fracciones constan de una secuencia de dígitos. Puede faltar la parte entera o la parte fraccionaria (pero no ambas). Puede faltar el punto decimal o [eE] y el exponente (pero no ambos).

(iii) Un identificador es una secuencia de letras, dígitos y guiones bajos (_). El primer carácter no puede ser un dígito. Las letras mayúsculas y minúsculas son distintas. La longitud del identificador es ilimitada.

b) Lenguaje C++

(i) Siempre está incluido ASCII, pero dependiendo del compilador podría codificar Unicode o UTF-8.

(ii) *Enteros*: una secuencia de dígitos que puede contar con ciertos prefijos y sufijos. Se considera como un octal si la secuencia comienza con 0, y solo puede estar conformado por dígitos del 0 al 7. Si comienza con otro dígito, se considera un decimal. Si la secuencia de dígitos está precedida por 0x o 0X se considera un numero hexadecimal. Los dígitos hexadecimales incluyen [aA] a [fF], que tienen valores de 10

a 15. Una constante decimal cuyo valor excede el entero con signo más grande se considera long; y la constante octal o hexadecimal que excede el entero sin signo más grande también se considera long; de lo contrario, las constantes enteras se toman como int.

Flotantes: consta de una parte entera, un punto decimal, una parte fraccionaria, un [eE] y un exponente entero opcionalmente con signo. Las partes de números enteros y fracciones constan de una secuencia de dígitos. Puede faltar la parte entera o la parte fraccionaria (pero no ambas). Puede faltar el punto decimal o [eE] y el exponente (pero no ambos).

(iii) Un identificador es una secuencia de letras, dígitos y guiones bajos (_). El primer carácter no puede ser un dígito. Las letras mayúsculas y minúsculas son distintas. La longitud del identificador es ilimitada.

c) Lenguaje C#

(i) Unicode, UTF-16.

(ii) *Enteros*: una secuencia de dígitos que puede contar con ciertos prefijos y sufijos. Solo cuenta con dos representaciones: decimal o hexadecimal. Si la secuencia de dígitos está precedida por 0x o 0X se considera un numero hexadecimal. Puede terminar con los sufijos [lL] y [uU].

Flotantes: consta de una parte entera, un punto decimal, una parte fraccionaria, un [eE], un exponente entero opcionalmente con signo y los sufijos [fF], [dD], [mM]. Las partes de números enteros y fracciones constan de una secuencia de dígitos. Puede faltar la parte entera o la parte fraccionaria (pero no ambas). Puede faltar el punto decimal o [eE] y el exponente, y si faltan los tres anteriores, debe estar presente alguno de los sufijos, de lo contrario los sufijos son opcionales.

(iii) Un identificador es una secuencia de letras, dígitos y guiones bajos (_). El primer carácter no puede ser un dígito. Las letras mayúsculas y minúsculas son distintas. Se permite utilizar @ como prefijo para usar palabras clave como identificadores, aunque el @ no es realmente parte del identificador.

d) Lenguaje Fortran

(i) Letras del alfabeto inglés en mayúsculas y minúsculas, dígitos arábigos y los siguientes caracteres especiales: espacio ' " ( ) * + - / : = _ ! & $; < >

(ii) *Enteros*: una secuencia de dígitos que puede comenzar con [+ -].

Reales: cuenta con dos representaciones, decimales y exponenciales. Los decimales tienen una parte entera, un punto decimal y una parte fraccionaria, además pueden comenzar con [+ -]. La parte entera y

fraccionaria constan de una secuencia de dígitos. Los exponenciales cuentan con una representación decimal, seguido por [eE] y un entero.

(iii) Un identificador no tiene más de 31 caracteres. El primer carácter debe ser una letra. Los caracteres restantes, si los hay, pueden ser letras, dígitos o (_). Los identificadores no distinguen entre mayúsculas y minúsculas.

e) Lenguaje Java

(i) Unicode, UTF-16.

(ii) *Enteros*: una secuencia de dígitos que puede comenzar con [+ -].

Reales: cuenta con dos representaciones, decimales y exponenciales. Los decimales tienen una parte entera, un punto decimal y una parte fraccionaria, además pueden comenzar con [+ -]. La parte entera y fraccionaria constan de una secuencia de dígitos. Los exponenciales cuentan con una representación decimal, seguido por [eE] y un entero.

