



**INSTITUTO POLITÉCNICO
NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**



**Reporte de práctica:
Buscador de palabras reservadas de C**

Grupo:
4CM2

Nombre del alumno:
Maldonado Hernández Alexander

Unidad de aprendizaje:
Teoría de la computación

Maestro:
Genaro Juárez Martínez

Fecha de entrega:
22 de junio de 2025

Índice

1. Introducción	1
2. Desarrollo	2
2.1. Diseño del NFA	2
2.2. Conversión del NFA a DFA	4
2.3. Implementación del analizador léxico	6
2.4. Visualización gráfica del DFA	8
3. Conclusiones	10
4. Referencias	10
5. Anexos	11
5.1. Anexo 1. Código fuente del programa	11
5.2. Anexo 2. Código de C analizado por el DFA (prueba.c)	24

1. Introducción

En esta práctica se desarrolló un sistema automatizado capaz de reconocer todas las palabras reservadas del lenguaje de programación C mediante el uso de autómatas finitos. El objetivo principal fue implementar un autómata finito no determinista (NFA) que reconociera dichas palabras, transformarlo en un autómata finito determinista (DFA) mediante el algoritmo de subconjuntos, y emplear dicho DFA para analizar archivos de texto o código fuente con el fin de identificar las palabras reservadas, contando su frecuencia de aparición y registrando sus posiciones dentro del archivo.

La práctica se dividió en varias etapas fundamentales. En primer lugar, se diseñó el NFA correspondiente a las palabras reservadas del compilador C, lo cual implicó modelar todas las posibles secuencias de caracteres que conforman dichas palabras. Posteriormente, se realizó el proceso de conversión a DFA utilizando el método de subconjuntos, el cual fue documentado paso a paso en el reporte, incluyendo la generación de la tabla de transiciones entre subconjuntos y el mapeo correspondiente a estados del DFA.

Una vez construido el DFA, se implementó un programa en Python que lee un archivo de entrada (ya sea un fragmento de código o un archivo de texto) y procesa carácter por carácter utilizando las transiciones definidas por el autómata. Cuando se reconoce una palabra reservada, el programa la registra, contabiliza su aparición y guarda su posición exacta (expresada en coordenadas de columna y fila) dentro del archivo. Además, se genera un archivo adicional donde se documenta toda la evaluación del autómata: cada símbolo leído, el estado actual, el estado siguiente y las transiciones realizadas.

Como complemento visual, el programa incluye una función que grafica el DFA utilizando la biblioteca `networkx` junto con `matplotlib`, mostrando los estados, las transiciones y resaltando los estados iniciales y finales. Esta gráfica permite al usuario tener una representación clara del funcionamiento del autómata y facilita el análisis del proceso de reconocimiento.

El desarrollo de esta práctica no solo reforzó los conocimientos teóricos sobre autómatas finitos y su aplicación práctica, sino que también permitió comprender el papel fundamental que estos modelos tienen en el diseño de compiladores y analizadores léxicos. La correcta identificación de palabras reservadas es un paso esencial en el análisis sintáctico y semántico de los lenguajes de programación, y esta práctica simula a pequeña escala cómo funciona dicho proceso en compiladores reales.

2. Desarrollo

2.1. Diseño del NFA

El primer paso fundamental en el desarrollo de esta práctica fue el diseño de un autómata finito no determinista (NFA) capaz de reconocer todas las palabras reservadas del lenguaje de programación C. Estas palabras, tales como `int`, `return`, `while`, `typedef`, entre muchas otras, son elementos clave del lenguaje y deben ser identificadas de forma precisa por el autómata.

Para ello, se construyó un NFA donde cada camino desde el estado inicial hasta un estado de aceptación representa una palabra reservada válida. La estructura del NFA fue diseñada de forma jerárquica: desde el estado inicial, se bifurcan múltiples caminos dependiendo del primer carácter leído (por ejemplo, ‘i’ puede conducir a `int` o `if`, mientras que ‘f’ puede conducir a `for` o `float`). Esta construcción permite que múltiples palabras compartan prefijos, reduciendo así la cantidad de transiciones redundantes.

El NFA fue implementado como un diccionario en Python, donde las claves representan los estados y los valores son diccionarios de transiciones. Cada transición puede conducir a uno o más estados (no determinismo) dependiendo del símbolo leído. El estado inicial fue designado como `q0`, y se establecieron múltiples estados de aceptación, cada uno representando el final de una palabra reservada.

Es importante destacar que el no determinismo del autómata se manifiesta principalmente en el estado `q0`, el cual presenta una transición cíclica para cada símbolo del alfabeto. Esto significa que, independientemente del carácter leído, siempre existe la posibilidad de permanecer en el estado `q0`. Esta característica permite que el autómata ignore caracteres no relevantes o que aparezcan prefijos superpuestos entre diferentes palabras reservadas, pero también introduce ambigüedad, ya que desde un mismo símbolo pueden surgir múltiples caminos posibles hacia estados distintos. Esta ambigüedad es precisamente lo que caracteriza a un NFA y hace necesaria su posterior transformación a un DFA. El NFA diseñado se puede apreciar en la Figura 1.

