


 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p><b>INSTITUTO TECNOLÓGICO DE CELAYA</b></p>	
<p><b>PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II</b></p>	<p><b>AUTOR: EQUIPO 4</b></p>	

CARRERA	NOMBRE DE LA ASIGNATURA
INGENIERIA EN SISTEMAS COMPUTACIONALES	LENGUAJES Y AUTÓMATAS II

PROYECTO FINAL NO.	NOMBRE DE LA PRACTICA	FECHA DE ENTREGA
1	Proyecto Final	01 de Diciembre, 2023

1	INTRODUCCION
	<p>Crear un nuevo lenguaje en NetBeans es una tarea fascinante que implica diseñar y desarrollar un entorno de programación personalizado para satisfacer las necesidades específicas de un dominio o tecnología particular. NetBeans, como entorno de desarrollo integrado (IDE), ofrece un marco sólido para la creación de nuevos lenguajes, lo que permite a los desarrolladores extender su funcionalidad más allá de los lenguajes de programación convencionales.</p> <p>Al emprender la creación de un nuevo lenguaje en NetBeans, los desarrolladores tienen la oportunidad de definir la sintaxis, semántica y características únicas que se alineen con los requisitos específicos de su proyecto. Esto implica la creación de gramáticas, analizadores léxicos y semánticos, así como la implementación de funcionalidades de resaltado de sintaxis, autocompletado y otras herramientas esenciales para mejorar la productividad del programador.</p> <p>El proceso puede abarcar desde la definición de las reglas fundamentales del nuevo lenguaje hasta la integración completa en el entorno de desarrollo NetBeans. Esto no solo facilita la escritura de código, sino que también mejora la legibilidad y comprensión del código fuente, ofreciendo una experiencia de desarrollo más intuitiva y eficiente.</p> <p>Crear un nuevo lenguaje en NetBeans no solo es un ejercicio técnico, sino también un medio para potenciar la productividad y la calidad del código en entornos especializados. Con NetBeans como plataforma de desarrollo, los desarrolladores pueden aprovechar las herramientas existentes y la infraestructura sólida del IDE para llevar a cabo esta tarea desafiante con mayor eficacia.</p>



2	OBJETIVO
	<p><b>Objetivo principal</b></p> <p>El objetivo principal al crear un lenguaje de programación propio es diseñar una herramienta que simplifique y potencie el proceso de desarrollo de software, adaptándola a necesidades específicas o desafíos particulares. Este lenguaje debe ser capaz de traducir las ideas y la lógica de un programador en instrucciones que una computadora pueda entender y ejecutar de manera eficiente además de que para que aborden problemas de manera efectiva y expresen sus ideas de una manera única, adaptada a sus necesidades y visión, lo que puede conducir a un desarrollo de software más eficiente y efectivo. Además de comprender los diferentes tipos de análisis para el caso de si se utiliza alguna librería con la que podamos apoyarnos.</p> <p><b>Objetivos secundarios</b></p> <ul style="list-style-type: none"> <li>-Optimizar rendimiento y eficiencia.</li> <li>-Ser flexible y adaptable a diferentes entornos.</li> </ul>

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p><b>INSTITUTO TECNOLÓGICO DE CELAYA</b></p>	
<p><b>PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II</b></p>	<p><b>AUTOR: EQUIPO 4</b></p>	

<ul style="list-style-type: none"> <li>-Priorizar seguridad del código.</li> <li>-Proporcionar documentación completa y clara.</li> <li>-Integrar análisis estático para detectar errores.</li> <li>-Permitir interoperabilidad con otros lenguajes.</li> <li>-Facilitar desarrollo colaborativo.</li> <li>-Admitir múltiples paradigmas de programación.</li> <li>-Ofrecer herramientas de desarrollo integradas.</li> <li>-Fomentar comunidad activa y retroalimentación continua.</li> </ul>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3	MATERIALES NECESARIOS
	<ul style="list-style-type: none"> <li>• Conocimiento en programación y teoría de lenguajes</li> <li>• Herramientas de desarrollo</li> <li>• Documentación</li> <li>• Conjunto de reglas (gramática)</li> <li>• Conocimiento sobre los diferentes análisis</li> <li>• Compilador o interprete</li> <li>• Pruebas y depuración</li> <li>• Hardware</li> <li>• Tiempo y esfuerzo</li> <li>• Colaboración</li> </ul>

4	MARCO TEÓRICO
	<p style="text-align: center;"><b>PROYECTO FINAL</b></p> <p>En un mundo donde la programación se ha convertido en una habilidad esencial, crear un lenguaje de programación propio puede ser una ambiciosa, pero valiosa empresa. Esto podría ser motivado por la necesidad de abordar desafíos específicos o aspirar a un enfoque de desarrollo único. Algunos argumentos para crear un lenguaje de programación propio incluyen:</p> <p><b>Innovación:</b> Desarrollar un nuevo lenguaje te brinda la oportunidad de innovar en términos de sintaxis, características y enfoques de resolución de problemas. Puedes experimentar con ideas revolucionarias que aún no se han implementado en otros lenguajes.</p> <p><b>Eficiencia y productividad:</b> Un lenguaje personalizado puede estar diseñado para ser altamente eficiente en un conjunto particular de tareas, lo que podría aumentar la productividad y reducir el tiempo de desarrollo en proyectos específicos.</p> <p><b>Educación y enseñanza:</b> Un lenguaje propio podría ser una herramienta valiosa para la enseñanza y el aprendizaje de la programación al simplificar conceptos, haciendo que la programación sea más accesible y adaptada a las necesidades educativas.</p> <p><b>Especialización:</b> Puede crear un lenguaje especializado que se adapte perfectamente a un dominio específico, como la ciencia de datos, la robótica o la inteligencia artificial, lo que facilita el desarrollo de aplicaciones en ese campo.</p> <p><b>Propiedad intelectual:</b> Al crear tu propio lenguaje, tienes un control completo sobre los derechos de autor y las licencias, lo que te permite proteger tu propiedad intelectual y establecer reglas</p>

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p><b>INSTITUTO TECNOLÓGICO DE CELAYA</b></p>	
<p><b>PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II</b></p>	<p><b>AUTOR: EQUIPO 4</b></p>	

personalizadas para su uso.

**Comunidad y cultura:** La creación de un lenguaje personalizado puede fomentar el desarrollo de una comunidad en torno a tu proyecto y promover una cultura única de desarrollo de software.

A pesar de estas ventajas, es importante recordar que crear un lenguaje de programación propio es una tarea desafiante que requiere tiempo, recursos y esfuerzo significativos. Debe hacerse con un propósito claro y una comprensión profunda de las necesidades y objetivos que se desean alcanzar.

### ACERCA DE NETBEANS

NetBeans es un entorno de desarrollo integrado (IDE) que proporciona un conjunto de herramientas y características para facilitar el desarrollo de software en varios lenguajes de programación, incluyendo Java, PHP, C++, HTML, entre otros. Si estás considerando crear un lenguaje de programación utilizando NetBeans, algunas ventajas podrían incluir:

**-Entorno de Desarrollo Integrado (IDE):** NetBeans ofrece un entorno de desarrollo completo con características como resaltado de sintaxis, autocompletado de código, depuración y perfiles de rendimiento, lo cual facilita el proceso de desarrollo de un nuevo lenguaje.

**-Soporte para múltiples lenguajes:** NetBeans es políglota, lo que significa que admite varios lenguajes de programación. Esto puede ser beneficioso si tu nuevo lenguaje está destinado a interoperar con otros lenguajes o si planeas agregar funcionalidades específicas de otros lenguajes.

**-Herramientas de desarrollo visual:** NetBeans incluye herramientas visuales que facilitan la creación de interfaces de usuario y la representación visual de estructuras de datos. Esto puede ser útil si tu nuevo lenguaje tiene componentes visuales o interfaces gráficas de usuario.

**-Facilidad de depuración:** NetBeans proporciona un sólido conjunto de herramientas de depuración que te ayudarán a identificar y corregir errores en tu lenguaje. Puedes establecer puntos de interrupción, inspeccionar variables y seguir la ejecución del programa paso a paso.



**-Gestión de proyectos:** NetBeans facilita la organización y gestión de proyectos. Puedes estructurar tu proyecto de lenguaje de manera eficiente, gestionar dependencias y automatizar tareas de construcción.

**-Soporte para sistemas operativos múltiples:** NetBeans es compatible con varios sistemas operativos, lo que significa que puedes desarrollar tu lenguaje en diferentes plataformas, lo cual es beneficioso si buscas la portabilidad de tu lenguaje a través de diferentes entornos.

**-Amplia comunidad y recursos:** NetBeans cuenta con una comunidad activa y una amplia base de usuarios. Esto significa que hay muchos recursos, tutoriales y foros disponibles que pueden ayudarte en el proceso de desarrollo de tu lenguaje.

**-Integración con otras tecnologías:** NetBeans puede integrarse con diversas tecnologías y frameworks, lo que podría ser beneficioso si planeas que tu nuevo lenguaje se integre con otras tecnologías existentes.

Es importante destacar que, aunque NetBeans puede ser una opción sólida para el desarrollo de un

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p><b>INSTITUTO TECNOLÓGICO DE CELAYA</b></p>	
<p><b>PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II</b></p>	<p><b>AUTOR: EQUIPO 4</b></p>	

lenguaje, la elección de la herramienta también dependerá de tus preferencias personales, los requisitos específicos de tu proyecto y la comunidad que rodea a la herramienta.

### ACERCA DE JAVA

La elección de Java para crear un lenguaje de programación puede tener varias ventajas debido a las características y capacidades de Java como lenguaje de programación y plataforma. Aquí algunas ventajas:

**-Portabilidad:** Java es conocido por su portabilidad. Si implementas tu lenguaje sobre la plataforma Java, los programas escritos en tu lenguaje podrán ejecutarse en cualquier sistema que tenga una máquina virtual Java (JVM). Esto facilita la distribución y el uso en una variedad de entornos.

**-Amplia Comunidad y Recursos:** Java tiene una comunidad grande y activa, lo que significa que puedes encontrar abundante documentación, tutoriales y recursos en línea. Esto puede ser útil durante el desarrollo y la resolución de problemas.

**-Herramientas de Desarrollo:** aJava cuenta con excelentes herramientas de desarrollo, como NetBeans, Eclipse y IntelliJ IDEA, que facilitan la escritura, la depuración y la administración de código. Estas herramientas pueden ser valiosas para crear y mantener tu lenguaje.

**-Rendimiento Razonable:** Aunque Java no es tan rápido como algunos lenguajes compilados directamente a código de máquina, su rendimiento es bastante bueno gracias a la optimización de la JVM. Además, la mejora continua en las versiones de Java puede brindar beneficios adicionales de rendimiento.

**-Gestión de Memoria:** La gestión automática de la memoria mediante el recolector de basura en Java puede aliviar la carga de trabajo del programador en comparación con lenguajes que requieren gestión manual de la memoria. Esto puede hacer que el desarrollo de tu lenguaje sea más seguro y menos propenso a errores de memoria.



**-Gran Ecosistema:** Java tiene un ecosistema extenso de bibliotecas y frameworks que podrías utilizar para construir y extender las capacidades de tu lenguaje. Esto puede acelerar el desarrollo y hacer que tu lenguaje sea más versátil.

**-Seguridad:** Java incluye características de seguridad integradas, como el control de acceso y la gestión de permisos. Al basar tu lenguaje en Java, puedes heredar estas características de seguridad.

**-Facilidad de Integración:** Java proporciona mecanismos fáciles para la integración con otros lenguajes y sistemas. Si deseas que tu lenguaje trabaje bien con código Java existente o con bibliotecas escritas en Java, esta integración puede ser ventajosa.

**-Soporte Multihilo:** Java tiene un sólido soporte para programación multihilo, lo que puede ser beneficioso si tu lenguaje pretende aprovechar la concurrencia y el paralelismo en la ejecución de programas.

**-Documentación Rica:** Java es conocido por tener una documentación rica y bien estructurada. Esto puede ser útil al documentar tu propio lenguaje y proporcionar recursos para los usuarios.

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p><b>INSTITUTO TECNOLÓGICO DE CELAYA</b></p>	
<p><b>PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II</b></p>	<p><b>AUTOR: EQUIPO 4</b></p>	

Aunque hay muchas ventajas al elegir Java, también es importante considerar los requisitos específicos de tu proyecto y tus preferencias personales. La elección de la tecnología adecuada dependerá de varios factores, incluidos los objetivos del lenguaje que estás creando y las necesidades de tus usuarios.

