INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Práctica 1. Programar un AFD

PRESENTA

REYES FRANCISCO PAOLA

5CM4 | COMPILADORES | CECILIA ALBORTANTE MORATO

Introducción.

Un autómata finito es un modelo matemático de una máquina que acepta cadenas de un lenguaje definido sobre un alfabeto Σ. Consiste en un conjunto finito de estados y un conjunto de transiciones entre esos estados, que dependen de los símbolos de la cadena de entrada. El autómata finito acepta una cadena x si la secuencia de transiciones correspondientes a los símbolos de x conduce desde el estado inicial a un estado final.

Si para todo estado del autómata existe como máximo una transición definida para cada símbolo del alfabeto, se dice que el autómata es determinístico (AFD). Si a partir de algún estado y para el mismo símbolo de entrada, se definen dos o más transiciones se dice que el autómata es no determinístico (AFND).

Formalmente un autómata finito se define como una 5-upla:

$$M = < \Sigma, Q, q_0, F, T >$$

Σ: Alfabeto o conjunto finito de símbolos de entrada

Q: $Conjunto\ de\ estados$

q_0: $estado\ inicial$

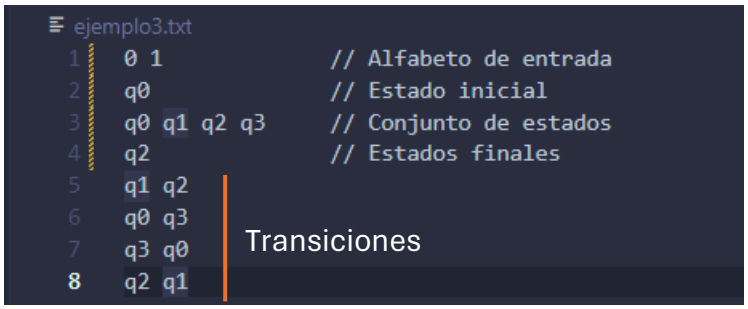
F: $F \subseteq Q\ Conjunto\ de\ estados\ finales$

T: $Qx\Sigma \rightarrow Q\ Función\ de\ transición$

Generalmente se asocia con cada autómata un grafo dirigido, llamado diagrama de transición de estados. Cada nodo del grafo corresponde a un estado. El estado inicial se indica mediante una flecha que no tiene nodo origen. Los estados finales se representan con un círculo doble.

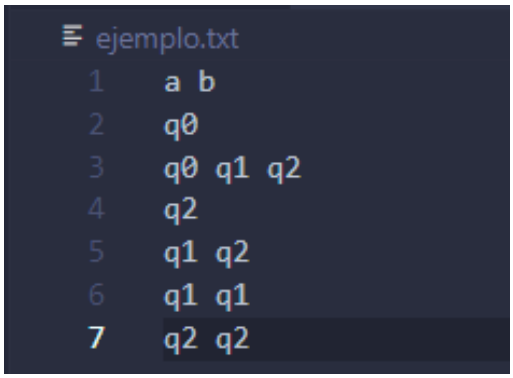
Desarrollo.

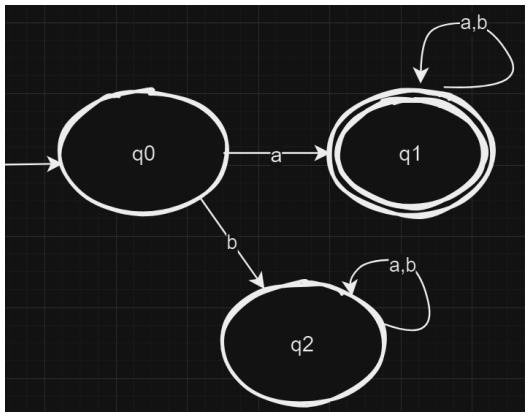
Para el desarrollo de esta práctica, se crearon tres archivos de texto que contenían la información de la tupla que define el autómata que queremos utilizar. Para ello, el archivo contenía la siguiente información:



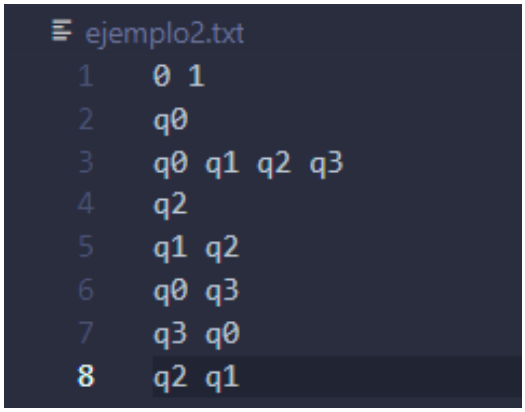
Para las transiciones sólo se consideró el contenido de la tabla de transiciones.

