# Instituto Nacional de Educación Nivel Diversificada INED Santa Cruz Naranjo

# Centro GNet

**Tema principal: Teoría de autómatas finitos deterministas**

**Materia: Producción**

**Nombre del alumno: Luis Mario López Paniagua**

**ID: 1701**

**Catedrático: Gustavo Blanco**

**Fecha: 5/04/2025**

**Grado: 5to Bachillerato**

**1. Teoría de autómatas**

* **Definición y usos de los autómatas finitos deterministas (AFD)**.

Un autómata determinista consiste de: • **Un estado finito de estados Q.** **Un estado finito de símbolos de entrada Σ.** **Una función de transición δ que toma como argumentos un estado y un símbolo de entrada y regresa un estado**.

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



Son conjunto de señales recibidas por el procesador de un ordenador, el que está compuesto por un número **finito** de compuertas (estados), cada una de las cuales pueden utilizar dos condiciones posibles, por lo general denotadas por 0 y 1(transiciones).

**2. Gramáticas formales**

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

Generan los lenguajes regulares (aquellos reconocidos por un autómata finito). Son las gramáticas

Más restrictivas

* **Gramáticas recursivas**: explicación y diferencias con las gramáticas regulares.
* Un **autómata finito** (**AF**) o **máquina de estado finito** es un [modelo computacional](https://es.wikipedia.org/wiki/Modelo_computacional) que realiza [cómputos](https://es.wikipedia.org/wiki/Proceso_%28inform%C3%A1tica%29) en forma automática sobre una [entrada](https://es.wikipedia.org/wiki/Entrada) para producir una [salida](https://es.wikipedia.org/wiki/Salida_%28inform%C3%A1tica%29).
* **Con la recursión derecha, no se realiza ninguna reducción hasta que se haya leído toda la lista de elementos; con la recursión izquierda, se realiza una reducción al encontrar cada nuevo elemento de la lista** . Por lo tanto, la recursión izquierda puede ahorrar mucho espacio en la pila.

**3. Análisis léxico**

* **Tokenización**: qué es y cómo se usa en el proceso de compilación.
* La **tokenización es un concepto que cada vez cobra más relevancia en el mundo de la tecnología, la economía y la sociedad**. Se trata de un proceso que permite [transformar cualquier tipo de activo o dato en una unidad digital](https://www.sydle.com/es/blog/smart-contracts-6568b780d11f781b770d9eee) llamada token, que puede ser almacenada, transferida y gestionada de forma segura y eficiente.
* **Expresiones regulares**: su función en el análisis léxico.

Las expresiones regulares se utilizan habitualmente en el análisis léxico para los siguientes fines: **Coincidir con palabras clave e identificadores** . Identificar literales, como enteros, números de coma flotante y caracteres. Reconocer símbolos, como operadores y signos de puntuación.

* **Reconocimiento de palabras clave**: cómo se identifican en los lenguajes de programación.
* El analizador léxico rompa las sintaxis en una serie de **fichas**, eliminando cualquier espacio en blanco o comentarios en el código fuente.
* Si el analizador léxico encuentra un sidario inválido, genera un error. Lee los flujos de caracteres del código fuente, comprueba si hay fichas legales y pasa los datos al analizador de sintaxis cuando lo exige.



* **Identificación de tokens**: clasificación y ejemplos prácticos.
* Un token de autenticación **transmite de manera segura información sobre las identidades de los usuarios entre aplicaciones y sitios web**. Permiten a las organizaciones fortalecer sus procesos de autenticación para dichos servicios.

**4. Análisis sintáctico**

* **Tipos de análisis sintáctico**:
	+ Análisis sintáctico descendente.

El análisis sintáctico descendente recursivo **es una de las técnicas de análisis sintáctico más sencillas que se utilizan en la práctica**. Los analizadores descendentes recursivos también se denominan analizadores descendentes , ya que construyen el árbol de análisis de arriba a abajo (en lugar de de abajo a arriba).

* + Análisis sintáctico ascendente.
	+ Al contrario que el analizador **descendente**, el **ascendente** (el tipo de **análisis ascendente** más conocido es el **análisis** por desplazamiento/reducción) utiliza derivaciones más a la derecha, construyendo el árbol **sintáctico** desde las hojas hasta la raíz.
* **Árboles de análisis sintáctico**: qué son y cómo ayudan en la estructura de los programas.

Árbol de sintaxis: Un árbol de sintaxis es una **representación gráfica de la estructura sintáctica abstracta del código fuente de un lenguaje de programación**. Se crea analizando el código fuente, lo que implica examinar el código y descomponerlo en sus componentes.

**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.

El **algoritmo de Boyer**-**Moore** utiliza la información recopilada durante el preprocesamiento para omitir secciones del **texto**, lo que resulta en un factor constante menor que el de muchos otros **algoritmos** de **búsqueda de cadenas**. En general, el **algoritmo** se ejecuta más rápido a medida que aumenta la longitud del patrón.

**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.