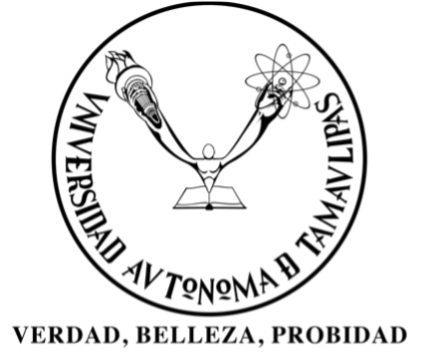
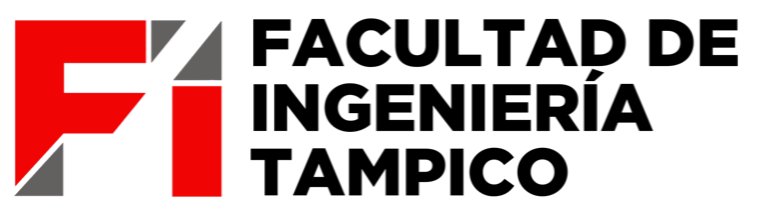
**UNIVERSIDAD AUTÓNOMA DE TAMAULIPAS**

**Facultad de Ingeniería Tampico**

**PROYECTO INTEGRADOR**

****

**Nombre de la Unidad**

**Desarrollo de un Analizador Léxico**

**Nombre de la Asignatura**

**Programación de sistemas de base I**

Docente: Muñoz Quintero Dante Adolfo

**8v**o. Semestre – Grupo “**I**”

2024-2

**Fecha de Entrega:** Lunes, 2 de Diciembre de 2024, 20:00

**Integrante(s):**

Cruz Bonifacio Luis Fernando - 2173390170

Contenido

[1.-Descripcion General. 3](#_Toc184036202)

[2.-Breve explicación de un Analizador Léxico. 4](#_Toc184036203)

[3.- Tareas del Proyecto 7](#_Toc184036204)

[1. Especificación del Lenguaje Personalizado(Simplex): 7](#_Toc184036205)

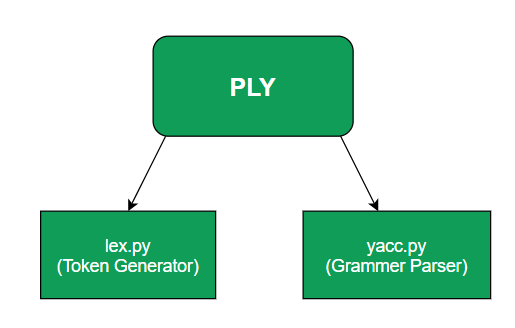
[2. Especificación del Analizador Léxico: 10](#_Toc184036206)

# 1.-Descripcion General.

El objetivo de este proyecto es diseñar un analizador léxico (lexer) para un lenguaje personalizado, que será utilizado en una aplicación creada por el estudiante. Este leer transformará un archivo de texto que contenga un “programa” en una lista de tokens correspondientes a la gramática definida. La especificación del lenguaje será diseñada de manera informal utilizando ejemplos ilustrativos de código.

El proyecto incluye las siguientes fases:

* Diseño de la especificación del lenguaje: Definición de la sintaxis básica del lenguaje.
* Implementación del analizador léxico: Creación de un lexer funcional utilizando Python y su librería Ply (Python Lex-Yacc) que es una herramienta de generación de analizadores en **Python**, inspirada en **Lex** y **Yacc**, que son herramientas tradicionales para la generación de analizadores léxicos y sintácticos en C.
* Pruebas y validación: Verificación del correcto funcionamiento del lexer, incluyendo la identificación de errores léxicos.



# 2.-Breve explicación de un Analizador Léxico.

**¿Qué es un analizador léxico (lexer)?**

Desde la entrada del usuario hasta la ejecución del código, existen tres pasos principales, tanto para intérpretes como para compiladores:

1. **Análisis léxico** (1)
2. **Análisis sintáctico** (2)
3. **Generación de código** (por ejemplo, a código de máquina o bytecode)
4. **Ejecución** (3)

Un **analizador léxico** (o lexer) es una herramienta que realiza el análisis léxico, que es el primer paso en este proceso.

**Diferencia entre análisis léxico y escaneo**

Al principio, el "escaneo" y el "análisis léxico" eran dos pasos distintos, pero debido a la mayor velocidad de los procesadores, ahora ambos términos se utilizan de forma intercambiable y se refieren al mismo proceso.