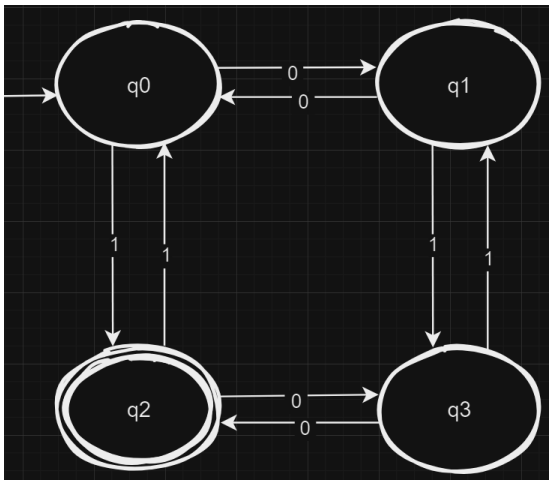
Ejemplo 1. *L = {w | w inicia con a}*





Ejemplo 2. *L = {w | w tiene un número par de ceros y un número impar de unos}*

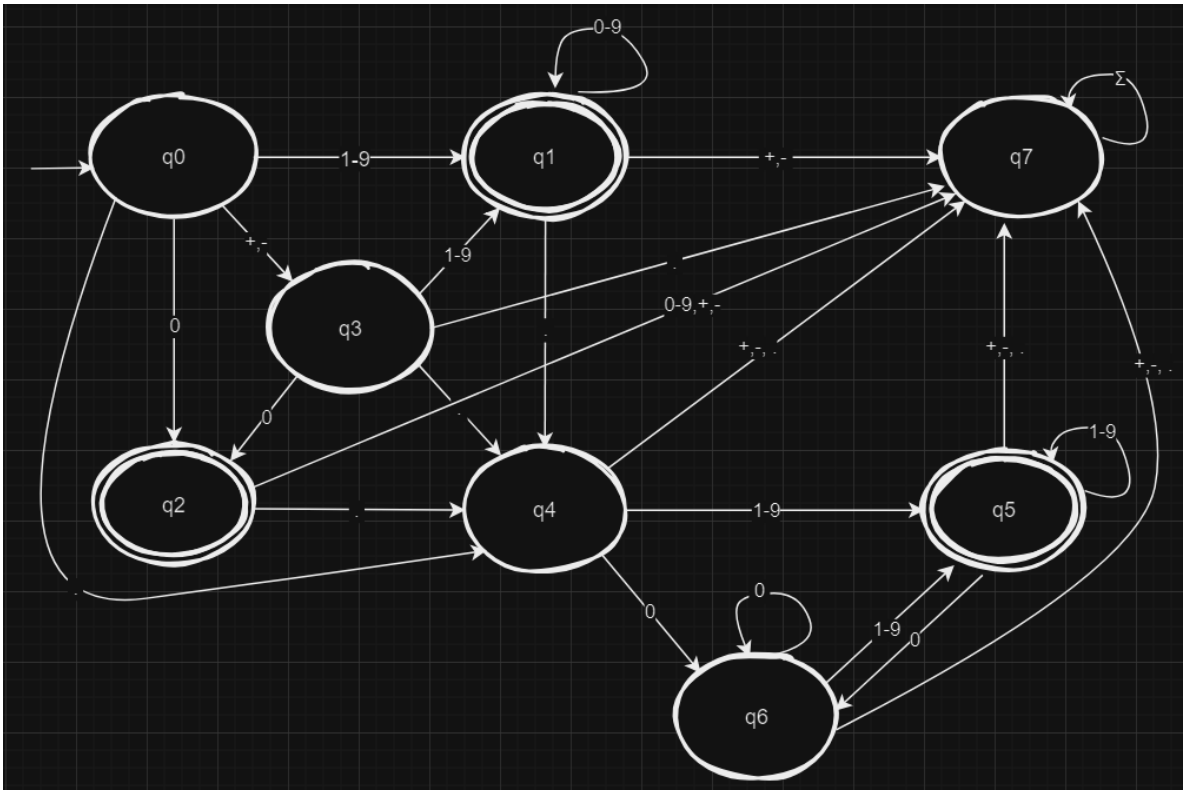




Ejemplo 3. *L = {w | w es un número entero o un número real, ambos con o sin signo}*