(iii) Un identificador es una secuencia de longitud ilimitada de letras, dígitos, ($), y caracteres de conexión. El primer caracter no pude ser un dígito.

f) Lenguaje Lisp

(i) Unicode.

(ii) *Enteros*: una secuencia de dígitos que puede comenzar con [+ -].

Reales: cuenta con una parte entera, un punto y una secuencia de dígitos después del punto.

(iii) Un identificador es una secuencia de longitud ilimitada de letras, dígitos y [_ - * (. ?]

g) Lenguaje SQL

(i) Unicode, UTF-8.

(ii) *Enteros*: una secuencia de dígitos que puede comenzar con [+ -]. Contiene un máximo de 128 dígitos.

Reales: cuenta con una parte entera, un punto, una secuencia de dígitos, que es la parte fraccionaria, [eE] y una secuencia de dígitos con signo opcional. Puede faltar la parte entera o la parte fraccionaria (pero no ambas). Puede faltar el punto decimal o [eE] y el exponente (pero no ambos).

(iii) Un identificador es una secuencia de letras, dígitos y (_). Debe comenzar con una letra. Los identificadores no distinguen entre mayúsculas y minúsculas, y tienen un límite de 128 caracteres.

Ejercicio 3.3.2

d) L = { bbb, abbb, abbba, aababab ...} describe todas las cadenas conformadas por a's y b's, que contienen exactamente 3 b's.

e) L = { ϵ, aabb, bbaabbbb, aaabba, abab...} describe todas las cadenas conformadas por a's y b's, con numero par de a's y b's.

Ejercicio 3.3.4

Para poder obtener cualquier combinación de mayúsculas y minúsculas de la palabra "select" sería necesario usar la siguiente definición regular:
select → [Ss][Ee][Ll][Ee][Cc][Tt]

Ejercicio 3.3.5

a) cons → [b-df-hj-np-tv-z]
string → cons*a(a|cons)*e(e|cons)*i(i|cons)*o(o|cons)*u(u|cons)*

b) string → a*b*c*...z*

c) letter → [a-zA-Z]
comment → /* (letter|"*/")*(letter)* */

d) string → string → [^ ([0-9]*0[0-9]*0[0-9]* | [0-9]*1[0-9]*1[0-9]* | ... | [0-9]*9[0-9]*9[0-9]*)*]

e) string → ([0-9]*0[0-9]*0[0-9]* | [0-9]*1[0-9]*1[0-9]* | ... | [0-9]*9[0-9]*9[0-9]*)*

f) string → (aa|bb)*((ab|ba)(aa|bb)*(ba|ab)(aa|bb)*)*((ab|ba)(bb|aa)*a|b)

g) string → [kqrbnp0-8]?[-x]?[kqrbn0-8]?

h) string → b*(ab?)*

i) string → b*a*b?a*

Ejercicio 3.3.7

\"\\

Ejercicio 3.3.8

Se va a usar la siguiente nomenclatura:

- UT = Unión de todos los caracteres
- UT-c = Unión de todos los caracteres menos c
- UT-cs = Unión de todos los caracteres menos c y menos s

Expresión	Complemento	Equivalente
c	^c	(UT-c)
\c	^\c	(UT-c)
"c"	^"c"	Sea c = $c_1c_2c_3...c_n$, entonces .*(UT-c_1)c_1(UT-c_1).*(UT-c_2)c_2(UT-c_2) ... .*(UT-c_n)c_n(UT-c_n).*
.	^.	ϵ
r*	^r*	(UT-r)*
r+	^r+	(UT-r)*
r?	^r?	(UT-r)*
r_1r_2	^r_1r_2	.*(UT-r_1)r_1(UT-r_1).*(UT-r_2)r_2(UT-r_2)
$r_1\|r_2$	^$r_1\|r_2$	(UT-r_1r_2)

Ejercicio 3.3.10

a) Si ^ tiene a la izquierda un paréntesis cuadrado y a la derecha una expresión, entonces es el complemento, de lo contrario indica el inicio de la cadena.

b) Si ^ tanto como $ pueden ser reemplazados con la expresión r_1/r_2. En el primer caso sería r_1 sería \n y r_2 sería .*; en el segundo caso r_1 sería la expresión a evaluar y r_2 sería \n.

