



DEPARTAMENTO DE SISTEMAS COMPUTACIONALES E INFORMÁTICA

ASUNTO: **SOLICITUD DE ACTIVIDADES**

Celaya, Guanajuato, 02 / octubre / 2022

LENGUAJES Y AUTÓMATAS II

DOCENTE DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ
SEMESTRE AGOSTO-DICIEMBRE 2023

ACTIVIDAD 5 (VALOR 44 PUNTOS)

LEA CUIDADOSAMENTE, Y REALICE LAS SIGUIENTE ACTIVIDADES, CONSIDERANDO LOS CRITERIOS DE CALIDAD PROPUESTOS EN LOS DOCUMENTOS DE LA [GUÍA TUTORIAL](#), Y LA [RÚBRICA DE EVALUACIÓN](#),

EL LECTOR DEBE TOMAR MUY EN CUENTA QUE ESTA ACTIVIDAD ES UN EXAMEN, Y NO UNA SIMPLE TAREA, PUES DEMANDA DEDICACIÓN PARA INVESTIGAR, LEER, ANALIZAR, REDACTAR, ILUSTRAR Y PROPOSER DE MANERA PROFESIONAL LOS TEMAS PROPUESTOS EN LA ESTRUCTURA TEMÁTICA DE ESTA ASIGNATURA.

3. ANÁLISIS SINTÁCTICO.

- A. INVESTIGUE, LEA, COMPREnda Y ELABORE UNA MONOGRAFÍA TÉCNICA COMPLETAMENTE APEGADA A LO SOLICITADO EN LA [GUÍA TUTORIAL](#) (PUNTO 3, INCISO a) ACERCA DE LOS SIGUIENTES TEMAS :

TEMA 3.3 PRECEDENCIA DE OPERADORES
TEMA 3.4.1 ANALIZADOR SINTÁCTICO DESCENDENTE (LL)
TEMA 3.4.2 ANALIZADOR SINTÁCTICO ASCENDENTE (LR, LALR)
TEMA 3.5 DISEÑO Y ADMINISTRACIÓN DE UNA TABLA DE SÍMBOLOS.
TEMA 3.6 MANEJO DE ERRORES SINTÁCTICOS Y SU RECUPERACIÓN,
TEMA 3.7 GENERADORES DE CÓDIGO PARA ANALIZADORES SINTÁCTICOS: YACC, BISON (REPASO).

CONSIDERACIÓN :

DEBE USTED ENTENDER EL VALOR QUE TIENE ESTA ACTIVIDAD Y QUE LOS TEMAS ANTES REFERIDOS, PARA NADA DEBEN SER ABORDADOS COMO SIMPLES CONCEPTOS REDACTADOS CON LA LIGEREZA QUE YA SE HA OBSERVADO EN ACTIVIDADES PREVIAS.

ANALICE CADA TEMA, SUS CARACTERÍSTICAS, SU IMPORTANCIA, SUS CONCEPTOS, SUS EJEMPLOS, SUS ILUSTRACIONES, Y LOS TIPOS DE EVIDENCIAS QUE USARÁ PARA DEMOSTRAR QUE USTED HA ADQUIRIDO UN VERDADERO CONOCIMIENTO ACERCA DE ÉSTOS.





A MODO DE PRÁCTICA REALICE ESTE PUNTO Y ELABORE EJERCICIOS PRÁCTICOS CON LOS CUÁLES USTED DEMUESTRE

¿ CÓMO FUNCIONA UN ANALIZADOR DESCENDENTE ?

¿ CÓMO FUNCIONA UN ANALIZADOR ASCENDENTE ?

¿ CUÁLES SON LOS OBJETIVOS Y LAS FUNCIONES DE UN ANALIZADOR SINTÁCTICO ?

A MODO DE PRÁCTICAS REALICE ESTE PUNTO Y ELABORE EJERCICIOS NECESARIOS CON LOS CUÁLES USTED DEMUESTRE

¿ CÓMO FUNCIONA EL GENERADOR DE CÓDIGO PARA ANALIZADORES SINTÁCTICOS YACC ?

¿ CÓMO FUNCIONA EL GENERADOR DE CÓDIGO PARA ANALIZADORES SINTÁCTICOS BISON ?.

¿ CUÁLES SON LAS CARACTERÍSTICAS Y LAS FUNCIONES DE ESTOS DOS ANALIZADORES SINTÁCTICO ?

ELABORE UN PAR DE VIDEOS DONDE EXPONGA CÓMO EMPLEÓ ESTAS DOS HERRAMIENTAS. COLOQUE SU MATERIAL EN YOUTUBE O EN ALGUNA OTRA PLATAFORMA E INCLUYA LA LIGA DENTRO DE SU EXAMEN.

IMPORTANTE : SI LO REQUIERE PUEDE CONSULTAR EL [SIGUIENTE DOCUMENTO](#) PARA ORIENTAR SU TRABAJO EN CONOCER QUÉ ES Y CÓMO HACER UNA MONOGRAFÍA CON EL RIGOR ACADÉMICO REQUERIDO.

POR ÚLTIMO, RECUERDE LEER LA [GUÍA TUTORIAL](#) PARA EL CORRECTO TRATAMIENTO DE ESTE INCISO.

¿ QUÉ SE CALIFICARÁ ? : LA RÚBRICA PARA EVALUAR ESTA ACTIVIDAD ESTARÁ INTEGRADA POR LOS SIGUIENTES CRITERIOS.

- a. **LA OPORTUNIDAD.** SI EL TRABAJO FUE ENTREGADO OPORTUNAMENTE.
- b. **LA COMPRENSIÓN.** SE VALORARÁ EL GRADO DE COMPRENSIÓN DEL TEMAS ANALIZADOS.
- c. **LA CALIDAD.** SI LAS EVIDENCIAS ENVIADAS CORRESPONDEN A LA CALIDAD ESPERADA PARA ESTE NIVEL PROFESIONAL QUE SE CURSA.
- d. **LA CAPACIDAD DE SÍNTESIS.** SI LAS EVIDENCIAS ENTREGADAS TIENEN EL NIVEL DE DETALLE Y PROFUNDIDAD REQUERIDA, O EN BIEN SI SE OMITIERON CONCEPTOS CON EL AFÁN DE SIMPLIFICAR Y ENTREGAR UN MATERIAL ACADÉMICA Y TÉCNICAMENTE POBRE.
- e. **LA CREATIVIDAD.** LA MANERA EN QUE SE EXPRESAN LOS CONCEPTOS Y EL TRATAMIENTO QUE SE DA A LA INFORMACIÓN ANALIZADA PARA QUE ÉSTA SEA COMPRESIBLE EN SU ESENCIA.

IMPORTANTE : CUENTA CON EL TIEMPO SUFFICIENTE PARA REALIZAR ESTA ACTIVIDAD Y SUMAR PUNTOS IMPORTANTES A SU CALIFICACIÓN DE ESTA EVALUACIÓN.

IMPORTANTE : TODO EL MATERIAL ESCRITO DEBERÁ SER HECHO A MANO.





- B. ELABORE **UN VIDEO DE AL MENOS 25 MINUTOS**, DONDE A MODO DE VIDEO CONFERENCIA COMENTE Y EXPLIQUE LO DESARROLLADO TODOS LOS INCISOS ANTERIORES.

SI TIENE REGISTRADO UN EQUIPO, HAGA LA VIDEO CONFERENCIA CON LOS INTEGRANTES Y EXPLIQUEN LO REALIZADO EN ESTA ACTIVIDAD.

NOTA: LA LIGA DEL VIDEO DEBERÁ SER INTEGRADA AL PDF DE LA ACTIVIDAD, PARA QUE AL HACER CLIC EN ÉSTE SE REPRODUZCA. POR ÚLTIMO, NO OLVIDE OTORGAR LOS PRIVILEGIOS NECESARIOS PARA COMPARTIR CORRECTAMENTE ESTE MATERIAL.



Av. Antonio García Cubas #600 esq. Av. Tecnológico, Colonia Alfredo V. Bonfil, C.P. 38010
Celaya, Gto. Tel. 01 (461) 611 75 75 e-mail: lince@celaya.tecnm.mx tecnm.mx | celaya.tecnm.mx





CONSIDERACIONES.

CADA UNO DE LOS PUNTOS ANTERIORES DEBE SER DESARROLLADO CON LA PROFUNDIDAD ACORDE A UN NIVEL PROFESIONAL, Y APEGÁNDOSE COMPLETAMENTE A LAS DIRETRICES DE LA GUÍA TUTORIAL.

NO CONCIBA ESTE TRABAJO, COMO UN SIMPLE RESUMEN O EJERCICIO DE TRANSCRIPCIÓN, PUES EL VALOR INDICADO AL INICIO DE ESTA ACTIVIDAD LE DARÁ A USTED UNA BUENA IDEA DE LO QUE SE ESPERA DE ELLA, EN CUANTO A CALIDAD Y EL APRENDIZAJE OBTENIDO, MISMO QUE SERÁ PUESTO A PRUEBA MEDIANTE UN EXAMEN ESCRITO O BIEN ORAL EN CLASE.

SI DECIDIÓ ELABORAR ESTA ACTIVIDAD EN EQUIPO, CADA INTEGRANTE DE ÉSTE DEBERÁ POSEER EL MISMO NIVEL DE CONOCIMIENTO, PUES TAN SOLO REPARTIR TEMAS ENTRE LOS INTEGRANTES DEL EQUIPO, SUPONDRIÁ UN GRAVE ERROR DE INTERPRETACIÓN A LA INTENCIÓN DIDÁCTICA REAL DE ESTA ACTIVIDAD.

POR ÚLTIMO, ESTA ACTIVIDAD SOLO SE PODRÁ DESARROLLAR EN EQUIPO, SI SE REGISTRÓ EN UNO PREVIAMENTE, UTILIZANDO EL FORMATO ENTREGADO EN LA ACTIVIDAD INICIAL. DE LO CONTRARIO DEBERÁ ELABORAR Y ENTREGAR LA ACTIVIDAD DE FORMA INDIVIDUAL.

LA ENTREGA DE DICHO REGISTRO SE HARÁ VÍA CORREO ELECTRÓNICO ENVIANDO ÉSTE AL PROFESOR DESIGNADO, Y POSTERIORMENTE EN CLASE ENTREGANDO LA HOJA EN FÍSICO.

OBSERVACIONES:

- CADA HOJA QUE ENTREGUE DE SU ACTIVIDAD, DEBERÁ ESTAR FIRMADA AL MARGEN DERECHO, INCLUIDA LA PROPIA SOLICITUD DE LA ACTIVIDAD.
- INTEGRE TODO SU TRABAJO EN UN SOLO ARCHIVO DE TIPO .PDF, Y ASIGNE EL NOMBRE QUE A CONTINUACIÓN SE INDICA.

NO OLVIDE ANEXAR LAS HOJAS DE ESTA ACTIVIDAD Y DE SU TRABAJO DESPUÉS DE SU PORTADA.

- UNA VEZ ELABORADA SU ACTIVIDAD, RECUERDE DIGITALIZARLA Y NOMBRARLA EN BASE A LA NOMENCLATURA QUE SE INDICA MÁS ADELANTE EN ESTE DOCUMENTO.
- SI SUS EVIDENCIAS ENVIADAS POR CORREO, NO CUMPLEN CON LA NOMENCLATURA SOLICITADA, NO SERÁN CONSIDERADAS COMO EVIDENCIAS PARA SU EVALUACIÓN.
- POR ÚLTIMO, POR FAVOR GESTIONE APROPIADAMENTE SU TIEMPO, Y SEA PUNTUAL EN SU ENTREGA Y ASÍ EVITAR PROBLEMAS DE NULIDAD POR EXTEMPORANEIDAD.





LA NOMENCLATURA SOLICITADA PARA ENVIAR SU TRABAJO ES LA SIGUIENTE :

AAAA-MM-
DD_TNM_CELAYA_MATERIA_DOCUMENTO_[EQUIPO]_NOCTROL_APELLIDOS_NOMBRE_SEM.PDF

(NOTA : * TODO DEBE SER ESCRITO USANDO LETRAS MAYÚSCULAS ***)**

DONDE :

TNM_CELAYA	:	INSTITUCIÓN ACADÉMICA
AAAA	:	AÑO
MM	:	MES
DD	:	DÍA
MATERIA	:	LAI ^{II} , LI MÁS EL GRUPO (-A , -B, -C)
DOCUMENTO	:	A1-ACTIVIDAD 1, P1-PRACTICA 1, R1-REPORTE 1, T1-TAREA 1, PG1-PROGRAMA, ETC. (CAMBIANDO EL NÚMERO CONSECUTIVO POR EL QUE CORRESPONDA)
[EQUIPO]	:	NÚMERO DEL EQUIPO QUE CORRESPONDA SEGÚN INDICACIÓN DEL PROFESOR. [OPCIONAL]
NOCTROL	:	SU NÚMERO DE CONTROL
APELLIDOS	:	SUS APELLIDOS
NOMBRE	:	SU NOMBRE
SEM	:	EL PERIODO SEMESTRAL EN CURSO: AGO-DIC

EJEMPLO :

SI EL TRABAJO SE SOLICITÓ EN EQUIPO.

2023-10-02_TNM_CELAYA_LAI^{II}-A_A5_EQUIPO_99_9999999_PEREZ_PEREZ_JUAN_AGO-DIC23.PDF

DONDE EL NOMBRE DEBERÁ CORRESPONDER AL JEFE DE EQUIPO QUE HACE LA ENTREGA DEL TRABAJO.

SI EL TRABAJO SE SOLICITÓ INDIVIDUALMENTE.

2023-10-02_TNM_CELAYA_LAI^{II}-A_A5_9999999_PEREZ_PEREZ_JUAN_AGO-DIC23.PDF





FECHA Y HORA DE ENTREGA:

LA INDICADA EN LA PLATAFORMA VIRTUAL.

EN CASO DE QUE EL TRABAJO SE HAYA SOLICITADO EN EQUIPO, EL JEFE DEL MISMO SERÁ EL ÚNICO RESPONSABLE DE ENVIAR LA ACTIVIDAD EN LA PLATAFORMA VIRTUAL.

MUY IMPORTANTE:

1. DESPUÉS DE LA HORA INDICADA EN LA PLATAFORMA VIRTUAL (AÚN CUANDO SOLO SEA UN MINUTO O VARIOS), LA ACTIVIDAD SERÁ CONSIDERADA COMO EXTEMPORÁNEA Y NO CONTARÁ COMO EVIDENCIA PARA SU EVALUACIÓN.