Nota: Dado que el NFA posee 148 estados, es necesario hacer zoom en la foto para poder apreciar de manera correcta los símbolos de transición que le corresponden a cada estado.

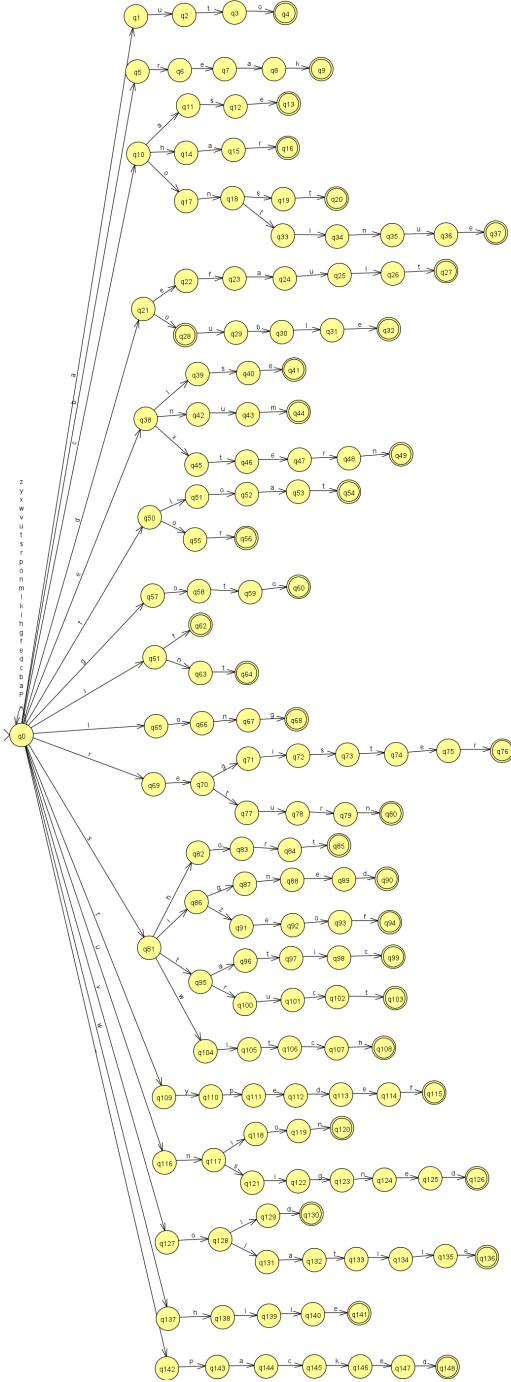


Figura 1: Diseño del NFA

2.2. Conversión del NFA a DFA

Una vez diseñado el autómata finito no determinista (NFA) capaz de reconocer todas las palabras reservadas del lenguaje C, el siguiente paso fue transformarlo en un autómata finito determinista (DFA), el cual es más adecuado para su implementación computacional directa, ya que no presenta ambigüedad en sus transiciones.

La conversión se realizó utilizando el método de subconjuntos, el cual consiste en crear nuevos estados en el DFA que representen subconjuntos de estados del NFA. Cada estado del DFA representa una posible combinación de estados del NFA que pueden alcanzarse mediante una secuencia de símbolos desde el estado inicial.

El proceso comenzó generando el subconjunto que contiene únicamente al estado inicial del NFA. A partir de este, se exploraron todas las posibles transiciones para cada símbolo del alfabeto, lo que dio lugar a nuevos subconjuntos que fueron mapeados a nuevos estados del DFA. Esta exploración se realizó de forma iterativa, asegurando que todos los subconjuntos alcanzables fueran considerados y almacenando las transiciones en una tabla estructurada.

Para facilitar este proceso, se desarrolló una función en Python: `generar_tabla(...)`, la cuál genera automáticamente la tabla de subconjuntos, escribe dicha tabla en un archivo CSV, y asigna identificadores únicos a cada nuevo estado del DFA. Durante esta construcción, se marcó claramente qué subconjuntos contenían estados de aceptación del NFA, lo que permitió determinar qué estados del DFA serían considerados finales.

Este método asegura que el DFA resultante cubre exactamente el mismo lenguaje reconocido por el NFA original, pero con la ventaja de tener una única transición definida para cada símbolo desde cualquier estado dado.

A continuación, en el Cuadro 1 se muestra la tabla de transiciones generada como parte del proceso de conversión del NFA a DFA.

