Pestaña 2

Instituto Nacional de Educación Diversificada
INED
Santa Cruz Naranjo, Santa Rosa.

Catedrático: Gustavo Blanco
Cátedra: Producción
Ciclo escolar: 2026

Temas:
Teoría de Autómatas, Gramáticas formales, Análisis léxico, Análisis Sintáctico, Algoritmos de análisis de cadena, Compiladores e intérpretes, Aplicaciones prácticas de autónomas, Lenguaje de programación C.

Estudiante: Sharon Abigail García.
Grado: 5to Compu.
Sección: Única.

Santa Cruz Naranjo, Santa Rosa
Fecha:31/03/2025

**01**
**Teoría de autómatas.**

- **Definición y usos de las autónomas finitos determinadas (AFD)**

Un Autómata Finito Determinista (AFD) es un modelo matemático computacional que reconoce lenguajes regulares. Se define por tener un conjunto finito de estados y transiciones precisas para cada símbolo de entrada, lo que significa que para cualquier estado y símbolo dado, existe exactamente una transición posible a un siguiente estado.

- **Ejemplos y aplicaciones en informática y ciencias de la computación**

**Aplicaciones en Software y Desarrollo (Aplicaciones Informáticas)**
Son programas diseñados para realizar tareas específicas.

- Ofimática: Suites como Microsoft Office o LibreOffice, utilizadas para el procesamiento de textos, hojas de cálculo y presentaciones.
- Navegadores Web: Google Chrome, Mozilla Firefox, fundamentales para la gestión de información y navegación.
- Gestión de Contenidos (CMS): Sistemas como WordPress, utilizados para la creación y administración de sitios web.
- Aplicaciones Multimedia: Reproductores como VLC o herramientas de diseño como Adobe Photoshop.

**Ciencias de la Computación (Áreas Teóricas y Aplicadas)**
Se centran en el estudio de algoritmos, complejidad y hardware.

- Algoritmos y Estructuras de Datos: Desarrollo de algoritmos eficientes para la búsqueda y clasificación de información, fundamentales en motores de búsqueda como Google.
- Teoría de Autómatas: Modelado de sistemas complejos y diseño de compiladores.
- Computación Cuántica: Exploración de nuevos paradigmas de cómputo para resolver problemas no tratables actualmente.

**02**
**Gramáticas Formales.**

**Gramáticas regulares:** Conceptos y ejemplos de su uso en lenguajes de programación.

- **Concepto.**

Las gramáticas regulares son un tipo de gramática formal (Tipo 3 en la Jerarquía de Chomsky) que define lenguajes regulares a través de reglas de producción restringidas, donde la parte derecha tiene un máximo de un símbolo no terminal. Son equivalentes a los autómatas finitos y expresiones regulares, fundamentales para el análisis léxico en compiladores.

- **Ejemplos y Uso en Lenguajes de Programación.**

Se utilizan para definir la sintaxis básica, principalmente en el análisis léxico (identificación de componentes como tokens, números, palabras clave).

**Identificadores (Variables):**

- Regla: Empieza con letra, seguido de letras o números.

- Gramática: S– aA|bA|...|zA
  A– aA|...|zA|0A|...|9A|e

- Expresión Regular: [a-zA-Z][a-zA-Z0-9]*.

**Gramáticas recursivas: explicación y diferencias con las grámaticas regulares.**

Las gramáticas recursivas son aquellas que contienen reglas de producción donde un símbolo no terminal se deriva a sí mismo, directa o indirectamente (ej. A—Aa), permitiendo generar conjuntos infinitos de cadenas mediante estructuras repetitivas. A diferencia de las gramáticas regulares (tipo 3), las cuales son lineales y reconocidas por autómatas finitos, las recursivas suelen ser libres del contexto (tipo 2) y requieren autómatas de pila para procesar estructuras anidadas o jerárquicas

**03**
**Análisis léxico.**

- **Tokenización:** qué es y cómo se usa en el proceso de compilación.

**Qué es?** La tokenización es el proceso de sustituir los datos sensibles o los activos físicos por símbolos de identificación únicos que conservan toda la información esencial de los datos o los activos sin comprometer su seguridad.

**Como se utiliza?**
Cuando se realiza un pago online, los datos bancarios reales no viajan por la red: en su lugar, el sistema genera un token que los representa y lo utiliza para completar la transacción.