SE LE SUGIERE ENVIAR CON ANTICIPACIÓN SU ACTIVIDAD A FIN DE EVITAR CONFLICTOS POR NO ENTREGAR ÉSTA A TIEMPO.

BAJO NINGÚN PRETEXTO O JUSTIFICACIÓN SE ACEPTARÁN LOS TRABAJOS EXTEMPORÁNEOS, EVITE LA PENA DE RECORDAR A USTED QUE EL VALOR DE LA PUNTUALIDAD ES PARTE IMPORTANTE DE SUS EVIDENCIAS Y ES EL PRIMER PUNTO QUE SE HA DE EVALUAR.

2. NO OLVIDE ANEXAR A SU ARCHIVO .PDF DE EVIDENCIAS UNA PORTADA PROFESIONAL, Y ESTA SOLICITUD DE ACTIVIDADES CON TODAS LAS HOJAS FIRMADAS EN EL MARGEN DERECHO.
3. POR ÚLTIMO, TODA EVIDENCIA GENERADA QUE CONTENGA AL MENOS UNA TRANSCRIPCIÓN DE CUALQUIER FUENTE Y DE CUALQUIER TIPO, ES DECIR CON MATERIAL PLAGIADO SERÁ ANULADA DE FORMA INCONTROVERTIBLE.



Av. Antonio García Cubas #600 esq. Av. Tecnológico, Colonia Alfredo V. Bonfil, C.P. 38010
Celaya, Gto. Tel. 01 (461) 611 75 75 e-mail: lince@celaya.tecnm.mx tecnm.mx | celaya.tecnm.mx



Tecnológico Nacional

De México en Celaya

Asignatura: Lenguajes y Autómatas II

Docente: Ricardo González González

Actividad 5

Equipo 4

Integrantes:

Cardoso Morales Cristian David
González Rodríguez Francisco
Núñez Servin Juan Ángel

11/10/2023

Tecnológico Nacional De México En Celaya

Asignatura : Lenguajes y Autómatas II

Docente : Ricardo González González

"Monografía General"

Integrantes :

* Cardoso Morales Cristian David

* González Rodríguez Francisco

* Núñez Scrivin Juan Ángel

11/10/2023

Tabla de contenido

Tema 3.3-Precedencia de operadores.....	4
3.3.1-Definicion de operadores.....	4
3.3.2-Jerarquia de precedencia.....	5
3.3.3-Aplicación en la programación.....	7
Tema 3.4.1-Analizador sintáctico descendente (LL)	9
3.4.1.1-Acerca de.....	9
3.4.1.2-Definición.....	10
3.4.1.3-Funcionamiento.....	10
3.4.1.4-Ventajas.....	12
3.4.1.5-Desafíos.....	13
Tema 3.4.2-Analizador sintáctico ascendente (LR, LALR)	14
3.4.2.1-Analizador sintáctico ascendente LR.....	15
3.4.2.2-Definición.....	15
3.4.2.3-Tabla LR.....	15
3.4.2.4-Autómata de pila.....	16
3.4.2.5-Analizador sintáctico ascendente LALR.....	16
3.4.2.6-Definición.....	16
3.4.2.7-Compactación de la tabla.....	16
3.4.2.8-Comparación LR vs LALR.....	17
3.4.2.9-Aplicaciones.....	18
Tema 3.5-Diseño y administración de una tabla de símbolos.....	19
3.5.1-Definición y propósito.....	19
3.5.2-Diseño de una tabla de símbolos.....	20
3.5.3-Administración de una tabla de símbolos.....	21
3.5.4-Importancia en la compilación.....	22
Tema 3.6-Manejo de errores sintácticos y su recuperación.....	24
3.6.1-Errores sintácticos: Definición y tipos.....	25
3.6.2-Detección de errores sintácticos.....	26
3.6.3-Reporte de errores sintácticos.....	26
3.6.4-Recuperación de errores sintácticos.....	27
3.6.5-Importancia del manejo de errores sintácticos.....	28
Tema 3.7-Generadores de código para analizadores sintácticos:	
Yacc, Bison (repaso)	29
3.7.1-Generadores de código para analizadores sintácticos.....	30
3.7.2-YACC (Yet Another Compiler Compiler)	31
3.7.3-BISON.....	33
Conclusión.....	34

Índice de cuadros

Figura 1-Nivel de precedencia	5
Tabla 1-Precedencia de operadores.....	8
Figura 2-Árbol de constituyentes.....	9
Figura 3-Árbol de dependencias.....	10
Figura 4-Análisis sintáctico predictivo no recursivo.....	12
Figura 5-Grámatica.....	13
Tabla 2-Estapa de análisis	14
Figura 6-LR CON LALR.....	17
Figura 7-Aplicaciones.....	18
Figura 8-Registro de símbolos.....	20
Figura 9-Proceso de compilación y tabla de símbolos.....	23
Figura 10-Declaración de constantes.....	24
Figura 11-Errores sintácticos.....	27
Figura 12-Fase 1.....	28
Figura 13-Fase 2.....	29
Figura 14-Acerca de YACC.....	32
Figura 15-Acerca de BISON.....	33

Tema 3.3 Precedencia De Operadores

La precedencia de operadores es un concepto fundamental en la programación y las matemáticas que determina el orden en el que se evalúan las operaciones en una expresión. Esta regla es crucial para asegurar que los cálculos se realicen de manera correcta y consistente. En este trabajo, exploraremos en detalle la precedencia de operadores, su importancia en la programación y cómo se aplica en diferentes lenguajes de programación.

3.3.1 - Definición de operadores

Los operadores son símbolos o palabras reservadas que indican una operación a realizar entre dos o más valores. Estos operadores pueden ser aritméticos (como la suma, la resta, la multiplicación,

y la división), lógicos (como AND, OR y NOT), de comparación (como igual, mayor que, menor que), entre otros. Cada operador tiene un nivel de precedencia que dicta cuándo se ejecuta en relación con otros operadores en una expresión.

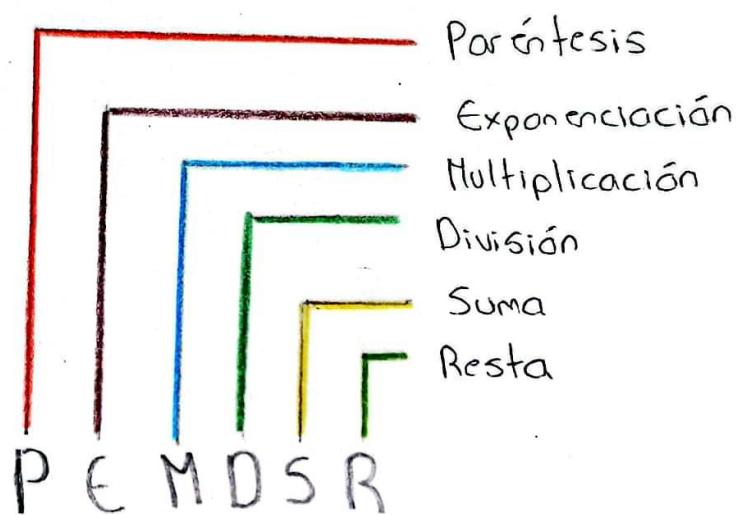


Figura 1.- Nivel de precedencia

3.3.2 - Jerarquía de precedencia

La jerarquía de precedencia establece un orden muy específico en el que se evalúan los operadores en una expresión. Por lo general, se sigue una jerarquía de arriba hacia abajo, donde todos los

Operadores con mayor precedencia se evalúan primero.

A continuación, se presenta una jerarquía, común de precedencia de operadores:

1.- Paréntesis - Los operadores dentro de paréntesis se evalúan primero.

2.- Exponentes - Las operaciones de exponentes, como la potenciación, tienen la segunda mayor precedencia.

3.- Multiplicación y División - Los operadores de multiplicación (*) y división (/) tienen una igual precedencia y se evalúan antes que la suma y la resta.

4.- Suma y Resta - Los operadores de suma (+) y resta (-) tienen la menor precedencia en toda esta jerarquía.

Ejemplo

Para comprender mejor como funciona la precedencia de operadores, consideramos la siguiente expresión:

$$8 + 6 * 4$$

Según la jerarquía de precedencia, primero se evalúa la multiplicación (6×4) y luego se realiza la suma ($8 +$ resultado de la multiplicación), dando como resultado 32. Si se quisiera cambiar el orden de evaluación, se pueden utilizar paréntesis para alterar la precedencia:

$$(8+6) * 4$$

En este caso primero se realiza la suma dentro del paréntesis ($8+6$), y luego se multiplica por 4, obteniendo el resultado de 56.

3.3.3 - Aplicación en la programación

La comprensión de la precedencia de operadores es esencial en la programación, ya que asegura que las expresiones matemáticas se evalúen de manera coherente y se obtengan resultados precisos.

Diferentes lenguajes de programación pueden tener reglas específicas de precedencia de operadores, por lo que es importante consultar la documentación de cada lenguaje para saber cómo se aplican estas reglas.

Asociatividad	Operadores	Información Adicional
no asociativo	clone new	clone and new
Izquierda	[array ()
no asociativo	++ --	incremento/decuento
no asociativo	instanceof	tipos
Derecha	!	lógico
Izquierda	* / %	aritmética
Izquierda	<< >>	bit a bit
no asociativo	<<= >>= <>	comparación
no asociativo	== != === !== ==	comparación
Izquierda	&	bit a bit y referencias
Izquierda	^	bit a bit
Izquierda		bit a bit
Izquierda	&&	bit a bit
Izquierda	? :	ternario
Izquierda	and	lógico
Izquierda	xor	lógico
Izquierda	or	lógico

Tabla 1.- Precedencia de operadores

Tema 3.4.1 Analizador Sintáctico Descendente (LL)

3.4.4.1.1 - Acerca de

Los analizadores sintácticos, una parte fundamental en la fase de análisis de un compilador, se encargan de verificar si una secuencia de tokens (generada por el analizador léxico) corresponde a una estructura sintáctica válida según la gramática del lenguaje de programación. Este analizador es de los más comunes y eficientes y también se conoce como análisis descendente predictivo.

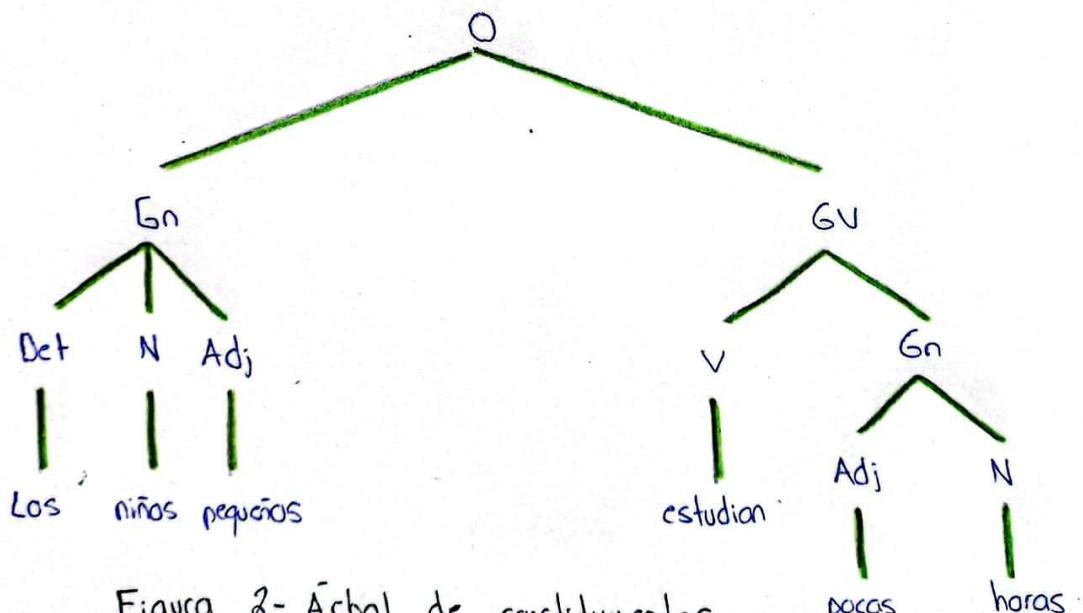




Figura 3 - Árbol de dependencias

3.4.1.2 - Definición

Un analizador sintáctico descendente LL es una herramienta que se utiliza en la compilación de programas para verificar la estructura gramatical de un código fuente. El término "LL" se refiere a "Left-to-Right, Leftmost derivation", lo que significa que el analizador comienza desde la izquierda del código fuente y construye una derivación de la gramática más a la izquierda.

3.4.1.3 - Funcionamiento

1.- Predictivo: El Analizador Sintáctico Descendente LL es predictivo en el sentido de que puede prever qué regla gramatical debe aplicarse en función del token actual.

No necesita retroceder ni realizar múltiples intentos para determinar la estructura sintáctica correcta del código. Esta característica lo hace rápido y eficiente.

2.- Tabla de Análisis Sintáctico: El analizador utiliza una tabla de análisis sintáctico (tabla LL) que se construye previamente o partir de la gramática del lenguaje. Esta tabla es esencial porque contiene información sobre qué producción de la gramática se debe utilizar en función del estado actual del análisis y del token actual en el código fuente.

3.- Recursión: En el proceso de análisis, el analizador utiliza funciones recursivas para aplicar las producciones gramaticales. Cada no terminal en la gramática se corresponde con una función de análisis. Cuando se encuentra un no terminal en el código fuente, se llama a la función de análisis más correspondiente para checar ese no terminal. Este enfoque recursivo es una característica distinta de los analizadores sintácticos descendentes.

4.- Top- Down: El Analizador Sintáctico Descendente LL sigue un enfoque "top-down". Comienza desde el símbolo inicial de la gramática y desciende hacia los símbolos terminales en el código fuente.