Cuadro 1: Tabla de transiciones del NFA

2.3. Implementación del analizador léxico

Una vez construido el DFA correspondiente al lenguaje de palabras reservadas del compilador C, se procedió a desarrollar un programa en Python capaz de utilizar dicho autómata para analizar archivos de texto o código fuente. Este programa actúa como un analizador léxico básico, cuyo objetivo es identificar las palabras reservadas válidas conforme al lenguaje, registrar cuántas veces aparecen y en qué posiciones exactas se encuentran dentro del archivo.

El analizador recorre el archivo línea por línea, y cada línea carácter por carácter. A medida que se leen los caracteres, el programa evalúa las transiciones correspondientes en el DFA. Si el símbolo actual no pertenece al alfabeto del autómata o no existe una transición válida desde el estado actual, el analizador reinicia el proceso desde el estado inicial. En caso de que se llegue a un estado de aceptación, se considera que se ha reconocido una palabra reservada.

Cuando se detecta una palabra válida, el programa registra su aparición junto con su posición en el archivo, utilizando coordenadas del tipo (x, y) donde x representa el número de columna (índice del carácter en la línea) y y el número de línea. Además, el programa cuenta cuántas veces se repite cada palabra reservada, acumulando dicha información en una estructura de datos que posteriormente se guarda en el archivo **resultados.txt**. Para ejemplificar el uso del programa, se analizó el código del Anexo 2; los resultados pueden ser observados en la Figura 2.

```
resultados.txt
1 ===== RESUMEN DE PALABRAS ACEPTADAS =====
2 Nota: Las posiciones se expresan en coordenadas (x,y) donde 'x' es el número de columna y 'y' el número de fila
3
4 Palabra: 'struct' | Apariciones: 2 | Posiciones: (11,3), (9,5)
5 Palabra: 'int' | Apariciones: 11 | Posiciones: (19,3), (5,6), (7,12), (7,13), (7,14), (1,17), (5,24), (10,27), (48,27), (11,35), (11,37)
6 Palabra: 'float' | Apariciones: 4 | Posiciones: (24,3), (5,7), (5,25), (5,32)
7 Palabra: 'return' | Apariciones: 3 | Posiciones: (31,3), (5,40), (19,40)
8 Palabra: 'if' | Apariciones: 3 | Posiciones: (39,3), (5,34), (36,34)
9 Palabra: 'else' | Apariciones: 3 | Posiciones: (43,3), (7,36), (18,36)
10 Palabra: 'for' | Apariciones: 3 | Posiciones: (49,3), (5,27), (43,27)
11 Palabra: 'typedef' | Apariciones: 2 | Posiciones: (54,3), (1,5)
12 Palabra: 'void' | Apariciones: 2 | Posiciones: (63,3), (1,11)
13 Palabra: 'const' | Apariciones: 3 | Posiciones: (69,3), (5,8), (21,11)
14 Palabra: 'char' | Apariciones: 1 | Posiciones: (11,8)
15 Palabra: 'sizeof' | Apariciones: 3 | Posiciones: (17,24), (33,24), (53,24)
16 Palabra: 'do' | Apariciones: 1 | Posiciones: (30,35)
```

Figura 2: Resultados del código del anexo 2

Adicionalmente, se genera un archivo llamado **evaluacion.txt** que contiene un registro detallado de todas las derivaciones del DFA durante el análisis. Este archivo documenta, carácter por carácter, el símbolo leído, el estado actual, el estado siguien-

te y, en su caso, la aceptación de una palabra. Este historial permite verificar paso a paso el comportamiento del autómata y validar que se estén siguiendo correctamente las transiciones definidas. Siguiendo con el análisis del Anexo 2, se puede observar en la Figura 3 las primeras 8 evaluaciones realizadas por el DFA para ese código.

```
≡ evaluacion.txt
1   Símbolo: '#'
2   q0 -> q0
3
4   Símbolo: 'i'
5   q0 -> q134
6
7   Símbolo: 'n'
8   q134 -> q4
9
10  Símbolo: 'c'
11  q4 -> q139
12
13  Símbolo: 'l'
14  q139 -> q133
15
16  Símbolo: 'u'
17  q133 -> q129
18
19  Símbolo: 'd'
20  q129 -> q138
21
22  Símbolo: 'e'
23  q138 -> q78
```

Figura 3: Primeras 8 evaluaciones del DFA

Esta implementación demuestra el uso práctico de los autómatas finitos deterministas en tareas reales de análisis léxico, como las que realiza un compilador al procesar código fuente. Además, sirve como base para futuros análisis sintácticos o semánticos más complejos.

2.4. Visualización gráfica del DFA

Como parte complementaria del desarrollo de esta práctica, se implementó la funcionalidad de generar una representación gráfica del autómata finito determinista (DFA) resultante de la conversión del NFA. Esta visualización no solo cumple una función estética, sino que también permite analizar visualmente la estructura del autómata, facilitando la validación de sus transiciones, estados y configuración general.