- **Expresiones regulares:** Su función en el análisis léxico:

Las expresiones regulares (regex) son patrones de caracteres fundamentales en el análisis léxico para identificar y clasificar componentes del código fuente (tokens). Actúan como moldes que describen lexemas válidos, convirtiéndose en autómatas finitos (DFA/NFA) para procesar texto rápidamente, siendo esenciales en herramientas como Flex.

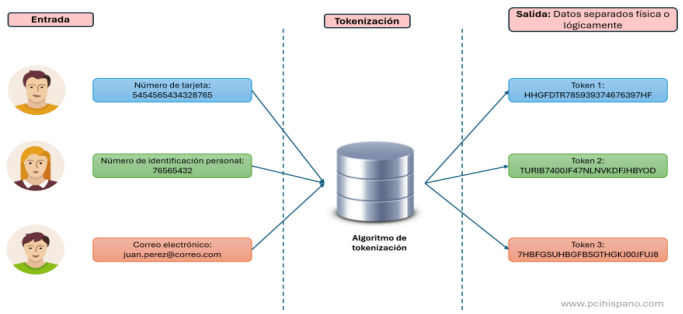
- **Reconocimiento de palabras claves:** cómo se indentifican en las lenguas de programación.

Las palabras clave en los lenguajes de programación son términos reservados con significados especiales y predefinidos (ej. if, for, class) que el compilador o intérprete identifica automáticamente al analizar la sintaxis del código. Estas no pueden usarse como nombres de variables o funciones y se distinguen por su ortografía exacta.

- **Identificación de tokens: clasificación y ejemplos prácticos.**

**Clasificación:** La clasificación de tokens es el proceso de asignar etiquetas específicas (como personas, organizaciones o lugares) a elementos individuales dentro de una secuencia de texto. Se utiliza en el procesamiento de lenguaje natural (PLN) para tareas como análisis de sentimientos, extracción de información y clasificación de documentos.

- **Ejemplo práctico:**



**04**
# Análisis Sintáctico.

**Tipos de análisis sintáctico:**

- **Análisis sintáctico descendente.**

El análisis sintáctico descendente (top-down) construye el árbol de derivación desde la raíz (símbolo inicial) hacia las hojas (tokens de entrada), siguiendo una derivación más a la izquierda. Intenta predecir la estructura de la gramática, descomponiendo no terminales en componentes más pequeños.

- **Análisis sintáctico ascendente.**

El análisis sintáctico ascendente (bottom-up parsing) es una técnica de compilación que construye el árbol de análisis desde las hojas (tokens de entrada) hacia la raíz (símbolo inicial),. Utiliza principalmente el método desplazamiento-reducción (shift-reduce), donde los tokens se desplazan a una pila hasta formar un mango (handle) que se reduce a un no terminal.

- **Árboles de análisis sintáctico:** qué son y cómo ayudan en la estructura de los programas.

**¿Qué son?.** Un árbol de análisis sintáctico es un árbol ordenado y enraizado que representa la estructura sintáctica de una cadena según una gramática libre de contexto, donde los nodos internos corresponden a símbolos no terminales y los nodos hoja corresponden a símbolos terminales de la gramática.

**Cómo ayudan en la estructura de los programas?.**
Ayudan descomponiendo los componentes de la cadena en unidades más pequeñas, mostrando los detalles de la estructura gramatical empleada para su creación.

**05**
## Algoritmos de análisis de cadena

- **Algoritmo de Boyer-Moore**: concepto, funcionamiento y su aplicación en la búsqueda de cadenas dentro del texto o programas.

**Concepto:** Es un método eficiente para buscar subcadenas (patrón) dentro de un texto, caracterizado por comparar de derecha a izquierda y utilizar heurísticas de preprocesamiento para dar saltos grandes, omitiendo caracteres innecesarios. Es más rápido cuanto más largo es el patrón.

