UNIVERSIDAD DEL MAGDALENA

TALLER – RECONOCEDOR DE TOKENS

COMPILADORES

INTEGRANTES:

JIMÉNEZ COLÓN ROSSIMAR

RINCONES SOLANO CARLOS DAVID

SOTO PACHECO YESID DAVID

TORRES MONTAÑEZ KEVIN LEONEL

DOCENTE:

ESMEIDE

Fecha de realización: 24/03/2024

Fecha de entrega: 25/03/2024

Informe Taller 2 – Compiladores

• ¿Qué tipo de autómata diseñaron, AFD o AFND?

Para el desarrollo de los autómatas utilizados como complemento para el reconocedor de tokens, autómatas que fueron el reconocedor de variables y de constantes numéricas, utilizamos un AFD

• ¿Cómo se diseñó el autómata escogido?

A continuación, se presenta el diseño de los autómatas utilizados

- Autómata que reconoce variables:

  En primer lugar, el grupo definió los caracteres que una variable puede tener (dígitos, letras, _ y $) para luego realizar el autómata, con cada uno de sus estados desde el inicial, comprobar si eran de aceptación o rechazo, dándonos como resultado:

	L	D	_/$	Aceptación (1) / rechazo (0)
Inicial	valido	error	valido	0
valido	valido	valido	valido	1
error	error	error	error	0

- Autómata que reconoce constantes numéricas:

  Para este autómata, definimos las partes que pueden confirmar una constante numérica (-, +, dígitos, e/E) para luego realizar el autómata, con cada uno de sus estados desde el inicial, y mientras desarrollábamos este, encontramos que podíamos eliminar algunos estados por ser repetitivos, luego comprobar si eran de aceptación o rechazo, y obtener lo siguiente:

Pos	Estado	-/+	dig	.	e/E	Aceptación/rechazo
0	Ini	Sig	Sdig	.	Error	0

1	Sig	Error	Sdig	S.	Error	0
2	Sdig	Error	Sdig	Sdig.	SdigE	1
3	.	Error	.dig	Error	Error	0
4	Sig.	Error	S.dig	Error	Error	0
5	Dec	Error	Dec	Error	decE	1
6	NE	NES	NEd	Error	Error	0
7	NEdig	Error	NEd	Error	Error	1
8	NEsig	Error	NEd	Error	Error	0
9	Error	Error	Error	Error	Error	0

• ¿Cómo se diseña la función de escaneo?

```
void tokenizar(string ruta) {
    ListaTokens tokens;
    tokens.llenarTokens();
    int numeroLinea = 1;
    try{
        fstream file(ruta);
        while (!file.eof()) {
            string linea;
            getline(file, linea);
            tokenizarLinea(linea, tokens, numeroLinea++);
        }
    }catch (const exception &e) {
        cout << e.what() << '\n';
    }
}
```

Imagen1: función tokenizar

```
void tokenizarLinea(string str, ListaTokens tokens, int linea) {
    bool isToken;
    vector<string> vec = splitString(str);
    for (string palabra : vec){
        isToken = false;
        for (int i = 0; i < tokens.nToken(); i++){
            if (palabra == tokens.list.at(i).getSymbol()) {
                cout << tokens.list.at(i).toString() << endl;
                isToken = true;
                continue;
            }
        }
        if (!isToken) {
            switch (evaluar(palabra)) {
            case 1:
                cout << Token(0, "STRING", palabra).toString() << endl;
                break;
            case 2:
                cout << Token(0, "ID", palabra).toString() << endl;
                break;
            case 3:
                cout << Token(0, "CONSTANT", palabra).toString() << endl;
                break;
            case 4:
                cout << "\033[1;31mError en la linea " << linea << " missing character "<<"["<<palabra<<"]\033[0m" << endl;
                break;
            default:
                cout << "\033[1;31mError en la linea " << linea << " token no definido para "<<"["<<palabra<<"]\033[0m" << endl;
                break;
            }
        }
    }
}
```

Imagen2: función tokenizar por línea

La función tokenizar está diseñada para procesar un archivo de texto ubicado en la ruta especificada. Comienza declarando una lista de tokens llamada tokens y la inicializa llenándola con datos utilizando una función no especificada llamada llenarTokens(). A continuación, inicializa una variable numeroLinea para llevar un registro del número de línea actual del archivo.

Dentro de un bloque try, intenta abrir el archivo especificado en modo lectura utilizando un objeto fstream llamado file. Luego, entra en un bucle while que continuará ejecutándose hasta que alcance el final del archivo (eof()). En cada iteración del bucle, lee una línea del archivo y la almacena en una variable llamada linea utilizando la función getline(). Posteriormente, llama a una función tokenizarLinea para procesar la línea actual y extraer tokens de ella. A medida que avanza a la siguiente línea, incrementa el número de línea actual.

Si ocurre alguna excepción durante el proceso de lectura del archivo, como un error de apertura o lectura, se captura mediante el bloque catch. La excepción se almacena en una variable e de tipo const exception&, y se imprime el mensaje de error correspondiente utilizando el método what() de la excepción