**¿Qué es el escaneo?**

El **escaneo** significa pasar por una cadena de caracteres, uno por uno, analizando cada carácter (carácter por carácter).

**¿Qué es el análisis léxico?**

Es el proceso mediante el cual los caracteres escaneados se convierten en **lexemas**.

**¿Qué es un lexema?**

Un **lexema** es una secuencia de caracteres reconocida y procesada como una unidad significativa dentro de un lenguaje.

Veamos un ejemplo con una frase:

*The quick brown fox*

Los lexemas serían:

* Lexema 1: the
* Lexema 2: quick
* Lexema 3: brown
* Lexema 4: fox

Ahora, con un fragmento de código similar:

*for(i; i<arr.length; i++){ }*

Los lexemas serían:

* Lexema 1: for
* Lexema 2: (
* Lexema 3: i
* Lexema 4: ;
* Lexema 5: i
* Lexema 6: <
* Lexema 7: arr
* Lexema 8: .
* Lexema 9: length
* Lexema 10: ;
* Lexema 11: i
* Lexema 12: ++
* Lexema 13: )
* Lexema 14: {
* Lexema 15: }

**¿Qué necesita un lexer como entrada?**

Un lexer necesita dos cosas:

1. El código fuente (source code).
2. Las palabras clave (keywords).

**¿Qué es una palabra clave (keyword)?**

Una **palabra clave** es un lexema que tiene un significado especial para el lexer. Normalmente, palabras como *print*, *from*, *to* son conocidas como palabras clave, pero también símbolos como *(* o *{* pueden ser considerados como palabras clave.

**Tipos de palabras clave**

* Existen palabras clave de un solo carácter, como los espacios en blanco.
* Existen palabras clave de múltiples caracteres, como *print*.

**Ignorando palabras clave**

Hay casos en los que queremos ignorar algunas palabras clave, como cuando aparecen entre comillas ("") o dentro de comentarios.

**¿Qué son los identificadores (IDs)?**

Los **identificadores** son nombres definidos por el usuario, como los nombres de espacios de nombres (namespaces), variables, clases y funciones. Las funciones de la biblioteca estándar son identificadores predefinidos. Normalmente, los nombres de palabras clave no pueden ser utilizados como identificadores.

# 3.- Tareas del Proyecto

## 1. Especificación del Lenguaje Personalizado(Simplex):

**Descripción del Lenguaje *Simplex***

**Simplex** es un lenguaje de programación de propósito general diseñado para ser fácil de aprender y utilizar. Su sintaxis es simple y está inspirada en lenguajes como Python y JavaScript, lo que lo hace accesible para programadores novatos, pero con suficiente capacidad para desarrollar proyectos pequeños y medianos.

**Características Principales:**

1. **Sintaxis Simple y Clara**: *Simplex* utiliza una sintaxis sencilla y legible, adecuada para programadores principiantes, eliminando palabras clave complicadas o redundantes.
2. **Control de Flujo**: El lenguaje soporta estructuras básicas de control de flujo, como condicionales (if, else), bucles (while, for) y sentencias de control de ejecución (break, continue).
3. **Tipos de Datos**: *Simplex* maneja tipos de datos primitivos como enteros (int), números decimales (float), cadenas de texto (string) y booleanos (bool). Además, permite declarar variables fácilmente.
4. **Operadores**: Incluye operadores aritméticos (+, -, \*, /), lógicos (and, or), de comparación (==, !=, >, <) y de asignación (=, +=, -=).
5. **Funciones**: *Simplex* permite la definición de funciones con la palabra clave function, sin necesidad de especificar tipos de retorno o parámetros, lo que permite una programación ágil y flexible.
6. **Estructuras de Datos**: Soporta la creación de arrays y listas para almacenar colecciones de datos, con índices numéricos o asociativos.
7. **Manejo de Errores**: El lenguaje proporciona un mecanismo sencillo para el manejo de errores mediante la detección de identificadores no válidos y símbolos desconocidos, generando mensajes claros que incluyen la línea y columna donde ocurrió el error.
8. **Comentarios**: Los comentarios de una sola línea se indican con // y los comentarios multi-línea con /\* \*/, permitiendo una documentación clara dentro del código.