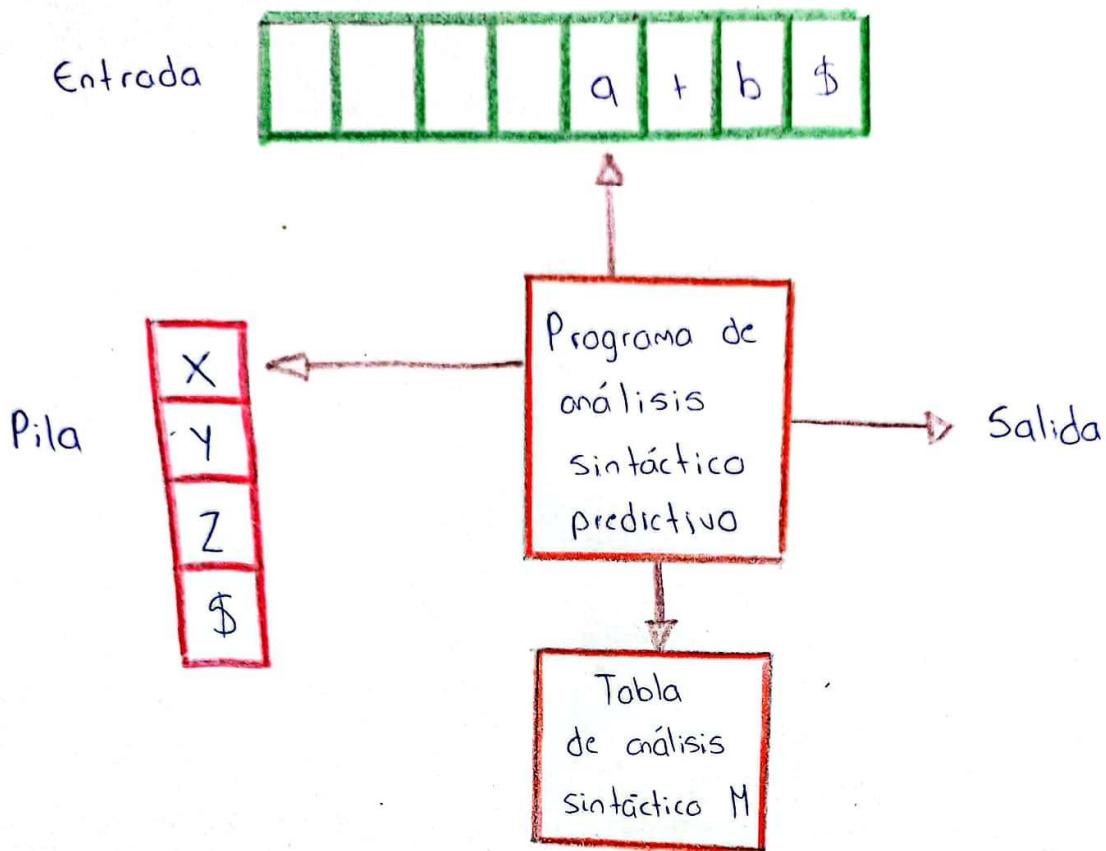


Figura 4- Análisis sintáctico predictivo no recursivo

3.4.1.4 - Ventajas

* **Eficiencia:** La capacidad predictiva y el uso de una tabla de análisis sintáctico hacen que ese tipo de analizador sea eficiente en términos de velocidad de análisis.

* **Mensajes de error útiles:** Puede proporcionar mensajes de error claros y útiles cuando se encuentra un error sintáctico en el código, lo que facilita la corrección por parte del programador.

$$E \rightarrow E + E$$

$$E \rightarrow E \wedge E$$

$$E \rightarrow id$$

$$E \rightarrow num$$

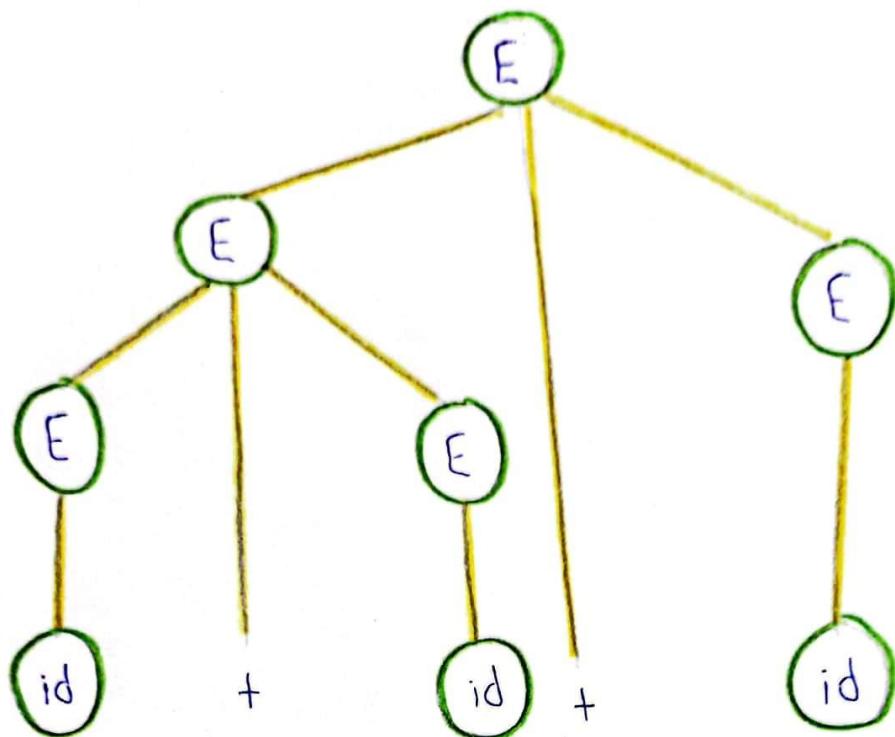


Figura 5 - Gramática

3.4.1.5 - Desafíos

* Limitaciones gramaticales: No puede manejar todas las gramáticas. En particular, no puede manejar gramáticos más ambiguos o gramáticas que requieren retroceso (backtracking) en el análisis.

* Limitaciones en lenguajes complejos: Puede no ser adecuado para lenguajes de programación extremadamente complejos, donde las reglas gramaticales son difíciles de definir de manera no ambigua.

Tema 3.4.2 Analizador Sintáctico Ascendente (LR, LALR)

Los analizadores sintácticos son componentes fundamentales en la construcción de compiladores y procesadores de lenguaje. Su función es verificar si un programa escrito en lenguaje de programación cumple con las reglas gramaticales definidas para ese lenguaje. Dentro de los analizadores sintácticos, los LR (Left to right) y LALR (Look Ahead LR) desempeñan un papel esencial en la etapa de análisis.

Forma sentencial	Handle	Producción reductora
$id_1 * id_2$	id_1	$F \rightarrow id$
$F * id_2$	F	$T \rightarrow F$
$T * id_2$	id_2	$F \rightarrow id$
$T * F$	$T * F$	$T \rightarrow T * F$
T	T	$E \rightarrow T$
S	E	$S \rightarrow E$

Tabla 2- Etapa de análisis

3.4.2.1 - Analizador Sintáctica

Ascendente LR

3.4.2.2 - Definición

Un analizador sintáctico ascendente LR opera de derecha a izquierda y realiza derivaciones de gramática desde la derecha hacia la izquierda. El término "L" este proviene de "Left-to-right" y "Rightmost derivation". Estos analizadores utilizan una pila y una tabla de análisis LR para determinar cómo reducir la pila en función del flujo de tokens en el código fuente.

3.4.2.3 - Tabla LR

El corazón del analizador LR es la tabla de análisis LR, que se deriva de la gramática del lenguaje. Esta tabla contiene información sobre cómo se deben realizar las reducciones y desplazamientos en función del estado actual y el token actual en la entrada.

3.4.2.4 - Autómata de Pila

Los analizadores sintácticos ascendentes LR funcionan mediante un autómata de pila. La pila se utiliza para rastrear el estado actual del análisis y las transiciones en el autómata de pilas se basan en las entradas de la tabla LR.

3.4.2.5 - Analizador Sintáctico Ascendente LALR

3.4.2.6 - Definición

Los analizadores sintácticos LALR (Look-Ahead LR) son una variante de los analizadores LR que emplean un enfoque más compacto para la tabla de análisis. La sigla "LALR" proviene de "Look-Ahead LR", lo que significa que consideran un conjunto de tokens de vistazo para tomar decisiones de reducción y desplazamiento.

3.4.2.7 - Compactación de la Tabla

En comparación con los analizadores LR estándar, los

analizadores LALR pueden tener una tabla más pequeña debido a su capacidad para agrupar estados similares. Esto hace que los tablas LALR sean más eficientes en términos de espacio de almacenamiento.

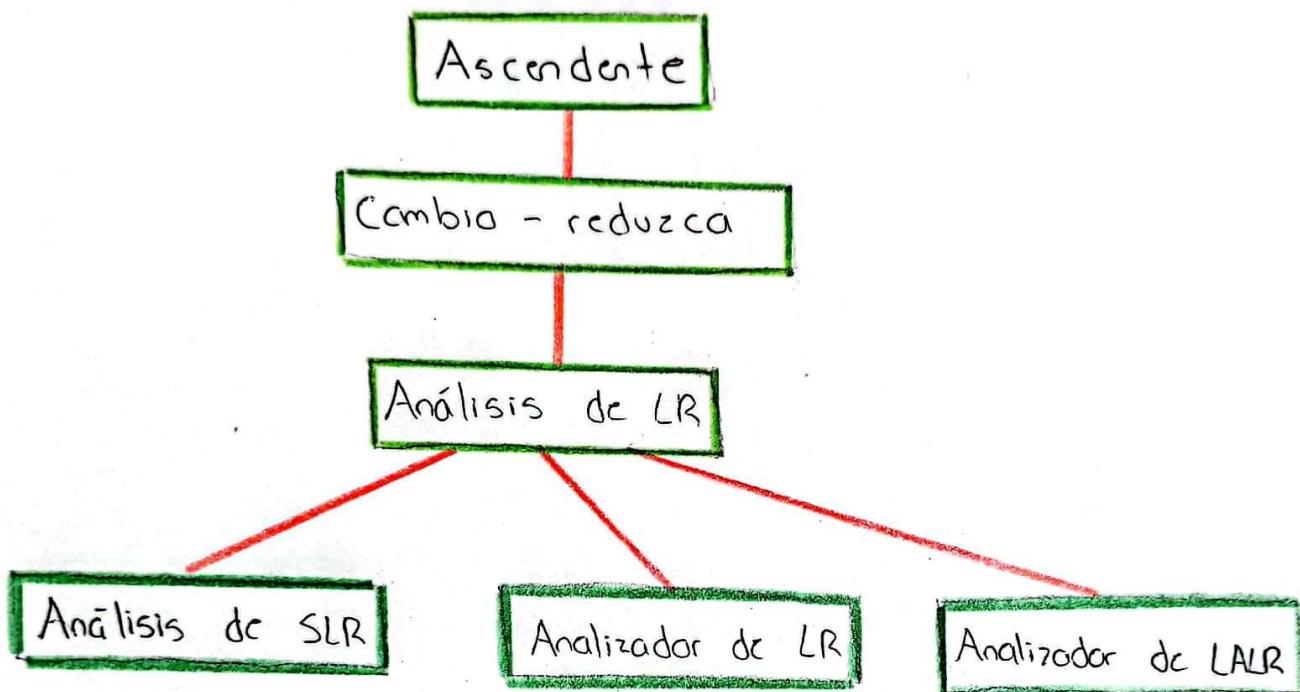


Figura 6.- LR con LALR

3.4.2.8 - Comparación LR vs LALR

* Los analizadores LR son más poderosos y pueden manejar un conjunto más amplio de gramáticas, pero sus tablas de análisis pueden ser más grandes y costosas en términos de memoria.

* Los analizadores LALR, aunque menos poderosos que los LR, son más eficientes en cuanto a memoria y, en muchos casos, pueden ser suficientes para analizar la mayoría de los lenguajes de programación.

3.4.2.9 - Aplicaciones

* Los analizadores sintácticos LR y LALR son más ampliamente utilizados en la construcción de compiladores y procesadores de lenguaje, donde se aplican para verificar la estructura sintáctica de un programa.

* También se emplean en la detección y manejo de errores sintácticos, lo que ayuda a los programadores a identificar y corregir problemas en su código.

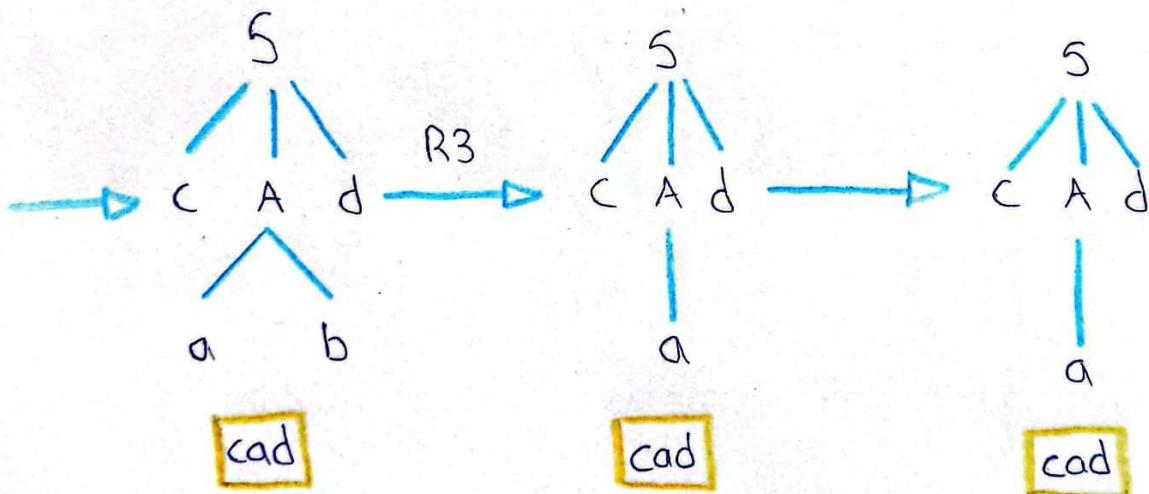


Figura 7- Aplicaciones

Tema 3.5 - Diseño y Administración de una tabla de símbolos

En el proceso de compilación de programas, la gestión de una tabla de símbolos es uno factor esencial. La tabla de símbolos es una estructura de datos que almacena información sobre las variables, constantes, funciones y otros identificadores utilizados en un programa. Se explorará en detalle el diseño y la administración de una tabla de símbolos, su importancia en la compilación y cómo se utiliza para garantizar la integridad y corrección de un programa compilado.