#### Librería JFlex

JFlex es una herramienta de generación de analizadores léxicos (scanners) escrita en Java. Se utiliza comúnmente en la construcción de compiladores para analizar el código fuente y dividirlo en tokens. Aquí hay una breve descripción de cómo funciona la librería JFlex:

- **Definición de Especificaciones:**

Creas un archivo de especificaciones (por ejemplo, con extensión .jflex) que contiene reglas para reconocer patrones en el código fuente. Estas reglas definen cómo se deben analizar los tokens.

- **Generación de Analizador Léxico:**

Utilizas JFlex para procesar el archivo de especificaciones y generar el código fuente Java del analizador léxico.

- **Integración con el Compilador:**

Incorporas el código generado por JFlex en tu compilador. Esto puede incluir la creación de clases y métodos específicos para manejar la entrada y salida del analizador léxico.

Ejecución del Analizador Léxico:

Cuando el compilador se ejecuta, el analizador léxico generado por JFlex toma el código fuente como entrada y produce una secuencia de tokens como salida. Cada token representa una unidad léxica (palabra clave, identificador, número, etc.).

Manejo de Errores:

JFlex también puede generar código para manejar errores léxicos, como tokens no reconocidos. Esto permite que el compilador informe errores de manera más precisa al programador.

#### Librería JCUP

JCUP es otra herramienta importante en la construcción de compiladores y se utiliza para generar analizadores sintácticos (parsers). A menudo, JCUP se utiliza en conjunto con JFlex para construir un analizador léxico y un analizador sintáctico completos. Aquí hay una breve descripción de cómo funciona JCUP:

1. Definición de Gramática:

- Creas un archivo que describe la gramática del lenguaje que estás compilando. Este archivo generalmente tiene una extensión .cup. En este archivo, especificas las reglas de producción y las precedencias entre ellas.

2. Generación de Analizador Sintáctico:

- Utilizas JCUP para procesar el archivo de gramática y generar el código fuente Java del analizador sintáctico. Este código contendrá las clases y métodos necesarios para realizar el análisis sintáctico del código fuente.

3. Integración con JFlex:

- Combinas el código generado por JCUP con el código generado por JFlex. Esto



TECNOLÓGICO  
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II

AUTOR: EQUIPO 4

- implica conectar el analizador léxico (JFlex) con el analizador sintáctico (JCUP) para construir un sistema completo de análisis léxico y sintáctico.
4. Manejo de Acciones Semánticas:
    - Puedes agregar acciones semánticas en el archivo de gramática para especificar qué hacer cuando se encuentra una determinada regla de producción. Estas acciones se ejecutan durante el análisis sintáctico y permiten realizar tareas como la construcción del árbol de sintaxis abstracta (AST) o la verificación semántica.
  5. Generación de Árbol de Sintaxis Abstracta (AST):
    - El analizador sintáctico construido con JCUP genera un árbol de sintaxis abstracta que representa la estructura jerárquica del código fuente.
    -



JCUP trabaja en conjunto con JFlex para completar la construcción de un compilador. Mientras que JFlex se encarga del análisis léxico, JCUP se ocupa del análisis sintáctico, permitiendo así la creación de un compilador que pueda comprender la estructura gramatical del lenguaje fuente y generar un árbol de sintaxis abstracta correspondiente.

### Compiler Tools

Conjunto de herramientas que se utilizan en el desarrollo de compiladores. Estas herramientas ayudan en la implementación de diversas fases del proceso de compilación, que convierte el código fuente de un programa en un código ejecutable. Aquí hay algunas de las herramientas comunes utilizadas en el desarrollo de compiladores:

1. Lexer Generators (Generadores de Analizadores Léxicos):
  - Ejemplos: JFlex, Lex, Flex
  - Descripción: Estas herramientas generan analizadores léxicos a partir de especificaciones que definen patrones de tokens en el código fuente. Los analizadores léxicos escanean el código fuente y generan tokens para ser procesados por el analizador sintáctico.
2. Parser Generators (Generadores de Analizadores Sintácticos):
  - Ejemplos: Bison, Yacc, ANTLR (también puede generar analizadores léxicos)
  - Descripción: Estas herramientas generan analizadores sintácticos a partir de especificaciones gramaticales. Los analizadores sintácticos analizan la estructura del código fuente según la gramática del lenguaje y generan un árbol de sintaxis abstracta (AST) o realizan acciones semánticas.
3. Semantic Analysis Tools (Herramientas de Análisis Semántico):
  - Descripción: Estas herramientas realizan análisis semántico en el código fuente para verificar la coherencia semántica y aplicar reglas específicas del lenguaje. Pueden incluir la generación de tablas de símbolos, comprobación de tipos, y otras tareas relacionadas con la semántica del programa.
4. Intermediate Code Generators (Generadores de Código Intermedio):
  - Descripción: Estas herramientas generan código intermedio, una representación intermedia del código fuente que es más fácil de optimizar y traducir a código de máquina. El código intermedio actúa como una capa intermedia entre el código fuente y el código de máquina final.
5. Code Optimizers (Optimizadores de Código):
  - Descripción: Estas herramientas realizan optimizaciones en el código intermedio para mejorar su rendimiento o reducir su tamaño. Pueden incluir optimizaciones como la eliminación de código muerto, la propagación de constantes, y la reordenación de instrucciones.



 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p><b>INSTITUTO TECNOLÓGICO DE CELAYA</b></p>	
<p><b>PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II</b></p>	<p><b>AUTOR: EQUIPO 4</b></p>	

#### 6. Code Generators (Generadores de Código):

- Descripción: Estas herramientas generan el código de máquina final a partir del código intermedio optimizado. El código de máquina es el conjunto de instrucciones ejecutado por la arquitectura específica del procesador.

#### 7. Linkers:

- Descripción: Los enlazadores se utilizan para combinar múltiples archivos de código objeto generados por el compilador en un programa ejecutable. También resuelven referencias a bibliotecas y símbolos externos.

Estas herramientas son esenciales en el desarrollo de compiladores y juegan un papel crucial en la transformación del código fuente de un programa en un ejecutable funcional. La combinación adecuada de estas herramientas permite construir un compilador eficiente y preciso.

### **LINCOTE: EL LENGUAJE DE PROGRAMACIÓN CON ESPIRITU LINCE**

Lincote será la propuesta de un lenguaje de programación con un nombre que va más allá de una mera etiqueta; es un homenaje a la fuerte identidad del tecnológico de Celaya, cuya mascota emblemática es el lince. Esta fusión de "lince" y "código" representa una simbiosis única entre la astucia del mundo animal y la lógica precisa del mundo de la programación.

La elección del nombre "Lincote" trae consigo una sensación de agilidad, ingenio y adaptabilidad, reflejando la naturaleza furtiva del lince en su entorno. Al igual que el lince es conocido por su habilidad para adaptarse a diferentes situaciones, Lincote se ha diseñado para adaptarse y evolucionar con las cambiantes necesidades tecnológicas.

La conexión con la mascota Estudiantil no solo agrega una dimensión de orgullo y pertenencia a la comunidad estudiantil y académica, sino que también subraya la importancia de la educación y la innovación en el desarrollo de Lincote. Es un lenguaje que simboliza el espíritu de aprendizaje y la destreza intelectual, combinando la esencia de un lince con la precisión del código.

### **GRAMÁTICA LIBRE DE CONTEXTO**

En este proyecto, el equipo hará una adopción de una gramática formal y libre de contexto, ya que esta se ha convertido en una práctica estándar y esencial en el diseño y desarrollo de lenguajes de programación. Esta norma se ha establecido gracias a una serie de ventajas y beneficios que ofrece a lo largo de todo el ciclo de vida del lenguaje y su utilización por parte de los programadores.

La principal razón detrás de esta práctica es la claridad y consistencia que aporta. Una gramática formal proporciona una estructura precisa y bien definida para el lenguaje, lo que facilita la comprensión del código fuente tanto para los programadores que lo escriben como para las herramientas que lo procesan. Esta claridad reduce la ambigüedad y las interpretaciones erróneas, lo que contribuye a la calidad del código y a la eficiencia en el desarrollo.

Además de la claridad, la gramática formal tiene un impacto significativo en la compatibilidad y la portabilidad. Al establecer reglas sintácticas sólidas, se garantiza la coherencia en diferentes implementaciones del lenguaje, lo que simplifica la tarea de migrar o ejecutar código en diversas plataformas y sistemas, lo que a su vez amplía la adopción del lenguaje.



TECNOLÓGICO  
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

La gramática formal también facilita el análisis sintáctico eficiente del código, lo que es fundamental para la creación de compiladores, analizadores y herramientas de desarrollo. Esta característica permite verificar rápidamente si el código cumple con las reglas del lenguaje, lo que ahorra tiempo y reduce errores.

### COMPILADOR

Lincode se basará en un enfoque de compilación en lugar de interpretación, lo que significa que su código se traduce en un formato ejecutable antes de la ejecución. Esta elección estratégica ofrece ventajas significativas para los desarrolladores y usuarios finales.

La compilación, en contraste con la interpretación, permite una mayor optimización del código. Es especialmente beneficioso para al aplicarlo se requieren un buen rendimiento .

La decisión de la compilación también conlleva ventajas en términos de seguridad y protección de la propiedad intelectual. El código fuente original se convierte en un formato que es más difícil de acceder y modificar, lo que protege la propiedad intelectual de los desarrolladores.

El proceso de compilación permite la detección temprana de errores de sintaxis y de lógica, lo que resulta en un código más confiable. Además, las herramientas de análisis estático pueden ofrecer un mayor nivel de seguridad, ya que pueden evaluar y mejorar el código antes de la ejecución.

### ESTRUCTURA GENERAL DEL LENGUAJE

<programa> ::= <directiva\_inclusion> <declaracion\_main>

<directiva\_inclusion> ::= "#include <nombre\_biblioteca>"

<declaracion\_main> ::= "int main()" "{" <declaraciones\_locales> <sentencias> "}"

<declaraciones\_locales> ::= <declaracion\_variable> | <declaracion\_variable> <declaraciones\_locales>

<declaracion\_variable> ::= <tipo> <identificador> ";"

<tipo> ::= "int" | "float" | "String" |

<sentencias> ::= <sentencia> | <sentencia> <sentencias>

<sentencia> ::= <if\_statement> | <while\_loop> | <expresion> ";"

<if\_statement> ::= "if" "(" <expresion> ")" "{" <sentencias> "}" "else" "{" <sentencias> "}" |  
"if" "(" <expresion> ")" "{" <sentencias> "}"

<while\_loop> ::= "while" "(" <expresion> ")" "{" <sentencias> "}"

<expresion> ::= <termino> | <termino> "+" <expresion> | <termino> "-" <expresion>

<termino> ::= <factor> | <factor> "\*" <termino> | <factor> "/" <termino>





TECNOLÓGICO  
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

<factor> ::= <entero> | <identificador> | "(" <expresion> ")"

<identificador> ::= <letra> <resto\_identificador>

<resto\_identificador> ::= <letra> | <digito> | "\_"

<letra> ::= "a" | "b" | ... | "z" | "A" | "B" | ... | "Z"

<digito> ::= "0" | "1" | ... | "9"