**Ejemplo de Código en *Simplex*:**

simplex

function main() {

// Declaración de variables

int x = 5;

float y = 10.5;

// Estructura condicional

if (x < y) {

print("x es menor que y");

} else {

print("x no es menor que y");

}

// Bucle while

while (x < 10) {

x = x + 1;

}

// Uso de una función

int z = suma(x, y);

print(z);

}

// Función para sumar dos números

function suma(int a, float b) {

return a + b;

}

**Propósito del Lenguaje:**

*Simplex* está diseñado para proyectos pequeños donde se busca rapidez en la escritura y legibilidad del código, pero también con la capacidad de extenderse para tareas de complejidad media. Ideal para programadores novatos, pero también útil en prototipos rápidos y proyectos de scripting.

**Casos de Uso:**

* **Educación**: Como herramienta de aprendizaje de programación.
* **Prototipos**: Para crear prototipos rápidamente sin complicaciones.
* **Automatización**: En scripts simples para automatización de tareas.

## 2. Especificación del Analizador Léxico:

**Objetivo del Analizador Léxico**

El analizador léxico (lexer) de *Simplex* se encargará de leer el código fuente del lenguaje *Simplex* y dividirlo en tokens. Un token es una secuencia de caracteres que tiene un significado en el lenguaje. El lexer identificará y clasificará estos tokens según su tipo, como identificadores, palabras reservadas, operadores, delimitadores y números.

**Elementos del Lexer:**

1. **Identificadores:**
   * **Reglas**: Los identificadores de *Simplex* son secuencias de caracteres alfanuméricos que comienzan con una letra (a-z, A-Z) o un guion bajo (\_), seguidos de letras, números o guiones bajos.
   * **Ejemplo**:
     + Válido: variable, \_variable123
     + Inválido: 123variable, @variable
   * **Expresión Regular**:[a-zA-Z\_][a-zA-Z0-9\_]\*
   * **Token**: IDENTIFIER.
2. **Palabras Reservadas:**
   * **Reglas**: Las palabras reservadas son términos predefinidos que tienen un significado especial en *Simplex*. Estas palabras no pueden usarse como identificadores.
   * **Palabras reservadas**: function, if, else, while, return, print, int, float, string, bool, true, false, etc.
   * **Expresión Regular**: El lexer puede tener un conjunto fijo de palabras clave, que se compararán al leer el texto.
   * **Token**: KEYWORD.
3. **Operadores:**
   * **Reglas**: *Simplex* soporta operadores aritméticos, lógicos y de asignación, entre otros.
   * **Operadores aritméticos**: +, -, \*, /, %
   * **Operadores lógicos**: and, or
   * **Operadores de asignación**: =, +=, -=, \*=, /=, %=
   * **Operadores de comparación**: ==, !=, <, >, <=, >=
   * **Expresión Regular**: [\+\-\\*/%]=?|==|!=|<=|>=|and|or
   * **Token**: OPERATOR.
4. **Delimitadores:**
   * **Reglas**: Los delimitadores son símbolos que se usan para separar sentencias, bloques de código o elementos dentro de las estructuras de datos.
   * **Delimitadores comunes**: ;, ,, {, }, (, ), [, ]
   * **Expresión Regular**: ;|,|\{|\}|\(|\)|\[|\]
   * **Token**: DELIMITER.
5. **Números:**
   * **Reglas**: *Simplex* soporta números enteros y flotantes. Los enteros pueden ser decimales, octales o hexadecimales. Los números flotantes pueden tener una parte decimal opcional.
   * **Números enteros**:
     + Decimales: 123, 456
     + Hexadecimales: 0x1A, 0xFF
     + Octales: 075
   * **Números flotantes**: 3.14, 0.99, 1.0e10
   * **Expresión Regular**:

0x[0-9A-Fa-f]+|0[0-7]+|\d+\.\d+|\d+(\.\d\*)?([eE][+]?\d+)?

* + **Token**: NUMBER.

**Manejo de Errores**

El lexer debe manejar los errores léxicos de manera adecuada. Si encuentra un símbolo que no puede clasificar según las reglas anteriores, generará un token especial de error. Esto es especialmente importante para caracteres no válidos en *Simplex*, como símbolos no permitidos o identificadores mal formateados.

* **Símbolos no válidos**: Cuando se encuentran símbolos no reconocidos (como caracteres fuera del rango alfabético o números mal formateados), el lexer generará un error.
* **Expresión Regular para caracteres no válidos**:

[^a-zA-Z0-9\_ \t\n\r;,.{}()\[\]=+\-\*\/<>!&|]

El token correspondiente será INVALID\_SYMBOL o similar.

