**Centro de Computación**

**Catedrático: Gustavo Blanco**

**Catedra: Producción**

**Tema: Analfabetismo Digital**

**Nombre: Marlen Rubí de Paz Donis**

**ID: 1438**

**Teoría de autómatas**

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

Un autómata finito determinista (AFD) se define como un modelo matemático que consta de un conjunto de estados, un estado inicial, un alfabeto, un conjunto de estados de aceptación y una función de transición. Utiliza la función de transición para determinar el siguiente estado en función de la entrada que recibe.

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

Considere el diseño de una máquina expendedora simple, en la que se introducen monedas para obtener un refresco. La máquina expendedora se puede modelar como un autómata finito que procesa una secuencia de entradas de monedas para realizar la transición entre estados (por ejemplo, 0, 5 centavos, 10 centavos, etc.) hasta que se alcanza el estado deseado (cambio exacto), después de lo cual dispensa un refresco.

Un ejemplo de autómata en la vida cotidiana es un elevador, ya que es capaz de memorizar las diferentes llamadas de cada piso y optimizar sus ascensos y descensos.

### ****Gramáticas formales****

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

En informática una gramática regular es una gramática formal (V, Σ, R, S) que puede ser clasificada como regular izquierda o regular derecha.

Ejemplo 7.2.3 Gramáticas regulares

1.

Defina la gramática regular G = ( N , Σ, P , S ) con no terminales N = { S }, terminales Σ = {0,1} y producciones

S→O,  S→ 0S,  S→1S

Es fácil ver que L ( G ) = {0,1}\*.

2.

Define la gramática regular G con las siguientes producciones:

S→O,  S→ aA,  S→bB, A→abS, B→baS

De las producciones podemos deducir que los no terminales son N = { S , A , B }, y los terminales son Σ = { a , b }. Si el símbolo de inicio es S , entonces L ( G ) es el lenguaje regular ( aab ∪ bba )\*. Si el símbolo de inicio es B , entonces por inspección, estamos prefijando todas las cadenas generadas por S con ba , y entonces L ( G ) = ba ( aaa ∪ bba )\*.

 **Gramáticas recursivas**: explicación y diferencias con las gramáticas regulares.

Una gramática formal es una estructura lógico-matemática con un conjunto de reglas de formación que definen las [cadenas de caracteres](https://es.wikipedia.org/wiki/Cadenas_de_caracteres) admisibles en un determinado [lenguaje formal](https://es.wikipedia.org/wiki/Lenguaje_formal) o [lengua natural](https://es.wikipedia.org/wiki/Lengua_natural). Las gramáticas formales aparecen en varios contextos diferentes: la [lógica matemática](https://es.wikipedia.org/wiki/L%C3%B3gica_matem%C3%A1tica), las [ciencias de la computación](https://es.wikipedia.org/wiki/Ciencias_de_la_computaci%C3%B3n) y la [lingüística](https://es.wikipedia.org/wiki/Ling%C3%BC%C3%ADstica) teórica, frecuentemente con métodos e intereses divergentes.

Los “lenguajes recursivamente enumerables”

que incluyen a los dos anteriores y Los “lenguajes regulares” es la clase más

pequeña, e incluye a los lenguajes más

simples. Por ejemplo, el conjunto de todos

los números binarios.

### ****Análisis léxico****

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

La tokenización es el proceso de sustituir los datos sensibles por símbolos de identificación únicos que conservan toda la información esencial de los datos sin comprometer su seguridad. Un token es un dato que no tiene ningún significado ni relación con los datos sensibles originales.

Actúa como un paso preliminar antes del análisis, que convierte los tokens en una representación estructurada, lo que permite la ejecución precisa de instrucciones.

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

Las expresiones regulares **son una forma de especificar patrones, entendiendo por patrón la forma de describir cadenas de caracteres**. Es la forma de definir los tokens o componentes léxicos y, como veremos, cada patrón concuerda con una serie de cadenas.

