Facultad	Ingeniería	**Carrera**	Informática y Sistemas
Curso	Compiladores	**Sección**	01
Nombre del docente	Pedro David Gabriel Wong		
Correo electrónico del docente	pdgabrielw@correo.url.edu.gt		

Contenido

1 Objetivo

Aplicar de forma práctica los conceptos de compiladores, principalmente la fase de Análisis Léxico para generar un programa que realice la primera fase de la compilación y genere una salida de tokens y su respectiva tabla de símbolos.

2 Objetivos Específicos

- Aplicar los conceptos de análisis léxico para la traducción de un lenguaje.
- Implementar el uso de expresiones regulares para la evaluación de patrones.
- Implementar mecanismos eficientes para la lectura del código fuente.
- Qué el alumno tenga una interacción más profunda con la generación de código y las diferentes tecnologías para desarrolladores.

3 Descripción General del Proyecto

Se tiene la necesidad de crear un nuevo lenguaje de programación orientado a programadores de Latinoamérica, por lo que se le solicita que aplique sus conocimientos en programación y diseño en software para completar el compilador de este nuevo lenguaje de programación.

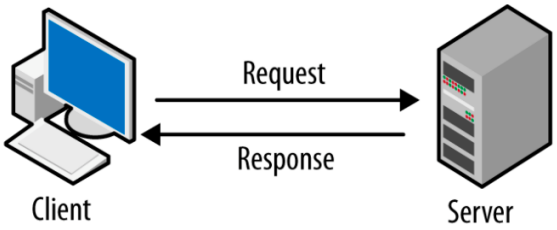
La primera fase del proyecto consta de la elaboración del analizador léxico, en donde debe de contar con un método de ingreso del código fuente a analizar.

3.1 Especificaciones de la solución

1. La lógica del analizador léxico se debe elaborar utilizando el lenguaje de Java.
2. Se debe de proveer una interfaz gráfica que permita la interacción siguiente:
    a. **Programa fuente**: Ingreso de texto o carga de archivo con el código del programa fuente. Debe de poder ver la línea del código fuente, así como permitir la edición de este.
    b. Accionar el proceso de análisis léxico del código fuente.
    c. Contar con los siguientes reportes:
        i. Reporte de Tokens generados
        ii. Reporte de Tabla de Símbolos
        iii. Reporte de Errores Léxicos encontrados.

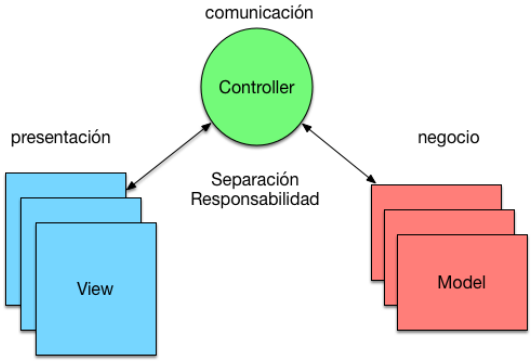
3.2 Arquitectura de la solución

La arquitectura de la solución consiste implementar un Cliente-Servidor.



El Cliente: contendrá la página web con toda la interfaz gráfica con la que interactúa el usuario final.
El Servidor: contendrá toda la lógica, clases y procedimientos para que funcione correctamente el proyecto.

Se recomienda utilizar el patrón Modelo-Vista-Controlador (MVC):



4 Descripción del Lenguaje

1. Lenguaje insensible a mayúsculas y minúsculas

El lenguaje aceptado no es sensible a mayúsculas y minúsculas (Case Insensitive)

Ejemplo

```
EnTEro Contador;
ContadOr = 1;
IF (contador==1){return=1;} Else {return=0;}
```

2. Comentarios

Se debe de aceptar el uso de comentarios dentro del código. Clases de comentario aceptado:

2.1. Comentarios en una sola línea: debe de considerar como comentario todo carácter de la línea posterior a encontrar doble diagonal //.
2.1.1. Inicia con //
2.1.2. Finaliza con salto de línea o fin de archivo.
2.2. Comentarios de varias líneas: comentarios encerrados con /* y */

