**1. Teoría de autómatas**

* **Definición y usos de los autómatas finitos deterministas (AFD)**.
* Un autómata finito determinista (AFD) es una máquina que reconoce o traduce cadenas de caracteres. Se compone de un conjunto finito de estados, símbolos de entrada, una función de transición, un estado inicial y un conjunto de estados finales.



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

Las ciencias de la computación y la informática tienen aplicaciones en muchos ámbitos, como la salud, la investigación científica, la educación, los negocios, y más.

Aplicaciones de la ciencia de la computación

* Desarrollo de inteligencia artificial
* Manejo de información a través de bases de datos
* Comunicación mediante redes de computadoras
* Modelado, simulación, visualización y exploración de problemas complejos en disciplinas científicas como la química, la física, la genética y la astronomía

Aplicaciones de la informática

* Diagnósticos asistidos por inteligencia artificial
* Telemedicina
* Automatización de tareas
* Control de procesos
* Almacenamiento de información
* Gestión de negocios
* Educación
* Medios de transportes
* Ingeniería

Ejemplos de aplicaciones de la informática en la ciencia

* Analizar datos
* Simplificar expresiones
* Controlar experimentos
* Identificar moléculas
* Medir áreas de figuras específicas
* Llevar información estadística de procesos

Aplicaciones de los AFD

* **Análisis de texto**: Se utilizan para buscar y reemplazar patrones en archivos de texto.

 **Análisis de protocolos**: Se utilizan para analizar protocolos.

 **Análisis de seguridad**: Se utilizan para analizar la seguridad.

 **Reconocimiento de voz**: Se utilizan para reconocer la voz.

 **Procesamiento de lenguaje natural**: Se utilizan para procesar el lenguaje natural.

 **Unidades de control de CPU**: Se utilizan en las unidades de control de CPU.

 **Comportamiento de personajes de videojuegos**: Se utilizan para modelar el comportamiento de los personajes de videojuegos.

**2. Gramáticas formales**

* **Gramáticas regulares**: concepto y ejemplos de su uso en lenguajes de programación.
* una **gramática regular** es una [gramática formal](https://es.wikipedia.org/wiki/Gram%C3%A1tica_formal) (V, Σ, R, S) que puede ser clasificada como regular izquierda o regular derecha. Las gramáticas regulares sólo pueden generar a los [lenguajes regulares](https://es.wikipedia.org/wiki/Lenguaje_regular) de manera similar a los autómatas finitos y las expresiones regulares.
* Dos gramáticas regulares que generan el mismo lenguaje regular se denominan equivalentes. Toda gramática regular es una [gramática libre de contexto](https://es.wikipedia.org/wiki/Gram%C3%A1tica_libre_de_contexto).
* **Gramáticas recursivas**: explicación y diferencias con las gramáticas regulares.

Una gramática recursiva es aquella que contiene reglas de producción que se repiten, lo que puede llevar a una cadena que incluya el mismo no terminal de nuevo.

**3. Análisis léxico**

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

La tokenización es el **proceso de intercambiar datos confidenciales por datos no confidenciales llamados “tokens” que pueden usarse en una base de datos o un sistema interno sin ponerlo dentro del alcance**.

* **Expresiones regulares**: su función en el análisis léxico.

En el análisis léxico, se usan para identificar y extraer tokens del código, como palabras clave o identificadores.

¿Qué son las expresiones regulares?

* Son una manera flexible de buscar o reconocer cadenas de texto.
* Son un lenguaje que se puede usar para describir otros lenguajes.
* Son una herramienta útil para cualquier desarrollador.

¿Cómo se usan en el análisis léxico?

* Se usan para especificar patrones para tokens, que son los componentes básicos de los lenguajes de programación.

  Se usan para definir los tokens legales y luego sintetizar automáticamente el analizador léxico.

¿Cómo se construyen?

Se construyen con una combinación de literales, operadores y símbolos especiales.

 Se pueden usar para realizar operaciones básicas como selección entre alternativas, concatenación, entre otras.

¿Qué ventajas tienen?

* Son una forma fácil y declarativa para el análisis léxico.
* Tienen más probabilidades de generar un código eficiente, fácil de mantener y seguro.
* **Reconocimiento de palabras clave**: cómo se identifican en los lenguajes de programación.
* **Identificación de tokens**: clasificación y ejemplos prácticos.

**4. Análisis sintáctico**

* **Tipos de análisis sintáctico**:
	+ Análisis sintáctico descendente.
	+ Análisis sintáctico ascendente.
* **Árboles de análisis sintáctico**: qué son y cómo ayudan en la estructura de los programas.

**5. Algoritmos de análisis de cadenas**

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

**6. Compiladores e intérpretes**

* **Estructura de un compilador**: definición y componentes.
* **Fases del compilador**:
	+ Análisis léxico.
	+ Análisis sintáctico.
	+ Generación de código.
	+ Optimización de código.
* **Lenguajes de programación utilizados en la creación de compiladores**: como **Lex y Yacc**.

**7. Aplicaciones prácticas de autómatas**

* **Reconocimiento de patrones** en diferentes áreas.
* **Procesamiento de lenguaje natural** y su uso en inteligencia artificial.
* **Análisis de texto y minería de datos**: su importancia en el análisis de información.

**8. Lenguaje de programación C y C++**

* **Librerías fundamentales**: **STDIO y CONIO,** un resumen de las funciones de cada una.
* **Lista de 10 librerías más comunes en C/C++**, además de las ya mencionadas.
* **Manejadores de formatos para diferentes tipos de datos**, con ejemplos de uso en código