#### Ejemplo de programa

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

int main(){
    int x = 10, y = 5;
    printf("Ingrese un numero por favor : ");
    scanf("%d", &x);
    printf("Ingrese otro numero por favor: ");
    scanf("%d", &y);
    printf("\nSENTENCIA IF-ELSE : \n");
    if(x > y){
        printf("Numero 1 (%d) es mayor que Numero 2 (%d) \n\n",x,y);
    }else{
        printf("Numero 2 (%d) es mayor que Numero 1 (%d) \n\n",y,x);
    }
    system("pause");
    system("cls");
    printf("\nIngrese otro numero para probar el \"FOR\" por favor: ");
    scanf("%d", &x);
    int num = 1;
    printf("\nSENTENCIA FOR : \n");
    for(int i = x; i < x+10;i++){
        printf("\n Iteracion %d, numero: %d ",num,i);
        ++num;
    }
    printf("\n");
    system("pause");
    return 0;
}
```

Este programa en lenguaje C realiza las siguientes acciones:

**\*Declaración e Inicialización de Variables:** Se declaran dos variables enteras, x e y, con valores iniciales de 10 y 5, respectivamente.

**\*Entrada de Datos desde el Usuario:** Se solicita al usuario ingresar dos números, que son almacenados en las variables x e y.



TECNOLÓGICO  
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

**\*Comparación de Números (Sentencia IF-ELSE):** Se utiliza una sentencia if-else para comparar los valores de x e y. Se imprime un mensaje indicando cuál de los dos números es mayor.

**\*Pausa y Limpieza de Pantalla:** Se pausa la ejecución del programa para que el usuario pueda ver los resultados hasta que presione una tecla. La pantalla de la consola se limpia (solo en sistemas Windows) para mejorar la presentación.

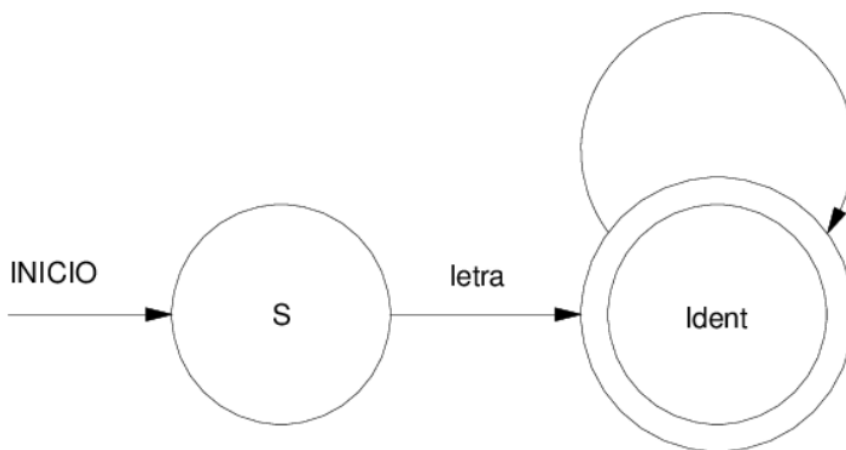
**\*Entrada de Otro Número para la Sentencia FOR:** Se solicita al usuario ingresar otro número, que se almacena en la variable x.

**\*Impresión de Números con Iteración (Sentencia FOR):** Utilizando una sentencia for, se imprimen los números desde x hasta x+9, junto con el número de iteración.

**\*Pausa antes de Cerrar el Programa:** Se pausa nuevamente para que el usuario pueda revisar los resultados antes de cerrar la consola.

**\*Indicador de Éxito al Sistema Operativo:** El programa devuelve 0 para indicar que se ejecutó correctamente al sistema operativo.

**Autómata para reconocer un identificador**





TECNOLÓGICO  
NACIONAL DE MÉXICO®

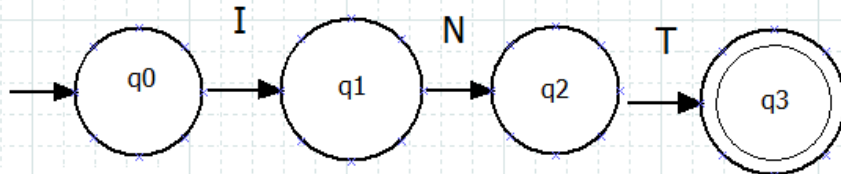
INSTITUTO TECNOLÓGICO DE CELAYA



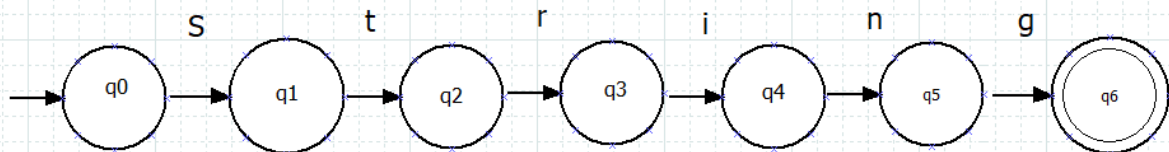
PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II

AUTOR: EQUIPO 4

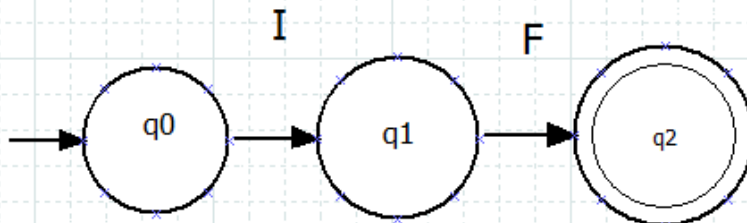
Autómata para tipo de dato INT



Autómata para tipo de dato String



Automata para IF





TECNOLÓGICO  
NACIONAL DE MÉXICO

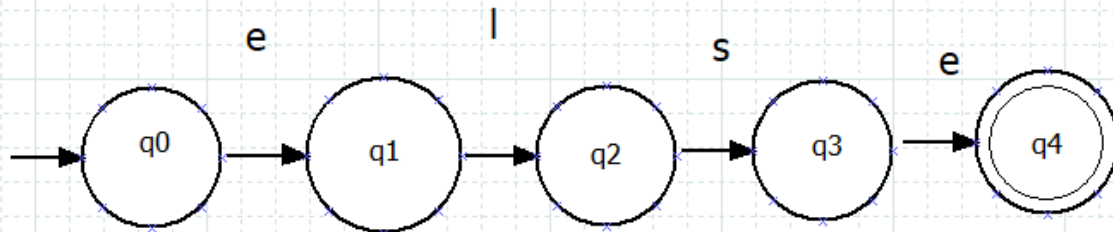
INSTITUTO TECNOLÓGICO DE CELAYA



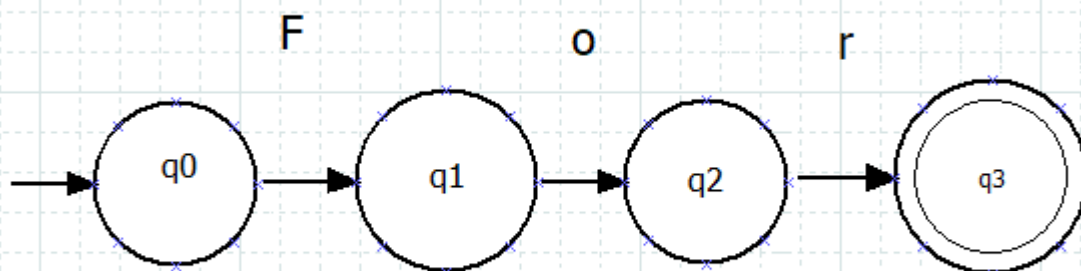
PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II

AUTOR: EQUIPO 4

Autómata para else



Autómata para For





TECNOLÓGICO  
NACIONAL DE MÉXICO®

## INSTITUTO TECNOLÓGICO DE CELAYA



### PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II

AUTOR: EQUIPO 4

Tabla de Tokens

Token	Declaración	Símbolo
1	return	Return
2	Llave_c	}
3	Incremento	++
4	ComparadorIgual	=
5	Llave_a	{
6	std	Std
7	Comilla_simple	'
8	Break	break
9	coma	,
10	Parent_c	)
11	Parent_a	(
12	Corchete_c	]
13	true	True
14	menorQue	<
15	Corchete_a	[
16	Punto	.
17	Default	Default
18	Decremento	--
19	For	for
20	ReservadoNumeral	#
21	ReservadoInclude	Include
22	Identificador	
23	MayorQue	>
24	int	Int
25	Numero	1-9
26	Punto y coma	;
27	scanf	scanf
28	cadena	Hola
29	AND	&
30	IF	If
31	Else	else
32	SYSTEM	System
33	PRINTF	printf
34	FLOAT	float
35	STRING	String
36	Suma	+
37	Resta	-
38	Multiplacion	*
39	MAIN	main
40	O_logico	
41	Division	/



TECNOLÓGICO  
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

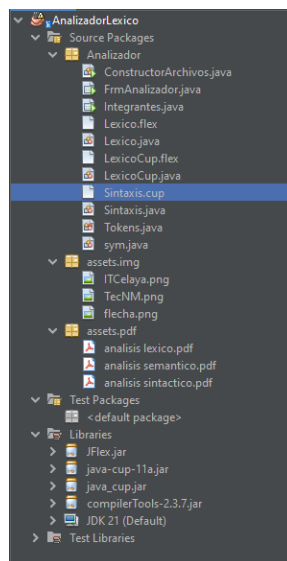
**AUTOR: EQUIPO 4**

## 5 Desarrollo

### Estructura General del proyecto

A continuación se muestra la estructura final del proyecto del compilador, la cual tiene tres paquetes donde el principal es el de analizador donde contienen todas las clases que se utilizaron en la elaboración del proyecto, la de assets.img que es el paquete donde se almacenan todas las imágenes que se utilizaron en la interfaz, por ultimo el de assets.pdf que al igual que las imágenes se almacenan todos los pdf de los análisis para así el usuario sepa que es y como funciona cada tipo de análisis en base a los trabajos e investigaciones que se realizaron a lo largo del semestres.

La parte de librerías tiene todas las librerías que Utilizamos como Jflex y JavaCup que nos ayuda a realizar más fácilmente los tres análisis presentados, y también tenemos el compilerTools que nos ayuda a enumerar las líneas de nuestro código.







TECNOLÓGICO  
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA

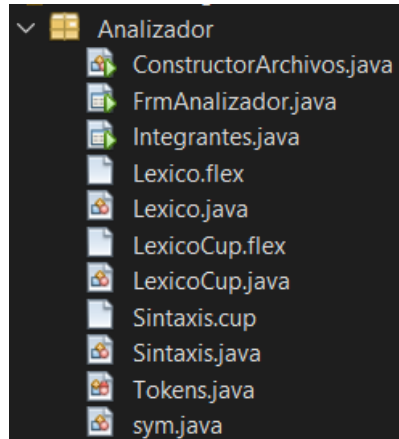


**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

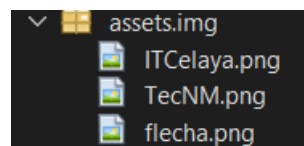
**AUTOR: EQUIPO 4**

### Estructura de las carpetas

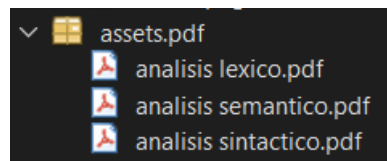
#### Analizador



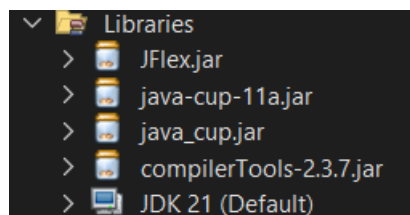
#### assets/img



#### assets/pdf



#### Librerías usadas (Libraries)





TECNOLÓGICO  
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

### Clases para poder llevar a cabo el compilador

Es importante mencionar parte del código de las clases que nos fueron más útiles, es más que claro que cualquier clase es de mucha importancia, pero en este caso las que nos pudieron validar todos los diferentes análisis son:

#### Clase para los tokens

```
package Analizador;  
public enum Tokens {  
    Bool,  
    BitAnd,  
    BitOr,  
    Break,  
    Byte,  
    Case,  
    Char,  
    Cin,  
    Coma,  
    Comilla_simple,  
    Comillas,  
    ComparadorIgual,  
    Const,  
    Continue,  
    Corchete_a,  
    Corchete_c,  
    Cout,  
    Default,  
    Define,  
    Decremento,  
    Diferente,  
    Division,  
    DivisionIgual,  
    Do,  
    Double,  
    DobleMayor,  
    DobleMenor,  
    DosPuntos,  
    Else,  
    Endl,  
    ERROR,  
    Et,  
    False,  
    Float,  
    For,  
    Identificador,  
    If,  
    Igual,  
    Int,  
    Include,
```



TECNOLÓGICO  
NACIONAL DE MÉXICO®

## INSTITUTO TECNOLÓGICO DE CELAYA



### PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II

AUTOR: EQUIPO 4

Incremento,  
Iostream,  
Linea,  
Llave\_a,  
Llave\_c,  
Long,  
Main,  
MasIgual,  
MenosIgual,  
MayorQue,  
MenorQue,  
MayorIgual,  
MenorIgual,  
Modulo,  
ModuloIgual,  
Multiplicacion,  
MultiplicacionIgual,  
Namespace,  
Negador,  
Numero,  
Numeral,  
O\_logico,  
Parent\_a,  
Parent\_c,  
P\_coma,  
Printf,  
Punto,  
Register,  
Resta,  
Return,  
Scanf,  
Short,  
Suma,  
Std,  
\_System,  
\_String,  
Struct,  
Switch,  
STRING\_LITERAL,  
Typedef,  
True,  
Unsigned,  
Using,  
Void,  
While,  
Y\_logico

}



**AUTOR: EQUIPO 4**

*James*

*[Signature]*





TECNOLÓGICO  
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

