Nombre:

Jhoel Alexander Ordoñez Coronel

Fecha:

01/06/2023

Docente:

Ruiz Vivanco Omar Alexander

Ciclo Académico:

Abril – Agosto

Estructura del lenguaje.

Palabras reservadas	
Tipo de datos	int = numeros enteros double = números decimales string = cadena
Condicional	if else
Ciclo repetitivo	for while do while
Casos	switch case break
Presentar Cadenas	print
Importar librerías	import

Operadores lógicos	**Operadores aritméticos**
logical_opera	artihmetic_opera
{&&,||,<,>,<=,>=,==}	{+,-,/,*}
Operadores Asignación	**Operadores de separación**
{=}	**{**(,), {, }, ;}

Condicionales

IF

IF (expresion){

<sentencia>};

IF ,ELSE

IF(expresion){

<sentencias>

} ELSE{

<expresion>

};

Ciclos

WHILE

WHILE (expresion){

<sentencia>};

DO WHILE

DO {

<sentencia>

WHILE (expresion);

<sentencia2>};

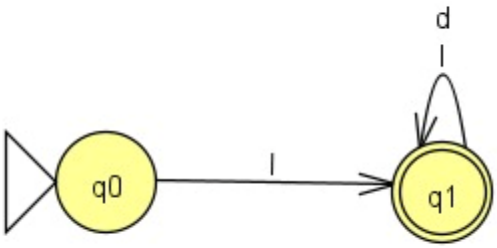
FOR

FOR (expresion1;expresion2;expresion3){

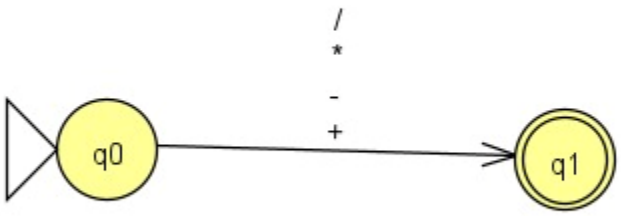
<sentencia>};