**Funcionamiento:**

- **Alineación inicial:** El patrón se alinea con el principio del texto.
- **Comparación de derecha a izquierda**: Se compara el último carácter del patrón con el carácter correspondiente en el texto, moviéndose hacia atrás (de derecha a izquierda).
- **Desplazamiento (Shift):** Si ocurre una falta de coincidencia (mismatch), el algoritmo calcula el mayor salto posible utilizando dos reglas:
- **Regla del Carácter Incorrecto:** Si el carácter del texto no coincide con el del patrón, el algoritmo busca la última aparición de ese carácter en el patrón. Si existe, alinea ese carácter; si no, desplaza el patrón completamente pasando el carácter.
- **Regla del Buen Sufijo:** Si parte del patrón ya coincidió antes de la falta de coincidencia, el algoritmo alinea el patrón con otra ocurrencia de ese sufijo dentro del patrón.
- **Finalización:** Se repite el proceso hasta encontrar el patrón o superar la longitud del texto.

**Aplicación en la búsqueda de cadenas dentro del texto o programas**

- **Editores de texto y navegadores:** Función "buscar" (Ctrl+F) en editores de texto (como Notepad + +) o navegadores web.
- **Bioinformática:** Búsqueda de secuencias genéticas (ADN) donde las cadenas son largas.
- **Seguridad de red:** Identificación de firmas de código malicioso o patrones sospechosos en el tráfico de red (IDS/IPS).
- **Detección de plagio:** Comparación de documentos para encontrar similitudes exactas.

## 06
## Compiladores e intérpretes

- **Estructura de un compilador:** definición y componentes.

1. **Definición:**Es un programa especializado que toma como entrada un código fuente escrito en un lenguaje de alto nivel (como C++, Java, Python) y lo transforma en un programa objetivo (código máquina o bytecode) ejecutable por la computadora. Durante este proceso, verifica la sintaxis y semántica del lenguaje.
2. **Componentes:**

- **Análisis Léxico (Escáner):** Lee el código fuente caracter por caracter y lo agrupa en unidades con significado llamadas tokens (palabras clave, identificadores, operadores).
- **Análisis Sintáctico (Analizador):** Agrupa los tokens en estructuras jerárquicas (árboles sintácticos) para comprobar si siguen las reglas gramaticales del lenguaje.
- **Análisis Semántico:** Verifica la consistencia del código, asegurando que las variables tengan sentido (chequeo de tipos, compatibilidad de operadores).
- **Generación de Código Intermedio:** Crea una representación de bajo nivel, independiente de la máquina, que facilita la optimización y traducción a diferentes arquitecturas.
- **Optimización de Código:** Mejora el código intermedio para que sea más eficiente (más rápido o más pequeño) sin cambiar su funcionalidad.
- **Generación de Código (Back-end):** Traduce el código intermedio optimizado al lenguaje de máquina o ensamblador específico de la CPU destino.

- **Fases principales del compilador:**
  **Análisis Léxico (Scanner):** Lee el código fuente como una secuencia de caracteres y lo agrupa en componentes significativos llamados tokens (palabras clave, identificadores, operadores).
  **Análisis Sintáctico (Parser):** Agrupa los tokens en estructuras jerárquicas (árboles de sintaxis) para verificar si la secuencia sigue las reglas gramaticales del lenguaje.
  **Análisis Semántico:** Comprueba la coherencia lógica, como la compatibilidad de tipos de datos, declaración de variables y alcance, asegurando que el programa tenga sentido.
  **Generación de Código Intermedio:** Crea una representación del código de bajo nivel, independiente de la máquina, que facilita la optimización y la portabilidad.
  **Optimización de Código:** Mejora el código intermedio para que sea más eficiente (rápido o compacto), eliminando cálculos innecesarios o código inalcanzable.

**Generación de Código Final:** Traduce el código intermedio optimizado al lenguaje de máquina o ensamblador específico de la arquitectura destino.

- **Lenguaje de programación utilizados en la creación de compiladores:** como **Lex y Yacc**.
  Lex y Yacc son herramientas clásicas utilizadas para crear compiladores, principalmente generando código fuente en lenguaje C. Lex crea analizadores léxicos (escáneres) a partir de expresiones regulares, mientras que Yacc genera analizadores sintácticos (parsers) basados en gramáticas, produciendo archivos .c (como y.tab.c) que se compilan para formar el compilador.

  - **Lenguajes Principales:** Lex y Yacc están diseñados principalmente para producir código fuente en C.
  - **Funcionamiento con C:** El analizador léxico (yylex) y el sintáctico (yyparse) generados se combinan con código C personalizado para el análisis funcional.
  - **Alternativas Modernas:** Aunque Lex/Yacc usan C, existen versiones para otros lenguajes como JFlex/Cup para Java.
  - **Proceso:** El código fuente se define en archivos .l (Lex) y y. (Yacc), que se transforman en archivos C.