```
/* Marcador define*/
( "define" ) {lexemas=yytext(); return Define;}
/* Marcador while*/
( "while" ) {lexemas=yytext(); return While;}
/* Operador Division*/
( "/" ) {lexemas=yytext(); return Division;}
/* Salto, tabulacion y espacio */
( "\\n" | "\\t" ) { /*Ignore*/ }
/* Marcador Do*/
( "do" ) {lexemas=yytext(); return Do;}
/* Double */
(double) {lexemas=yytext(); return Double;}
/* Marcador Else*/
( "else" ) {lexemas=yytext(); return Else;}
/* Ciclo repetitivo for */
( "for" ) {lexemas=yytext(); return For;}
/* Condicional if */
( "if" ) {lexemas=yytext(); return If;}
/* Comparador Mayor igual*/
( ">=" ) {lexemas=yytext(); return MayorIgual;}
/* Comparador Menor igual*/
( "<=" ) {lexemas=yytext(); return MenorIgual;}
/* Operador Más igual*/
( "+=" ) {lexemas=yytext(); return MasIgual;}
/* Operador Menos igual*/
( "-=" ) {lexemas=yytext(); return MenosIgual;}
/* Operador Multiplica igual*/
( "*" ) {lexemas=yytext(); return MultiplicacionIgual;}
/* Operador Division igual*/
( "/" ) {lexemas=yytext(); return DivisionIgual;}
/* Operador Comparador igual*/
( "==" ) {lexemas=yytext(); return ComparadorIgual;}
/* Operador Diferente */
( "!=" ) {lexemas=yytext(); return Diferente;}
/* Operador Negador*/
( "!" ) {lexemas=yytext(); return Negador;}
/* Incremento */
( "++" ) {lexemas=yytext(); return Incremento;}
/* Decremento*/
( "--" ) {lexemas=yytext(); return Decremento;}
/* Operador Igual */
( "=" ) {lexemas=yytext(); return Igual;}
/* Doble mayor*/
( ">>" ) {lexemas=yytext(); return DobleMayor;}
/* Modulo igual*/
( "%" ) {lexemas=yytext(); return ModuloIgual;}
/* Doble menor*/
( "<<" ) {lexemas=yytext(); return DobleMenor;}
```



TECNOLÓGICO  
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

```
/* Operador Mayor que */
( ">" ) {lexemas=yytext(); return MayorQue;}
/* Operador Menor que */
( "<" ) {lexemas=yytext(); return MenorQue;}
/* Signo dos puntos */
( ":" ) {lexemas=yytext(); return DosPuntos;}
/* operador y logico */
( "&&" ) {lexemas=yytext(); return Y_logico;}
/* operador bitand */
( "&" ) {lexemas=yytext(); return BitAnd;}
/* operador o logico */
( "||" ) {lexemas=yytext(); return O_logico;}
/* operador bitor */
( "|" ) {lexemas=yytext(); return BitOr;}
/* Integer */
(int) {lexemas=yytext(); return Int;}
/* Bool */
(bool) {lexemas=yytext(); return Bool;}
/* Long */
(long) {lexemas=yytext(); return Long;}
/* Short */
(short) {lexemas=yytext(); return Short;}
/* Float */
(float) {lexemas=yytext(); return Float;}
/* Byte */
(byte) {lexemas=yytext(); return Byte;}
/* Char */
(char) {lexemas=yytext(); return Char;}
/* Marcador include */
( "include" ) {lexemas=yytext(); return Include;}
/* Marcador namespace */
( "namespace" ) {lexemas=yytext(); return Namespace;}
/* Marcador std */
( "std" ) {lexemas=yytext(); return Std;}
/* Marcador iostream */
( "iostream" ) {lexemas=yytext(); return Iostream;}
/* Marcador using */
( "using" ) {lexemas=yytext(); return Using;}
/* Marcador false */
( "false" ) {lexemas=yytext(); return False;}
/* Marcador true */
( "true" ) {lexemas=yytext(); return True;}
/* Salto de linea endl */
( "endl" ) {lexemas=yytext(); return Endl;}
/* Salto de linea */
( "\n" ) {return Linea;}
/* Llave de apertura */
( "{" ) {lexemas=yytext(); return Llave_a;}
```





TECNOLÓGICO  
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

```
/* Llave de cierre */
( "}" ) {lexemas=yytext(); return Llave_c;}
/* Marcador de inicio de algoritmo */
( "main" ) {lexemas=yytext(); return Main;}
/* Marcador register*/
( "register" ) {lexemas=yytext(); return Register;}
/* Salida por consola*/
( "cout" ) {lexemas=yytext(); return Cout;}
/* Marcador Void*/
( "void" ) {lexemas=yytext(); return Void;}
/* Marcador Typedef*/
( "typedef" ) {lexemas=yytext(); return Typedef;}
/* Marcador switch*/
( "switch" ) {lexemas=yytext(); return Switch;}
/* Marcador unsigned*/
( "unsigned" ) {lexemas=yytext(); return Unsigned;}
/* Entrada por consola */
( "cin" ) {lexemas=yytext(); return Cin;}
/* Operador Multiplicación */
( "*" ) {lexemas=yytext(); return Multiplicacion;}
/* Operador Modulo*/
( "%" ) {lexemas=yytext(); return Modulo;}
/* Numeral # */
( "#" ) {lexemas=yytext(); return Numeral;}
/* Parentesis de apertura */
( "(" ) {lexemas=yytext(); return Parent_a;}
/* Parentesis de cierre */
( ")" ) {lexemas=yytext(); return Parent_c;}
/* P_coma */
( ";" ) {lexemas=yytext(); return P_coma;}
/* punto */
( "." ) {lexemas=yytext(); return Punto;}
/* coma */
( "," ) {lexemas=yytext(); return Coma;}
/* Operador Resta */
( "-" ) {lexemas=yytext(); return Resta;}
/* Return */
( "return" ) {lexemas=yytext(); return Return;}
/* Operador Suma */
( "+" ) {lexemas=yytext(); return Suma;}
/* Identificador */
{L}{L}{D}* {lexemas=yytext(); return Identificador;}
/* Numero */
("-( "{D}+" )" | ("-{D}+.{D}+" )" | -({D})+ | {D}+ | {D}+.{D}+ | -{D}+.{D}+ {lexemas=yytext(); return
Numero;}
/* Error de analisis */
. {return ERROR;}
```



TECNOLÓGICO  
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

### Clase para la sintaxis

```
package Analizador;
import java_cup.runtime.Symbol;
parser code
{
    private Symbol s;
    public void syntax_error(Symbol s){
        this.s = s;
    }
    public Symbol getS(){
        return this.s;
    }
};

terminal Bool, BitAnd, BitOr, Break, Byte, Case, Char, Cin, Coma, Comilla_simple, Comillas,
    ComparadorIgual, Const, Continue, Corchete_a, Corchete_c, Cout, Default, Define, Decremento,
    Diferente, Division, DivisionIgual, Do, Double, DobleMayor, DobleMenor, DosPuntos, Else, Endl,
    ERROR, Et, False, Float, For, Identificador, If, Igual, Int, Include, Incremento, Iostream, Linea,
    Llave_a, Llave_c, Long, Main, MasIgual, MenosIgual, MayorQue, MenorQue, MayorIgual,
    MenorIgual,
    Modulo, ModuloIgual, Multiplicacion, MultiplicacionIgual, Namespace, Negador, Numero, Numeral,
    O_logico, Parent_a, Parent_c, P_coma, Printf, Punto, Register, Resta, Return, Scanf, Short, Suma,
    System,
    Std, _String, Struct, Switch, Typedef, True, Unsigned, Using, Void, While, Y_logico, STRING_LITERAL;
non terminal INICIO, SENTENCIA, DECLARACION, IF, FOR, COMPLEMENTO,
    SENTENCIA_BOOLEANA_COMPUESTA, SENTENCIA_BOOLEANA_SIMPLE,
    OPERADORES_RELACIONALES, DECLARACION_ANIDADA, IMPORTACION,
    SENTENCIA_BOOLEANA_ANIDADA, FOR_PARTE_UNO, OPERACIONES_COMPLEMENTARIAS,
    WHILE, DO_WHILE, OPERADORES_ARITMETICOS, COUT, SALIDA, CIN, ENTRADA,
    CASE, SWITCH, CREACION_ARREGLO, DECLARACION_ARREGLO_ANIDADA,
    DECLARACION_ARREGLO, IDENTIFICADORES, PRINTF, SCANF;
start with INICIO;
IMPORTACION ::=
    Numeral Include MenorQue Identificador Punto Identificador MayorQue |
    Numeral Include STRING_LITERAL |
    Numeral Include MenorQue Identificador MayorQue |
    Numeral Include MenorQue Iostream MayorQue |
    IMPORTACION Numeral Include MenorQue Identificador Punto Identificador MayorQue |
    IMPORTACION Numeral Include STRING_LITERAL |
    IMPORTACION Numeral Include MenorQue Identificador MayorQue |
    IMPORTACION Numeral Include MenorQue Iostream MayorQue
;
INICIO ::=
    IMPORTACION Using Namespace Std P_coma Int Main Parent_a Parent_c Llave_a SENTENCIA
    Llave_c |
```



TECNOLÓGICO  
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

```
IMPORTACION Int Main Parent_a Parent_c Llave_a SENTENCIA Llave_c |
Int Main Parent_a Parent_c Llave_a SENTENCIA Llave_c |
Main Parent_a Parent_c Llave_a SENTENCIA Llave_c
;
SENTENCIA ::=
SENTENCIA DECLARACION |
DECLARACION |
SENTENCIA IF |
IF |
SENTENCIA FOR |
FOR |
SENTENCIA WHILE |
WHILE |
SENTENCIA DO_WHILE |
DO_WHILE |
SENTENCIA OPERACIONES_COMPLEMENTARIAS P_coma |
OPERACIONES_COMPLEMENTARIAS P_coma |
SENTENCIA COUT |
COUT |
SENTENCIA CIN |
CIN |
SENTENCIA SWITCH |
SWITCH |
SENTENCIA PRINTF |
PRINTF |
SENTENCIA SCANF |
SCANF
;
DECLARACION_ANIDADA ::=
Identificador Igual Numero |
Identificador Igual Numero Coma DECLARACION_ANIDADA |
Identificador Igual STRING_LITERAL |
Identificador Igual STRING_LITERAL Coma DECLARACION_ANIDADA |
Identificador Coma DECLARACION_ANIDADA |
Identificador Igual Identificador |
Identificador Igual True Coma DECLARACION_ANIDADA |
Identificador Igual False Coma DECLARACION_ANIDADA |
Identificador Igual True |
Identificador Igual False |
Identificador
;
CREACION_ARREGLO ::=
Numero Coma CREACION_ARREGLO |
Numero |
STRING_LITERAL Coma CREACION_ARREGLO |
STRING_LITERAL
;
DECLARACION_ARREGLO_ANIDADA ::=
```



TECNOLÓGICO  
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

```
Int DECLARACION_ARREGLO P_coma |
_String DECLARACION_ARREGLO P_coma |
Char DECLARACION_ARREGLO P_coma
;
DECLARACION_ARREGLO::=
  Identificador Corchete_a Corchete_c Igual Llave_a CREACION_ARREGLO Llave_c Coma
DECLARACION_ARREGLO |
  Identificador Corchete_a Corchete_c Igual Llave_a CREACION_ARREGLO Llave_c |
  Identificador Corchete_a Numero Corchete_c Igual Llave_a CREACION_ARREGLO Llave_c Coma
DECLARACION_ARREGLO |
  Identificador Corchete_a Numero Corchete_c Igual Llave_a CREACION_ARREGLO Llave_c |
  Identificador Corchete_a Numero Corchete_c Coma DECLARACION_ARREGLO |
  Identificador Corchete_a Numero Corchete_c
;
DECLARACION ::=
  Int DECLARACION_ANIDADA P_coma |
  Unsigned Int DECLARACION_ANIDADA P_coma |
  Short Int DECLARACION_ANIDADA P_coma |
  Unsigned Short Int DECLARACION_ANIDADA P_coma |
  Long Int DECLARACION_ANIDADA P_coma |
  Unsigned Long Int DECLARACION_ANIDADA P_coma |
  Double DECLARACION_ANIDADA P_coma |
  Long Double DECLARACION_ANIDADA P_coma |
  Float DECLARACION_ANIDADA P_coma |
  Char DECLARACION_ANIDADA P_coma |
  Unsigned Char DECLARACION_ANIDADA P_coma |
  _String DECLARACION_ANIDADA P_coma |
  Bool DECLARACION_ANIDADA P_coma |
  DECLARACION_ARREGLO_ANIDADA
;
OPERADORES_ARITMETICOS::=
  Suma |
  Resta |
  Multiplicacion |
  Division |
  Modulo
;
OPERADORES_RELACIONALES::=
  BitOr |
  BitAnd |
  ComparadorIgual |
  MenorIgual |
  MayorIgual |
  Diferente |
  MenorQue |
  MayorQue
;
SENTENCIA_BOOLEANA_SIMPLE::=
```



TECNOLÓGICO  
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

