División de Tecnologías para la Integración Ciber-Humana

Departamento de Ciencias Computacionales

Subproducto 0. Analizador léxico básico

Corona Padilla Aldo. 217474545

Seminario de Solución de Problemas de Traductores de Lenguaje 1

Sección: D12

Profesor: Roberto Patiño Ruiz

Fecha de entrega: 30/08/2023

Subproducto 0. Analizador léxico básico

Objetivo Particular: Analizar y probar ejemplos de código para un analizador léxico básico que sea capaz de leer flujo de caracteres de entrada y transformarlo en una secuencia de componentes léxicos.

Desarrollo:

Código de programa 1:

```
#include <iostream>
int main(int argc, char **argv)
{
   // filas representa 3 estados: 0, 1, 2, mientras que el de aceptación es 3 y el estado de error es -1
   // columnas representan las entradas (0 y 1)
   // Este analizador solo puede leer tokens de un solo dígito binario
   int MT[3][2] = {
      {1, 0},
      {-1, 2},
      {-1, 3}
   };
   int simbolo;
   int estado = 0;
   while (estado != 3)
   {
      std::cout << "dame la nueva entrada: ";
      std::cin >> simbolo;
      estado = MT[estado][simbolo];
      std::cout << estado << "\n";
   }
   return 0;
}
```

Este código simula un autómata determinista de tres estados que procesa una secuencia de entradas binarias (0, 1) y cambia de estado en función de las transiciones definidas en la matriz “MT”. La matriz define las transiciones entre estados en función de los símbolos de entrada. Tiene 3 filas (estados) y 2 columnas (símbolos de entrada). Cada elemento de la matriz indica a qué estado se debe transicionar desde el estado actual cuando se recibe un símbolo de entrada específico.

Una optimización que le encontraría al código sería checar que la entrada del usuario esté dentro de los límites del índice de MT y no nos de segfault.

Código de programa 2:

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv)
{
   // Abre un archivo llamado "prueba.txt" en modo lectura de texto
   FILE *fichero;
   fichero = fopen("prueba.txt", "rt");

   // Definición de la matriz de transición
   int MT[3][2] = {{1, 0}, {-1, 2}, {-1, 3}};

   // Variable para almacenar el símbolo de entrada
   int simbolo;

   // Estado inicial de la máquina
   int estado = 0;

   // Bucle que se ejecuta hasta que el estado sea 3
   while (estado != 3)
   {
      // Mostrar mensaje para indicar que se está esperando una nueva entrada
      std::cout << "La nueva entrada: ";

      // Leer un carácter del archivo de entrada
      fscanf(fichero, "%c", &simbolo);

      // Convertir el carácter a un número entero
      simbolo = simbolo - '0';

      // Mostrar el símbolo de entrada leído
      std::cout << "es: " << simbolo << "\n";
```

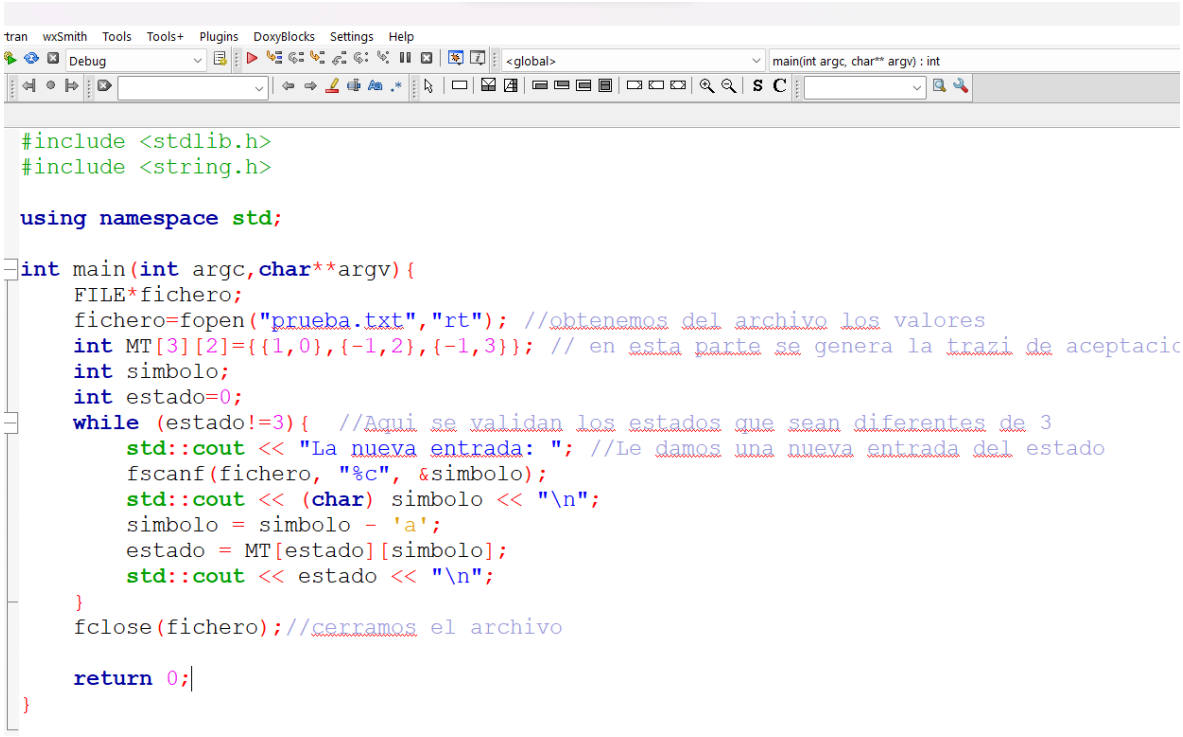
```
        // Realizar una transición de estado basada en la matriz de transición
        estado = MT[estado][simbolo];
        // Mostrar el nuevo estado después de la transición
        std::cout << estado << "\n";
    }