 **Reconocimiento de palabras clave**: cómo se identifican en los lenguajes de programación.

El significado de las palabras clave —y el significado específico de la noción de *palabra clave*— difiere ampliamente de un lenguaje de programación a otro.

En muchos lenguajes de programación, como por ejemplo [C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)), una palabra reservada identifica una forma sintáctica. Las palabras usadas en construcciones de [flujo de control](https://es.wikipedia.org/wiki/Estructuras_de_control), talesif, then, y else, son palabras clave. En estos lenguajes, las palabras clave no pueden usarse también como nombres de variables, constantes o funciones. En lenguajes con [macros](https://es.wikipedia.org/wiki/Macro) o evaluación débil, las construcciones de flujo de control se pueden implementar como macros o funciones.

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

**Para que una palabra clave se identifique como un token válido, el patrón es la secuencia de caracteres que componen la palabra clave** .

Es una agrupación de caracteres reconocidos por el analizador léxico que constituyen

los símbolos con los que se forman las sentencias del lenguaje y también se les

denomina componentes léxicos.

Constituyen los símbolos terminales de la gramática:

❖ Palabras reservadas.

❖ Identificadores.

❖ Operadores y constantes.

❖ Símbolos de puntuación y especiales.

### ****4. Análisis sintáctico****

**Tipos de análisis sintáctico**:

 Análisis sintáctico descendente.

Se puede considerar el análisis sintáctico descendente como un **intento de encontrar una derivación por la izquierda para una cadena de entrada** También se puede considerar como un intento de construir un árbol de análisis sintáctico para la entrada comenzando desde la raíz y creando los nodos del árbol en orden previo.

 Análisis sintáctico ascendente.

Análisis Ascendente (BU): **Es un tipo de análisis que construye el árbol sintáctico desde las hojas (la parte inferior) hasta la raíz (la parte superior)**. Aplica reducciones (pasos de la derivación en reversa) y desplazamientos (lecturas de tokens de entrada).

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

Un árbol de análisis **representa la estructura sintáctica de una cadena generada por una gramática libre de contexto**. Es una estructura de árbol jerárquico donde cada nodo representa un símbolo (terminal o no terminal) en la gramática.

### ****5. Algoritmos de análisis de cadenas****

### **Algoritmo de Boyer-Moore**: concepto

### l algoritmo de búsqueda de cadenas Boyer-Moore es un particularmente eficiente [algoritmo de búsqueda de cadenas](https://es.wikipedia.org/wiki/Algoritmos_de_b%C3%BAsqueda_de_subcadenas" \o "Algoritmos de búsqueda de subcadenas), y ha sido el punto de referencia estándar para la literatura de búsqueda de cadenas práctica

### A la gente frecuentemente le sorprende el algoritmo de Boyer-Moore, cuando lo conoce, porque en su verificación intenta comprobar si hay una coincidencia en una posición particular marchando hacia atrás. Comienza una búsqueda al principio de un texto para la palabra "[ANPANMAN](https://es.wikipedia.org/wiki/Anpanman)", por ejemplo, comprueba que la posición octava del texto en proceso contenga una "N". Si encuentra la "N", se mueve a la séptima posición para ver si contiene la última "A" de la palabra, y así sucesivamente hasta que comprueba la primera posición del texto para una "A".

### ****6. Compiladores e intérpretes****

###  **Estructura de un compilador**: definición y componentes.

¿Qué es un compilador?

Un compilador **es un programa informático que traduce todo el código fuente de un proyecto de software a código máquina antes de ejecutarlo**. Solo entonces el procesador ejecuta el software, obteniendo todas las instrucciones en código máquina antes de comenzar.

¿Cuál es la estructura de un compilador?

El funcionamiento de un compilador está dividido fundamentalmente en dos partes: **Una de análisis del código fuente y otra de síntesis del código objeto**, para lo cual, al igual que el ensamblador, debe construir y analizar varias tablas (ver Figura 1).