Ejercicio 3.3.11

- 's': s
- \': \'
- *: [unión de todos los caracteres]*
- ?: [unión de todos los caracteres]
- [s]: [s]

Ejercicio 3.3.12

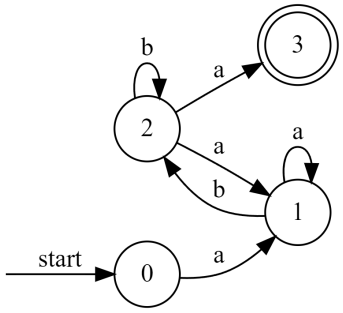
Se pueden reemplazar los caracteres %, _, y e por: .*, .{m}, siendo m la cantidad de veces seguidas que se ponga el caracter _ y \respectivamente. Ejemplos de patrones en SQL reemplazados con expresiones regulares:

- C%: C.*
- __C%: .{2}C.*
- %r_m%: .*r.{1}m.*
- 15e%: 15\%

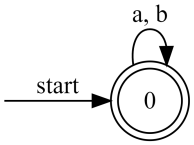
Ejercicios sección 3.4

Ejercicio 3.4.1

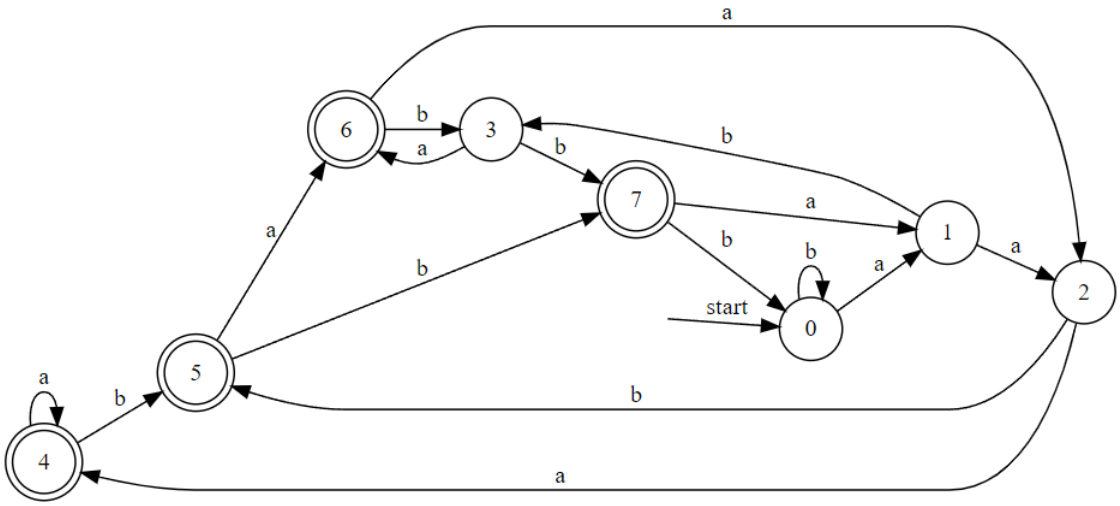
a)


b
a
3
2
a
a
b
1
start
0
a

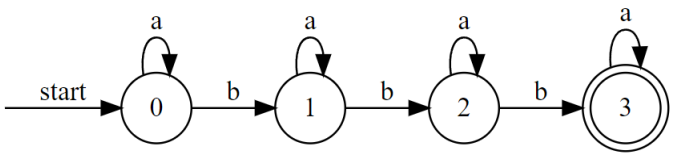
b)


a, b
start
0

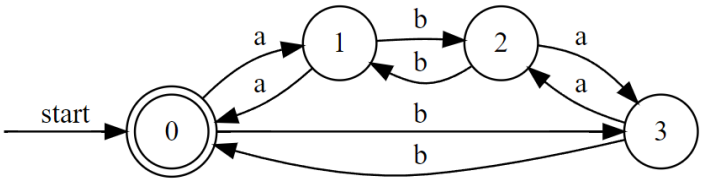
c)


a
6
b
3
a
b
b
7
a
1
a
2
b
b
a
start
0
a
5
b
a
b
4
a

d)