```
True |
False |
IDENTIFICADORES |
IDENTIFICADORES OPERADORES_ARITMETICOS IDENTIFICADORES |
IDENTIFICADORES OPERADORES_ARITMETICOS Numero |
Numero OPERADORES_ARITMETICOS IDENTIFICADORES |
Numero OPERADORES_ARITMETICOS Numero |
Numero |
STRING_LITERAL |
Parent_a SENTENCIA_BOOLEANA_SIMPLE Parent_c
;
SENTENCIA_BOOLEANA_COMPUESTA::=
    Negador          SENTENCIA_BOOLEANA_SIMPLE          OPERADORES_RELACIONALES
SENTENCIA_BOOLEANA_SIMPLE |
    SENTENCIA_BOOLEANA_SIMPLE OPERADORES_RELACIONALES SENTENCIA_BOOLEANA_SIMPLE |
    SENTENCIA_BOOLEANA_SIMPLE          OPERADORES_RELACIONALES          Negador
SENTENCIA_BOOLEANA_SIMPLE |
    Negador          SENTENCIA_BOOLEANA_SIMPLE          OPERADORES_RELACIONALES          Negador
SENTENCIA_BOOLEANA_SIMPLE |
    Parent_a SENTENCIA_BOOLEANA_COMPUESTA Parent_c |
    Negador Parent_a SENTENCIA_BOOLEANA_COMPUESTA Parent_c |
    Negador SENTENCIA_BOOLEANA_SIMPLE
;
SENTENCIA_BOOLEANA_ANIDADA::=
    SENTENCIA_BOOLEANA_COMPUESTA |
    SENTENCIA_BOOLEANA_SIMPLE |
    SENTENCIA_BOOLEANA_COMPUESTA Y_logico SENTENCIA_BOOLEANA_ANIDADA |
    SENTENCIA_BOOLEANA_COMPUESTA O_logico SENTENCIA_BOOLEANA_ANIDADA |
    SENTENCIA_BOOLEANA_SIMPLE Y_logico SENTENCIA_BOOLEANA_ANIDADA |
    SENTENCIA_BOOLEANA_SIMPLE O_logico SENTENCIA_BOOLEANA_ANIDADA
;
IF ::=
    If Parent_a SENTENCIA_BOOLEANA_ANIDADA Parent_c Llave_a SENTENCIA Llave_c |
    If Parent_a SENTENCIA_BOOLEANA_ANIDADA Parent_c Llave_a SENTENCIA Llave_c Else Llave_a
SENTENCIA Llave_c |
    If Parent_a SENTENCIA_BOOLEANA_ANIDADA Parent_c Llave_a SENTENCIA Llave_c Else IF
;
FOR_PARTE_UNO::=
    Int IDENTIFICADORES Igual Numero |
    IDENTIFICADORES |
    IDENTIFICADORES Igual Numero |
    Int IDENTIFICADORES Igual IDENTIFICADORES
;
IDENTIFICADORES::=
    Identificador |
    Identificador Corchete_a Numero Corchete_c
    Identificador Corchete_a Identificador Corchete_c
;
```



TECNOLÓGICO  
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

OPERACIONES\_COMPLEMENTARIAS::=

Return Numero |  
Incremento IDENTIFICADORES |  
Decremento IDENTIFICADORES |  
IDENTIFICADORES Incremento |  
IDENTIFICADORES Decremento |  
IDENTIFICADORES MasIgual IDENTIFICADORES |  
IDENTIFICADORES MenosIgual IDENTIFICADORES |  
IDENTIFICADORES MultiplicacionIgual IDENTIFICADORES |  
IDENTIFICADORES DivisionIgual IDENTIFICADORES |  
IDENTIFICADORES ModuloIgual IDENTIFICADORES |  
IDENTIFICADORES Igual IDENTIFICADORES |  
IDENTIFICADORES Igual IDENTIFICADORES OPERADORES\_ARITMETICOS IDENTIFICADORES |  
IDENTIFICADORES MasIgual Numero |  
IDENTIFICADORES MenosIgual Numero |  
IDENTIFICADORES MultiplicacionIgual Numero |  
IDENTIFICADORES DivisionIgual Numero |  
IDENTIFICADORES ModuloIgual Numero |  
IDENTIFICADORES Igual Numero |  
IDENTIFICADORES Igual Numero OPERADORES\_ARITMETICOS IDENTIFICADORES |  
IDENTIFICADORES Igual IDENTIFICADORES OPERADORES\_ARITMETICOS Numero |  
IDENTIFICADORES Igual Numero OPERADORES\_ARITMETICOS Numero |  
\_System Parent\_a STRING\_LITERAL Parent\_c

;

FOR::=

For Parent\_a FOR\_PARTE\_UNO P\_coma SENTENCIA\_BOOLEANA\_ANIDADA P\_coma  
OPERACIONES\_COMPLEMENTARIAS Parent\_c Llave\_a SENTENCIA Llave\_c

;

WHILE::=

While Parent\_a SENTENCIA\_BOOLEANA\_ANIDADA Parent\_c Llave\_a SENTENCIA Llave\_c

;

DO\_WHILE::=

Do Llave\_a SENTENCIA Llave\_c While Parent\_a SENTENCIA\_BOOLEANA\_ANIDADA Parent\_c  
P\_coma

;

COUT::=

Cout DobleMenor SALIDA P\_coma

;

SALIDA::=

IDENTIFICADORES |  
STRING\_LITERAL |  
STRING\_LITERAL DobleMenor SALIDA |  
IDENTIFICADORES DobleMenor SALIDA |  
Endl

;

ENTRADA::=

IDENTIFICADORES |  
IDENTIFICADORES DobleMayor ENTRADA





TECNOLÓGICO  
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

```
;
CIN::=
  Cin DobleMayor ENTRADA P_coma
;
CASE::=
  Case Numero DosPuntos SENTENCIA Break P_coma |
  Case STRING_LITERAL DosPuntos SENTENCIA Break P_coma |
  Case IDENTIFICADORES DosPuntos SENTENCIA Break P_coma |
  CASE Case Numero DosPuntos SENTENCIA Break P_coma |
  CASE Case IDENTIFICADORES DosPuntos SENTENCIA Break P_coma |
  CASE Case STRING_LITERAL DosPuntos SENTENCIA Break P_coma
;
SWITCH::=
  Switch Parent_a IDENTIFICADORES Parent_c Llave_a CASE Default DosPuntos SENTENCIA Break
P_coma Llave_c |
  Switch Parent_a IDENTIFICADORES Parent_c Llave_a CASE Default DosPuntos SENTENCIA Llave_c |
  Switch Parent_a IDENTIFICADORES Parent_c Llave_a CASE Llave_c
;
PRINTF::=
  Printf Parent_a STRING_LITERAL Coma COMPLEMENTO Parent_c P_coma |
  Printf Parent_a STRING_LITERAL Parent_c P_coma
;
COMPLEMENTO::=
  Identificador |
  Identificador Coma COMPLEMENTO
;
SCANF::=
  Scanf Parent_a STRING_LITERAL Coma BitAnd Identificador Parent_c P_coma
;
Clase validaciones
package Analizador;
import java_cup.runtime.Symbol;
%%
%class LexicoCup
%type java_cup.runtime.Symbol
%cup
%full
%line
%char
L=[a-zA-Z_]+
D=[0-9]+
espacio=[\t\r\n]+
%{
  private Symbol symbol(int type, Object value){
    return new Symbol(type, yyline, yycolumn, value);
  }
  private Symbol symbol(int type){
    return new Symbol(type, yyline, yycolumn);
  }
}
```



TECNOLÓGICO  
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

```
}
%}
%%
/* Espacios en blanco */
{espacio} { /*Ignore*/ }
/* Palabra reservada break */
( break ) {return new Symbol(sym.Break, yychar, yyline, yytext());}
/* Palabra reservada system */
( system ) {return new Symbol(sym._System, yychar, yyline, yytext());}
/* Tipo de dato String */
( string ) {return new Symbol(sym._String, yychar, yyline, yytext());}
/* Printf */
( printf ) {return new Symbol(sym.Printf, yychar, yyline, yytext());}
/* Scanf */
( scanf ) {return new Symbol(sym.Scanf, yychar, yyline, yytext());}
/* Palabra reservada case */
( case ) {return new Symbol(sym.Case, yychar, yyline, yytext());}
/* Comentarios */
( "//"(.)* | "/"(.)* ) { /*Ignore*/ }
/* Comillas */
( "\"" ) {return new Symbol(sym.Comillas, yychar, yyline, yytext());}
/* Comilla simple */
( "'" ) {return new Symbol(sym.Comilla_simple, yychar, yyline, yytext());}
/* Cadena - texto entre comillas */
L?\"(\\.[^\\\"])*\" {return new Symbol(sym.STRING_LITERAL, yychar, yyline, yytext());}
/* Palabra reservada const */
( const ) {return new Symbol(sym.Const, yychar, yyline, yytext());}
/* Palabra reservada continue */
( continue ) {return new Symbol(sym.Continue, yychar, yyline, yytext());}
/* Corchete de apertura */
( "[" ) {return new Symbol(sym.Corchete_a, yychar, yyline, yytext());}
/* Corchete de cierre */
( "]" ) {return new Symbol(sym.Corchete_c, yychar, yyline, yytext());}
/* Palabra reservada default */
( default ) {return new Symbol(sym.Default, yychar, yyline, yytext());}
/* Palabra reservada define */
( define ) {return new Symbol(sym.Define, yychar, yyline, yytext());}
/* Salto, tabulacion y espacio */
( "\\n" | "\\t" ) { /*Ignore*/ }
/* Palabra reservada while */
( while ) {return new Symbol(sym.While, yychar, yyline, yytext());}
/* Operador Division */
( "/" ) {return new Symbol(sym.Division, yychar, yyline, yytext());}
/* Palabra reservada Do */
( do ) {return new Symbol(sym.Do, yychar, yyline, yytext());}
/* Tipo de dato bool */
( bool ) {return new Symbol(sym.Bool, yychar, yyline, yytext());}
/* Tipo de dato double */
```



TECNOLÓGICO  
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

```
( double ) {return new Symbol(sym.Double, yychar, yyline, yytext());}  
/* Palabra reservada Else */  
( else ) {return new Symbol(sym.Else, yychar, yyline, yytext());}  
/* Palabra reservada For */  
( for ) {return new Symbol(sym.For, yychar, yyline, yytext());}  
/* Palabra reservada If */  
( if ) {return new Symbol(sym.If, yychar, yyline, yytext());}  
/* Operador Mayor Igual */  
( ">=" ) {return new Symbol(sym.MayorIgual, yychar, yyline, yytext());}  
/* Operador Menor Igual */  
( "<=" ) {return new Symbol(sym.MenorIgual, yychar, yyline, yytext());}  
/* Operador Mas Igual */  
( "+=" ) {return new Symbol(sym.MasIgual, yychar, yyline, yytext());}  
/* Operador Menos Igual */  
( "-=" ) {return new Symbol(sym.MenosIgual, yychar, yyline, yytext());}  
/* Operador Multiplica Igual */  
( "*=" ) {return new Symbol(sym.MultiplicacionIgual, yychar, yyline, yytext());}  
/* Operador Division Igual */  
( "/=" ) {return new Symbol(sym.DivisionIgual, yychar, yyline, yytext());}  
/* Comparador Igual */  
( "==" ) {return new Symbol(sym.ComparadorIgual, yychar, yyline, yytext());}  
/* Operador Diferente*/  
( "!=" ) {return new Symbol(sym.Diferente, yychar, yyline, yytext());}  
/* Operador negador*/  
( "!" ) {return new Symbol(sym.Negador, yychar, yyline, yytext());}  
/* Operador Incremento */  
( "++" ) {return new Symbol(sym.Incremento, yychar, yyline, yytext());}  
/* Operador decremento */  
( "--" ) {return new Symbol(sym.Decremento, yychar, yyline, yytext());}  
/* Operador Igual */  
( "=" ) {return new Symbol(sym.Igual, yychar, yyline, yytext());}  
/* Relacional doble mayor*/  
( ">>" ) {return new Symbol(sym.DobleMayor, yychar, yyline, yytext());}  
/* Operador Modulo igual*/  
( "%=" ) {return new Symbol(sym.ModuloIgual, yychar, yyline, yytext());}  
/* Relacional doble menor*/  
( "<<" ) {return new Symbol(sym.DobleMenor, yychar, yyline, yytext());}  
/* Relacional Mayor que */  
( ">" ) {return new Symbol(sym.MayorQue, yychar, yyline, yytext());}  
/* Relacional Menor que */  
( "<" ) {return new Symbol(sym.MenorQue, yychar, yyline, yytext());}  
/* Signo dos puntos*/  
( ":" ) {return new Symbol(sym.DosPuntos, yychar, yyline, yytext());}  
/* operador y logico*/  
( "&&" ) {return new Symbol(sym.Y_logico, yychar, yyline, yytext());}  
/* operador bitand*/  
( "&" ) {return new Symbol(sym.BitAnd, yychar, yyline, yytext());}  
/* operador o logico*/
```



TECNOLÓGICO  
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