### **Fases del compilador**:

###  Análisis léxico.

### Habitualmente el término ``análisis léxico'' se refiere al tratamiento de la entrada que produce como salida la lista de tokens. Un token hace alusión a las unidades mas simples que tiene significado. Habitualmente un token o lexema queda descrito por una expresión regular.

###  Análisis sintáctico.

¿Qué es el análisis sintáctico?

El **análisis sintáctico** es el **análisis** de las funciones sintácticas o relaciones de concordancia y jerarquía que guardan las palabras cuando se agrupan entre sí en forma de sintagmas, oraciones simples y oraciones compuestas de proposiciones

### Generación de código

### En programación, la generación de código es una de las fases mediante el cual un compilador convierte un programa sintácticamente correcto en una serie de instrucciones a ser interpretadas por una máquina.

###  Optimización de código.

### La optimización eficaz del código implica realizar cambios en el código para reducir el consumo de recursos, minimizar el tiempo de ejecución y mejorar el rendimiento general. Al optimizar el código, puede identificar y eliminar las ineficiencias que podrían dar lugar a un mayor consumo de recursos y mayores costos

###  **Lenguajes de programación utilizados en la creación de compiladores**: como **Lex y Yacc**.

Yacc también se utilizó en Unix para implementar el compilador C portátil, así como analizadores para lenguajes de programación como FORTRAN 77, Ratfor, APL, bc, m4, etc. Yacc también se ha reescrito para otros lenguajes, incluidos **OCaml, Ratfor, ML, Ada, Pascal, Java, PHP, Python, Ruby, Go, Common Lisp y Erlang**

### ****7. Aplicaciones prácticas de autómatas****

###  **Reconocimiento de patrones** en diferentes áreas.

### Existen dos métodos de clasificación en el reconocimiento de patrones: supervisada y no supervisada. Para aplicar el reconocimiento de patrones supervisado, se necesita un conjunto de datos etiquetados de gran tamaño; en caso de no contar con ellos, se puede aplicar un enfoque no supervisado.

### **Procesamiento de lenguaje natural**

### El Procesamiento del Lenguaje Natural es el campo de conocimiento de la Inteligencia Artificial que se ocupa de la investigar la manera de comunicar las máquinas con las personas mediante el uso de lenguas naturales, como el español, el inglés o el chino.

###  **Análisis de texto y minería de datos**: su importancia en el análisis de información.

### La minería de datos es una técnica asistida por computadora que se utiliza en los análisis para procesar y explorar grandes conjuntos de datos. Gracias a las herramientas y métodos de minería de datos, las organizaciones pueden descubrir patrones y relaciones ocultas en sus datos.

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

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

### ¿Qué hace la librería Stdio?

### h proporciona funciones clave, como fread() y fwrite(), que permiten la lectura y escritura directa de bloques de datos binarios.

### ¿Qué es conio en C?

### La librería conio. h ofrece funciones para manipular la consola, como clrscr(), getch(), y textcolor(), optimizando la interacción con el usuario. La librería conio. h en programación en C ofrece funciones que permiten la entrada y salida de datos en la consola de una manera más controlada y eficiente.

Lista de 10 librerías más comunes en C/C++

Algunas de las librerías más comunes en C++ son:

* **<algorithm>**: Algoritmos de la plantilla estándar
* **<chrono>**: Mediciones de tiempo
* **<complex>**: Números complejos
* **<exception>**: Excepciones estándar
* **<functional>**: Objetos para funciones
* **<initializer\_list>**: Inicializar listas
* **<iterator>**: Definir iteradores
* **<limits>**: Calcular límites numéricos
* **<locale>**: Localizaciones
* **<memory>**: Elementos de memoria

### **Manejadores de formatos para diferentes tipos de datos**

### Los tipos de datos de C entran en categorías generales. Los tipos enteros incluyen int , char , short , long y long long . Estos tipos se pueden calificar con signed o unsigned , y unsigned solo se puede usar como una abreviatura para unsigned int .