    // Cerrar el archivo de entrada
    fclose(fichero);

    // Devolver 0 como código de salida
    return 0;
}
```

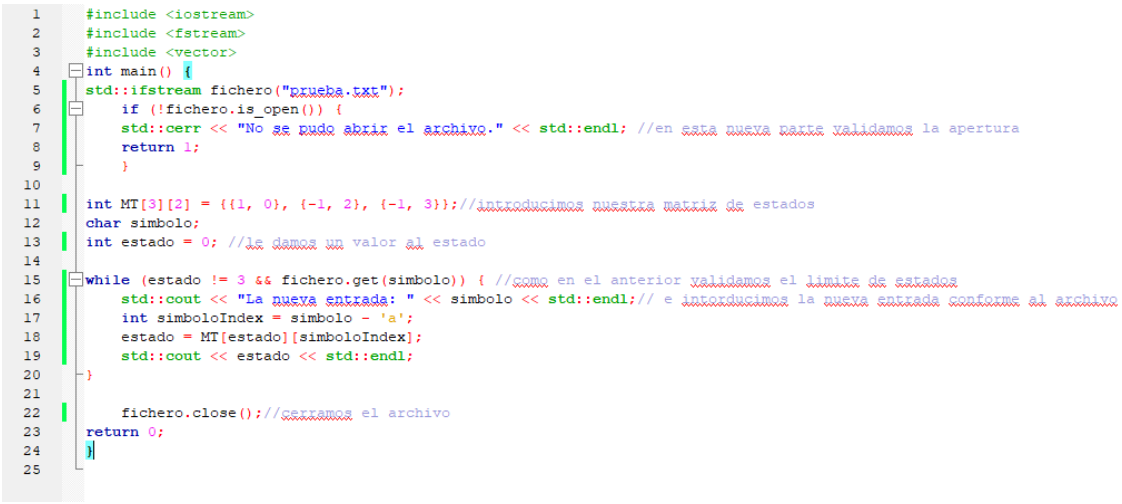
Este código representa una simulación simple de una máquina de Turing que lee una secuencia de entrada desde un archivo llamado "prueba.txt" y realiza transiciones de estado basadas en una matriz de transición predefinida. La máquina de Turing se detiene cuando alcanza el estado 3. Cada vez que se lee un símbolo de entrada, se muestra en la consola junto con el estado actual después de la transición.

Código de programa 3:



para este programa nos genera una matriz de 3 estados con dos entradas ya sea 0,1 u 2 ,en donde cada vez que lo corre, mientras el estado de entrada sea diferente de 3, te pedira una nueva entrada y al igual que una máquina de turing generara nuevos estados hasta llegar a su límite con una mejora en la eficiencia conforme al anterior código.

Código de programa 4:



Para este código al igual que el anterior realizamos una máquina de estados finita en la cual de la misma forma creamos nuestra matriz con estados y validamos conforme los datos extraídos de nuestro txt, pero ahora también mejoramos la validación del archivo que introducimos en caso de que no entre, simplemente finaliza el programa.

Código de programa 5:

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

// Definición de tokens
enum Token
{
   TOKEN_LETRA_A,
   TOKEN_LETRA_B,
   TOKEN_FINAL,
   TOKEN_DESCONOCIDO
};

int main(int argc, char **argv)
{
   // Apertura del archivo "prueba.txt" en modo de lectura
   FILE *fichero;
   fichero = fopen("prueba.txt", "rt");
   if (!fichero)
   {
      std::cerr << "No se pudo abrir el archivo." << std::endl;
      return 1;  // Termina el programa con un código de error
   }

   // Definición de una matriz de transición de estados
   int MT[3][2] = {{1, 0}, {-1, 2}, {-1, 3}};

   char simbolo;
   int estado = 0;

   // Bucle de procesamiento: lee caracteres del archivo y realiza transiciones
   while (estado != 3 && fscanf(fichero, "%c", &simbolo) != EOF)
   {
      // Imprime el símbolo leído en esta iteración
      std::cout << "La nueva entrada: " << simbolo << std::endl;
```

```
        // Determina el token asociado al símbolo leído
        Token token;
        if (simbolo == 'a')
        {
            token = TOKEN_LETRA_A;
        }
        else if (simbolo == 'b')
        {
            token = TOKEN_LETRA_B;
        }
        else if (simbolo == '\n' || simbolo == '\r' || simbolo == EOF)
        {
            token = TOKEN_FINAL;
        }
        else
        {
            token = TOKEN_DESCONOCIDO;
        }

        // Realiza la transición de estado según la matriz de transición
        estado = MT[estado][token];

        // Imprime el token y el nuevo estado
        std::cout << "Token: " << token << ", Estado: " << estado << std::endl;
    }

    // Cierra el archivo después de procesar todos los caracteres
    fclose(fichero);

    return 0;
}
```

