Ejercicio 2, capitulo 1:

¿Cuales son los cinco cualidades que usted como usuario considera mas importantes en un compilador que compras.

1º Pruebas de software

2º Optimizacion del programa

3º Deteccion de codigo malicioso

4º Diseño de nuevas arquitecturas informaticas.

5º Programacion dinamica - seccion de instrucciones

¿Esa lista cambia cuando Tu eres el Compilador?

En principio, depurar programas que van a ser interpretados es mas facil que depurar programas a compilar por los siguientes razones.

- Con el interprete tienes una sola version del ejecutable.
- Hay menos errores especificos de plataforma cuando se usa un interprete

Por lo tanto esa lista no cambia ya que trabajaria con un interprete.

Ejercicio 3, Capitulo 1.

- Muchas aplicaciones para la Tecnologia de compiladores.
- Generacion de codigo maquina para lenguajes de alto nivel.
- Pruebas de software
- Diseño de nuevas arquitecturas informaticas
- Tecnicas de optimizacion, seleccion de instrucciones
- Interprete de consultas de base de datos

10, 11, 12, 14, 15, 16 Capitulo 2.

10) Muestra que el conjunto de lenguajes regulares que esta cerrado bajo la interseccion

• La reflexión de un lenguaje L, que se expresa como LR, es el lenguaje formado por las reflexiones de todas sus cadenas por ejemplo:

si $L = \{001, 10, 111\}$,

entonces $LR = \{001, 10, 111\}$

* Propiedades de clausura de los regulares

Ademas por el camino hemos visto:

1° La union de dos lenguajes regulares es regular. Los regulares son cerrados por la operacion de la union.

2° La concatenación de dos lenguajes regulares es regular. Los regulares son cerrados por la operacion de concatenación.

3° La estrella de Kleene de un lenguaje regular es regular. Los regulares son cerrados por la operacion de concatenación.

Otras propiedades de clausura de los regulares.

1° El complementario de un lenguaje regular es regular. Los regulares son cerrados por la operacion de complemento.

2° El reverso de un lenguaje regular es regular. Los regulares son cerrados por la operacion de reverso.

3° La intersección de dos lenguajes regulares es regular. Los regulares son cerrados por la operacion de interseccion

El complementario y el reverso son muy faciles, la interseccion tiene algo mas de trabajo.

$\{w \in \{a,b\} \mid |w|_a = 2n, |w|_b = 2m, \quad n, m \in N\} =$

$= \{w \in \{a,b\} \mid |w|_a = 2n, n \in N\} \cap \{w \in \{a,b\} \mid |w|_b = 2m, m \in N\}$

Capitulo 2).

```
T ← {DA, {D-DA}};
P ← Ø
while (P ≠ T) do
P ← T;
T ← Ø;
for each set p ∈ P do
T ← T ∪ Split(p);
end;
end;

Split(S) {
For each c ∈ Σ do
if c splits S into S1 and S2
        then return {S1, S2};
end;
return S;
}
```

12. Construya un DFA para cada una de las siguientes construcciones del lenguaje c, y luego construya la tabla correspondiente para una implementación basada en tablas para cada uno de ellos:

a. Constantes enteras

b. Identificadores

c. Comentarios

13. Para cada uno de los dfas del ejercicio anterior, cree un escáner de código directo.

```
DIGIT   [0-9]
ID      [a-z] [a-z0-9]*
%%
{DIGIT}+     {
          printf ("An integer: %s (%d)\n", yytext,
                  atoi (yytext));
          }
{DIGIT}+ "." {DIGIT}*     {
          printf ( "A float : %s  (%g)\n", yytext
               atof (yytext));
     }
| then | begin | end | procedure | function
printf ( "A keyword: %s\n", yytext );
}
```

12) Capitulo 2
Construya un DFA para cada una de las siguientes construcciones del lenguaje C, y luego construya la tabla correspondiente para la implementación basada en tablas para cada uno de ellos:

a) Constantes enteras
b) Identificadores
c) Comentarios

13) Capitulo 2
Para cada uno de los dfas del ejercicio anterior cree un escaner de codigo directo.

UN DFA ~~[illegible]~~ que acepta $aa^*|bb^*$

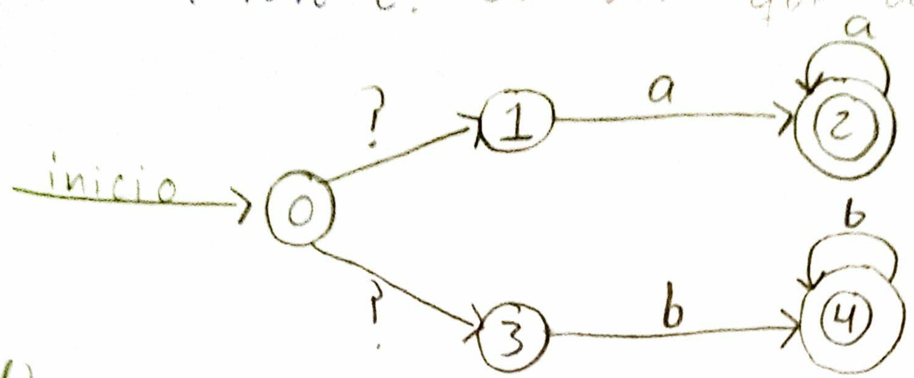
Un automata finito determinista (AFD) es un caso especial de un automata finito no determinista en el cual:

1. Ningun estado tien transición, es decir una transición con la entrada ?, y.
2. Para cada estado s y cada simbolo de entrada a, hay exactamente una arista etiquetada a que sale de s.

Un automata finito determinista tiene una transición desde cada estado con cualquier entrada. Si se esta usando una tabla de transiciones para representar la funcion de transicion de un AFD, Entonces cada entrada de transicionesnes un solo estado.