```
( "||" ) {return new Symbol(sym.O_logico, yychar, yyline, yytext());}  
/* operador bitor*/  
( "|" ) {return new Symbol(sym.BitOr, yychar, yyline, yytext());}  
/* Palabra reservada Int */  
( int ) {return new Symbol(sym.Int, yychar, yyline, yytext());}  
/* Palabra reservada Long */  
( long ) {return new Symbol(sym.Long, yychar, yyline, yytext());}  
/* Palabra reservada Short*/  
( short ) {return new Symbol(sym.Short, yychar, yyline, yytext());}  
/* Palabra reservada Float*/  
( float ) {return new Symbol(sym.Float, yychar, yyline, yytext());}  
/* Palabra reservada Byte*/  
( byte ) {return new Symbol(sym.Byte, yychar, yyline, yytext());}  
/* Palabra reservada Char*/  
( char ) {return new Symbol(sym.Char, yychar, yyline, yytext());}  
/* Marcador include */  
( "include" ) {return new Symbol(sym.Include, yychar, yyline, yytext());}  
/* Marcador namespace */  
( "namespace" ) {return new Symbol(sym.Namespace, yychar, yyline, yytext());}  
/* Marcador Std */  
( "std" ) {return new Symbol(sym.Std, yychar, yyline, yytext());}  
/* Marcador ostream */  
( "ostream" ) {return new Symbol(sym.ostream, yychar, yyline, yytext());}  
/* Marcador using*/  
( "using" ) {return new Symbol(sym.Using, yychar, yyline, yytext());}  
/* Marcador false */  
( "false" ) {return new Symbol(sym.False, yychar, yyline, yytext());}  
/* Marcador true */  
( "true" ) {return new Symbol(sym.True, yychar, yyline, yytext());}  
/* Marcador endl */  
( "endl" ) {return new Symbol(sym.Endl, yychar, yyline, yytext());}  
/* Llave de apertura */  
( "{" ) {return new Symbol(sym.Llave_a, yychar, yyline, yytext());}  
/* Llave de cierre */  
( "}" ) {return new Symbol(sym.Llave_c, yychar, yyline, yytext());}  
/* Marcador de inicio de algoritmo */  
( "main" ) {return new Symbol(sym.Main, yychar, yyline, yytext());}  
/* Marcador register*/  
( "register" ) {return new Symbol(sym.Register, yychar, yyline, yytext());}  
/* Salida por consola */  
( "cout" ) {return new Symbol(sym.Cout, yychar, yyline, yytext());}  
/* Marcador void*/  
( "void" ) {return new Symbol(sym.Void, yychar, yyline, yytext());}  
/* Marcador typedef*/  
( "typedef" ) {return new Symbol(sym.Typedef, yychar, yyline, yytext());}  
/* Marcador switch*/  
( "switch" ) {return new Symbol(sym.Switch, yychar, yyline, yytext());}  
/* Marcador unsigned*/
```



TECNOLÓGICO  
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



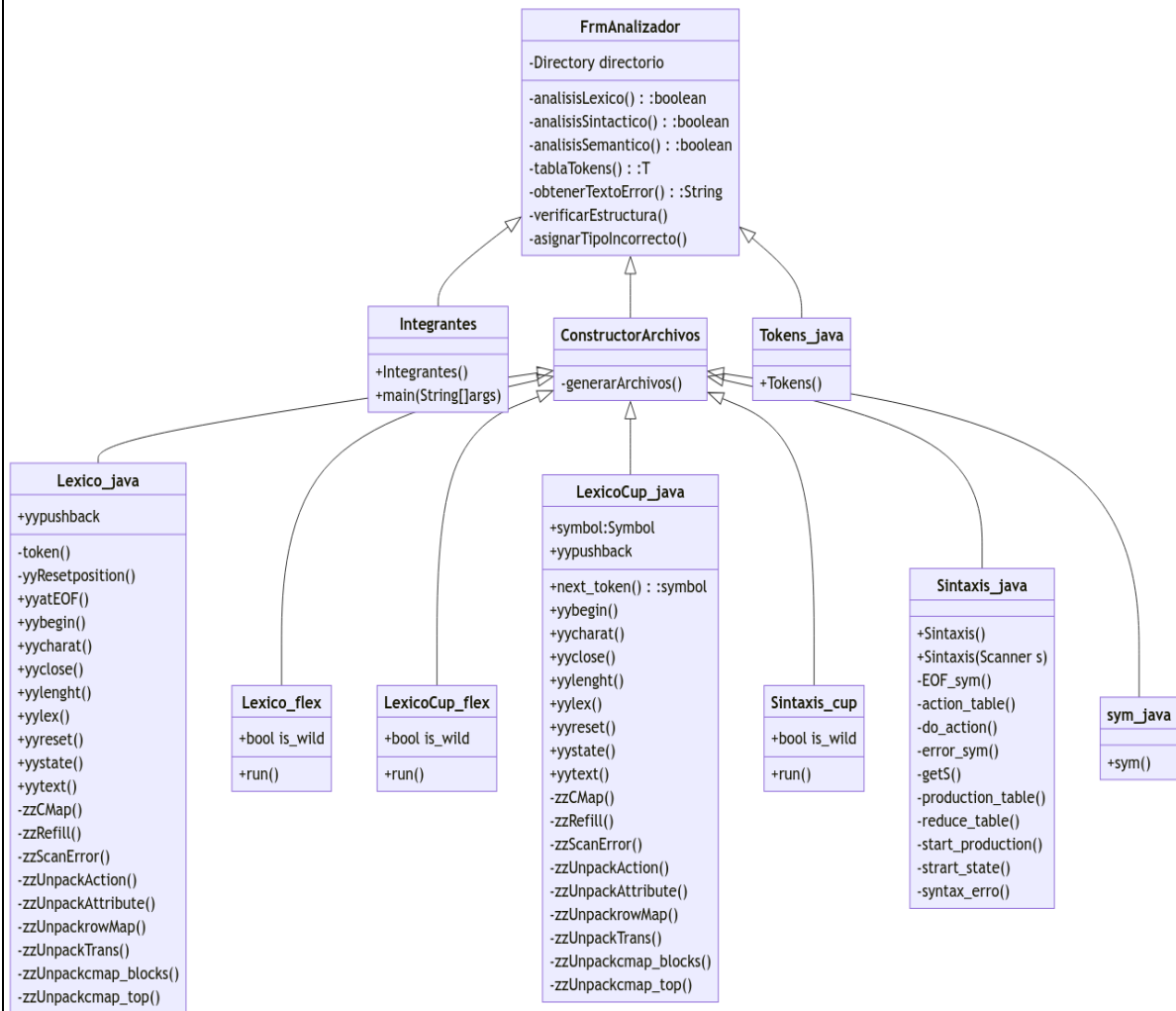
**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**