3.5.1 - Definición y Propósito

Una tabla de símbolos es una estructura de datos que actúa como una base de datos para almacenar información sobre los símbolos (como variables, funciones, nombres de tipos, etc), utilizados en un programa. Cada entrada en la tabla de símbolos contiene información importante, como el nombre del símbolo, su tipo, su

valor (si es una constante), su ubicación en la memoria y otros atributos relevantes. El propósito principal de la tabla de símbolos es permitir al compilador llevar un registro de los símbolos utilizados en el programa y garantizar que se utilicen de una manera coherente y según las reglas del lenguaje.

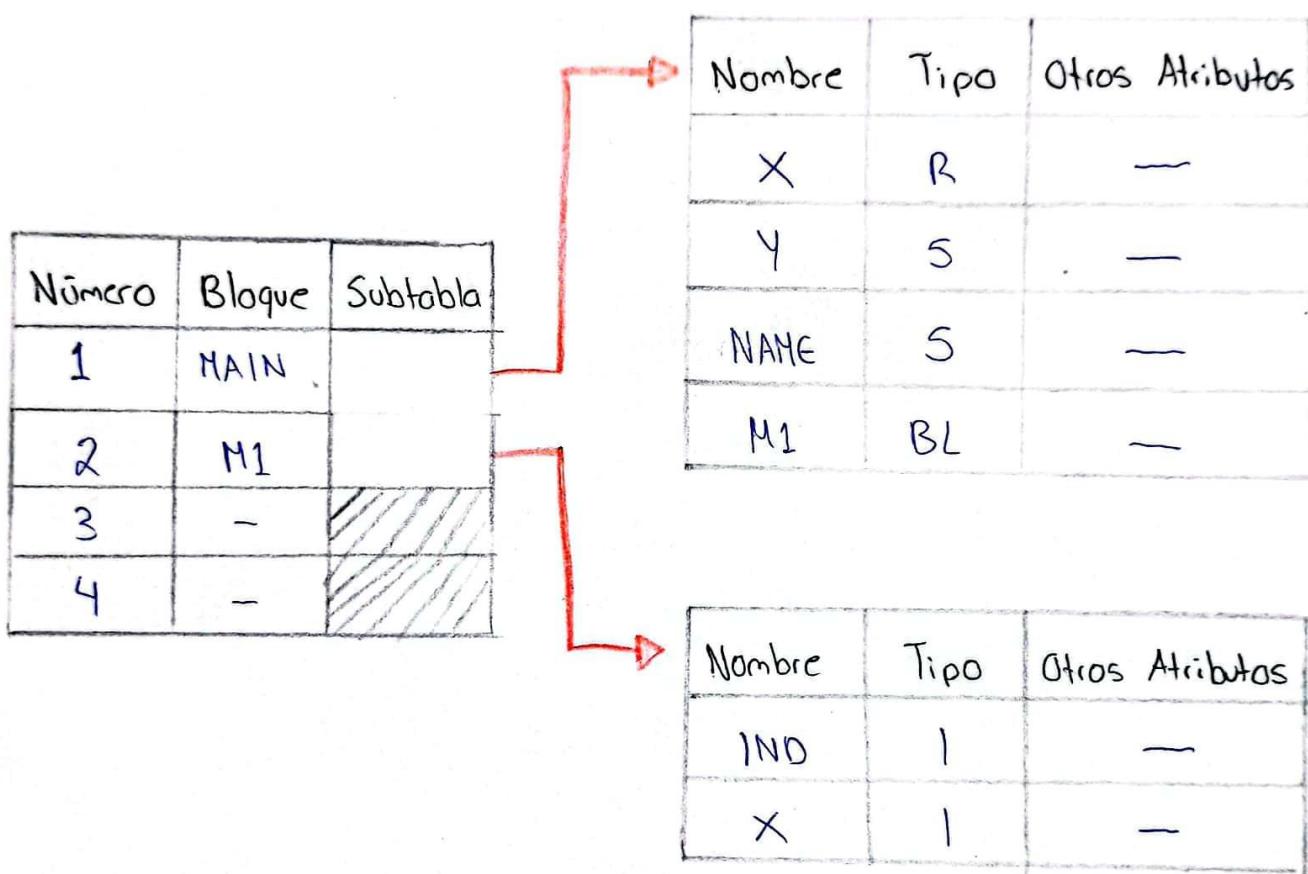


Figura 8.- Registro de símbolos

3.5.2 - Diseño de una tabla de símbolos

El diseño de una tabla de símbolos debe de ser cuidadosamente planificado pero que sea eficiente

y efectivo en su función. Aquí hay algunos elementos clave en el diseño de una tabla de símbolos:

1- Estructura de datos: La tabla de símbolos puede implementarse utilizando diversas estructuras de datos, como tablas hash, árboles de búsqueda binaria, listas enlazadas o arreglos. La elección de la estructura depende de la eficiencia requerida y de la cantidad de símbolos a administrar.

2- Atributos de los símbolos: Cada entrada en la tabla debe almacenar una serie de atributos relevantes para el símbolo, como su nombre, tipo, ámbito de visibilidad y otros datos relacionados con la semántica del programa.

3- Resolución de colisiones: Si se utiliza una estructura de datos como una tabla hash para implementar la tabla de símbolos, se debe considerar cómo manejar colisiones para garantizar un acceso eficiente a símbolos.

3.5.3 - Administración de una tabla de símbolos

La administración de una tabla de símbolos implica varios operaciones clave, como la inserción, búsqueda

y eliminación de símbolos. Aquí están las principales actividades relacionadas con la administración:

- 1.- Inserción: Cuando se encuentra un nuevo símbolo en el programa, se debe insertar en la tabla de símbolos junto con sus atributos relevantes.
- 2.- Búsqueda: Antes de utilizar un símbolo en el programa, se debe buscar en la tabla de símbolos para verificar su existencia y obtener información sobre él.
- 3.- Actualización: Si se permite la redefinición de símbolos, la tabla de símbolos debe ser capaz de manejar las actualizaciones de atributos cuando se encuentre una nueva declaración para un símbolo existente.
- 4.- Eliminación: Cuando un símbolo sale de alcance o ya no se necesita, se debe eliminar de la tabla de símbolos para liberar recursos.

3.S.4- Importancia en la compilación

La tabla de símbolos es un componente crítico en el proceso de compilación, ya que ayuda a garantizar la coherencia semántica y la corrección del programa.

fuente. Al verificar los símbolos para que se utilicen de acuerdo con las reglas del lenguaje y almacenar información importante sobre ellos, la tabla de los símbolos facilita la generación de código abierto objeto y la detección de errores semánticos durante la compilación.

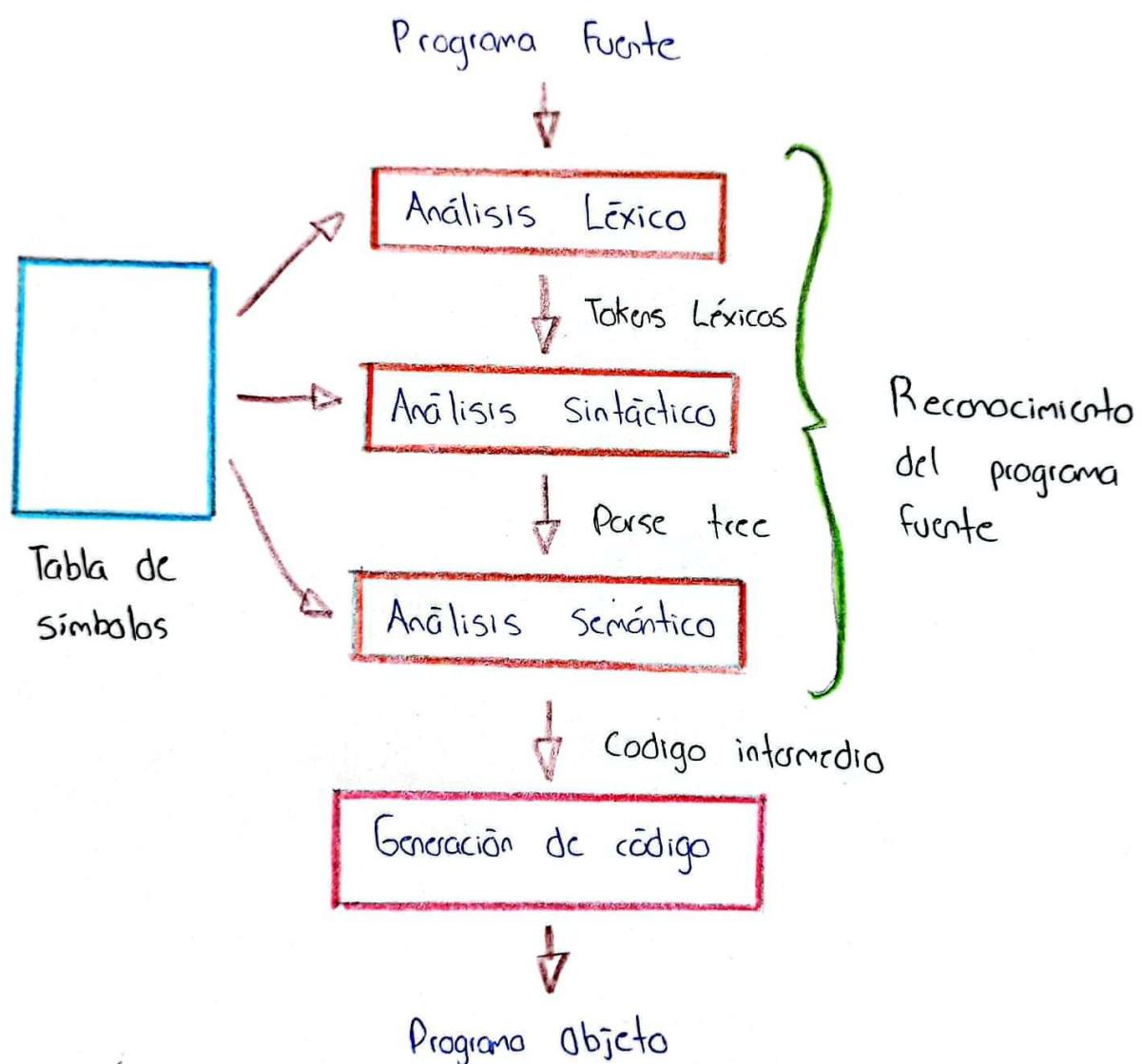


Figura 9:- Proceso de compilación y tabla de símbolos

Tema 3.6 - Manejo De Errores Sintácticos Y Su Recuperación

El proceso de compilación es una fase crítica en el desarrollo de programas, donde el código fuente escrito por el programador se traduce en código ejecutable. Durante esta fase, es esencial detectar y manejar errores sintácticos en el código fuente, ya que pueden afectar la correcta compilación del programa. Exploraremos en profundidad el manejo de los errores sintácticos en compiladores, incluida su detección, reporte y recuperación, así como su importancia en la construcción de compiladores robustos.

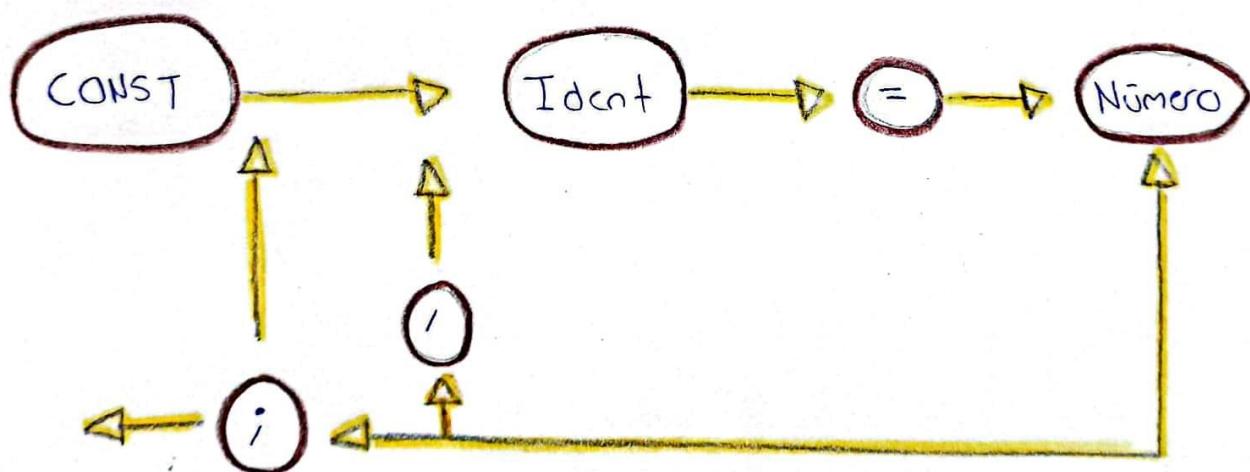


Figura 10- Declaración de constantes

3.6.1- Errores Sintácticos : Definición y Tipos

Los errores sintácticos son violaciones de las reglas gramaticales de un lenguaje de programación. Estos errores ocurren cuando el código fuente no sigue la estructura sintáctica correcta. Algunos ejemplos comunes de errores sintácticos incluyen la falta de un punto y coma al final de una línea, la omisión de un paréntesis de cierre en una expresión, o la incorrecta colocación de llaves en bloques de código.

Los errores sintácticos se pueden dividir en dos categorías principales :

1.- Errores de emisión : Estos errores ocurren cuando falta un elemento necesario en el código, como un paréntesis o una llave de cierre.

2.- Errores de adición : Estos errores ocurren cuando se agrega un elemento no deseado en el código, como un carácter o un operador innecesario.

3.6.2 - Detección de Errores Sintácticos

La detección de errores sintácticos es una parte crucial del proceso de compilación y generalmente se realiza mediante el uso de analizadores sintácticos. Cuando se encuentra un error sintáctico, el analizador informa sobre la ubicación del error y la naturaleza del problema en el código fuente. Esto ayuda al programador a identificar y corregir el error.

3.6.3 - Reporte de Errores Sintácticos

El reporte de errores sintácticos debe ser claro y preciso para que el programador pueda entender y corregir el problema de manera eficiente. La información proporcionada generalmente incluye:

- * La ubicación exacta del error (línea y columna).
- * Una descripción del tipo de error.
- * Una indicación visual, como una flecha o resaltado, que señala el lugar exacto del error en el código fuente.