Ejemplo

//Esto es un comentario en una sola línea
Contador = 0; *//Inicialización de contador*

*/*Esto es un comentario de varias líneas*
Todo esto es parte del
comentario
fin del comentario de varias líneas/*

3. Tipos de Dato

Se debe de contemplar los siguientes tipos de datos para las variables:

Tipo	*Equivalente*	*Descripción*	*Nota*	*Valor por defecto*	*Ejemplo*
Entero	Int	Número enteros	De -2147483648 a 2147483647	0	123, 1, 10, 55320
Real	Double	Número con decimales	Sin límite de decimales	0.0	1.0, 10.523, 123.99, 0.123
Booleano	boolean	Verdadero o Falso		true	true/false
Caracter	char	Acepta un carácter	Delimitado por comilla simple ‘ ’	\u0000 (null/0)	‘1’, ‘p’, ‘a’, ‘=’, ‘&’
Cadena	string	Conjunto de caracteres	Delimitado por comillas dobles “ ”	“” (cadena vacía)	“hola”, “proyecto”, “compilador”

Nota: se debe de poder reconocer números negativos (signo unario “-“ seguido de número real o entero)

4. Declaración de variables

Toda variable que se quiera utilizar en el lenguaje fuente debe de ser declarada previo a su uso. El formato para definir variables es como sigue:
[tipo de dato] [identificador][,identificador]*[=<expresión>];

Ejemplo

Entero intVar;
Entero i, j, k;
Entero i = 1+1;

```
Cadena strVar, palabra = “”;
Real a;
Real a = 5.30;
```

5. Operadores Aritméticos

- **Suma**: operador binario de suma aritmética. Símbolo “+”.
- **Resta**: operador binario de resta aritmética. Símbolo “-”.
- **Multiplicación**: operador binario de multiplicación aritmética. Símbolo “*”.
- **División**: operador binario de división aritmética. Símbolo “/”.
- **Potencia**: operador binario para multiplicar un factor “a” por él mismo un número “b” de veces: a^b. Símbolo “^”.
- **Módulo**: operador binario que devuelve el resto de la división de un número entre otro. Símbolo “#”.
- **AutoIncremento**: operador unario que devuelve el valor del factor aumentado en una unidad. Símbolo “++”.
- **AutoDecremento**: operador unario que devuelve el valor del factor disminuido en una unidad. Símbolo “--” (doble signo menos).

Ejemplo

```
//Operación simple aritmética
5+3*4/3-2

//Uso de operadores en asignación
Contador = intVar*10;
dblVar = 5^3;

//Módulo
Contador = intVar#10; //si intVar=65, el resultado es 5

//Autoincremento
i=0;
i++; //retorna el valor de 1

//Autodecremento
i=1;
i--; //retorna el valor de 0
```

6. Operadores Relacionales

Los operadores relacionales generan un resultado booleano. El objetivo es comparar entre dos expresiones y devolver el valor lógico correspondiente.

Operador	*Descripción*	*Ejemplo*
==	true=si las expresiones son iguales, sino false.	5 == 8 → false "a" == "a"→ true var1 == var3
>=	true=si la expresión izquierda es mayor o igual a la expresión derecha, sino false.	5 >= 8 → false "a" >= "a"→ true var1 >= var3
<=	true=si la expresión izquierda es menor o igual a la expresión derecha, sino false.	5 <= 8 → true "a" <= "a"→ true var1 <= var3
!=	true=si las expresiones NO son iguales, sino false.	5 != 8 → true "a" != "a"→ fasle var1 != var3
<	true=si la expresión izquierda es menor a la expresión derecha, sino false.	5 < 8 → true "a" < "a"→ false var1 < var3
>	true=si la expresión izquierda es mayor a la expresión derecha, sino false.	5 > 8 → false "a" > "a"→ false var1 > var3

Ejemplo

```
//Sobre una expresión condicional
If (variable1 == variable2)

//Asignación
boolVariable = Contador >= 10

//Sobre condición de bucles
For (int i=0; i <= Centinela; i++)
```

7. Operadores Lógicos

Los operadores lógicos comparan entre valores booleanos y devuelven el resultado también como booleano (true/false).