```
( "unsigned" ) {return new Symbol(sym.Unsigned, ychar, yyline, yytext());}
/* Entrada por consola*/
( "cin" ) {return new Symbol(sym.Cin, ychar, yyline, yytext());}
/* Operador Multiplicacion */
( "*" ) {return new Symbol(sym.Multiplicacion, ychar, yyline, yytext());}
/* Operador Modulo */
( "%" ) {return new Symbol(sym.Modulo, ychar, yyline, yytext());}
/* Numeral # */
( "#" ) {return new Symbol(sym.Numeral, ychar, yyline, yytext());}
/* Parentesis de apertura */
( "(" ) {return new Symbol(sym.Parent_a, ychar, yyline, yytext());}
/* Parentesis de cierre */
( ")" ) {return new Symbol(sym.Parent_c, ychar, yyline, yytext());}
/* Punto y coma */
( ";" ) {return new Symbol(sym.P_coma, ychar, yyline, yytext());}
/* Punto */
( "." ) {return new Symbol(sym.Punto, ychar, yyline, yytext());}
/* Coma */
( "," ) {return new Symbol(sym.Coma, ychar, yyline, yytext());}
/* Operador Resta */
( "-" ) {return new Symbol(sym.Resta, ychar, yyline, yytext());}
/* return */
( "return" ) {return new Symbol(sym.Return, ychar, yyline, yytext());}
/* Operador Suma */
( "+" ) {return new Symbol(sym.Suma, ychar, yyline, yytext());}
/* CADENA
{L}{L}{D}* {return new Symbol(sym.Identificador, ychar, yyline, yytext());}
*/
/* Identificador */
{L}{L}{D}* {return new Symbol(sym.Identificador, ychar, yyline, yytext());}
/* Numero */
{"-("{D}+)" ) | ("-{D}+.{D}+)" ) | -({D})+ | {D}+ | {D}+.{D}+ | -{D}+.{D}+ {return new
Symbol(sym.Numero, ychar, yyline, yytext());}
/* Error de analisis */
. {return new Symbol(sym.ERROR, ychar, yyline, yytext());}
```



Diagrama de clases





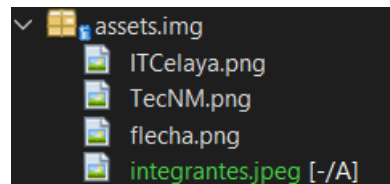
 <p>TECNOLÓGICO NACIONAL DE MÉXICO®</p>	<p><b>INSTITUTO TECNOLÓGICO DE CELAYA</b></p>	
<p><b>PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II</b></p>	<p><b>AUTOR: EQUIPO 4</b></p>	

## Manual técnico del funcionamiento del compilador

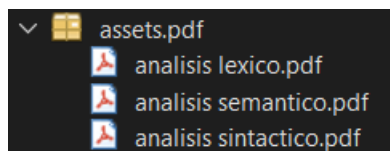
### 1.- Vista general del proyecto (Clases y Paquetes)

Se constan de diferentes paquetes donde se almacenan diferentes objetos, en este caso los paquetes son:

-Paquete de imágenes: En este paquete se encuentran los logos a la Universidad a la cual pertenecemos, además de una imagen extra de una flecha que nos servirá en nuestro compilador para poder regresar a nuestra pantalla principal, además de nuestra foto de integrantes quienes fueron los creadores de este proyecto.



-Paquetes con actividades pdf: En esta parte se encuentran tres diferentes actividades para que el usuario final pueda visualizar información acerca de cualquier tipo de análisis, en este caso el léxico, semántico y sintáctico.





### 2.- Vista principal de la interfaz

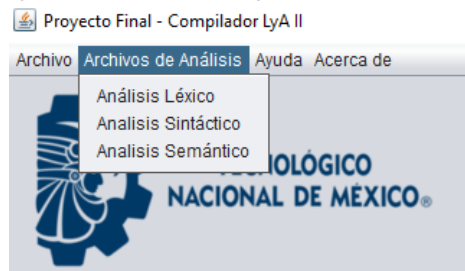
Como primera parte la interfaz se divide en diferentes secciones, cada sección tiene diferente propósito para el usuario final, estas secciones serian:

-Sección de archivo: Esta sección sirve para poder hacer un nuevo programa, abrir un programa que ya habíamos creado alguna vez e incluso si creamos algún programa nuevo poder guardarlo en nuestro dispositivo donde estemos programando.



 <p>TECNOLÓGICO NACIONAL DE MÉXICO®</p>	<p><b>INSTITUTO TECNOLÓGICO DE CELAYA</b></p>	
<p><b>PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II</b></p>	<p><b>AUTOR: EQUIPO 4</b></p>	

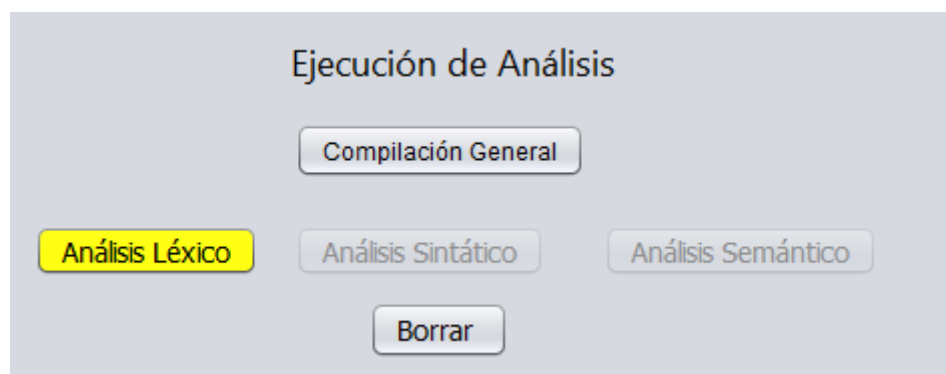
-Sección archivos de análisis: En esta sección es donde el usuario final podrá consultar los diferentes tipos de pdf de los diferentes tipos de análisis, como son léxico, semántico y sintáctico, con el fin de que pueda tener una guía de a que se refiere cada tipo de análisis.



-Sección de ayuda: En esta sección el usuario final podrá consultar un manual de cómo funciona el compilador y así mismo una documentación de cómo podría empezar a crear un programa con todas las variables que son requeridas, que palabras son reservadas y muchas más cosas que tienen que ver con la creación de este compilador.

-Sección acerca de: En esta parte te lleva a otra ventana donde podrá visualizar quienes fueron los creadores de este compilador y así mismo podrás regresar a la interfaz principal sin ningún problema.

-Sección ejecución de análisis: En esta sección del usuario podrá ir probando análisis por análisis una vez que realice algún programa en la entrada donde se empieza a compilar cualquier programa. Así mismo tiene un botón de Compilación General ya que este nos servirá para poder ejecutar los 3 tipos de Análisis sin necesidad de ir uno por uno. Los Análisis manejan los colores del semáforo, el color rojo nos indicara que sucedió algún error durante la compilación, el color amarillo nos dirá que está en espera de ser ejecutado y el color verde nos indicara que el análisis paso correctamente.



Como podemos observar cuenta con un botón de borrar ya que este nos servirá para limpiar donde escribimos nuestro código, la salida que nos arroja y limpiar nuestra tabla de tokens.

-Sección escribir código: En esta parte de la interfaz es donde el usuario final empezara a teclear el código que desea compilar.



TECNOLÓGICO  
NACIONAL DE MÉXICO®

**INSTITUTO TECNOLÓGICO DE CELAYA**



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**



**AUTOR: EQUIPO 4**

1

-Sección salida de código: En esta parte una vez que el usuario final escribió algún código o cargo algún código desde su dispositivo, cuando desee analizar algunos de los tipos de análisis en el cuadro de salida le aparecerá un mensaje de si el análisis paso el compilado correctamente.

Salida:

Sección tabla de tokens: En esta sección se ira analizando cada parte del código donde se le asignara un id, el token de esa palabra y la línea a la que pertenece esta palabra.

 <p>TECNOLÓGICO NACIONAL DE MÉXICO®</p>	<p><b>INSTITUTO TECNOLÓGICO DE CELAYA</b></p>	
<p><b>PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II</b></p>	<p><b>AUTOR: EQUIPO 4</b></p>	

### Generación de Token's

### Pruebas del compilador

#### Ejecutar la compilación general o el análisis sin código:

En esta parte si queremos hacer la ejecución general nos marcará un error en la salida de que no se encuentra algún código en la entrada de texto, esto será por parte de la compilación general:

Ejecución de Análisis

Compilación General

Análisis Léxico

Análisis Sintático

Análisis Semántico

Borrar

1

Salida:

Introduce Código, está vacío.

Generación de Token's

Ahora si queremos ejecutar directamente el análisis también nos mandara un error de que no se encuentra ningún código:



TECNOLÓGICO  
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

Ejecución de Análisis

Compilación General

Análisis Léxico    Análisis Sintático    Análisis Semántico

Borrar

Generación de Token's

1

Salida:

Introduce código, esta vacío.

Abrir y Guardar archivos

Le daremos un nombre a nuestro código y así mismo lo procedemos a guardar donde nosotros queramos, y se visualizará en nuestro ordenador.

Ejecución de Análisis

Compilación General

Análisis Léxico    Análisis Sintático    Análisis Semántico

Borrar

Generación de Token's

21    printf("\nSENTENCIA\n");  
22    for(int i = x; i < x+10; i++)  
23    printf("\n Iteracion: %d", i);  
24    ++i;  
25    }  
26    printf("\n");  
27    system("pause");  
28    return 0;  
29   }

Salida:

Guardar

Look In: Documents



Archivos Ejemplo Basico React  
Arduino  
CITTEC PROTEUS 8.13 + TODAS LAS LIBRERIAS  
Dell  
GitHub



File Name:

Files of Type: All Files

Guardar    Cancel

Visualizar el código guardado:

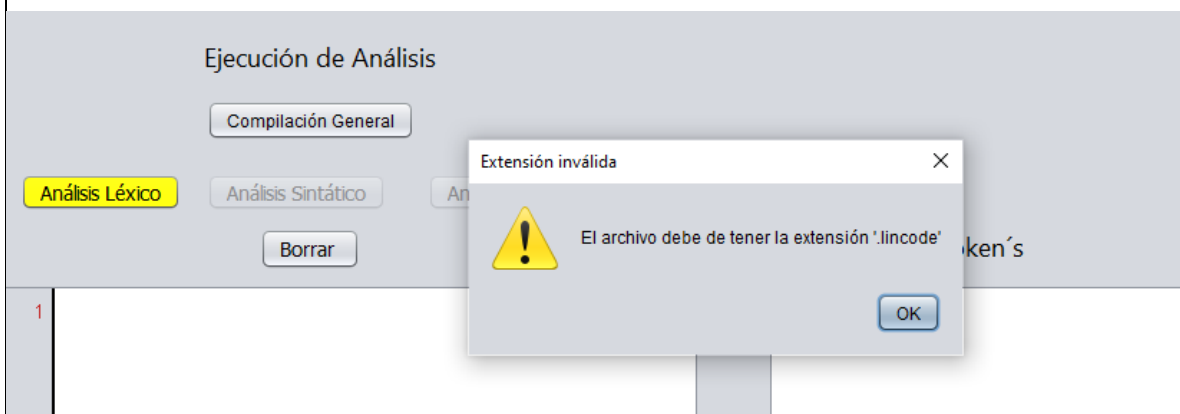
 <b>TECNOLÓGICO NACIONAL DE MÉXICO</b>	<b>INSTITUTO TECNOLÓGICO DE CELAYA</b>		
<b>PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II</b>		<b>AUTOR: EQUIPO 4</b>	

Nombre	Fecha de modificación	Tipo	Tamaño
 ProgramaFinal.lincode	30/11/2023 10:50 p. m.	Archivo LINCOTE	1 KB
 programa 3.lincode	30/11/2023 10:37 p. m.	Archivo LINCOTE	1 KB

Asi mismo podemos abrir estos códigos:



Asi mismo si deseamos abrir otra archivo que no tenga la extensión correspondiente, enviara un mensaje de error donde indique que este tipo de archivo no es válido:



### -Compilación general

En esta parte haremos diferentes pruebas dónde haremos que los análisis pasen todas las pruebas correctamente, además de que uno por uno estén marcando errores al quererlo compilar de una sola forma.



TECNOLÓGICO  
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II

AUTOR: EQUIPO 4

### 1.- Análisis completados correctamente

Ejecución de Análisis

Compilación General

Análisis Léxico    Análisis Sintático    Análisis Semántico

Borrar

```
21 printf("\nSENTENCIA FOR : \n");
22 for(int i = x; i < x+10;i++){
23     printf("\n Iteracion %d, numero: %d ",num,i);
24     ++num;
25 }
26 printf("\n");
27 system("pause");
28 return 0;
29 }
```

Salida:

Análisis Completado

Generación de Token's

67	<Reservado Printf>	printf
64	<Parentesis de apertura>	(
87	<Cadena>	"\n"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
75	<System>	system
64	<Parentesis de apertura>	(
87	<Cadena>	"pause"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
71	<Reservada return>	return
61	<Numero>	0
66	<Punto y coma>	;
46	<Llave de cierre>	}

Como podemos observar los 3 tipos de análisis de completaron de manera correcta respetando la tabla de tokens y así mismo mandando un mensaje de la salida de que se ejecuto exitosamente.

### 2.- Errores de análisis

Primeramente, ejecutaremos un error en el análisis léxico:

Ejecución de Análisis

Compilación General

Análisis Léxico    Análisis Sintático    Análisis Semántico

Borrar

```
21 printf("\nSENTENCIA FOR : \n");
22 for(int i = x; i < x+10;i++){
23     printf("\n Iteracion %d, numero: %d ",num,i);
24     ++num;
25 }
26 printf("\n");
27 system("pause");
28 return 0;
29 @
```

Salida:

Análisis con errores. Verifica el código.

Generación de Token's

64	<Parentesis de apertura>	(
87	<Cadena>	"\n"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
75	<System>	system
64	<Parentesis de apertura>	(
87	<Cadena>	"pause"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
71	<Reservada return>	return
61	<Numero>	0
66	<Punto y coma>	;
< ; >		
46	<Llave de cierre>	}

Como podemos observar al querer hacer la compilación nos marca un error de que existen errores en el código, esto se debe a que agregamos un símbolo que no está definido.

Segundo, ejecutaremos un error en el análisis léxico:



TECNOLÓGICO  
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

Ejecución de Análisis

Compilación General

Análisis Léxico Análisis Sintáctico Análisis Semántico

Borrar

```
21 printf("\nSENTENCIA FOR : \n");
22 for(int i = x; i < x+10;i++){
23     printf("\n Iteracion %d, numero: %d ",num,i);
24     ++num;
25 }
26 prin("\n");
27 system("pause");
28 return 0 ;
29 }
```

Salida:

Análisis con errores. Verifica el código.

Generación de Token's

37	<Identificador>	prin
64	<Parentesis de apertura>	(
87	<Cadena>	"\n"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
75	<System>	system
64	<Parentesis de apertura>	(
87	<Cadena>	"pause"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
71	<Reservada return>	return
61	<Numero>	0
66	<Punto y coma>	;
46	<Llave de cierre>	}

Pasa el análisis léxico, pero marca error en el sintáctico debido a que no está bien escrita la salida del mensaje.

#### -Ejecución de Análisis uno por uno

1.- Análisis Léxico: En esta primera parte haremos que el primer análisis en este caso el léxico marque un error, donde el botón cambiara de color amarillo a color rojo.

Ejecución de Análisis

Compilación General

Análisis Léxico Análisis Sintáctico Análisis Semántico

Borrar

```
21 printf("\nSENTENCIA FOR : \n");
22 for(int i = x; i < x+10;i++){
23     printf("\n Iteracion %d, numero: %d ",num,i);
24     ++num;
25 }
26 printf("\n");
27 system("pause");
28 return 0;
29 $
```

Salida:

Error 100 <Símbolo no definido>

Generación de Token's

64	<Parentesis de apertura>	(
87	<Cadena>	"\n"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
75	<System>	system
64	<Parentesis de apertura>	(
87	<Cadena>	"pause"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
71	<Reservada return>	return
61	<Numero>	0
66	<Punto y coma>	;
< ; >		
46	<Llave de cierre>	}

En esta parte podemos observar que agregamos un símbolo que no esta definido y de esta manera no nos habilita los otros botones ya que hasta que no pase el análisis no nos cambiara tanto de color y no nos habilitara el siguiente botón.

Ahora haremos que pase la prueba este análisis:





TECNOLÓGICO  
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

Ejecución de Análisis

Compilación General

Análisis Léxico Análisis Sintático Análisis Semántico

Borrar

```
21 printf("\nSENTENCIA FOR : \n");
22 for(int i = x; i < x+10;i++){
23     printf("\n Iteracion %d, numero: %d ",num,i);
24     ++num;
25 }
26 printf("\n");
27 system("pause");
28 return 0;
29 }
```

Salida:

Análisis Léxico realizado correctamente

Generación de Token's

67	<Reservado Printf>	printf
64	<Parentesis de apertura>	(
87	<Cadena>	"\n"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
75	<System>	system
64	<Parentesis de apertura>	(
87	<Cadena>	"pause"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
71	<Reservada return>	return
61	<Numero>	0
66	<Punto y coma>	;
46	<Llave de cierre>	}

Si bien podemos observar que se habilito el siguiente botón y esta en espera de si le haremos alguna modificación al código o simplemente haremos que corra de manera efectiva. En este caso haremos una prueba de error:

Ejecución de Análisis

Compilación General

Análisis Léxico Análisis Sintático Análisis Semántico

Borrar

