ITCR – IC5701.COMPILADORES E INTÉRPRETES


# TAREA #2 – CORRECCIÓN DE ERRORES: FASE Léxica ignorancia de caracteres

**Estudiantes:**

William Alfaro Quirós – Carné: 2022437996

Jesus Cordero Díaz - Carné: 2020081049

Pablo Venegas Sánchez - Carné: 2018083497

**Profesor:**

Emmanuel Ramírez S.

**II Semestre 2024**



# Fase Léxica

- **Descripción**

La fase léxica de un compilador se encarga de analizar el código fuente y dividirlo en unidades de significado conocidas como tokens (por ejemplo, palabras clave, operadores, literales, etc.). Estos tokens son esenciales para las etapas posteriores de análisis. Sin embargo, el código fuente puede incluir caracteres o secuencias inesperadas que no corresponden a ningún token válido.

Durante esta fase, los errores léxicos ocurren cuando el analizador léxico encuentra caracteres no reconocidos o inesperados. En lugar de interrumpir el análisis, la técnica de ignorancia de caracteres permite al compilador omitir estos caracteres, continuando con el análisis de los tokens válidos.



# Técnica de Corrección de Errores Léxicos por Ignorancia de Caracteres.

- **Corrección de errores**

*Esta técnica se basa en un enfoque "no disruptivo" para manejar caracteres inesperados. En lugar de emitir errores o detener el proceso, el analizador léxico simplemente descarta (o ignora) los caracteres que no reconoce. Esto permite que el compilador continúe procesando los tokens válidos sin interrupciones y evita la acumulación de errores léxicos que podrían dificultar el análisis sintáctico y semántico posterior.*



# ALGORITMO

- **Algoritmo**

*El algoritmo de ignorancia de caracteres puede integrarse en el flujo de un analizador léxico mediante un simple conjunto de reglas para comparar cada carácter con patrones válidos. Si un carácter no coincide con ningún patrón, se omite en lugar de generar un error.*



# ALGORITMO

- **Pseudocódigo**

```
algoritmo IgnorarCaracteresInvalidos(entrada):
    lista_tokens = []
    para cada carácter en entrada:
        si carácter coincide con algún patrón válido:
            token = identificarToken(caracter)
            añadir token a lista_tokens
        sino:
            imprimir "Advertencia: Carácter no válido ignorado -", carácter
    devolver lista_tokens
```

Flujo:

El analizador revisa cada carácter en la entrada.

Si el carácter corresponde a un patrón definido (ej., dígito, letra, operador), se convierte en un token y se añade a lista_tokens.

Si el carácter no es reconocible, se imprime una advertencia, lo que permite al usuario conocer los caracteres omitidos.

Este enfoque minimiza las interrupciones y facilita la identificación de caracteres erróneos en el contexto general del código.



# Ejemplo de Gramática en EBNF:

- **Gramática utilizada**

<identificador> ::= <letra> { <letra> | <digito> }
<letra> ::= "a" | "b" | ... | "z" | "A" | "Z"
<operador> ::= "+" | "-" | "*" | "/"
<numero> ::= <digito> { <digito> }
<digito> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"



# CÓDIGO

- **Código**

Error Error

```
String entrada = "abc@123+def#456";
AnalizadorLexico analizador = new AnalizadorLexico();
List<String> tokens = analizador.procesar(entrada);
```

# DEMOSTRACIÓN

- **Uso del Visual Studio Code**

```java
// Método para procesar la entrada
public List<String> procesar(String entrada) {
    List<String> listaTokens = new ArrayList<>();
    StringBuilder tokenActual = new StringBuilder();

    for (char caracter : entrada.toCharArray()) {
        if (LETRAS.indexOf(caracter) >= 0) {
            // Identificadores (inician con letra)
            tokenActual.append(caracter);
        } else if (DIGITOS.indexOf(caracter) >= 0) {
            // Continuar formando identificador o número
            tokenActual.append(caracter);
        } else if (OPERADORES.indexOf(caracter) >= 0) {
            // Guarda el token actual antes de añadir el operador
            if (tokenActual.length() > 0) {
                listaTokens.add(tokenActual.toString());
                tokenActual.setLength(newLength:0); // Reiniciamos el token actual
            }
            listaTokens.add(String.valueOf(caracter)); // Añadir el operador como token
        } else {
            // Ignorar caracteres no válidos
            System.out.println("Advertencia: Carácter no válido ignorado - '" + caracter + "'");
        }
    }
```

```
> java .\AnalizadorLexico.java
Advertencia: Carácter no válido ignorado - '@'
Advertencia: Carácter no válido ignorado - '#'
Tokens generados: [abc123, +, def456]
```

# DUDAS Y/O CONSULTAS

- **Espacio para consultas**



# FASE Léxica ignorancia de caracteres

- **Bibliografía**

Muñoz Ruiz, J. C. (2003). Analizador léxico-sintáctico para el lenguaje de animacuento. https://repositorio.uniandes.edu.co/server/api/core/bitstreams/e65cf1d5-fdac-4edc-aca5-e61464766749/content

del Dago, G., Aguirre, J., Rojo, G. A., Novick, F., Macchi, D. G., & Murchio, A. (2013). DE TECNOLOGÍA E INFORMÁTICA. https://museodeinformatica.org.ar/wp-content/uploads/2017/02/ReTIH-3.pdf