Operador	*Descripción*	*Ejemplo*
||	Equivalente al operador lógico OR	(exp1)||(exp2) true||false→true
&&	Equivalente al operador lógico AND	(exp1)&&(exp2) True&&false→false
!	Negación de una expresión booleana. Devuelve el valor contrario: true→false, false→true	!(5 == 8) → true !('a'=='a')→ false

Ejemplo

```
//ejemplo en sentencia if
If ((a==b) && (a<=10 || !varBool)) { resultado=a*b;}
```

8. Signos especiales

- **Agrupación**: Como signo de agrupación se utilizará los paréntesis "(" y ")".
- **Control**: Como signo de control se tendrá el fin de una sentencia por punto y coma ";". Agrupación de expresiones como if, for, while, funciones; por medio de llaves "{" y "}".

Ejemplo

```
//ejemplo en sentencia if
If ((a==b)&&(a<=(10+5))) { resultado=a*b;}
```

9. Sentencias de control

Se utilizarán las siguientes sentencias de control:

- **IF, IF-Else:** La sentencia IF se forma con la siguiente estructura:

$$\boldsymbol{if}\ (<\boldsymbol{expresión\ booleana}>)\{<\boldsymbol{instrucciones}>\}$$
$$\boldsymbol{if}\ (<\boldsymbol{expr}>)\{<\boldsymbol{instr}>\}\boldsymbol{else}\{<\boldsymbol{instr}>\}$$
$$\boldsymbol{if}\ (<\boldsymbol{expr}>)\{<\boldsymbol{instr}>\}\boldsymbol{else\ if}\ (<\boldsymbol{expr}>)\{<\boldsymbol{instr}>\}$$

Ejemplo

```
//Ejemplo if simple
If (variable == contador) { var01 = 10;} else {var01 = 20;}

//Ejemplo if else
If ((Contador >= 10)&&(boolCentinela)) {
   Operar = true;
} else {
   Operar = false;
   Contador = 0;
```

```
}

//Ejemplo else if
If ((Contador >= 10)&&(boolCentinela)) {
   Operar = true;
} else if (boolCentinela){
   Operar = false;
   Contador = 0;
} else if (!boolCentinela){
   Operar = true;
} else { Contador = 1; }
```

10. Sentencias cíclicas

Se podrán utilizar dos sentencias cíclicas for y while:

- **Sentencia FOR:** la estructura de la sentencia es de la siguiente forma:

$$\boldsymbol{for}(< \boldsymbol{def}.\boldsymbol{variable} >; < \boldsymbol{condición} >; < \boldsymbol{paso} >)\{< \boldsymbol{instr}. >\}$$

- **Sentencia DO-WHILE:** la estructura de la sentencia es de la siguiente forma:

$$\boldsymbol{do}\{< \boldsymbol{instrucciones} >\}\ \boldsymbol{while}(< \boldsymbol{expresión\ booleana} >);$$

Ejemplo

```
//Ejemplo de sentencia for
For (int i=0; i<= Centinela;i++){
   Resultado = Resultado * i;
}

//Ejemplo sentencia do-while
do{
   i++;
}while(i<Maximo);
```

11. Funciones reservadas del Compilador

- **Instrucciones de salida:**
  - EscribirLinea: función que permite escribir en la consola un texto específico ingresado como parámetro de la función. Estructura: $\boldsymbol{EscribirLinea}$("hola Mundo");
  - Escribir: función similar a escribir línea**,** pero no genera un cambio de línea en la consola. Estructura: $\boldsymbol{Escribir}$("Hola");
- **Funciones predefinidad:**
  - Longitud(<var>);: devuelve la longitud del factor indicado.
  - aCadena(<var>);: devuelve un tipo string con el valor que contenga el factor pasado como parámetro.