**07**

## Aplicaciones prácticas de autónomas

- **Reconocimiento de patrones:** en diferentes áreas.

Es una rama de la inteligencia artificial y el aprendizaje automático que identifica regularidades, tendencias y estructuras en datos (imágenes, voz, texto, números) mediante métodos supervisados o no supervisados. Permite a las máquinas clasificar información, tomar decisiones y anticipar necesidades.

**Seguridad y Biometría**: Reconocimiento facial, huellas dactilares, análisis de comportamiento de movimiento y detección de intrusos.

**Medicina y Salud:** Diagnóstico por imágenes (radiografías, resonancias), detección de células infectadas, análisis genético y monitoreo de constantes vitales.

**Interacción Hombre-Máquina:** Reconocimiento de voz (asistentes virtuales), OCR (reconocimiento óptico de caracteres), procesamiento de lenguaje natural y traducción automática.

**Visión Artificial e Industria:** Reconocimiento de objetos, control de calidad en líneas de producción, conducción autónoma y robótica.

**Finanzas y Datos:** Detección de fraudes bancarios, predicción de tendencias en el mercado de valores y análisis de comportamiento de compra. Ciencia y Medio **Ambiente:** Análisis de patrones meteorológicos, astronomía (identificación de astros) y clasificación de especies.

- **Procesamiento del lenguaje natural** y su uso en la inteligencia artificial.

El Procesamiento de Lenguaje Natural (PLN o NLP) es una rama de la inteligencia artificial (IA) que permite a las computadoras entender, interpretar y generar lenguaje humano (texto o voz). Combina lingüística computacional con aprendizaje automático (machine learning) para analizar contextos, sentimientos e intenciones, facilitando la interacción hombre-máquina.

**<u>Uso en la Inteligencia Artificial</u>:**

**Asistentes Virtuales:** Siri, Alexa y chatbots entienden y responden comandos de voz y texto.

**Traducción automática:** Herramientas como Google Translate utilizan el PLN para convertir idiomas con precisión.

**Análisis de Sentimiento:** Evalúa opiniones en redes sociales o reseñas de productos para determinar si son positivas o negativas.

**Generación de Texto:** Modelos de IA crean contenido, resúmenes automáticos y corrigen ortografía.

**Detección de Spam:** Filtra correos electrónicos no deseados analizando el contenido del mensaje.

**08**

**Lenguaje de programación C**

- **Librerías fundamentales: STDIO y CONIO**, un resumen de las funciones de cada una.

**STDIO:** (Entrada/Salida Estándar).
Es la biblioteca base para interactuar con el usuario a través de consola o archivos.
**printf():** Muestra datos formateados en pantalla.
**scanf():** Lee datos formateados desde el teclado.
**getchar() / putchar():** Lee/escribe un solo carácter.
**gets() / puts():** Lee/escribe cadenas de caracteres.
**fopen() / fclose() / fprintf():** Funciones para crear, abrir, cerrar y escribir en archivos (manejo de archivos).

**CONIO:** (Entrada/Salida de consola - Turbo C)
Diseñada para compiladores antiguos de 16 bits (MS-DOS) para manipular la interfaz de usuario.
**clrscr():** Borra la pantalla y coloca el cursor en la esquina superior izquierda.
**getch():** Lee un carácter directamente del teclado sin esperar Enter y sin mostrarlo en pantalla (usado para pausar).
**getche():** Igual que getch(), pero muestra el carácter presionado.
**gotoxy(x, y):** Mueve el cursor a una posición específica de la pantalla.
**kbhit():** Verifica si se ha presionado alguna tecla sin detener la ejecución del programa.