**Proceso del Lexer**

1. **Lectura del Código Fuente**: El lexer toma el código fuente como entrada y lo lee carácter por carácter.
2. **Clasificación de Tokens**: Según las expresiones regulares y las reglas definidas para cada tipo de token (identificadores, palabras reservadas, operadores, etc.), el lexer identifica y clasifica los tokens.
3. **Manejo de Errores**: Si se encuentra un símbolo no reconocido, se reporta un error léxico con la ubicación (número de línea y columna).
4. **Generación de Tokens**: Cada token identificado se genera y se almacena en una lista o estructura adecuada para su posterior procesamiento en el analizador sintáctico (parser).

**Ejemplo de Entrada y Salida**

**Entrada:**

Simplex

function main() {

int x = 10;

if (x > 5) {

print("x is greater than 5");

}

}

**Salida del Lexer:**

| **Token** | **Valor** |
| --- | --- |
| KEYWORD | function |
| IDENTIFIER | main |
| DELIMITER | ( |
| DELIMITER | ) |
| KEYWORD | int |
| IDENTIFIER | x |
| OPERATOR | = |
| NUMBER | 10 |
| DELIMITER | ; |
| KEYWORD | if |
| DELIMITER | ( |
| IDENTIFIER | x |
| OPERATOR | > |
| NUMBER | 5 |
| DELIMITER | ) |
| DELIMITER | { |
| KEYWORD | print |
| DELIMITER | ( |
| STRING | "x is greater than 5" |
| DELIMITER | ) |
| DELIMITER | ; |
| DELIMITER | } |

**Resumen del Flujo del Lexer:**

* El lexer recibe un archivo de texto con código fuente en *Simplex*.
* Clasifica las secuencias de caracteres en tokens (identificadores, palabras clave, operadores, delimitadores, números, etc.).
* Si encuentra un símbolo no reconocido o una sintaxis incorrecta, lo marca como error.
* Los tokens generados se pasan al siguiente paso del compilador o intérprete para el análisis sintáctico.

Este enfoque proporcionará una base sólida para el analizador léxico de *Simplex*, asegurando que se puedan procesar correctamente los archivos fuente y que cualquier error léxico sea reportado adecuadamente.

**3. Implementación del Analizador Léxico**

El analizador léxico se implementó utilizando la biblioteca **PLY (Python Lex-Yacc)**, siguiendo las funcionalidades especificadas en el proyecto. Este lexer procesa archivos de texto que contienen código escrito en el lenguaje **Simplex** y realiza las siguientes tareas:

**Funcionalidades del Lexer:**

1. **Identificación de Tokens**.

El lexer identifica y clasifica los elementos válidos en el programa de entrada, generando una lista de tokens. Los tokens incluyen:

* + **T\_KEYWORD**: Palabras clave reservadas como if, else, while.
  + **T\_IDENTIFIER**: Identificadores válidos como mi\_variable o funcion.
  + **T\_INVALID\_IDENTIFIER**: Identificadores inválidos, como aquellos que no cumplen con las reglas del lenguaje (por ejemplo, 123variable).
  + **T\_OPERATOR**: Operadores aritméticos, lógicos y de comparación (+, -, \*, /, >, <, etc.).
  + **T\_ASSIGNMENT\_OPERATOR**: Operadores de asignación (=, +=, -=, etc.).
  + **T\_DELIMITER**: Delimitadores como ;, ,, {, }, ( y ).
  + **T\_NUMBER**: Números enteros, flotantes o hexadecimales (como 10, 3.14, 0x1F).
  + **T\_STRING**: Literales de cadenas de texto, delimitadas por comillas simples o dobles.
  + **T\_COMMENT**: Comentarios de una línea (//) y de múltiples líneas (/\* ... \*/).
  + **T\_UNRECOGNIZED\_SYMBOL**: Símbolos no reconocidos, como $ y #.

1. **Manejo de Errores**.

El lexer detecta e informa errores relacionados con:

* + **Identificadores no válidos**: Identificadores que no cumplen con las reglas del lenguaje, como aquellos que comienzan con un número o contienen caracteres no permitidos.
  + **Símbolos desconocidos**: Caracteres no válidos en el lenguaje, como $ y #.

1. **Mensajes de Error Informativos.**

Cada error léxico incluye:

* + **Número de línea**: Línea del archivo donde ocurrió el error.
  + **Columna**: Posición exacta en la línea donde se detectó el error.
  + **Descripción clara del problema**: Mensaje detallado explicando la naturaleza del error.

4. Pruebas del Lexer:

* Probar el lexer con varios ejemplos que incluyan:
* Programas válidos que cumplan con la especificación del lenguaje.
* Casos que generen errores, para verificar la utilidad de los mensajes de error.