Autómatas

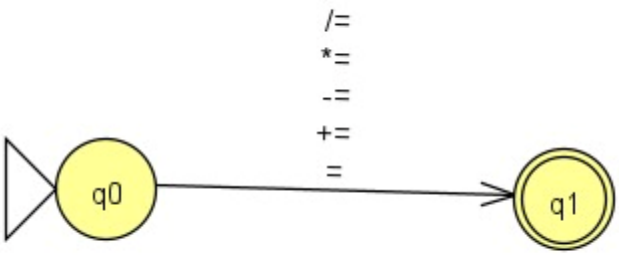
Autómata de identificadores



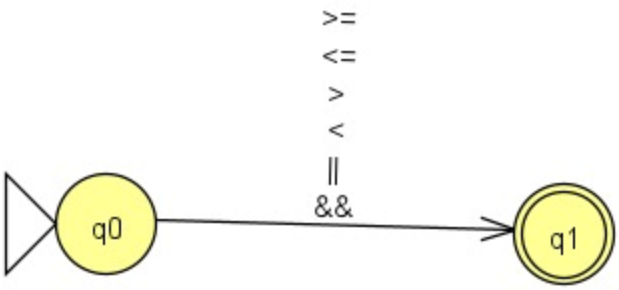
Autómata de Operadores aritméticos



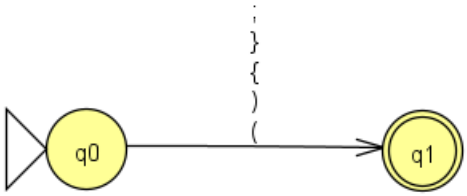
Autómata de Asignación



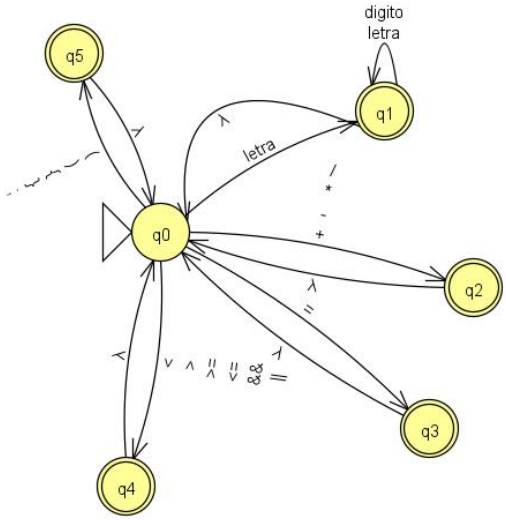
Autómata de Operadores Lógicos



Autómata de separador



Autómata Final



Código del lenguaje

```
#include <iostream>

#include <fstream>
#include <string>
#include <sstream>
#include <unordered_set>

using namespace std;

// Definición de los estados
enum Estado {
    q0,  // Estado inicial
```

```
    q1,  // Estado para identificadores
    q2,  // Estado para operadores aritméticos
    q3,  // Estado para operadores de asignación
    q4,  // Estado para operadores relacionales y separadores
    q5   // Estado para separadores
};

// Verifica si un carácter es una letra minúscula
bool esLetra(char c) {
    return (c >= 'a' && c <= 'z');
}

// Verifica si un carácter es un dígito
bool esDigito(char c) {
    return (c >= '0' && c <= '9');
}

// Verifica si un carácter es un operador aritmético
bool esOperadorAritmetico(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
}

// Verifica si un carácter es un operador de asignación
bool esOperadorAsignacion(char c) {
    return c == '=';
}

// Verifica si un carácter es un operador relacional
bool esOperadorLogico(char c) {
    return c == '<' || c == '>' || c == '=';
}

// Verifica si es un carácter es un separador
bool esSeparador(char c) {
    return c == '(' || c == ')' || c == '{' || c == '}' || c == ';';
}

// Función de transición
Estado transicion(Estado estadoActual, char simbolo) {
    switch (estadoActual) {
        case q0:
            if (esLetra(simbolo)){
                return q1;
            }
            else{
```

```
                return q0;
            }
            break;
        case q1:
            if (esLetra(simbolo) || esDigito(simbolo))
                return q1;
            break;
        case q2:
            if (esOperadorAritmetico(simbolo))
                return q2;
            break;
        case q3:
            if (esOperadorAsignacion(simbolo))
                return q3;
            break;
        case q4:
            if (esOperadorLogico(simbolo))
                return q4;
            break;
        case q5:
            if(esSeparador(simbolo))
                return q5;
            break;
    }
    // Si no se encuentra una transición válida, se devuelve el estado inicial
    return q0;
}

// Función para validar un identificador
bool validarIdentificador(const string& identificador) {
    if (!identificador.empty() && esLetra(identificador[0])) {
        Estado estadoActual = q1;
        for (char simbolo : identificador) {
            estadoActual = transicion(estadoActual, simbolo);
        }
        return estadoActual == q1;
    }
    return false;
}

// Función para validar un operador aritmético
bool validarArimetico(const string& identificador) {
    Estado estadoActual = q0;
    for (char simbolo : identificador) {
        estadoActual = transicion(estadoActual, simbolo);
```

```
    }
    if (estadoActual == q0){
        estadoActual = q2;
        for (char simbolo : identificador) {
            estadoActual = transicion(estadoActual, simbolo);
        }
    }
    return estadoActual == q2;
}

// Función para validar un operador de asignación
bool validarAsignacion(const string& identificador){
    Estado estadoActual = q0;
    for (char simbolo : identificador) {
        estadoActual = transicion(estadoActual, simbolo);
    }
    if(estadoActual == q0){
        estadoActual = q3;
        for (char simbolo : identificador) {
            estadoActual = transicion(estadoActual, simbolo);
        }
    }
    return estadoActual == q3;
}

// Función para validar un operador relacional
bool validarLogico(const string& identificador){
    Estado estadoActual = q0;
    for (char simbolo : identificador) {
        estadoActual = transicion(estadoActual, simbolo);
    }
    if(estadoActual == q0){
        estadoActual = q4;
        for (char simbolo : identificador) {
            estadoActual = transicion(estadoActual, simbolo);
        }
    }
    return estadoActual == q4;
}

// Función para validar un separador
bool validarSepa(const string& identificador){
    Estado estadoActual = q0;
    for (char simbolo : identificador) {
        estadoActual = transicion(estadoActual, simbolo);
```

```
    }
    if(estadoActual == q0){
        estadoActual = q5;
        for (char simbolo : identificador) {
            estadoActual = transicion(estadoActual, simbolo);
        }
    }
    return estadoActual == q5;
}

// Función principal
int main() {
    string nombreArchivo;
    cout << "Ingrese el nombre del archivo de texto: ";
    cin >> nombreArchivo;

    // Abrir el archivo
    ifstream archivo(nombreArchivo);
    if (!archivo.is_open()) {
        cout << "Error al abrir el archivo." << endl;
        return 1;
    }

    // Conjunto de palabras reservadas
    unordered_set<string> palabrasReservadas = {"if", "else", "switch", "while",
"for", "do", "int", "short", "long",
        "double", "import", "retorn", "string", "print", "case", "break"};

    string linea;
    int numeroLinea = 1;
    while (getline(archivo, linea)) {
        cout << "Línea " << numeroLinea << ":" << endl;

        // Procesar cada palabra en la línea utilizando un stringstream
        stringstream ss(linea);
        string palabra;
        while (ss >> palabra) {
            cout << "Palabra: " << palabra;

            // Validar el identificador
            if (validarIdentificador(palabra)) {
                cout << " - Es un identificador";
                if (palabrasReservadas.count(palabra) > 0) {
                    cout << " (Palabra reservada)";
                }
```

```
            }
            // Validar operadores aritméticos
            else if (validarArimetico(palabra)) {
                cout << " - Operador aritmético";
            }
            // Validar operadores de asignación
            else if (validarAsignacion(palabra)){
                cout << " - Operador asignación";
            }
            // Validar operadores relacionales
            else if (validarLogico(palabra)) {
                cout << " - Operador relacional";
            }
            // Validar separadores
            else if (validarSepa(palabra)){
                cout << " - Operador separador";
            }
            // Si no cumple ninguna de las validaciones anteriores, no es un
identificador válido
            else {
                cout << " - No es un identificador";
            }

            cout << endl;
        }

        numeroLinea++;
    }

    // Cerrar el archivo
    archivo.close();

    // Pausar la ejecución para ver los resultados
    system("PAUSE");
    return 0;
}
```