**Lista de 10 librerías más comunes en C**, además de las ya mencionadas.
**1.stdlib.h:** Funciones de utilidad general (gestión de memoria dinámica malloc/free, conversión de tipos, números aleatorios rand, control de procesos exit).
**2.string.h:** Manipulación de cadenas de caracteres y arreglos (strcpy, strcat, strlen, strcmp, memcpy).
**3.math.h:** Funciones matemáticas comunes (sin, cos, sqrt, pow, log).
**4.ctype.h:** Clasificación y transformación de caracteres (isalpha, isdigit, tolower, toupper).
**5.time.h:** Manipulación de fecha y hora (time, ctime, difftime, clock).
**6.stdbool.h:** Define el tipo de dato booleano (bool, true, false) (introducido en C99).
**7.stdint.h:** Define tipos de enteros con anchos precisos (int8_t, uint32_t, int64_t) (C99).
**8.limits.h:** Define constantes con los límites de los tipos enteros (INT_MAX, CHAR_BIT).
**9.float.h:** Define constantes con los límites de los tipos de coma flotante (FLT_MAX, DBL_MIN).
**10.stddef.h:** Define tipos y macros estándar útiles, como size_t, NULL, y offsetof.

- **Manejadores de formatos para diferentes tipos de datos**, con ejemplos de uso de código.

**JSON (JavaScript Object Notation) - El estándar web**

**Utilizado para APIs y configuraciones, estructurado como pares clave-valor.**

```
import json

# Datos a JSON (Serialización)
datos = {"nombre": "Ana", "edad": 28, "activo": True}
json_string = json.dumps(datos) # Convierte dict a JSON
print(json string)

# JSON a Datos (Deserialización)
json input = '{"nombre": "Ana", "edad": 28}'
objeto = json.loads(json input) # Convertir JSON a dict
print(objeto["nombre"])
```

**CSV (Comma Separated Values) - Datos tabulares**

**Ideal para hojas de cálculo y bases de datos.**

```
import csv

# Escribir CSV
with open('usuarios.csv', 'w', newline='') as file:
  writer = csv.writer(file)
  writer.writerow(["Nombre", "Edad"])
writer.writerow(["Ana", 28])

# Leer CSV
with open('usuarios.csv', 'r') as file:
 reader = csv.reader(file)
for row in reader:
 print(row)
```

ID: 1808

# Investigación 01 Sharon García

## 1) Teoría de autómatas.

Un Autómata Finito Determinista (AFD) es un modelo matemático computacional que reconoce lenguaje regulares.

- Ejemplos y aplicaciones en informática y ciencias de la computación.

Ejemplo:
- Un usuario quiere guardar un documento de word en una USB.

Aplicaciones en informática.
Gestión de archivos: El sistema operativo utiliza formatos de almacenamiento para organizar y acceder a archivos en discos duros, SSD, USB, etc.

Ciencias de la Computación:
Se centran en el estudio de algoritmos, complejidad y hardware.

- Algoritmos y Estructuras de datos: Desarrollo de algoritmos eficientes para la búsqueda y clasificación.

- Teoría de Autómatas: Modelo de sistemas complejos y diseño de compiladores.

Sharon Abigail Garcia ID: 1808

## 2) Gramáticas formales.

**Gramáticas regulares: Conceptos y ejemplos de su uso en lenguajes de programación**

**Concepto:** Las gramáticas regulares son un tipo de gramática formal (Tipo 3 en la Jerarquía de Chomsky) que define lenguajes regulares.

**Ejemplos de su uso en lenguajes de programación.**

Se utilizan para definir la sintaxis básica, principalmente en el análisis léxico (identificación de componentes como tokens, números, palabras clave).

**Gramáticas recursivas: Explicación y diferencias con las gramáticas regulares.**

Las gramáticas recursivas son aquellas que contienen reglas de producción donde un símbolo no terminal se deriva a sí mismo, directa o indirectamente (ej. A -> Aa). A diferencia de las gramáticas regulares (tipo 3), las cuales son lineales y reconocidas por autómatas finitos.

## 3) Análisis léxico

- **Tokenización: Qué es y como se usa en el proceso de compilación.**

**Qué es:** Es el proceso de sustituir los datos sensibles o los activos físicos por símbolos de identificación únicos que conservan toda la información.

Sharon Abigail Garcia ID: 1008

¿Como se utiliza? Cuando se realiza un pago online, los datos bancarios reales no viajan por la red.

- Expresiones regulares: Su función en el analisis léxico:

Las expresiones regulares (regex) son patrones de caracteres fundamentales en el análisis léxico para identificar y clasificar componentes del código fuente.