Ejemplo:

```
//Llamadas de funciones prestablecidas en el compilador
Longitud(“Hola”); //devuelve el valor 4
aCadena(525); //devuelve el valor “525”
```

```
EscribirLinea("Hola Mundo"); // genera una salida en consola con el texto "Hola Mundo"
```

5 Tabla de Símbolos

En el proceso del análisis léxico del programa fuente, se debe de crear y alimentar la tabla de símbolos respectiva de los compiladores. Esta tabla es de utilidad para las siguientes fases del compilador.

La estructura mínima de la tabla debe de ser:

Columna	**Objetivo**
Identificador	Nombre del identificador
Tipo de Token	Nombre del token
Línea	Número de línea donde se encontró
Columna	Posición del primer carácter donde se encontró

Ejemplo:

Identificador	**Tipo de Token**	**Línea**	**Columna**
varCentinela	ID	3	1
Resultado	ID	8	5
i	ID	4	5

6 Control de Errores

Como parte fundamental del analizador léxico, es dar visibilidad de forma detallada de los errores léxicos que se han encontrado durante el análisis del código. Por tal motivo, se debe de generar una salida al programador indicando los errores encontrados en el código fuente.

Ejemplo de salida:
1: Error léxico en línea 5 y columna 10: el carácter "@" es inválido.
2: Error léxico en línea 10, columna 2: el carácter "." no pudo ser reconocido como parte del patrón de número.

El diseño para llevar el control de errores se deja a criterio del estudiante.

Nota: El analizador léxico no debe de detenerse ante el primer error encontrado, sino debe de continuar y exponer todos los errores identificados en el código.

7 Aspectos importantes del proyecto

Requerimientos mínimos

El proyecto debe de cumplir con los siguientes aspectos para que pueda ser revisado:

1. El entregable debe de ser funcional y no tener errores de compilación.
2. Documentación del diseño y arquitectura de la solución.
    a. Lógica del analizador léxico
    b. Expresiones regulares utilizadas
    c. Listado de palabras reservadas
    d. Librerías y dependencias
    e. Arquitectura de su solución
    f. Tecnología utilizada para backend y frontend
    g. Link de GitHub con pantallazo del historial de commits.
3. El proyecto debe de ser trabajo utilizando la plataforma de GitHub (versión gratuita), en donde deberá de demostrar que se tienen un registro de versiones. Mínimo 3 commits.

Datos de Entrega

- **Fecha de entrega**: 19 de marzo 2025
- **Forma de entrega**: El proyecto deberá de ser cargado a la plataforma en Moodle. Se debe de entregar el último commit realizado a la fecha máxima de entrega.
- **El proyecto podrá realizarse en grupos de máximo 4 integrantes**. Cada alumno deberá indicar su grupo utilizando el Portal de la URL a más tardar el 19 de febrero 2025.

8 Distribución de calificación

No	Entregable	Criterio calificación	Puntaje
01	Documentación del proyecto	Documentación Completa	15
02	Manual Técnico-Usuario	Documento claro y preciso que guie al usuario a completar el flujo de la solución.	15
03	Código Fuente – enlace del repositorio	Repositorio en GitHub que demuestre el trabajo colaborativo. Debe de coincidir con la versión subida en el Portal.	20
04	Presentación de proyecto	Breve explicación de funcionalidad del proyecto. Responder preguntas de conceptos del curso sobre su proyecto.	20
05	Ejecución correcta de escenarios	Ejecución correcta de los escenarios de prueba, se harán 5 pruebas.	30
		TOTAL	**100**

9 OBSERVACIONES IMPORTANTES:

- Toda acción de plagio o fraude será penalizada de acuerdo con el **Reglamento de Convivencia del Estudiante Landivariano**, según Artículo 12. Faltas académicas, literales i) *Todas las modalidades de plagio o fraude y en general, cualquier conducta contraria a la verdad y a la honradez encaminada a engañar al docente con intención de obtener un provecho académico personal o ajeno. j) Defraudar el sistema de comprobación del rendimiento académico, ya sea individual o en colaboración con otros para su ejecución. k) Brindar o recibir información por cualquier medio, durante una evaluación; intercambiar exámenes o sustracción de los mismos. l) Suplantar a una persona en cualquier evaluación o actividad académica.*