Para esta tarea se utilizó la biblioteca `networkx` junto con `matplotlib` en el lenguaje Python. El graficador construye un grafo dirigido, donde cada nodo representa un estado del DFA y cada arista está etiquetada con el símbolo del alfabeto correspondiente a la transición. Los estados se distinguen visualmente de la siguiente manera:

- El estado inicial se representa con un color verde claro.
- Los estados de aceptación (finales) se muestran en color naranja.
- El resto de los estados se dibujan en azul claro.

Además, las transiciones entre estados se representan mediante líneas curvas con etiquetas que indican el símbolo correspondiente. Esta elección de diseño mejora la legibilidad del grafo cuando existen múltiples transiciones que se cruzan entre los mismos pares de nodos.

La generación de esta gráfica se realiza automáticamente al finalizar el análisis del archivo. La imagen se muestra en pantalla y también se guarda como un archivo de alta calidad (`DFA.png`), lo cual permite su inclusión directa en este reporte tal cual como se observa en la Figura 4.

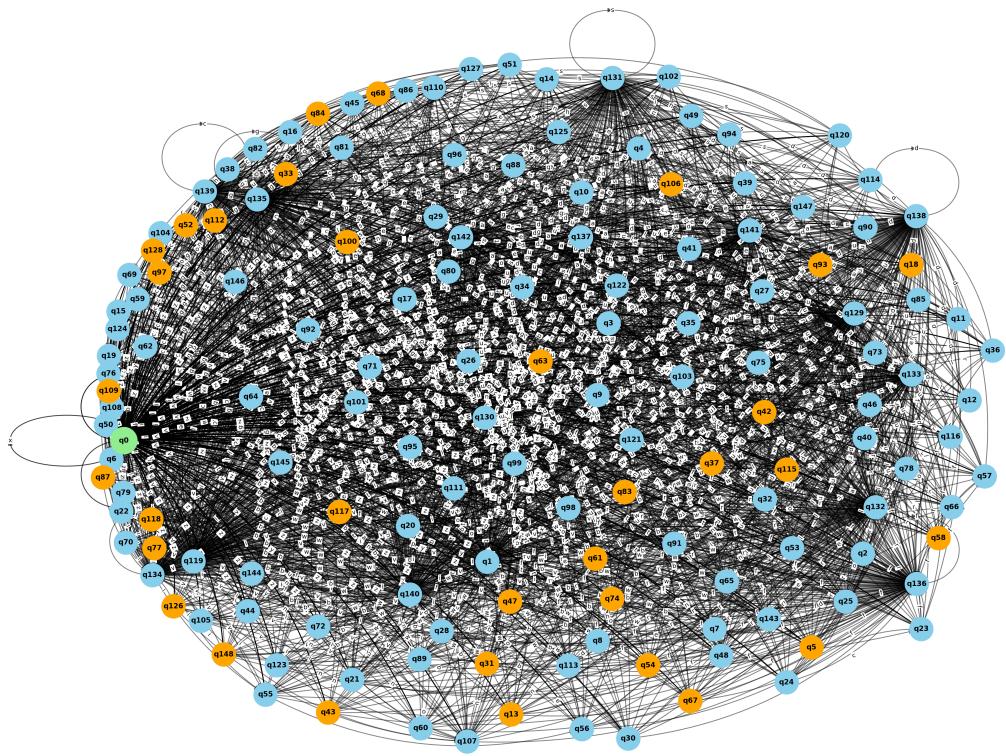


Figura 4: Diseño del DFA

Esta representación gráfica sirve como una herramienta visual poderosa para comprender cómo el autómata procesa cadenas de entrada y proporciona una validación visual del correcto diseño y conversión del NFA original.

3. Conclusiones

La realización de esta práctica permitió aplicar de manera integral los conceptos fundamentales de los autómatas finitos en el contexto del análisis léxico, una etapa esencial en la construcción de compiladores. A través del diseño de un NFA que reconoce todas las palabras reservadas del lenguaje C, y su posterior conversión a un DFA, se pudo comprobar que es posible representar lenguajes reales mediante modelos formales, y además automatizar su procesamiento.

Durante el desarrollo, se implementó una herramienta práctica que no solo identifica correctamente las palabras reservadas en archivos de texto o código, sino que también registra detalladamente cada transición realizada por el autómata. Esto no solo refuerza la comprensión del comportamiento de los DFA, sino que también demuestra su utilidad para tareas de escaneo y análisis de tokens, tal como lo hacen los analizadores léxicos en compiladores reales.

Asimismo, la generación de la tabla de conversión y su visualización gráfica facilitaron la validación y comprensión del funcionamiento del autómata. Estas representaciones permitieron observar de forma clara la relación entre los subconjuntos del NFA y los estados del DFA, así como verificar visualmente las transiciones y estados finales.