ejemplo3.txt

```
+ - . 0 1 2 3 4 5 6 7 8 9
q0
q0 q1 q2 q3 q4 q5 q6 q7
q1 q2 q5
q3 q3 q4 q2 q1 q1 q1 q1 q1 q1 q1 q1 q1
q7 q7 q4 q1 q1 q1 q1 q1 q1 q1 q1 q1 q1
q7 q7 q4 q7 q7 q7 q7 q7 q7 q7 q7 q7 q7
q7 q7 q4 q2 q1 q1 q1 q1 q1 q1 q1 q1 q1
q7 q7 q7 q6 q5 q5 q5 q5 q5 q5 q5 q5 q5
q7 q7 q7 q6 q5 q5 q5 q5 q5 q5 q5 q5 q5
q7 q7 q7 q6 q5 q5 q5 q5 q5 q5 q5 q5 q5
q7 q7 q7 q7 q7 q7 q7 q7 q7 q7 q7 q7 q7
```



Primero, como variables globales se declararon las siguientes.

```
vector<vector<string>> transiciones;
vector<char> alfabeto;
vector<string> estados;
string estadoInicial;
vector<string> estadosFinales;

string cadena,nombreArchivo;
```

El programa al momento de ejecutarse va a recibir dos argumentos, el primer argumento es el archivo de texto, es decir, el autómata que queremos utilizar y el segundo es la cadena que vamos a probar con dicho autómata.

```
int main(int argc, char *argv[]) {
    if (argc > 2) {
        nombreArchivo = argv[1];
        cadena = argv[2];
    } else {
        cout << "nombreArchivo.txt cadena" << endl;
    }

    ifstream archivo(nombreArchivo);

    if(!archivo.is_open()){
        cout << "No se pudo abrir el archivo" << endl;
        return 1;
    }
```

Después, se lee cada línea del archivo con nuestra función *getline*, en la primera línea leemos el alfabeto y mandamos a llamar a nuestra función *agregarAlfabeto*, pasa lo mismo para la siguiente línea, leemos el estado inicial y mandamos a una función que agrega el estado. Lo mismo ocurre para agregar el conjunto de estados y los estados finales.

Para las transiciones se ocupó un vector auxiliar donde se fueron añadiendo los datos que el programa iba leyendo en cada línea y después se agregaban a nuestro vector de transiciones. Se hizo una condicional para que no agregara filas vacías, ya que se tenía este problema al principio.

```
    // para almacenar cada línea leída del archivo
    string linea;
    // para saber el número de línea actual
    int num_linea = 1;

    // lee cada línea del archivo
    while(getline(archivo,linea)){

        if(num_linea == 1){ // alfabeto
            istringstream alfa(linea);
            char simbolo;
            // para cada palabra de la línea
            while(alfa >> simbolo) agregarAlfabeto(simbolo);

        }else if(num_linea == 2){ // estado inicial
            istringstream inicial(linea);
```

```
            string estado;
            while(inicial >> estado) agregarInicial(estado);

        }else if(num_linea == 3){ // conjunto de estados
            istringstream estado(linea);
            string estados;
            while(estado >> estados) agregarEstados(estados);

        }else if(num_linea == 4){ // estados finales
            istringstream final(linea);
            string finales;
            while(final >> finales) agregarFinales(finales);

        }else{ // transiciones
            vector<string> aux;
            istringstream fila(linea);
            string casilla;
            while(fila >> casilla) aux.push_back(casilla);
            // para no agregar filas vacías
            if(aux.size() != 0) transiciones.push_back(aux);
        }

        num_linea++;
    }

    imprimir();
    automata();

    cout << "\n";

    archivo.close();

    return 0;
}
```

Funciones para agregar la tupla del autómata (excepto las transiciones).

```
void agregarAlfabeto(char simbolo){
    alfabeto.push_back(simbolo);
}

void agregarInicial(const string& inicial){
    estadoInicial = inicial;
}

void agregarEstados(const string& simbolo){
    estados.push_back(simbolo);
}

void agregarFinales(const string& simbolo){
    estadosFinales.push_back(simbolo);
}
```