Como consecuencia es muy facil determinar si un Automata finito determinista acepta o no una cadena de entrada, puesto que hay a lo sumo un estado desde el camino de inicio etiquetado con una o eso cadena.

12) Capitulo 2. Un DFA que acepta $aa^*|bb^*$



Q: {1, 2, 3, 4}

Σ: {a, b}

S: {0}

F: {2, 4}.

Un AFN que reconoce $aa^*|bb^*$.

- La cadena aaa es aceptada recorriendo los estados 0, 1, 2, 2 y 2.
- Los etiquetas de estas aristas son ? a, a y a, cuya concatenacion es aaa. Observece, que los simbolos ? "desaparecen" en una concatenacion.

Simulaccion de un AFD en C.
Entrada:
Una cadena de entrada "x" que termina con un caracter de fin de archivo eof. Un AFD D con un estado de inicio "s0" y un conjunto "F" de estados de Aceptacion.
Salida:
La respuesta "si", si D acepta "x", "no" en caso contrario.

Aplicar el algoritmo a la cadena de entrada "x". La funcion mover (s, c) da el estado al cual hay una transicion desde el estado "s" en un caracter de entrada "c". La funcion "sgtecar" devuelve el siguiente caracter de la cadena de entrada "x". Simulaccion de un AFD

```
s:=s0;
c:=sgtecar(car);
while c<> eof do
      s:= movers (s, c);
      c:=sg tecar (car);
end
if s esta en F then
      return "si"
else. retur "no";
```

12) Capitulo 2 "Ejemplo No 2"

DFA $(a|b)^*abb$

$S = \{0, 1, 2, 3\}$

$\Sigma = \{a, b\}$

$S\epsilon = \{0\}$

3 = Estado de Aceptacion.

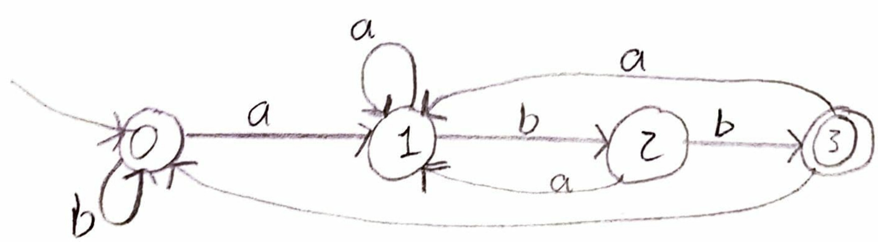
Lo siguiente Funcion define dicho Automata

$$F = (\{0, 1, 2, 3\}, \{a, b\}, T, 0, \{3\})$$

Donde T es la tabla de Transicion que refleja el patron de la expresion regular que estamos evaluando $T(s_i, c) = s_{i+1}$

	simbolos de entrada	
Estado	a	b
0	1	0
1	1	2
2	1	3
3	1	0

Grafo de Transición.



Consideremos que la Función de transición (siguiente estado) desde un estado s en un caracter de entrada c, esta dado por una operacion de seccion de estado (case en s). Ademas consideremos la existencia de una funcion sigtecar() que devuelve el siguiente caracter de la cadena de entrada, Obteniendo el siguiente algoritmo:

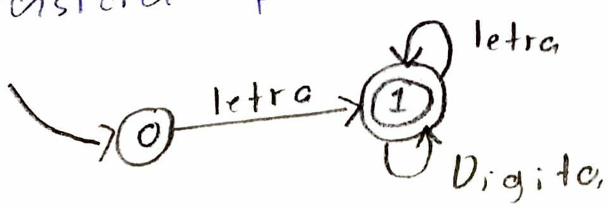
Codigo "C"

```
s = 0;
c = sigtecar (cadena);
while (c != '\') {
    case (s) {
    0: s = (c == 'b') ? 0 : 1
    1: s = (c == 'b') ? 2 : 1
    2: s = (c == 'b') ? 3 : 1
    3: s = (c == 'b') ? 0 : 1
    }
    c = sigtecar (cadena)
}
if (s == 3) "Cadena valida"
```

Ejem con Identificador

Pensemos en la definicion regular para un identificador que esta compuesta por la siguiente expresión regular letra (letra | digito)*, para este estado o caso consideremos el conjunto de estados como ?{0, 1}, el alfabeto de entrada estara compuesto de las definiciones regulares letra y digito. El 0 es considerado como el inicio y 1 como el fin de la transición.

De esta manera generamos el siguiente diagrama de transicion para un identificador.



14) Capitulo 3:

Dado un lenguaje regular o, lo que es lo mismo, dado un automata finito deterministo, puede construirse una maquina de turing que lo simule haciendo:

$Q' = Q \cup \{q'\}$

$\Sigma' = \Sigma$

$\Gamma = \Sigma \cup \{h\}$

$F' = \{q'\}$

$\delta'(q,\sigma) = (\delta(q,\sigma), \sigma, R)$

Es decir que si L es regular entonces tambien es recursivo.

De este modo tambien podemos afirmar que si L es independiente del contextos entonces tambien es recursivo.

14) Capitulo 2.

Dado un lenguaje regular o, lo que es lo mismo, dado un automata finito determinista, puede construirse una maquina de turin que lo simule haciendo:

$Q' = Q \cup \{q'\}$

$\Sigma' = \Sigma$

$\Gamma = \Sigma \cup \{K\}$

$F' = \{q'\}$

$\delta'(q,\sigma) = (\delta(q,\sigma),\sigma, R)$

Es, decir que si L es regular entonces tambien es recursivo.

De este modo tambien podemos afirmar que si L es independiente del contextos entonces tambie es recursivo.