Este código toma un archivo de texto como entrada y lo procesa caracter por caracter, determinando el tipo de carácter (token) y realizando transiciones de estado en una máquina de estados finitos definida por la matriz MT. Cada transición y estado se imprime en la consola durante el proceso.

Código de programa 6:

```c
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
// Definición de tokens
enum Token
{
   TOKEN_IDENTIFICADOR,
   TOKEN_ASIGNACION,
   TOKEN_ENTERO,
   TOKEN_SUMA,
   TOKEN_FIN
};
// Función para obtener el siguiente token
enum Token obtenerToken(char *lexema)
{
   // checamos que el caracter actual sea un token
   // lo que hacemos es iterar sobre cada caracter y vemos si el lexema es token o no y a
que token
   // corresponde
   int c = getchar();
   if (c == 'a')
   {
      strcpy(lexema, "a");
      return TOKEN_IDENTIFICADOR;
   }
   else if (c == '=')
   {
      strcpy(lexema, "=");
      return TOKEN_ASIGNACION;
   }
   else if (isdigit(c))
   {
      ungetc(c, stdin);
      scanf("%s", lexema);
      return TOKEN_ENTERO;
   }
   else if (c == '+')
   {
      strcpy(lexema, "+");
      return TOKEN_SUMA;
```

```
    }
    else if (c == '\n' || c == EOF)
    {
        return TOKEN_FIN;
    }
    else
    // podriamos dar error de sintaxis ya que dicho carácter (lexema) no se encuentra en
nuestros tokens
    {
        strcpy(lexema, "DESCONOCIDO");
        return TOKEN_FIN;
    }
}
int main()
{
    char lexema[100];
    enum Token token;
    std::cout << "escribe: ";
    // declaramos un arreglo de caracteres (que contendrá el lexema), después llamamos
la función la cual
    // hará un getchar (entrada del usuario) y compara dicho carácter con nuestros
posibles tokens y copia el
    // char al array de lexema para posteriormente imprimirlo junto con el token devuelto
    do
    {
        token = obtenerToken(lexema);
        printf("Token: %d, Lexema: %s\n", token, lexema);
    } while (token != TOKEN_FIN);

    return 0;
}
```

Este código implementa un analizador léxico básico en C/C++ para reconocer y clasificar tokens en una entrada de texto. Los tokens son las unidades léxicas más pequeñas que conforman un lenguaje de programación, como identificadores, operadores, números y otros elementos (en este caso nuestros tokens son IDENTIFICADOR, ASIGNACIÓN, ENTERO, SUMA, FIN). El analizador léxico procesa el flujo de caracteres de entrada y los agrupa en tokens específicos para su posterior procesamiento.

Conclusiones

En resumen, un analizador léxico es una piedra angular esencial en la construcción de compiladores e intérpretes, desempeñando un papel fundamental en la transformación de código fuente en una forma que el sistema pueda comprender y procesar. Al descomponer el código en unidades léxicas significativas, como tokens, el analizador léxico allana el camino para pasos posteriores en el proceso de compilación o interpretación. Al implementar un analizador léxico básico, hemos explorado la importancia de reconocer y clasificar correctamente los componentes léxicos, como palabras clave, identificadores, operadores y símbolos, sentando así las bases para el procesamiento exitoso de programas informáticos. Aunque nuestro analizador léxico actual puede ser básico, su comprensión subyacente sienta las bases para abordar desafíos más complejos en el desarrollo de software y la creación de herramientas de procesamiento de lenguaje.

Bibliografía

Escobar, R. (s. f.). ***Máquina de estados finitos en C [PIC18F45K50]*****.** http://stg-pepper.blogspot.com/2019/03/maquina-de-estados-finitos-en-c.html

Llamas, L. (2018). Implementar una máquina de estados finitos en Arduino. ***Luis Llamas*****.** https://www.luisllamas.es/maquina-de-estados-finitos-arduino/