Posteriormente, se creó una función para evaluar la cadena de entrada, al final de la evaluación se indica si la cadena fue aceptada, si no fue aceptada o si no es válida (cuando tenemos un símbolo que no es parte del alfabeto). Se utilizaron otras dos funciones auxiliares para obtener el índice de la fila y la columna para la tabla de transiciones.

```
bool automata(){
    // Primero se busca el índice del estado inicial (fila)
    size_t estadoActual = getFila(estadoInicial);

    // itera sobre cada símbolo de la cadena de entrada
    for(int i=0; i<cadena.size(); i++){
        // busca el símbolo en la posición i de la cadena a través
        // de nuestro vector que contiene el alfabeto
        if(find(alfabeto.begin(), alfabeto.end(), cadena.at(i)) != alfabeto.end()){
            // se busca el estado al que transita en la fila de nuestro
            // estado actual y en la columna en donde se encuentra el
            // símbolo dentro de nuestro alfabeto, se guarda en una variable
            // ese estado
            string s = transiciones.at(estadoActual).at(getColumna(cadena.at(i)));
            // se actualiza el valor del índice de la fila de
            // nuestro estado actual
            estadoActual = getFila(s);
        }else{
            // si no encuentra el símbolo dentro del alfabeto
            cout << "\n\033[91mCadena NO valida\033[0m" << endl;
            return false;
        }
    }

    // Una vez que termina de evaluar la cadena, iteramos a lo
    // largo de nuestro vector de estados finales para
    // verificar si el estado actual es uno de los estados finales
    for(int i=0; i<estadosFinales.size(); i++){
        if(estados.at(estadoActual) == estadosFinales.at(i)) {
            cout << "\n\033[92mCadena aceptada\033[0m";
            return true;
        }
    }
    // si no es parte de los estados finales
    cout << "\n\033[91mCadena NO aceptada\033[0m";
    return false;
}
```

Funciones para obtener el índice de la fila y la columna.

```
// para obtener el índice de la fila
size_t getFila(string estadoActual) {
    // busca el estado en el vector
    auto estado = find(estados.begin(), estados.end(), estadoActual);
    // calcula la distancia (el índice)
    // entre el inicio del vector y el estado
    return distance(estados.begin(), estado);
}
// para obtener el índice de la columna
size_t getColumna(char c) {
    auto caracter = find(alfabeto.begin(), alfabeto.end(), c);
    return distance(alfabeto.begin(), caracter);
}
```

Finalmente, como extra se creó una función que imprime la tupla del autómata, a excepción de las tabla de transiciones.

```
void imprimir() {
    cout << "\n\033[96mAlfabeto:\033[0m";
    for(int i=0; i<alfabeto.size(); i++) {
        cout << " " << alfabeto.at(i);
    }

    cout << "\n\033[93mEstado Inicial:\033[0m " << estadoInicial;

    cout << "\n\033[93mEstados:\033[0m";
    for(int i=0; i<estados.size(); i++) {
        cout << " " << estados.at(i);
    }

    cout << "\n\033[93mEstados Finales:\033[0m";
    for(int i=0; i<estadosFinales.size(); i++) {
        cout << " " << estadosFinales.at(i);
    }
}
```

Resultados.

Ejemplo 1. *Cadenas que inician con a.*

```
D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo1.txt aaabbbbabbba

Alfabeto: a b
Estado Inicial: q0
Estados: q0 q1 q2
Estados Finales: q1
Cadena aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo1.txt bbabba

Alfabeto: a b
Estado Inicial: q0
Estados: q0 q1 q2
Estados Finales: q1
Cadena NO aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo1.txt bbabbae

Alfabeto: a b
Estado Inicial: q0
Estados: q0 q1 q2
Estados Finales: q1
Cadena NO valida
```

Como se puede observar, tenemos un ejemplo al inicio de una palabra aceptada, otra no aceptada y la última es para cuando ingresamos una cadena con algún símbolo que no forma parte de nuestro alfabeto.