```
21 printf("\nSENTENCIA FOR : \n");
22 for(int i = x; i < x+10;i++){
23     printf("\n Iteracion %d, numero: %d ",num,i);
24     ++num;
25 }
26 prin("\n");
27 system("pause");
28 return 0;
29 }
```

Salida:

Error de sintaxis. Línea: 26 Columna: 700, Texto: "("

Generación de Token's

67	<Reservado Printf>	printf
64	<Parentesis de apertura>	(
87	<Cadena>	"\n"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
75	<System>	system
64	<Parentesis de apertura>	(
87	<Cadena>	"pause"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
71	<Reservada return>	return
61	<Numero>	0
66	<Punto y coma>	;
46	<Llave de cierre>	}

Podemos observar que nos marcó un error de sintaxis debido a que no pusimos bien la salida del printf y por esas cuestiones maca error como si no estuviera completo la parte de los paréntesis, asi mismo nos deja el mensaje de en qué línea se encuentra el error y a que error pertenece, asi mismo no incluye esta palabra en la tabla de tokens.

Ahora procederemos a corregir este error para verificar que el análisis se realizó correctamente:



TECNOLÓGICO  
NACIONAL DE MÉXICO

## INSTITUTO TECNOLÓGICO DE CELAYA



### PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II

AUTOR: EQUIPO 4

Ejecución de Análisis

Compilación General

Análisis Léxico Análisis Sintático Análisis Semántico

Borrar

```
21 printf("\nSENTENCIA FOR : \n");
22 for(int i = x; i< x+10;i++){
23     printf("\n Iteracion %d, numero: %d ",num,i);
24     ++num;
25 }
26 printf("\n");
27 system("pause");
28 return 0;
29 }
```

Salida:

Analisis Sintático realizado correctamente

Generación de Token's

67	<Reservado Printf>	printf
64	<Parentesis de apertura>	(
87	<Cadena>	"\n"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
75	<System>	system
64	<Parentesis de apertura>	(
87	<Cadena>	"pause"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
71	<Reservada return>	return
61	<Numero>	0
66	<Punto y coma>	;
46	<Llave de cierre>	}

Así mismo podemos observar que el siguiente análisis está en espera de ser ejecutado. Procederemos a marcar un error para saber que el análisis está fallando:

Ejecución de Análisis

Compilación General

Análisis Léxico Análisis Sintático Análisis Semántico

Borrar

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main(){
5     string x = 10, y = 5;
6     printf("Ingrese un numero por favor : ");
7     scanf("%d", &x);
8     printf("Ingrese otro numero por favor: ");
9     scanf("%d", &y);
}
```



Salida:

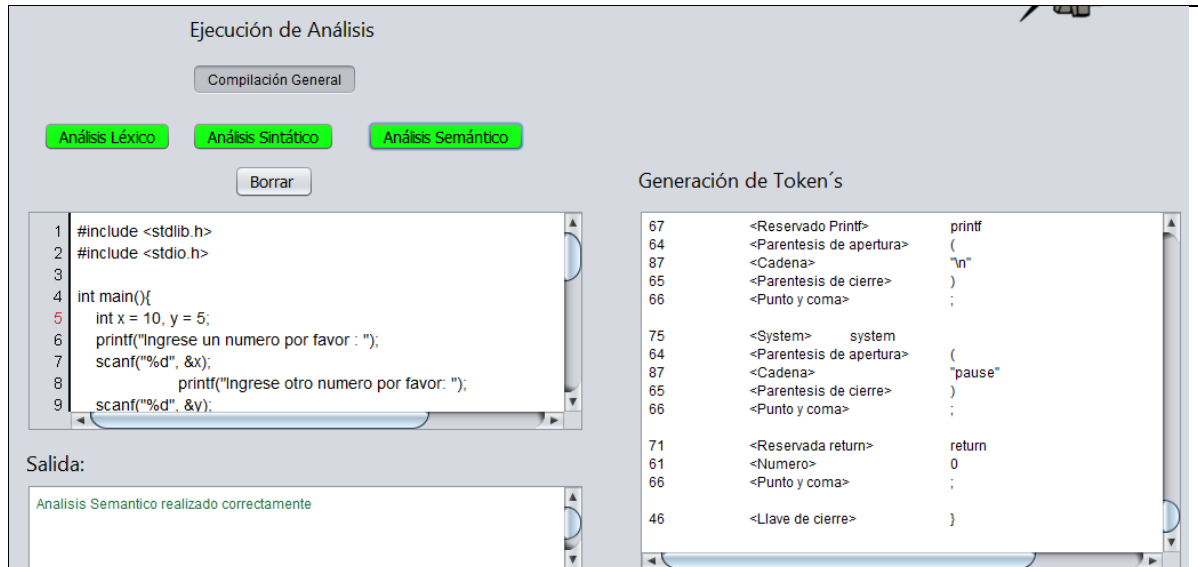
Error de asignación de tipo de dato

Generación de Token's

67	<Reservado Printf>	printf
64	<Parentesis de apertura>	(
87	<Cadena>	"\n"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
75	<System>	system
64	<Parentesis de apertura>	(
87	<Cadena>	"pause"
65	<Parentesis de cierre>	)
66	<Punto y coma>	;
71	<Reservada return>	return
61	<Numero>	0
66	<Punto y coma>	;
46	<Llave de cierre>	}



Definimos una variable de tipo String y le pasamos un entero, de esta forma nos manda un error y así mismo nos cambia de color el botón. No podemos meter otros tipos de datos que no corresponden. Ahora procederemos a que el análisis se ejecute correctamente:

 <b>TECNOLÓGICO NACIONAL DE MÉXICO</b>	<b>INSTITUTO TECNOLÓGICO DE CELAYA</b>		
<b>PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II</b>		<b>AUTOR: EQUIPO 4</b>	



Como podemos observar el análisis ha funcionado correctamente con el mensaje debido y así mismo los análisis funcionan correctamente.



6	Bitácora de incidencias		
Fecha	Problema encontrado	Solución	
07/11/2023	Manejador del entorno para la programación y la interfaz	Ver tutoriales	
14/11/2023	Error de la interfaz, ya que al clonarse se había realizado en un monitor de mayor tamaño	Empezar a modificarlo trabajando desde una laptop	
16/11/2023	Tokens para hacer commits en el GitHub	Crear nuevos tokens cada que hacemos commits en el repositorio	
21/11/2023	No arrojaba errores en los análisis	Visualizar diferentes videos de como trabajaban las librerías de los diferentes tipos de análisis.	

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p><b>INSTITUTO TECNOLÓGICO DE CELAYA</b></p>	
<p><b>PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II</b></p>	<p><b>AUTOR: EQUIPO 4</b></p>	

23/11/2023	No reconoció en el análisis semántico el error de sintaxis	Hacer más validaciones en ese tipo de análisis
28/11/2023	No compilaba de manera general los tres tipos de análisis	Ver que metodos estabamos mandando llamar al presionar el botón y vimos que nos faltaba mandar llamar uno de ellos
28/11/2023	El código de análisis semántico no funcionaba del todo bien	Se recurrió a mapear token por token para encontrar la solución e implementarla.

7	OBSERVACIONES
<p>A lo largo del proceso, nos encontramos en diversas situaciones que nos plantearon ciertos desafíos para comprender los distintos tipos de análisis. Aunque teníamos un conocimiento general sobre cómo funcionaban, experimentamos dificultades al aplicarlos en la programación debido a perspectivas divergentes. En un esfuerzo por superar estos obstáculos, decidimos complementar nuestra comprensión mediante la visualización de videos explicativos sobre el uso de las librerías pertinentes.</p> <p>La elección de emplear un repositorio y organizar reuniones a través de plataformas como Meet demostró ser beneficioso. Esta metodología facilitó una colaboración más estrecha entre los tres integrantes que estábamos desarrollando este compilador. La posibilidad de pensar de manera conjunta, discutir ideas y abordar desafíos de manera colectiva contribuyó significativamente a la culminación exitosa del proyecto.</p> <p>El intercambio continuo de conocimientos y la sinergia resultante de estas interacciones enriquecieron nuestra comprensión y habilidades, permitiéndonos superar las dificultades iniciales y lograr nuestros objetivos de manera más eficiente.</p> <p>En retrospectiva, la combinación de recursos visuales, herramientas colaborativas y reuniones periódicas resultó ser una estrategia efectiva para alcanzar el éxito en nuestro emprendimiento conjunto.</p>	

8	CONCLUSIÓN
<p>En conclusión, el proyecto "LINCODE: El Lenguaje de Programación con Espíritu de Lince" no solo se destaca por su singularidad al fusionar la agilidad del estudiante lince con la precisión del código, sino que también se fundamenta en principios sólidos de diseño que potencian la experiencia de programación. La adopción de una gramática formal y libre de contexto establece las bases para un código claro, consistente y compatible, características cruciales que contribuyen a elevar la calidad del software desarrollado y facilitan su adaptabilidad a diferentes contextos.</p> <p>La decisión estratégica de optar por un enfoque de compilación en lugar de interpretación refuerza aún más la robustez del lenguaje. Este enfoque no solo mejora el rendimiento de las aplicaciones escritas en LINCODE, sino que también aporta beneficios en términos de seguridad y la detección temprana de errores, factores esenciales para un desarrollo de software eficiente y confiable.</p> <p>El proyecto LINCODE no se limita simplemente a la definición de la sintaxis y semántica del lenguaje; también considera aspectos prácticos al contemplar una variedad de tipos de operadores y datos.</p>	

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p><b>INSTITUTO TECNOLÓGICO DE CELAYA</b></p>	
<p><b>PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II</b></p>	<p><b>AUTOR: EQUIPO 4</b></p>	

Además, la implementación de una interfaz de usuario intuitiva para interactuar con el compilador refleja el compromiso de facilitar el proceso de desarrollo y fomentar una mayor adopción por parte de la comunidad estudiantil.

En última instancia, LINCODE emerge como una propuesta integral que no solo busca ofrecer un nuevo lenguaje de programación, sino también establecer una conexión única con la comunidad estudiantil, proporcionando las herramientas necesarias para potenciar la creatividad y la eficiencia en el desarrollo de software.

9	REFERENCIAS
	<p><i>Crea tu propio compilador – Cap. 1 – Introducción.</i> (s/f). Blog de Tito Hinostroza. Recuperado el 1 de diciembre de 2023, de <a href="https://blogdetito.com/2019/01/05/crea-tu-propio-compilador-casero-parte-1/">https://blogdetito.com/2019/01/05/crea-tu-propio-compilador-casero-parte-1/</a></p> <p><i>El compilador de Java.</i> (s/f). Grupo Codesi. Recuperado el 1 de diciembre de 2023, de <a href="https://www.buscaminegocio.com/cursos-de-java/compilador-de-java.html">https://www.buscaminegocio.com/cursos-de-java/compilador-de-java.html</a></p> <p><i>De los compiladores e interpretes, E.</i> (s/f). <i>II26 Procesadores de lenguaje.</i> Uji.es. Recuperado el 1 de diciembre de 2023, de <a href="https://repositori.uji.es/xmlui/bitstream/handle/10234/5876/estructura.apun.pdf">https://repositori.uji.es/xmlui/bitstream/handle/10234/5876/estructura.apun.pdf</a></p> <p><i>De un Compilador, E. G.</i> (s/f). <i>Diseño de Compiladores I.</i> Edu.ar. Recuperado el 1 de diciembre de 2023, de <a href="https://users.exa.unicen.edu.ar/catedras/compila1/index_archivos/Introduccion.pdf">https://users.exa.unicen.edu.ar/catedras/compila1/index_archivos/Introduccion.pdf</a></p> <p><i>Luisa González Díaz, M.</i> (s/f). <i>Introducción a la construcción de compiladores.</i> Uva.es. Recuperado el 1 de diciembre de 2023, de <a href="https://www.infor.uva.es/~mluisa/talf/docs/aula/A7.pdf">https://www.infor.uva.es/~mluisa/talf/docs/aula/A7.pdf</a></p> <ul style="list-style-type: none"> <li>• <b>Video Explicativo del Proyecto:</b> <a href="https://youtu.be/OiGXl_yLKYQ">https://youtu.be/OiGXl_yLKYQ</a></li> <li>• <b>Código Fuente:</b></li> </ul>