En conjunto, esta práctica integró teoría, diseño formal, programación y visualización, brindando una experiencia completa sobre cómo se utilizan los autómatas finitos en problemas reales de la ingeniería de software. Además, sienta una base sólida para abordar fases posteriores del procesamiento de lenguajes, como el análisis sintáctico y semántico.

4. Referencias

- GeeksforGeeks. (2021, septiembre). Transition Table in Automata. <https://www.geeksforgeeks.org/theory-of-computation/transition-table-in-automata/>
- TutorialsPoint. (2025, marzo). NDFA to DFA Conversion. https://www.tutorialspoint.com/automata_theory/ndfa_to_dfa_conversion.htm
- University of Baghdad Digital Repository. (2019, marzo). Steps for converting NFA to DFA: Lecture 8. <https://repository.uobaghdad.edu.iq/user/109234074914052017789/12c898c6-b118-45fc-94b1-f93a45eb5cb3.pdf>

5. Anexos

5.1. Anexo 1. Código fuente del programa

```
import csv
import networkx as nx
import matplotlib.pyplot as plt

def graficar_automata(alfabeto, transiciones, estado_inicial,
                     estados_finales):
    G = nx.MultiDiGraph()

    #Agregar estados
    for estado in transiciones:
        G.add_node(estado)

    #Agregar transiciones
    for origen, caminos in transiciones.items():
        for simbolo in alfabeto:
            if simbolo in caminos:
                destino = caminos[simbolo]
                G.add_edge(origen, destino, label=simbolo)

    pos = nx.spring_layout(G, k=3, iterations=100)
    plt.figure(figsize=(18, 13.5))

    #Dibujar estados
    nx.draw_networkx_nodes(G, pos, node_color='skyblue',
                           node_size=700)

    #Estado inicial en verde
    nx.draw_networkx_nodes(G, pos, nodelist=[estado_inicial],
                           node_color='lightgreen', node_size=900)

    #Estados finales en naranja
    nx.draw_networkx_nodes(G, pos, nodelist=estados_finales,
                           node_color='orange', node_size=700)
```

```

#Etiquetas estados
nx.draw_networkx_labels(G, pos, font_size=8, font_weight='bold')

#Dibujar aristas con curvas para mejor visibilidad
nx.draw_networkx_edges(G, pos, arrows=True,
→ connectionstyle='arc3,rad=0.2',alpha=0.5)

# Etiquetas aristas
etiquetas = nx.get_edge_attributes(G, 'label')
nx.draw_networkx_edge_labels(
    G, pos, edge_labels=etiquetas,
    font_color='black',
    font_size=7,
    → bbox=dict(facecolor='white',edgecolor='none',boxstyle='round,pad=0.1')
    #bbox=dict(facecolor='none', edgecolor='none', pad=0)
)

plt.axis('off')
plt.tight_layout()
plt.savefig("DFA.png", dpi=300, bbox_inches='tight') # Guardar
→ PNG con buena calidad
plt.show()

def
→ formatear_subconjunto(subconjunto,est_ini,est_fin,es_estadoNFA=False):
    subconjunto_ordenado = sorted(subconjunto,key=lambda x:
        → int(x[1:]))
    cadena = "{" + ",".join(subconjunto_ordenado) + "}"

    if es_estadoNFA == True and len(subconjunto) == 1 and
        → subconjunto[0] == est_ini: #Marca el estado inicial en la
        → tabla
            return f"->{cadena}"
    elif es_estadoNFA == True and any(estado in est_fin for estado in
        → subconjunto): #Marca subconjuntos con estados finales
            return f"*{cadena}"
    else:

```

```

    return cadena

def generar_tabla(alfabeto,NFA,est_ini,est_fin):
    encabezado = ['Estados NFA','Estado DFA'] + alfabeto

    with open("tabla.csv","w",newline='') as t:
        escritor = csv.writer(t,delimiter=';')
        escritor.writerow(encabezado)

        checados = set()           #Estados ya procesados
        pendientes = [[est_ini]]   #Subconjuntos por procesar
        fila = []                  #Texto a escribir en la fila
        contadorDFA = 0            #Mapeo de subconjunto NFA ->
        ↳ estado NFA

    while pendientes:
        subconjunto = pendientes.pop()
        fila = []

        if tuple(subconjunto) in checados:
            continue

        ↳ fila.append(formatear_subconjunto(subconjunto,est_ini,est_fin,True))
        checados.add(tuple(subconjunto))

        fila.append(f"q{contadorDFA}")
        contadorDFA += 1

        for simbolo in alfabeto:
            estados_siguientes = set()

            for estado in subconjunto:
                if simbolo in NFA[estado]:
                    ↳ estados_siguientes.update(NFA[estado][simbolo])

            ↳ fila.append(formatear_subconjunto(estados_siguientes,est_ini,est_
```

```

estados_ordenados =
    ↳ tuple(sorted(estados_siguientes,key=lambda x:
    ↳ int(x[1:])))

if estados_ordenados not in checados and
    ↳ estados_ordenados not in map(tuple,pendientes):
    pendientes.append(list(estados_ordenados))

escritor.writerow(fila)

def tabla_a_dfa():
    DFA = {}
    mapeoDFA = {} #Relaciona los subconjuntos del NFA con estados
    ↳ del DFA

    with open("tabla.csv",newline='') as t:
        lector = csv.reader(t,delimiter=';')
        encabezado = next(lector)
        alfabeto = encabezado[2:]
        filas = list(lector)

        inicial = ''
        finales = set()

    #Construcción de diccionario subconjunto NFA -> estado DFA
    for fila in filas:
        subconjuntoNFA = fila[0]
        estadoDFA = fila[1]
        if '->' in subconjuntoNFA:
            mapeoDFA[subconjuntoNFA[2:]] = estadoDFA
            inicial = mapeoDFA[subconjuntoNFA[2:]]
        elif '*' in subconjuntoNFA:
            mapeoDFA[subconjuntoNFA[1:]] = estadoDFA
            finales.add(mapeoDFA[subconjuntoNFA[1:]])
        else:
            mapeoDFA[subconjuntoNFA] = estadoDFA

    #Transiciones

```

```

for fila in filas:
    estadoDFA = fila[1]
    DFA[estadoDFA] = {}
    for i,simbolo in enumerate(alfabeto):
        subconjunto = fila[i+2]
        estado = mapeoDFA[subconjunto]
        DFA[estadoDFA][simbolo] = estado

return DFA,inicial,finales

def validar_palabra(palabra):
    validas = [
        "auto", "else", "long", "switch",
        "break", "enum", "register", "typedef",
        "case", "extern", "return", "union",
        "char", "float", "short", "unsigned",
        "const", "for", "signed", "void",
        "continue", "goto", "sizeof", "volatile",
        "default", "if", "static", "while",
        "do", "int", "struct", "_Packed",
        "double"]

    for palabraValida in validas:
        if palabraValida in palabra:
            ajustarPos = len(palabra) - len(palabraValida)
            return palabraValida,ajustarPos

    return palabra,len(palabra)

def analizar_texto(archivo, DFA, est_ini, est_fin, alfabeto):
    resultados = {}

    try:
        with open(archivo, "r") as a, open("evaluacion.txt", "w") as
        e:
            y = 0 #Número de linea

            for linea in a:

```

```

y += 1
linea = linea.rstrip('\n')
longitud = len(linea)
x = 0 #Número de columna
estado_actual = est_ini
palabra = ""
inicio_palabra = 0

while x < longitud:
    simbolo = linea[x]

    if simbolo not in alfabeto:
        e.write(f"Símbolo: '{simbolo}'\n")
        e.write(f"{estado_actual} -> {est_ini}\n\n")
        estado_actual = est_ini
        palabra = ""
        x += 1
        continue

    if simbolo not in DFA[estado_actual]:
        e.write(f"Símbolo: '{simbolo}'\n")
        e.write(f"{estado_actual} -> {est_ini}\n\n")
        estado_actual = est_ini
        palabra = ""
        x += 1
        continue

    if not palabra:
        inicio_palabra = x

    estado_siguiente = DFA[estado_actual][simbolo]
    e.write(f"Símbolo: '{simbolo}'\n{estado_actual}\n-> {estado_siguiente}\n\n")
    palabra += simbolo
    estado_actual = estado_siguiente

if estado_actual in est_fin:
    palabra,ajustarPos = validar_palabra(palabra)

```

```

        e.write(f"Palabra aceptada: '{palabra}' en
        ↵  línea {y}, columna
        ↵  {inicio_palabra+ajustarPos+1}\n\n")
    if palabra not in resultados:
        resultados[palabra] = [0, []]
    resultados[palabra][0] += 1

        ↵  resultados[palabra][1].append((inicio_palabra+ajustarPos+
        ↵  y))
    palabra = ""
x += 1

with open("resultados.txt", "w") as r:
    r.write("===== RESUMEN DE PALABRAS ACEPTADAS =====\n")
    r.write("Nota: Las posiciones se expresan en coordenadas
    ↵  (x,y) donde 'x' es el número de columna y 'y' el
    ↵  número de fila\n\n")
    for palabra, (conteo, posiciones) in resultados.items():
        posiciones = ", ".join(f"({x},{y})" for x, y in
        ↵  posiciones)
        r.write(f"Palabra: '{palabra}' | Apariciones:
        ↵  {conteo} | Posiciones: {posiciones}\n")

    print("Archivo analizado correctamente.")
    print("Las evaluaciones del DFA han sido guardadas en
    ↵  'evaluacion.txt'.")
    print("Las palabras reservadas detectadas han sido guardadas
    ↵  en 'resultados.txt'.")

except FileNotFoundError:
    print("El archivo no existe.")
    exit()

def main():
    #Definición del automata
    alfabeto =
    ↵  ['_','P','a','b','c','d','e','f','g','h','i','k','l','m','n','o','p','r','s'],
    NFA = {