a
a
a
a
start
0
b
1
b
2
b
3

e)


b
a
1
2
a
b
a
a
start
0
b
3
b

Ejercicio 3.4.3 (Ver en Anexos el código del Ejercicio 3.4.3)

a) abababaab

s	1	2	3	4	5	6	7	8	9
$f(s)$	0	0	1	2	3	4	5	1	2

b) aaaaaa

s	1	2	3	4	5	6
$f(s)$	0	1	2	3	4	5

c) abbaabb

s	1	2	3	4	5	6	7
$f(s)$	0	0	0	1	1	2	3

Ejercicio 3.4.4

1. Cuando s = 1:
   Como t = 0, entonces tiene dos opciones:
   - Si $b_2 = b_1$ entonces $t \leftarrow 1$ y $f(2) \leftarrow 1$
   - Si $b_2 \neq b_1$ entonces $f(2) \leftarrow 0$
2. Cuando s = m: asumimos que se calcula correctamente el ciclo.
3. Cuando s = m + 1: sabemos hasta s = m se ha calculado correctamente el ciclo, entonces veamos que pasa específicamente cuando s = m + 1, tenemos dos opciones:
   - Si $t = 0$, es porque nos encontramos comparando el primer carácter de la cadena, entonces hay dos caminos
     - Cuando $b_{s+1} = b_1$, $f(s+1) \leftarrow 1$
     - Cuando $b_{s+1} \neq b_1$, $f(s+1) \leftarrow 0$
   - Si $t > 0$ entonces estamos comparando un carácter mayor al primero, si esto pasa es porque los $t-1$ caracteres anteriores ya fueron validados, por lo tanto los $s-t-1$ caracteres anteriores también ya fueron validados. Si $b_{s+1} \neq b_{t+1}$ entonces t va a ir disminuyendo hasta que $b_{s+1} = b_{t+1}$ o hasta que $t = 0$
     - En el caso que $t = 0$, sale del while, y se pueden dar los dos casos que se presentaron cuando $t = 0$
     - Pero si $b_s + 1 = b_t + 1$, y $t = h > 0$, sale del while y entra a la condición de $b_{s+1} = b_{t+1}$, por lo cual ahora $f(s+1) \leftarrow h+1$, esto es válido porque recordemos que t fue disminuyendo su valor en cada ciclo del while, por lo que h es menor a t , pero si antes de ingresar al while, $t-1$ caracteres ya habían sido validados, por lo mismo $h-1$ caracteres también ya habrán sido validados.

Ejercicio 3.4.5

La variable t solo puede incrementar 1 en cada iteración, entonces t al iniciar cada iteración s tiene un valor de a lo mucho $s - 1 < n$. Este incremento se hace fuera del while por lo tanto es de valor constante. Cuando entra al while, t disminuye al menos en 1, por lo cual el while se corre a lo mucho s veces (ya que la condición del while se corre incluso cuando t = 0), y si corre esa cantidad de veces significa que t disminuyó un valor de $s - 1$ unidades, por lo tanto en la siguiente iteración no puede disminuir más, y aumentaría en 1; si disminuyó $s - 2$ en la siguiente iteración puede disminuir a lo mucho 1 unidad; si disminuyó $s - 3$ en la siguiente iteración puede disminuir a lo mucho 2 unidades, y asi sucesivamente. Luego el while se podría volver a correr s veces (es decir llevaría un total de $2s$ veces de haber corrido), solo cuando la iteración actual sea $2s$, por lo tanto cuando la iteración actual sea n se habrá corrido a lo mucho n veces.

Ejercicio 3.4.6 (Ver en Anexos el código del Ejercicio 3.4.6)

a) Verdadero

b) Falso

Ejercicio 3.4.9 (Ver en Anexos el código del Ejercicio 3.4.9)

a) El tamaño de s_n será igual al Fibonacci número n f_n.

b) s_6 = abaababa

s	1	2	3	4	5	6	7	8
$f(s)$	0	0	1	1	2	3	2	3

c) s_7 = abaababaabaab

s	1	2	3	4	5	6	7	8	9	10	11	12	13
$f(s)$	0	0	1	1	2	3	2	3	4	5	6	4	5