- Reconocimiento de palabras claves. Como se identifican en las lenguas de programación. Las palabras son terminos reservados con significados especiales y predefinidos (ej. if, for, class) que el compilador o intérprete identifica.

- Identificación de tokens: clasificación y ejemplo práctico.

Clasificación: La clasificación de tokens es el proceso de asignar etiquetas específicas (como personas, organizaciones o lugares) a elementos individuales dentro de una secuencia de texto.

4) Análisis sintáctico.

- Análisis sintáctico descendente: El análisis sintáctico descendente (top-down) construye el árbol de derivación desde la raíz (símbolo inicial) hacia las hojas (tokens de entrada), siguiendo una derivación más a la izquierda.

Sharon Abigail García ID: 1308

- Análisis sintáctico ascendente.

El análisis sintáctico ascedente (bottom-up parsing) es una técnica de compilación que construye el árbol de análisis desde la raíz (símbolo inicial), utiliza principalmente el método desplazamiento-reducción.

- Árboles de análisis sintáctico

¿Qué son? Es un árbol ordenado y enraizado que representa la estructura sintáctica de una cadena según una gramática libre de contexto.

Cómo ayudan a la estructura de los programas?
Ayudar descomponiendo los componentes de la cadena en unidades más pequeñas, mostrando los detalles de la estructura.

5) Algoritmos de análisis de cadena.

- Algoritmo de Boyer - Moore.

Concepto: Es un método eficiente para buscar subcadenas (patrón) dentro de un texto, caracterizado por comparar de derecha a izquierda y utilizar heurísticas de procesamiento para dar saltos grandes.

Funcionamiento:

- Alineación inicial: El patrón se alinea con el principio del texto.
- Comparación de derecha a izquierda: Se compara el último carácter del patrón con el carácter correspondiente en el texto moviendose hacia atrás.
- Finalización: Se repite el proceso hasta encontrar el patron o superar la longitud del texto.

Sharon Abigail Garcia ID: 1808

Aplicación en la búsqueda de cadenas dentro del texto o programas.

- Editores de texto y navegadores: función "buscar" (Ctrl+F) en editores de texto (como Notepad++ o navegadores web.
- Bioinformática: Búsqueda de secuencias genéticas (ADN) donde las cadenas son largas.
- Detección de plagio: Comparación de documentos para encontrar similitudes exactas.

## 6) Compiladores e intérpretes.

- Estructura de un compilador:

1. Definición: Es un programa especializado que toma como entrada un código fuente escrito en un lenguaje de alto nivel (como C++, Java, Python) y lo transforma en un programa objetivo (código máquina o bytecode).

2. Componentes:
- Análisis léxico (Escáner): Lee el código fuente caracter por caracter y lo agrupa.
- Análisis Sintactico (Analizador): Agrupa los tokens en estructuras jerárquicas (árboles sintácticos).
- Análisis semántico: Verifica la consistencia del código asegurando que las variables tengan sentido (chequeo de tipos).
- Generación de código final: Traduce el código intermedio optimizado al lenguaje de máquina o ensamblador.

Sharon Abigail García ID: 1808

- Lenguaje de programación utilizados en la creación de compiladores: como lex y yacc.

lex y Yacc son herramientas clásicas utilizadas para crear compiladores, principalmente generando código fuente en lenguaje C. lex crea analizadores léxicos (escáneres) a partir de expresiones regulares. Mientras yacc genera analizadores sintácticos (parsers) basados en gramáticas.

7) Aplicaciones prácticas de autónomas.

- Reconocimiento de patrones: Es una rama de la inteligencia artificial y el aprendizaje automático que identifica regularidades, tendencias y estructuras en datos.

Seguridad y Biometría: Reconocimiento facial, huellas dactilares, análisis de comportamiento.

Medicina y salud: Diagnóstico por imágenes (radiografías, resonancias), detección de células infectadas, análisis genético y monitoreo.

- Procesamiento del lenguaje natural.

El procesamiento de lenguaje natural (PLN o NLP), es una rama de la inteligencia (IA) que permite a las computadoras entender, interpretar y generar lenguaje humano.

Uso en la inteligencia Artificial:

Asistente virtuales: Siri, Alexa y chatbots entienden y responden comandos de voz y texto.

Traducción automática: Herramientas como Google translate utilizan el PLN para convertir idiomas con precisión.