```

```

'q0' : {'_': {'q0', 'q142'},  

         'P' : {'q0'},  

         'a' : {'q0', 'q1'},  

         'b' : {'q0', 'q5'},  

         'c' : {'q0', 'q10'},  

         'd' : {'q0', 'q21'},  

         'e' : {'q0', 'q38'},  

         'f' : {'q0', 'q50'},  

         'g' : {'q0', 'q57'},  

         'h' : {'q0'},  

         'i' : {'q0', 'q61'},  

         'k' : {'q0'},  

         'l' : {'q0', 'q65'},  

         'm' : {'q0'},  

         'n' : {'q0'},  

         'o' : {'q0'},  

         'p' : {'q0'},  

         'r' : {'q0', 'q69'},  

         's' : {'q0', 'q81'},  

         't' : {'q0', 'q109'},  

         'u' : {'q0', 'q116'},  

         'v' : {'q0', 'q127'},  

         'w' : {'q0', 'q137'},  

         'x' : {'q0'},  

         'y' : {'q0'},  

         'z' : {'q0'},},  

'q1' : {'u' : {'q2'}},  

'q2' : {'t' : {'q3'}},  

'q3' : {'o' : {'q4'}},  

'q4' : {},  

'q5' : {'r' : {'q6'}},  

'q6' : {'e' : {'q7'}},  

'q7' : {'a' : {'q8'}},  

'q8' : {'k' : {'q9'}},  

'q9' : {},  

'q10' : {'a' : {'q11'},  

           'h' : {'q14'},  

           'o' : {'q17'}}},

```

```

'q11' : {'s' : {'q12'}},
'q12' : {'e' : {'q13'}},
'q13' : {},
'q14' : {'a' : {'q15'}},
'q15' : {'r' : {'q16'}},
'q16' : {},
'q17' : {'n' : {'q18'}},
'q18' : {'s' : {'q19'},
          't' : {'q33'}},
'q19' : {'t' : {'q20'}},
'q20' : {},
'q21' : {'e' : {'q22'},
          'o' : {'q28'}},
'q22' : {'f' : {'q23'}},
'q23' : {'a' : {'q24'}},
'q24' : {'u' : {'q25'}},
'q25' : {'l' : {'q26'}},
'q26' : {'t' : {'q27'}},
'q27' : {},
'q28' : {'u' : {'q29'}},
'q29' : {'b' : {'q30'}},
'q30' : {'l' : {'q31'}},
'q31' : {'e' : {'q32'}},
'q32' : {},
'q33' : {'i' : {'q34'}},
'q34' : {'n' : {'q35'}},
'q35' : {'u' : {'q36'}},
'q36' : {'e' : {'q37'}},
'q37' : {},
'q38' : {'l' : {'q39'},
          'n' : {'q42'},
          'x' : {'q45'}},
'q39' : {'s' : {'q40'}},
'q40' : {'e' : {'q41'}},
'q41' : {},
'q42' : {'u' : {'q43'}},
'q43' : {'m' : {'q44'}},
'q44' : {},

```

```

'q45' : {'t' : {'q46'}},
'q46' : {'e' : {'q47'}},
'q47' : {'r' : {'q48'}},
'q48' : {'n' : {'q49'}},
'q49' : {},
'q50' : {'l' : {'q51'}},
    'o' : {'q55'}},
'q51' : {'o' : {'q52'}},
'q52' : {'a' : {'q53'}},
'q53' : {'t' : {'q54'}},
'q54' : {},
'q55' : {'r' : {'q56'}},
'q56' : {},
'q57' : {'o' : {'q58'}},
'q58' : {'t' : {'q59'}},
'q59' : {'o' : {'q60'}},
'q60' : {},
'q61' : {'f' : {'q62'}},
    'n' : {'q63'}},
'q62' : {},
'q63' : {'t' : {'q64'}},
'q64' : {},
'q65' : {'o' : {'q66'}},
'q66' : {'n' : {'q67'}},
'q67' : {'g' : {'q68'}},
'q68' : {},
'q69' : {'e' : {'q70'}},
'q70' : {'g' : {'q71'}},
    't' : {'q77'}},
'q71' : {'i' : {'q72'}},
'q72' : {'s' : {'q73'}},
'q73' : {'t' : {'q74'}},
'q74' : {'e' : {'q75'}},
'q75' : {'r' : {'q76'}},
'q76' : {},
'q77' : {'u' : {'q78'}},
'q78' : {'r' : {'q79'}},
'q79' : {'n' : {'q80'}},