Ejemplo 2. *Cadena con un número par de ceros e impar de unos.*

```
D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo2.txt 01

Alfabeto: 0 1
Estado Inicial: q0
Estados: q0 q1 q2 q3
Estados Finales: q2
Cadena NO aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo2.txt 0100000000

Alfabeto: 0 1
Estado Inicial: q0
Estados: q0 q1 q2 q3
Estados Finales: q2
Cadena NO aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo2.txt 01000000000

Alfabeto: 0 1
Estado Inicial: q0
Estados: q0 q1 q2 q3
Estados Finales: q2
Cadena aceptada
```

En los dos primeros ejemplos se probó con cadenas con un número impar de ceros.

```
D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo2.txt 010001111

Alfabeto: 0 1
Estado Inicial: q0
Estados: q0 q1 q2 q3
Estados Finales: q2
Cadena aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo2.txt 0100011110

Alfabeto: 0 1
Estado Inicial: q0
Estados: q0 q1 q2 q3
Estados Finales: q2
Cadena NO aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo2.txt 01000111

Alfabeto: 0 1
Estado Inicial: q0
Estados: q0 q1 q2 q3
Estados Finales: q2
Cadena NO aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo2.txt 1

Alfabeto: 0 1
Estado Inicial: q0
Estados: q0 q1 q2 q3
Estados Finales: q2
Cadena aceptada
```

Otros ejemplos donde se probó también con la cadena *1*, como el número de ceros es par, puede no estar en la cadena.

Ejemplo 3. *Cadena con un número entero o un número real, con o sin signo.*

Para esta última prueba se comentó la función imprimir con la finalidad de mostrar más ejemplos.

```
D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt +3.138

Cadena aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt -0.232

Cadena aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt -0.232.+

Cadena NO aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt .19820093

Cadena aceptada
```

```
D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt 0.00000

Cadena NO aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt 0019

Cadena NO aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt 19

Cadena aceptada
```

En el diseño del AFD se pensó en que, si el usuario ingresaba un número entero con 0 al inicio, se iba a marcar como inválida la cadena, es decir, no se puede poner 019 por ejemplo, solo 19. De igual manera, si se ingresa un 0 y seguido de este un punto para después poner solo ceros (0.0, 0.00, etc.), también te lo marca como inválido, te lo marca como válido si está el 0 solo:

```
D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt 0

Cadena aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt 0.100290

Cadena NO aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt 0.10029

Cadena aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt 0.0000000001

Cadena aceptada
```

También se pensó en sí, el usuario ingresa un número decimal como en el ejemplo (0.100290), el autómata lo toma como una cadena no aceptada, pues pasa lo mismo con los enteros, puede agregarse un 0 al inicio (019) y no afecta, pero no es común. En el caso de los decimales se realizó el diseño así para evitar cadenas como 0.10000 que equivale a 0.1.

Sin embargo, esa parte de los decimales puede cambiarse en la tabla de transiciones del autómata.

```
D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt 2+7

Cadena NO aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt 2.4903

Cadena aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt +.9322

Cadena aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt +..22

Cadena NO aceptada
```

Cambiando el diseño del autómata para que acepte las cadenas con decimales que terminan en 0 tenemos:

```
D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt 0.0000000

Cadena NO aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt 0.00000001

Cadena aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt 0.100290

Cadena aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt 0.10029

Cadena aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt -0.28900920

Cadena aceptada

D:\Escuela\Documentos\IPN\5to Semestre\Compiladores\Práctica 1. AFD>a ejemplo3.txt -0.2890092

Cadena aceptada
```

Se puede apreciar también, que las cadenas como 0.000 siguen sin ser aceptadas, pues para que sea aceptada tiene que ser sólo 0.

ejemplo3.txt

```
1  + - . 0 1 2 3 4 5 6 7 8 9
2  q0
3  q0 q1 q2 q3 q4 q5 q6 q7
4  q1 q2 q5
5  q3 q3 q4 q2 q1 q1 q1 q1 q1 q1 q1 q1 q1
6  q7 q7 q4 q1 q1 q1 q1 q1 q1 q1 q1 q1 q1
7  q7 q7 q4 q7 q7 q7 q7 q7 q7 q7 q7 q7 q7
8  q7 q7 q4 q2 q1 q1 q1 q1 q1 q1 q1 q1 q1
9  q7 q7 q7 q6 q5 q5 q5 q5 q5 q5 q5 q5 q5
10 q7 q7 q7 q5 q5 q5 q5 q5 q5 q5 q5 q5 q5
11 q7 q7 q7 q6 q5 q5 q5 q5 q5 q5 q5 q5 q5
12 q7 q7 q7 q7 q7 q7 q7 q7 q7 q7 q7 q7 q7
```