3.6.4- Recuperación de Errores Sintácticos

La recuperación de errores sintácticos se refiere a la estrategia utilizada para continuar el proceso de análisis sintáctico después de detectar un error: Hoy varias técnicas de recuperación de errores sintácticos, que incluyen:

- 1- Inserción y Eliminación de Símbolos: En algunos casos, el analizador puede intentar insertar o eliminar símbolos en el código para corregir el error.
- 2- Reemplazo de símbolos: Otra técnica es reemplazar un símbolo incorrecto por uno correcto.
- 3- Saltar tokens: El analizador puede optar por omitir ciertos tokens para continuar el análisis.

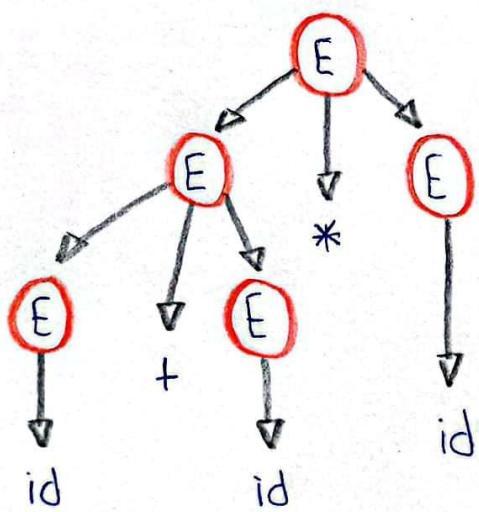


Figura 11- Errores sintácticos

3.6.5 - Importancia del Manejo de Errores Sintácticos

El manejo adecuado de errores sintácticos es crucial para la experiencia del programador y la construcción de compiladores robustos. Proporciona información útil para corregir errores y permite que los programadores que comprenden dónde y por qué ocurrieron los problemas. Además, el diseño de un sistema de recuperación de errores bien pensado puede evitar que un solo error sintáctico cause la interrupción de todo el proceso de compilación.

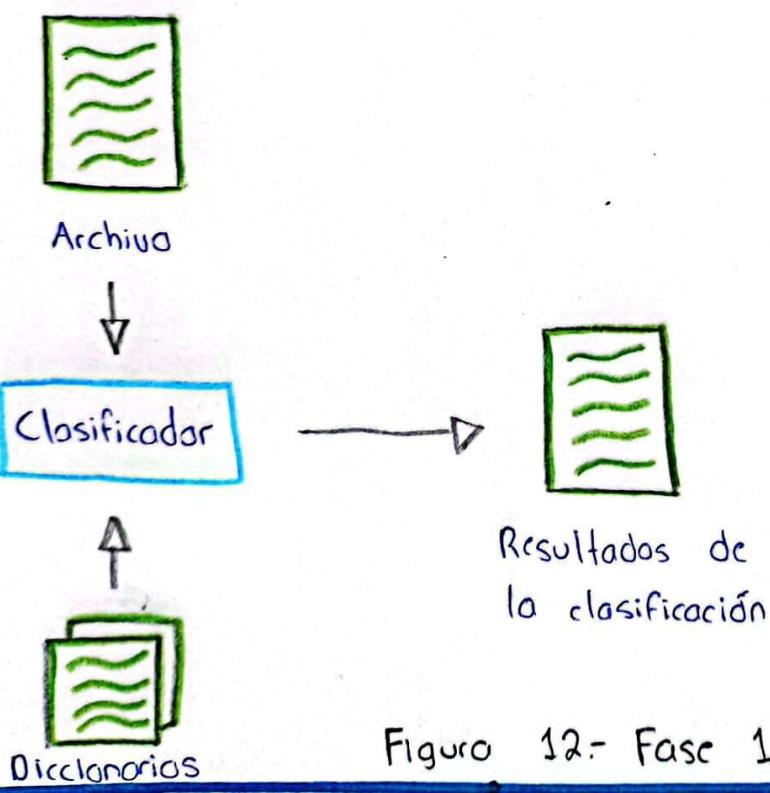


Figura 12- Fase 1



Reglas gramaticales
y léxicas



Detector automático
de errores



Resultados: errores
detectados y marcados

Figura 13- Fase 2

Tema 3.7- Generadores de
código para analizadores
sintácticos : Yacc, Bison
(repaso)

En el desarrollo de compiladores y procesadores de un lenguaje, la creación de analizadores sintácticos es una tarea crucial. Estos analizadores desempeñan un papel muy esencial al verificar si un programa escrito en cualquier lenguaje de programación cumple con la estructura sintáctica correcta.

3.7.1 - Generadores de código para Analizadores Sintácticos

Los generadores de código son herramientas que ayudan a los desarrolladores a crear analizadores sintácticos a partir de especificaciones de gramáticas formales. Estas herramientas generan código en lenguajes de la programación, como C o C++, que implementa el analizador sintático de acuerdo con la gramática especificada. Dos de las herramientas más conocidas y ampliamente utilizadas para este propósito son YACC y BISON.

3.7.2 - YACC (Yet Another Compiler Compiler)

1- Definición y Características : YACC es una herramienta que permite generar analizadores sintácticos basados en gramáticas LALR (Look-Ahead LR). Utiliza una notación específica para describir la gramática del lenguaje y produce un programa en C o C++ que puede reconocer y analizar el código fuente según la estructura definida en la gramática.

2- Gramáticas BNF : YACC utiliza una notación llamada

Backus-Naur Form (BNF) para definir la gramática del lenguaje. La especificación gramatical se describe en un archivo de entrada y se utiliza para generar el código del analizador sintáctico.

3.- Generación de Árboles de análisis : Uno de los resultados típicos de un analizador YACC es un árbol de análisis que representa la estructura jerárquica del código fuente según la gramática.

Especificación
YACC



Función de análisis
sintáctico

y_TAB.C (BYACC)

y.tab.c (Unix)

Figura 14: Acerca
de YACC

3.7.3 - BISON

1.- Definición y relación con YACC: BISON es un derivado de YACC y mantiene una compatibilidad significativa con YACC. Ofrece funcionalidades similares, pero a menudo se elige debido a su disponibilidad en sistemas Unix y su licencia de código abierto.

2.- Ventajas sobre YACC: BISON es preferido por algunos desarrolladores debido a su flexibilidad y capacidad para generar analizadores sintácticos más eficientes en términos de memoria y velocidad de análisis.

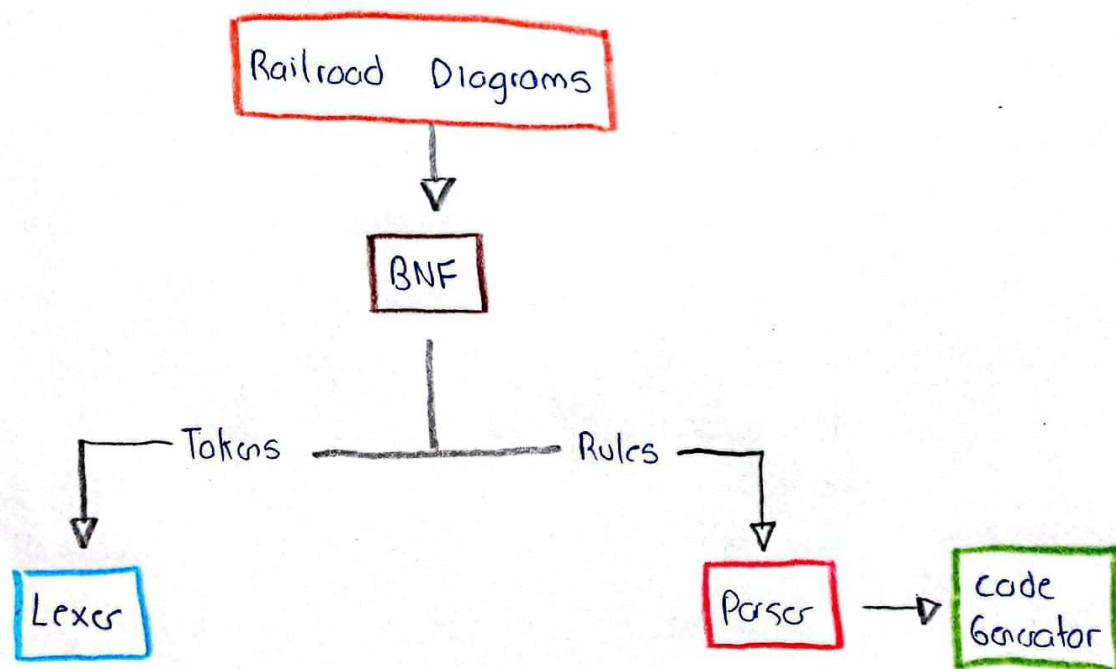


Figura 15 - Arco de BISON

Conclusión

Los temas relacionados con la construcción de los compiladores y análisis sintáctico son cruciales para el desarrollo de lenguajes de programación y de las herramientas de software. Estos temas abordan aspectos fundamentales que van desde la definición de los gramáticos y la precedencia de operadores hasta la implementación de analizadores sintácticos y la generación de código.

En conjunto, estos temas proporcionan los fundamentos que son necesarios para diseñar y desarrollar compiladores, intérpretes y otras herramientas relacionadas con el procesamiento de lenguajes de programación. La elección de enfoques de herramientas dependerá de las necesidades específicas del proyecto, la complejidad del lenguaje y los objetivos de desarrollo de software. La comprensión de estos temas es esencial para aquellos involucrados en la creación y mejora de lenguajes de programación y sus entornos de desarrollo.

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4**

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

PRACTICA No.	NOMBRE DE LA PRACTICA	FECHA DE ENTREGA
4	Análisis Sintáctico	11 de octubre, 2023

1	INTRODUCCION
<p>El análisis sintáctico es una etapa fundamental en el proceso de compilación e interpretación de lenguajes de programación. Su propósito radica en la verificación y comprensión de la estructura gramatical de un programa fuente. Que, en otras palabras, busca responder a la pregunta crucial de si un código cumple con las reglas sintácticas del lenguaje en el que está escrito.</p> <p>Este proceso desempeña un papel esencial en el ámbito del desarrollo de software, ya que es la base para garantizar la coherencia y corrección de los programas antes de que sean ejecutados o traducidos a código ejecutable. Y Para llevar a cabo esta tarea, se utilizan técnicas y enfoques específicos, siendo dos de los más comunes los analizadores descendentes y los analizadores ascendentes.</p> <p>Dentro de esta práctica correspondiente a la actividad 5, exploraremos en detalle estos dos enfoques en particular. Y desglosaremos los objetivos y las funciones esenciales de los analizadores descendente y ascendente, proporcionando una base sólida para comprender cómo se verifica y estructura la sintaxis en el proceso de desarrollo de software.</p>	

2	OBJETIVO
<ol style="list-style-type: none">1. Comprender los conceptos fundamentales del análisis sintáctico en la compilación e interpretación de programas.2. Explorar cómo funcionan los analizadores descendentes y ascendentes en la etapa de análisis sintáctico.3. Analizar los objetivos y funciones de un analizador sintáctico en un proceso de compilación o interpretación.	

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

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4****4****MARCO TEÓRICO****Análisis Sintáctico**

El análisis sintáctico es la etapa del proceso de compilación o interpretación que se encarga de verificar la estructura gramatical de un programa fuente.

Objetivos:

- Verificar que el programa cumple con las reglas de sintaxis del lenguaje.
- Construir una estructura de datos que represente la jerarquía gramatical del programa.

Funciones:

- Escanear y tokenizar el programa fuente.
- Aplicar reglas gramaticales para construir una estructura de árbol de sintaxis abstracta.
- Detectar errores de sintaxis y proporcionar mensajes de error claros.

Analizador Descendente (Parser Descendente)

Un analizador descendente es un tipo de analizador sintáctico que comienza desde el símbolo de inicio y desciende en el árbol de sintaxis abstracta siguiendo las reglas gramaticales.

• Funcionamiento:

- Utiliza una pila (stack) para rastrear las reglas gramaticales que se deben aplicar.
- Compara los tokens de entrada con las reglas gramaticales en un proceso de "emparejamiento descendente".
- Construye el árbol de sintaxis abstracta a medida que avanza.

Analizador Ascendente (Parser Ascendente)

Un analizador ascendente es otro tipo de analizador sintáctico que comienza desde los tokens de entrada y "asciende" en el árbol de sintaxis abstracta hasta llegar al símbolo de inicio.

• Funcionamiento:

- Utiliza una pila para rastrear los tokens de entrada y los símbolos gramaticales.
- Aplica reglas gramaticales de "reducción" para combinar elementos en un nodo superior en el árbol.
- El resultado es un árbol de sintaxis abstracta construido desde abajo hacia arriba

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4****5****DESARROLLO**

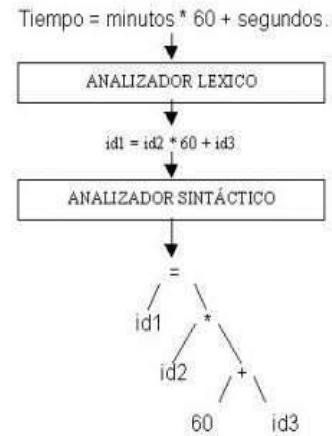
Con el marco teórico anterior dado, dando una pincelada o términos generales de como es el comportamiento de los analizadores sintácticos tanto descendente como ascendente, ahora toca profundizar:

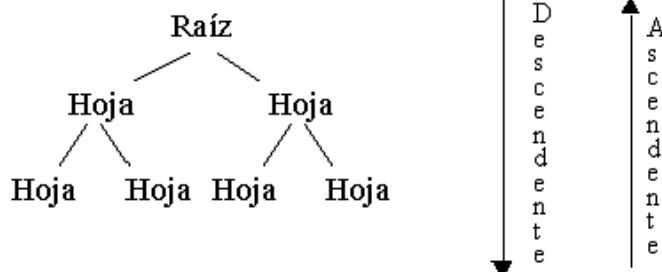
RECORDANDO QUE ES UN ANALIZADOR SINTACTICO

Es una herramienta esencial que desempeña un papel crítico en el análisis y comprensión de la estructura gramatical de la elaboración de un lenguaje o software. Su función principal es asegurar que el código fuente se ajuste a las reglas sintácticas establecidas por el lenguaje en el que está escrito.