```

```

'q80' : {},
'q81' : {'h' : {'q82'},
          'i' : {'q86'},
          't' : {'q95'},
          'w' : {'q104'}},
'q82' : {'o' : {'q83'}},
'q83' : {'r' : {'q84'}},
'q84' : {'t' : {'q85'}},
'q85' : {},
'q86' : {'g' : {'q87'},
          'z' : {'q91'}},
'q87' : {'n' : {'q88'}},
'q88' : {'e' : {'q89'}},
'q89' : {'d' : {'q90'}},
'q90' : {},
'q91' : {'e' : {'q92'}},
'q92' : {'o' : {'q93'}},
'q93' : {'f' : {'q94'}},
'q94' : {},
'q95' : {'a' : {'q96'},
          'r' : {'q100'}},
'q96' : {'t' : {'q97'}},
'q97' : {'i' : {'q98'}},
'q98' : {'c' : {'q99'}},
'q99' : {},
'q100' : {'u' : {'q101'}},
'q101' : {'c' : {'q102'}},
'q102' : {'t' : {'q103'}},
'q103' : {},
'q104' : {'i' : {'q105'}},
'q105' : {'t' : {'q106'}},
'q106' : {'c' : {'q107'}},
'q107' : {'h' : {'q108'}},
'q108' : {},
'q109' : {'y' : {'q110'}},
'q110' : {'p' : {'q111'}},
'q111' : {'e' : {'q112'}},
'q112' : {'d' : {'q113'}}}

```

```

'q113' : {'e' : {'q114'}},
'q114' : {'f' : {'q115'}},
'q115' : {},
'q116' : {'n' : {'q117'}},
'q117' : {'i' : {'q118'},
           's' : {'q121'}},
'q118' : {'o' : {'q119'}},
'q119' : {'n' : {'q120'}},
'q120' : {},
'q121' : {'i' : {'q122'}},
'q122' : {'g' : {'q123'}},
'q123' : {'n' : {'q124'}},
'q124' : {'e' : {'q125'}},
'q125' : {'d' : {'q126'}},
'q126' : {},
'q127' : {'o' : {'q128'}},
'q128' : {'i' : {'q129'},
           'l' : {'q131'}},
'q129' : {'d' : {'q130'}},
'q130' : {},
'q131' : {'a' : {'q132'}},
'q132' : {'t' : {'q133'}},
'q133' : {'i' : {'q134'}},
'q134' : {'l' : {'q135'}},
'q135' : {'e' : {'q136'}},
'q136' : {},
'q137' : {'h' : {'q138'}},
'q138' : {'i' : {'q139'}},
'q139' : {'l' : {'q140'}},
'q140' : {'e' : {'q141'}},
'q141' : {},
'q142' : {'P' : {'q143'}},
'q143' : {'a' : {'q144'}},
'q144' : {'c' : {'q145'}},
'q145' : {'k' : {'q146'}},
'q146' : {'e' : {'q147'}},
'q147' : {'d' : {'q148'}},
'q148' : {}

```

```
}

inicial = 'q0'
finales =
→ { 'q4', 'q9', 'q13', 'q16', 'q20', 'q37', 'q27', 'q32', 'q28', 'q41', 'q44', 'q49', 'q54', 

generar_tabla(alfabeto,NFA,inicial,finales)
DFA,inicialDFA,finalesDFA = tabla_a_dfa()
archivo = input("Escriba el nombre del archivo de texto o código
→ a ser analizado: ")

analizar_texto(archivo,DFA,inicialDFA,finalesDFA)
graficar_automata(alfabeto,DFA,inicialDFA,finalesDFA)

if __name__ == "__main__":
    main()
```

5.2. Anexo 2. Código de C analizado por el DFA (prueba.c)

```
#include <stdio.h>

// Uso de struct, int, float, return, if, else, for, typedef, void,
→ const

typedef struct {
    int id;
    float promedio;
    const char* nombre;
} Alumno;

void imprimirAlumno(const Alumno* a) {
    printf("ID: %d\n", a->id);
    printf("Nombre: %s\n", a->nombre);
    printf("Promedio: %.2f\n", a->promedio);
}

int main() {
    Alumno grupo[] = {
        {1, 8.5, "Ana"}, 
        {2, 9.2, "Luis"}, 
        {3, 7.8, "Marta"} 
    };

    int total = sizeof(grupo) / sizeof(Alumno); // sizeof
    float suma = 0;

    for (int i = 0; i < total; ++i) { // for, int
        imprimirAlumno(&grupo[i]);
        suma += grupo[i].promedio;
    }

    float promedioGrupo = suma / total;

    if (promedioGrupo > 8.0) { // if
        printf("Grupo destacado\n");
    } else { // else
```

```
    printf("Grupo promedio\n");
}

return 0; // return
}
```