Y para verificar lo anterior se deriva un proceso de análisis estructurado del cual están:

- **Tokenización:** El primer paso en el análisis sintáctico es dividir el código fuente en unidades más pequeñas llamadas "tokens".
- **Construcción de Gramática:** Para cada lenguaje, se define una gramática formal que establece las reglas sintácticas. Y se compone de reglas de producción que describen cómo se deben combinar los tokens para formar estructuras válidas en el lenguaje. Estas reglas de producción definen la jerarquía y la secuencia permitida de elementos.
- **Análisis Descendente o Ascendente:** Dado el analizador sintáctico utiliza la gramática definida para analizar los tokens.
Hay dos enfoques comunes para realizar este análisis: analizadores descendentes y analizadores ascendentes.
En ambos casos, el objetivo es emparejar los tokens con las reglas de producción de la gramática para construir una estructura de árbol que refleje la jerarquía sintáctica del programa.
 - **Analizadores Descendentes:** Comienzan con el símbolo inicial de la gramática y descienden en el árbol de análisis siguiendo las reglas de producción. Intentan emparejar los tokens en orden descendente desde la raíz hasta las hojas del árbol.
 - **Analizadores Ascendentes:** Comienzan con los tokens de entrada y "ascienden" en el árbol de análisis, utilizando las reglas de producción para combinar tokens en estructuras más grandes a medida que avanzan. El objetivo es llegar a la raíz del árbol.
- **Construcción del Árbol de Análisis Sintáctico:** A medida que se encuentran coincidencias entre los tokens y las reglas de producción, el analizador sintáctico construye un árbol de análisis sintáctico o un árbol de sintaxis abstracta. Cada nodo en el árbol representa una construcción sintáctica válida en el programa.



TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4****¿CÓMO FUNCIONA UN ANALIZADOR DESCENDENTE?**

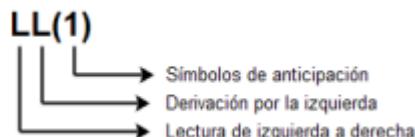
Un analizador descendente, también conocido como "parser descendente" y pueden ser conceptualizados como una forma de descubrir una derivación que avanza hacia la izquierda para una cadena de entrada.

Y en el contexto de la generación del árbol sintáctico, este tipo de analizador lo construyen desde arriba (raíz) hacia abajo (hojas).

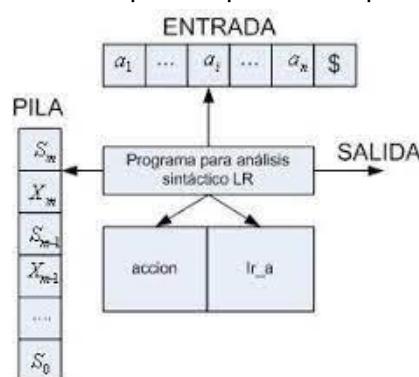
Y dentro este rubro existen 2 vertientes:

- **Análisis sintáctico descendente LL(1):** es una variante de analizador es una variante especializada de los analizadores descendentes que se caracteriza por su capacidad de mirar adelante en la entrada de manera limitada (uno o más tokens) para tomar decisiones de análisis.

La notación "LL" significa que el análisis se realiza de izquierda a derecha y que se genera una derivación izquierda en la gramática. El "(1)" indica que se mira solo un token adelante para tomar decisiones.



Los analizadores LL(1) se basan en tablas de análisis que indican las acciones a tomar en función del token actual y el símbolo en la parte superior de la pila de análisis.



TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4****Ejercicio**

- 1) Gramática que define expresiones aritméticas (sumas y productos)

La gramática sería de forma

Gramática

- E (Expresión aritmética)
- T (Término, puede ser un producto de factores)
- F (Factor, expresión encerrada en paréntesis con id)

Reglas de producción

1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

Entrada de expresión aritmética **id + id * (id)**.

Pasos:

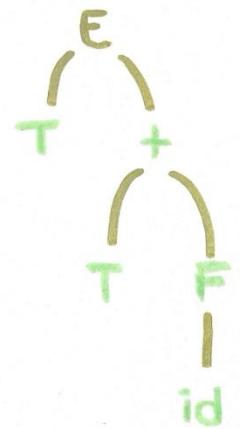
1. Inicializamos una pila de análisis y una tabla de análisis LL(1) que relaciona los tokens y los símbolos no terminales con las reglas gramaticales. La tabla de análisis nos indica qué regla aplicar en función del token actual y el símbolo en la parte superior de la pila.
2. Comenzamos con el símbolo de inicio E en la pila y el primer token **id** en la entrada.
3. Consultamos la tabla de análisis para saber qué regla aplicar. En este caso, la tabla indica que, dado el símbolo E en la pila y el token **id** en la entrada, debemos aplicar la regla E → T. Entonces, reemplazamos E en la pila con T.
4. Continuamos procesando la entrada y la pila de manera similar, siguiendo las reglas en la tabla de análisis y avanzando a través de los tokens.
5. Construimos un árbol de análisis sintáctico que refleja la estructura de la expresión. Al final del análisis, habremos construido un árbol que representa la jerarquía de la expresión aritmética **id + id * (id)**.

Pila	Entrada	Token	Regla a aplicar
E	id	id	E → T
T	id	id	T → F
F	id	id	F → id
F	id	+	E → T
T	id	+	E → E + T
E	+	+	E → E + T
T	+	+	E → E + T
T	+	id	T → T * F
F	+	id	F → id
F	+	(F → (E)
)	+	id	E → T

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4**

T	+	id	T -> F
F	+	id	F -> id
F	+	\$	aceptacion

El árbol de análisis sintáctico :



- **Análisis sintáctico por descenso recursivo (o con retroceso):** es un tipo de analizador que opera mediante una técnica recursiva en la que las reglas gramaticales se traducen directamente en funciones o procedimientos. Cada regla gramatical en la gramática del lenguaje se asocia con una función de análisis sintáctico.

Este enfoque es poderoso y flexible, ya que puede manejar gramáticas más complejas. Sin embargo, también puede ser menos eficiente en términos de memoria y tiempo de ejecución en comparación con otros métodos.

Ejercicio

- 2) Gramática simple para expresiones aritméticas (sumas y productos)

La gramática sería de forma

Gramática

- E (Expresión aritmética)
- T (Término, puede ser un producto de factores)
- F (Factor, expresión encerrada en paréntesis o un número)

Reglas de producción

1. E -> E + T
2. E -> T



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4

3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow \text{num}$

Entrada de expresión aritmética $2 + 3 * (4 - 1)$.

```
def expresion():      # Regla E
    termino()
    while token_actual == '+':
        emparejar('+')
        termino()

def termino():         # Regla T
    factor()
    while token_actual == '*':
        emparejar('*')
        factor()

def factor():          # Regla F
    if token_actual == '(':
        emparejar('(')
        expression()
        emparejar(')')
    else:
        emparejar('num')

def emparejar(terminal):
    if token_actual == terminal:
        obtener_siguiente_token()
    else:
        error()

def obtener_siguiente_token():
    # Obtener el siguiente token del flujo de entrada
    # y actualizar token_actual

def error():
    # Manejar errores sintácticos

# Iniciar el análisis sintáctico
token_actual = obtener_siguiente_token()
expresion()
if token_actual == '$':
    print("Expresión sintácticamente correcta.")
else:
    error()
```

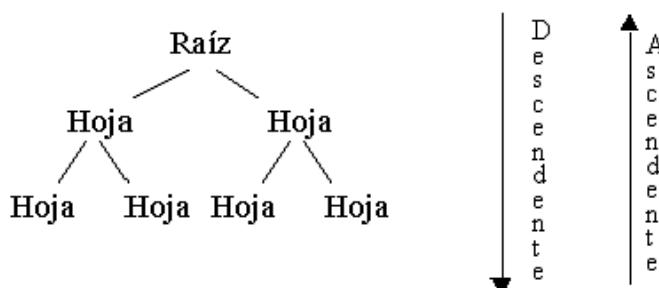
TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4**

Las características o proceso se resumen en esta terminación de conceptos:

- + **Escaneo léxico:** El proceso comienza con un analizador léxico, que divide el programa fuente en unidades más pequeñas llamadas "tokens". Cada token representa una palabra clave, un identificador, un operador, un número o algún otro elemento léxico del lenguaje de programación.
- + **Pila (stack):** Utiliza una estructura de datos llamada "pila" (stack) para llevar un registro de las reglas gramaticales que se deben aplicar y los símbolos que se esperan en la entrada.
- + **Comparación:** El analizador comienza desde el símbolo de inicio de la gramática y compara los tokens de entrada con las reglas gramaticales en un proceso de "emparejamiento descendente". Busca reglas que coincidan con los tokens actuales y las aplica.
- + **Construcción del árbol de análisis sintáctico:** A medida que se aplican las reglas gramaticales, el analizador construye un árbol de análisis sintáctico. Cada nodo en el árbol representa una construcción sintáctica válida en el programa fuente. El árbol se va desarrollando de arriba hacia abajo, siguiendo las reglas gramaticales.
- + **Manejo de errores:** Si el analizador encuentra un token que no coincide con las reglas gramaticales, puede generar un error de sintaxis. En algunos casos, intenta recuperarse de manera elegante para continuar analizando el programa, buscando una construcción válida.
- + **Finalización:** Una vez que el analizador ha recorrido todo el programa fuente y ha construido un árbol de análisis sintáctico válido, se considera que el análisis sintáctico ha terminado con éxito.

¿CÓMO FUNCIONA UN ANALIZADOR ASCENDENTE ?

Un analizador descendente, también conocido como "parser ascendente" y a diferencia de los analizadores descendentes, que comienzan desde el símbolo de inicio de la gramática y descienden en el árbol de análisis, los analizadores ascendentes operan desde los tokens de entrada y "ascienden" a través del árbol de análisis sintáctico.



Y dentro este rubro existen diversas variantes de analizadores, descritos de la siguiente manera :



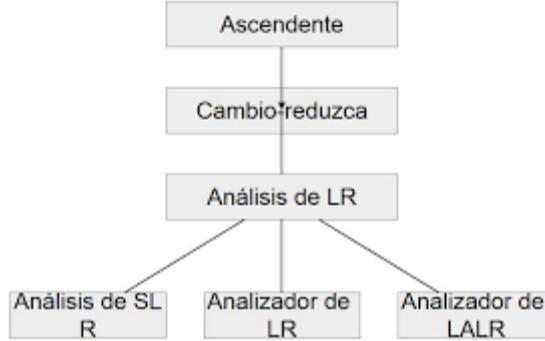
TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4



- **Ascendentes SL-LR(k):** Estos analizadores utilizan una combinación de análisis ascendente y descendente, lo que los hace adecuados para una variedad de gramáticas, incluyendo algunas que no son LR(k) ni LL(k).

Este analizador se ha desarrollado para abordar situaciones en las que las técnicas puras de análisis ascendente o descendente podrían resultar insuficientes o inefficientes.

Este tipo de analizador tiene como características la:

- Combinación de enfoques
- Versatilidad en la gramática
- Necesidad de conocimiento avanzado
- Balance entre potencia y eficiencia

- **Ascendentes LR(1):** Son notables por su capacidad para manejar gramáticas más complejas y ambiguas en la construcción de analizadores sintácticos, ya que pueden mirar más allá de un token y tomar decisiones basadas en más contexto.

Y sus características son:

- Potencia de Análisis
- Identificación de Reducciones
- Manejo de Gramáticas Complejas
- Requisitos de Implementación
- Eficiencia y Recursos

- **LALR(1):** Este analizador (Look-Ahead LR) es una variante de los analizadores LR(1) que utilizan información más compacta para reducir el espacio requerido en las tablas de análisis. Son más eficientes en términos de memoria que los analizadores LR(1), aunque pueden perder un poco de poder expresivo.

Se destacan por su capacidad para lograr un equilibrio entre la potencia de análisis y la eficiencia en el uso de recursos.

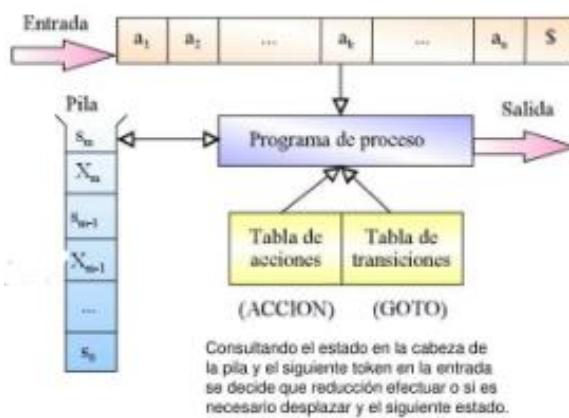
Donde sus características principales abarcan:

- Eficiencia en el Uso de Memoria
- Derivados de los Analizadores LR(1)
- Pérdida de Poder Expresivo Limitada

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II****AUTOR: EQUIPO 4**

- Implementación Práctica
- Contexto de Uso

Estructura general de un analizador LR

**Ejercicio**

- 3) Construir un compilador para un lenguaje de programación simple
Code4 es el lenguaje y se desea analizar y comprender programas escritos en este lenguaje

Programa en Code4 que calcula la **suma de 2 números**.

```
PROGRAM Suma;
VAR
    numero1, numero2, resultado: INTEGER;
BEGIN
    numero1 := 10;
    numero2 := 20;
    resultado := numero1 + numero2;
END.
```

El proceso a seguir para el análisis sería el siguiente:

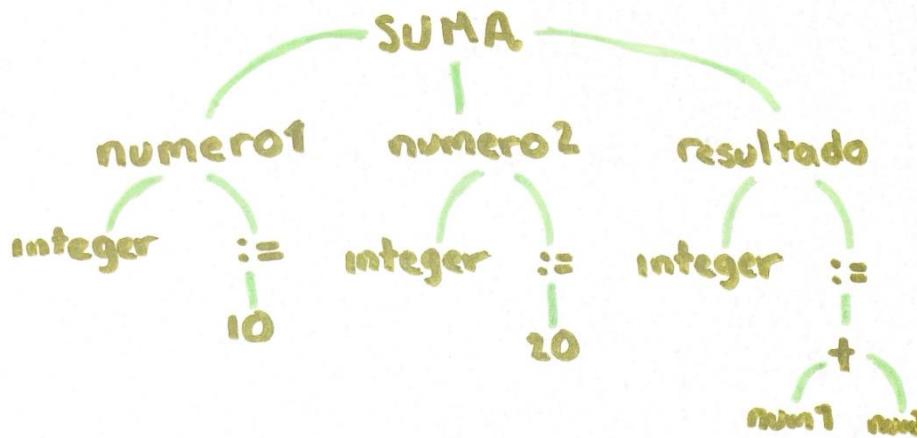
- **Escaneo Léxico:** Primero, un analizador léxico dividiría el código en tokens, identificando palabras clave como "PROGRAM", "VAR", "BEGIN", etc., y números, identificadores, operadores y otros elementos léxicos.
- **Construcción del Árbol de Análisis Sintáctico:** El analizador sintáctico ascendente construiría un árbol de análisis sintáctico ascendente a medida que procesa los tokens. Comienza con la raíz y agrega nodos a medida que identifica estructuras gramaticales, como declaraciones de



variables, asignaciones, bloques de código, etc.

- ⊕ **Identificación de Reglas Gramaticales:** El analizador ascendente identificaría las reglas gramaticales que rigen la estructura del programa. Por ejemplo, reconocería la regla PROGRAM -> VAR BEGIN para el comienzo del programa, y las reglas VAR -> IDENTIFIER : TYPE para las declaraciones de variables.
- ⊕ **Manejo de la Jerarquía Sintáctica:** El analizador ascendente manejaría la jerarquía sintáctica al identificar los bloques de código (como BEGIN ... END) y las estructuras de control (si las hubiera) en el programa. Construiría el árbol de análisis de manera que refleje la estructura del programa de manera precisa.
- ⊕ **Validación y Generación de Código:** A medida que el analizador sintáctico ascendente avanza, también podría realizar comprobaciones semánticas y generar código intermedio o código de máquina, dependiendo de la fase del compilador en la que se encuentre.

El árbol de análisis sintáctico:



¿CUÁLES SON LOS OBJETIVOS Y LAS FUNCIONES DE UN ANALIZADOR SINTÁCTICO ?

Una vez investigado todo lo anterior se deduce que su propósito u objetivo radica en la verificación y comprensión de la estructura gramatical de un programa fuente, respondiendo a la pregunta crucial de si un código cumple con las reglas sintácticas del lenguaje en el que está escrito.

Con las características mencionadas durante la práctica se pudo saber que los objetivos de este proceso de un análisis sintáctico se deriva en varias fases o componentes:

1. **Validación de la Sintaxis:** El objetivo principal del análisis sintáctico es validar si el programa fuente sigue las reglas sintácticas del lenguaje. Esto implica asegurarse de que las construcciones del programa estén correctamente organizadas y que no haya errores en la estructura.
2. **Construcción del Árbol de Análisis:** El analizador sintáctico debe construir una estructura de datos llamada árbol de análisis sintáctico que representa la jerarquía de las construcciones sintácticas en el programa. Este árbol es esencial para comprender la estructura y la lógica del programa.

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II****AUTOR: EQUIPO 4**

3. **Detección de Errores Sintácticos:** Si se encuentran errores sintácticos en el programa, el analizador debe ser capaz de identificar y notificar estos errores al programador. Esto garantiza que los desarrolladores puedan corregir problemas en una etapa temprana del desarrollo.

Y sus funciones clave son las siguientes:

- ✓ **Escaneo Léxico:** El analizador sintáctico trabaja en conjunto con el analizador léxico para dividir el código fuente en una secuencia de tokens. Esto implica la identificación de palabras clave, identificadores, operadores y otros elementos léxicos.
- ✓ **Análisis Descendente o Ascendente:** Dependiendo de la estrategia utilizada (analizadores descendentes o ascendentes), el analizador sintáctico sigue reglas gramaticales predefinidas para analizar la estructura del programa.
- ✓ **Construcción del Árbol Sintáctico:** Durante el análisis, el analizador sintáctico construye el árbol de análisis sintáctico que refleja la estructura jerárquica del programa. Esto implica la creación de nodos para representar constructores sintácticos como declaraciones, expresiones, y más.
- ✓ **Manejo de Ambigüedades:** En ocasiones, las gramáticas pueden ser ambiguas, y el analizador sintáctico debe resolver estas ambigüedades para interpretar el código de manera coherente.
- ✓ **Generación de Mensajes de Error:** Cuando se encuentra un error sintáctico, el analizador sintáctico genera mensajes de error que describen la naturaleza del problema y su ubicación en el código. Esto facilita la corrección por parte del programador.
- ✓ **Interfaz con el Analizador Semántico:** En muchos casos, el analizador sintáctico se comunica con el analizador semántico para realizar verificaciones adicionales y construir una representación más completa del programa.

6

BITÁCORA DE INCIDENCIAS

Fecha	Problema encontrado	Solución
03/10/2023	Mientras un compañero hacia investigación y recabado de información, se crasheó su computadora. Como ya no daba imagen ni prendía, se revisó y se concluyó que la batería había llegado a su fin.	Comprar otra batería del mismo modelo que la laptop.
09/10/2023	Se desconocía los tipos de analizador sintáctico ascendentes y su proceso de análisis.	Investigar y realizar ejercicios prácticos.

7

OBSERVACIONES

Se pudo contemplar que el proceso del análisis sintáctico es una fase crítica en el proceso de desarrollo de software y en la construcción de compiladores, pues garantiza que un programa fuente cumpla con las reglas sintácticas del lenguaje, lo que es fundamental para evitar errores y comportamientos inesperados en tiempo de ejecución.

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4****8****CONCLUSIÓN**

En el transcurso de esta práctica, hemos adentrado en el fascinante mundo del análisis sintáctico y todo lo que abarca, siendo así una etapa crítica en el proceso de desarrollo de software y compilación de lenguajes de programación. Hemos explorado en detalle los objetivos y las funciones fundamentales de los analizadores sintácticos, cuya misión principal es verificar y comprender la estructura gramatical de un programa fuente, asegurando que cumpla con las reglas sintácticas del lenguaje en el que está escrito.

Hemos destacado el papel esencial del análisis sintáctico como la base para garantizar la coherencia y corrección de los programas antes de su ejecución o traducción a código ejecutable, destacando su importancia en el desarrollo de software de alta calidad.

Y por último, hemos profundizado en dos enfoques comunes para llevar a cabo el análisis sintáctico: los analizadores descendentes y ascendentes, pues estos enfoques ofrecen diferentes perspectivas sobre cómo se puede abordar la tarea de verificar y comprender la estructura sintáctica de un programa, por lo que enriquece nuestra comprensión de las técnicas involucradas.

9**REFERENCIAS**

- 3.5 Análisis sintáctico descendente. (s. f.).
http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/35_analisis_sintctico_descendente.html
http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/35_analisis_sintctico_descendente.html
- ANALIZADOR SINTÁCTICO ASCENDENTE y DESCENDENTE. (2010, 15 diciembre). Soraya Carrion Blog. <https://sycg.wordpress.com/2010/12/14/analizador-sintactico-ascendente-y-descendente/>
- colaboradores de Wikipedia. (2023a). Analizador Sintáctico LR. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Analizador_sint%C3%A1ctico_LR
- colaboradores de Wikipedia. (2023b). Analizador sintáctico. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Analizador_sint%C3%A1ctico
- EcuRed. (s. f.). Analizador Sintáctico descendente - ECURED.
https://www.ecured.cu/Analizador_sint%C3%A1ctico_descendente
- https://www.cartagenagg.com/recursos/alumnos/apuntes/ININF2_M4_U3_T4.pdf
- https://www.cartagenagg.com/recursos/alumnos/apuntes/ININF2_M4_U3_T2.pdf
- https://www.cartagenagg.com/recursos/alumnos/apuntes/ININF2_M4_U4_T1.pdf

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II****AUTOR: EQUIPO 4**

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

PRACTICA No.	NOMBRE DE LA PRACTICA	FECHA DE ENTREGA
5	Análisis Sintáctico	11 de octubre, 2023

1	INTRODUCCION
Yacc y Bison son dos herramientas relacionadas que se utilizan en la construcción de compiladores y analizadores sintácticos para procesar lenguajes de programación y otros lenguajes formales. Ambas herramientas son generadores de analizadores sintácticos que ayudan a automatizar la generación de código para analizar la estructura gramatical de un lenguaje. Ambas herramientas, Yacc y Bison, son fundamentales para el desarrollo de compiladores y analizadores sintácticos. Permiten a los desarrolladores definir y analizar la estructura sintáctica de lenguajes de programación y otros lenguajes formales de manera eficiente, lo que facilita la creación de herramientas de procesamiento de lenguaje.	

2	OBJETIVO
Crear un pequeño programa de análisis sintáctico con yacc y Bison para poder comprender su funcionamiento más a detalle.	

3	MATERIALES NECESARIOS
<ul style="list-style-type: none">• Conocimiento en programación y teoría de lenguajes• Documentación• Conjunto de reglas (gramática)• Compilador o interprete byacc• Software cygwin64• Sistema operativo Windows o Linux• Pruebas y depuración• Tiempo y esfuerzo• Colaboración	

4	MARCO TEÓRICO
<p>YACC, abreviatura de "Yet Another Compiler Compiler", es una herramienta que se utiliza para generar analizadores sintácticos o parsers. Su función principal es analizar un flujo de tokens, que se obtiene mediante el análisis léxico (generalmente con una herramienta como Lex), y construir una representación interna de la estructura de un programa de acuerdo con la gramática especificada. Estas representaciones internas, típicamente en forma de árboles sintácticos o estructuras de datos, son cruciales para la posterior generación de código.</p> <p>Este se caracteriza por:</p>	



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II

AUTOR: EQUIPO 4

Gramática Formal: YACC se basa en una gramática formal, que define las reglas sintácticas de un lenguaje. Esta gramática se describe mediante un conjunto de reglas de producción que especifican cómo se forman las sentencias válidas en el lenguaje.

Analizador Sintáctico: YACC genera un analizador sintáctico que se encarga de verificar si una secuencia de tokens (producidos por el analizador léxico) sigue las reglas gramaticales definidas en la gramática formal. El analizador sintáctico crea una estructura de datos que representa la estructura jerárquica del código fuente, a menudo en forma de un árbol sintáctico.

Acciones Semánticas: En YACC, se pueden asociar acciones semánticas a las reglas gramaticales. Estas acciones se ejecutan cuando se reconoce una regla, lo que permite realizar tareas como construir una tabla de símbolos, generar código intermedio o realizar optimizaciones.

Generación de Código: Una vez que se ha realizado el análisis sintáctico y se ha construido una representación interna del programa, YACC se utiliza típicamente para generar código de salida. Esto puede ser código de máquina, código intermedio, código fuente en otro lenguaje o cualquier otro formato necesario.

BISON es un generador de analizadores sintácticos, a menudo utilizado en la construcción de compiladores y analizadores léxicos. Es una herramienta que ayuda a generar analizadores sintácticos a partir de una especificación de gramática.

Bison utiliza una gramática formal para definir las reglas sintácticas de un lenguaje. Esta gramática se describe en un archivo de entrada que suele tener extensión .y.

La gramática en Bison sigue el formato de las gramáticas libres de contexto (CFG) y utiliza reglas de producción para definir la estructura sintáctica del lenguaje que se está analizando.

Relación con Lex o Flex:

Bison se usa en conjunto con un analizador léxico, que generalmente se genera utilizando herramientas como Lex o Flex.

Mientras que el analizador léxico se encarga de dividir el flujo de entrada en tokens, Bison se encarga de realizar el análisis sintáctico, construyendo la estructura jerárquica del programa.

Bison genera analizadores sintácticos LR(1), que son un tipo de analizadores descendentes (bottom-up) utilizados en la construcción de compiladores.

Estos analizadores son capaces de analizar gramáticas más amplias y resolver conflictos de manera eficiente.

En Bison, cada regla de producción puede estar asociada con una acción semántica que se ejecuta cuando se aplica la regla.

Estas acciones semánticas permiten realizar acciones específicas durante el análisis sintáctico, como construir un árbol de sintaxis abstracta (AST) o generar código.

Aquí como instalar YACC y BISON:

Necesitaremos un software que nos emule los comandos de Unix-linux, en este caso se utilizó cygwing64 lo podemos descargar de la siguiente página: <https://www.cygwin.com/>.



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II

AUTOR: EQUIPO 4

This is the home of the Cygwin project

What...

...is it?

Cygwin is:

- a large collection of GNU and Open Source tools which provide functionality similar to a [Linux distribution](#) on Windows.
- a DLL (cygwin1.dll) which provides substantial POSIX API functionality.

...isn't it?

Cygwin is not:

- a way to run native Linux apps on Windows. You must rebuild your application *from source* if you want it to run on Windows.
- a way to magically make native Windows apps aware of UNIX® functionality like signals, pts, etc. Again, you need to build your apps *from source* if you want to take advantage of Cygwin functionality.

Cygwin version

The most recent version of the Cygwin DLL is [3.4.9](#).

The Cygwin DLL currently works with all recent, commercially released x86_64 versions of Windows, starting with Windows 7. For more information see the [FAQ](#).

DEPRECATION NOTE

Cygwin 3.4 is the last major version supporting

- Windows 7
- Windows 8
- Windows Server 2008 R2
- Windows Server 2012

Installing Cygwin

Install Cygwin by running [setup-x86_64.exe](#)

Use the setup program to perform a [fresh install](#) or to [update](#) an existing installation.

Keep in mind that individual packages in the distribution are updated separately from the DLL so the Cygwin DLL version is not useful as a general Cygwin distribution release number.

Support for Cygwin

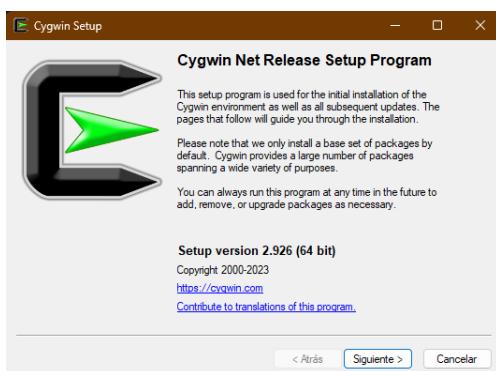
For all Cygwin-related questions, observations, suggestions and bug reports, please check the resources available at this site, such as the [FAQ](#), the [User's Guide](#) and the [mailing list archives](#). If you've exhausted these resources then please send email to the [appropriate mailing list](#).

Hacemos click en esta parte para descargar

Installing Cygwin

Install Cygwin by running [setup-x86_64.exe](#)

Una vez descargado comenzamos la instalación y ponemos siguiente.





TECNOLÓGICO
NACIONAL DE MÉXICO

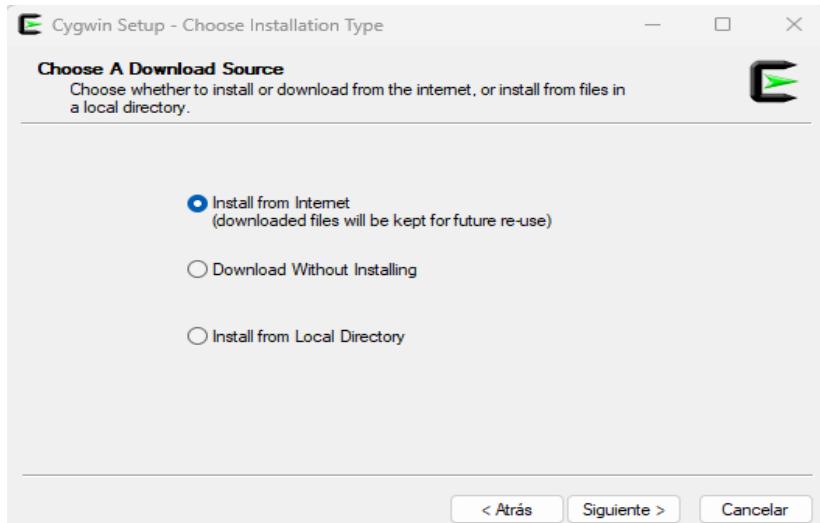
INSTITUTO TECNOLÓGICO DE CELAYA



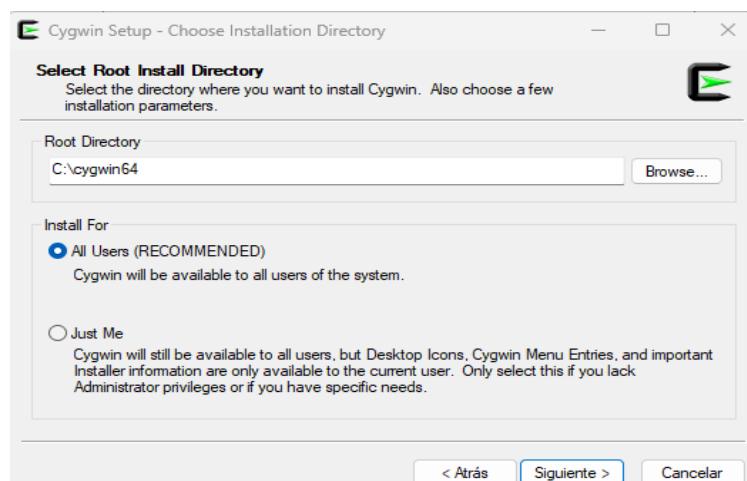
PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4

Ponemos que los paquetes los queremos instalar de internet.



Usamos todo lo recomendado.





TECNOLÓGICO
NACIONAL DE MÉXICO

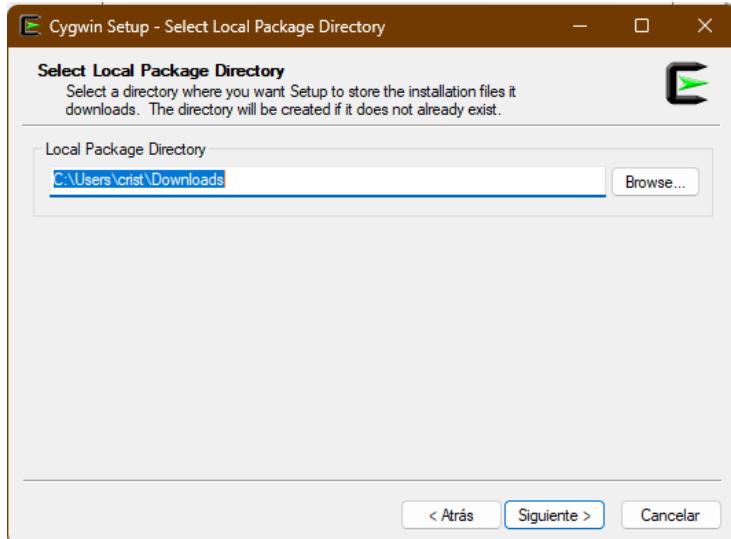
INSTITUTO TECNOLÓGICO DE CELAYA



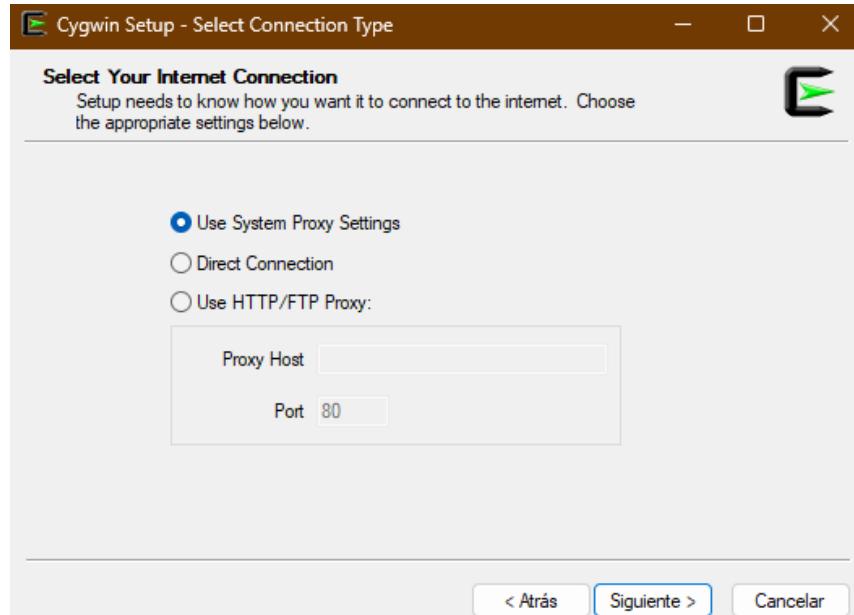
PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4

Ponemos nuestra dirección de instalación.



Seleccionamos por donde queremos conectarnos a internet, ponemos la primera.





TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4

Escogemos de donde queremos descargar todos los paquetes, lo recomendable es la primera opción.

The screenshot shows the 'Cygwin Setup - Choose Download Site(s)' window. The title bar has a green icon with a white arrow pointing right. The main area is titled 'Choose A Download Site' with the sub-instruction 'Choose a site from this list, or add your own sites to the list.' Below this is a list titled 'Available Download Sites:' containing several URLs. The first URL, 'https://mirrors.163.com', is highlighted with a blue selection bar. At the bottom of the list is a 'User URL:' input field and an 'Add' button. At the very bottom are three buttons: '< Atrás' (Back), 'Siguiente >' (Next), and 'Cancelar' (Cancel).

Available Download Sites:
https://mirrors.163.com
https://mirrors.aliyun.com
https://mirror.clientvps.com
https://cygwin.mirror.constant.com
https://polish-mirror.evolution-host.com
https://cygwin.mirrors.hoobly.com
https://mirrors.huaweicloud.com
http://mirror.team-cymru.com
https://mirrors.tencent.com
https://mirrors.xmission.com
https://mirror.clarkson.edu
http://www.gtlb.gatech.edu
https://mirrors.rit.edu

TECNOLÓGICO
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA

PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4

Aquí veremos lo que se nos instalara por default

Cygwin Setup - Select Packages						
Select Packages		Select packages to install.				
Package	Current	New	Src?	Categories	Size	Description
2048-cli	Skip			Games, Unmaintained	10k	2048 game for terminal
2048-cli-debuginfo	Skip			Debug, Unmaintained	29k	Debug info for 2048-cli
2048-qt	Skip			Games, Unmaintained	1.371k	2048 game for Qt/KDE
2048-qt-debuginfo	Skip			Debug, Unmaintained	2.257k	Debug info for 2048-qt
AMF	Skip			Devel	56k	Advanced Media Framework (AMF) SDK
AtomicParsley	Skip			Audio, Unmaintained	109k	Command-line program to read and set MPEG-4 tags compatible with iPod/iTunes
AtomicParsley-debuginfo	Skip			Debug, Unmaintained	281k	Debug info for AtomicParsley
CUnit	Skip			Lbs, Math	103k	CUnit is a Unit testing framework for C
CUnit-debuginfo	Skip			Devel	76k	Debinfo for CUnit
ELFIO	Skip			Devel, Lbs, System, Util	694k	ELF file reader and producer implemented as a C++ library
GConf2	Skip			GNOME, Unmaintained	310k	GNOME configuration database system
GConf2-debuginfo	Skip			Debug, Unmaintained	595k	Debug info for GConf2
GeoIP	Skip			Net, Unmaintained	24.604k	GeoIP Lite (free) databases
GeoIP-database	Skip			Net, Unmaintained	118k	Debug info for GeoIP
GeoIP-debuginfo	Skip			Debug, Unmaintained	930k	GraphicsMagick library
GraphicsMagick	Skip			Graphics	4.210k	Debug info for GraphicsMagick
GraphicsMagick-debuginfo	Skip			Graphics	105k	Image processing suite (utilities)
ImageMagick	Skip			Debug	7.034k	Debug info for ImageMagick
ImageMagick-debuginfo	Skip			Graphics	5.389k	Image processing suite (documentation)
ImageMagick-doc	Skip			GNOME, Unmaintained	84k	CORBA 2.4 Object Request Broker library
ORB12	Skip			Debug, Unmaintained	802k	Debug info for ORB12
ORB12-debuginfo	Skip			Graphics, Unmaintained	946k	Color management for computer animation
OpenColorIO	Skip			Debug, Unmaintained	4.647k	Debug info for OpenColorIO
OpenColorIO-debuginfo	Skip			Graphics, Unmaintained	746k	Color management for computer animation
OpenSP	Skip			Text, Unmaintained	314k	SGML parser utilities
OpenSP-debuginfo	Skip			Debug, Unmaintained	4.422k	Debug info for OpenSP
R	Skip			Math, Science	44.500k	R Statistical computing language
R-debuginfo	Skip			Debug	6.713k	Debug info for R
R_autorebase	Skip			Math	1k	Incremental autorebase for R modules
SDL-debuginfo	Skip			Debug, Unmaintained	512k	Debug info for SDL
SDL2-debuginfo	Skip			Debug, Unmaintained	2.409k	Debug info for SDL2
SDL2_image-debuginfo	Skip			Debug	253k	Debug info for SDL2_image
SDL2_mixer-debuginfo	Skip			Debug, Unmaintained	222k	Debug info for SDL2_mixer
SDL2_net-debuginfo	Skip			Debug, Unmaintained	34k	Debug info for SDL2_net
SDL2_ttf-debuginfo	Skip			Debug, Unmaintained	50k	Debug info for SDL2_ttf
SDL_Pango-debuginfo	Skip			Debug, Unmaintained	37k	Debug info for SDL_Pango
SDL_gfx-debuginfo	Skip			Debug, Unmaintained	154k	Debug info for SDL_gfx
SDL_image-debuginfo	Skip			Debug, Unmaintained	95k	Debug info for SDL_image
SDL_mixer-debuginfo	Skip			Debug, Unmaintained	228k	Debug info for SDL_mixer
SDL_net-debuginfo	Skip			Debug, Unmaintained	37k	Debug info for SDL_net
SDL_sound-debuginfo	Skip			Debug, Unmaintained	202k	Debug info for SDL_sound
SDL_ttf-debuginfo	Skip			Debug, Unmaintained	55k	Debug info for SDL_ttf
TeXmacs	Skip			Editors	30.473k	WYSIWYG editor for science and math
TeXmacs-debuginfo	Skip			Debug	40.739k	Debug info for TeXmacs
Thunar	Skip			Unmaintained, Xfce	1.125k	Xfce4 file manager
Thunar-debuginfo	Skip			Debug, Unmaintained	1.405k	Debug info for Thunar

Y Buscaremos byacc (una opción gratuita de yacc pero funciona de la misma forma) y elegiremos su ultima versión.

Package	Current	New	Src?	Categories	Size	Description
byacc	20220128-1	Keep		Devel, Unmaintained	95k	Berkeley Yacc
byacc-debuginfo	Skip			Uninstall	166k	Debug info for byacc
				Skip		
				20160606-1		
				20170430-1		
				Keep		
				Reinstall		

Buscaremos también gcc-core que será el que compile los archivos.

Cygwin Setup - Select Packages						
View Full		Search gcc-core	Clear			
Package	Current	New	Src?	Categories	Size	Description
djgpp-gcc-core	Skip			Devel, Unmaintained	7.926k	gcc for DJGPP toolchain (C)
gcc-core	11.4.0-1	Keep		Devel	30.795k	GNU Compiler Collection (C, OpenMP)
mingw64::gcc-core	Skip			Devel	28.203k	gcc for Win32 (i686-w64-mingw32) toolchain (C, OpenMP)
mingw64::x64_64-gcc-core	Skip			Devel	29.025k	gcc for Win64 toolchain (C, OpenMP)

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4**

Buscaremos BISON el cual de igual forma será en su ultima versión.

Select Packages
Select packages to install.

View	Full	Search Bison	Clear			
Package	Current	New	Src?	Categories	Size	De
bison	3.8.2-1	Keep	<input checked="" type="checkbox"/>	Devel	954k	G
bison-debuginfo		Skip	<input type="checkbox"/>	Uninstall	1,087k	D
			<input type="checkbox"/>	Skip		
			<input type="checkbox"/>	3.7.6-1		
			<input type="checkbox"/>	3.8.1-1		
			<input checked="" type="checkbox"/>	Keep		
			<input type="checkbox"/>	Reinstall		

Buscaremos Flex que será el que nos ayudará en la parte léxica para trabajar en conjunto con Yacc y BISON.

Select packages to install.

View	Full	Search flex	Clear			
Package	Current	New	Src?	Categories	Size	Description
flex	2.6.4-2	Keep	<input checked="" type="checkbox"/>	Devel	340k	A fast lexical analyzer generator
flex-debuginfo		Skip	<input type="checkbox"/>	Debug	253k	Debug info for flex
flexdll		Skip	<input type="checkbox"/>	Devel	291k	Creates DLLs with runtime symbol resolution

Buscaremos nano que será nuestro editor de texto para crear nuestros archivos.

Select packages to install.

View	Full	Search nano	Clear			
Package	Current	New	Src?	Categories	Size	Description
nano	4.9-1	Keep	<input checked="" type="checkbox"/>	Editors, Unmaintained	583k	Enhanced clone of Pico editor
nano-debuginfo		Skip	<input type="checkbox"/>	Debug, Unmaintained	628k	Debug info for nano



TECNOLÓGICO
NACIONAL DE MÉXICO

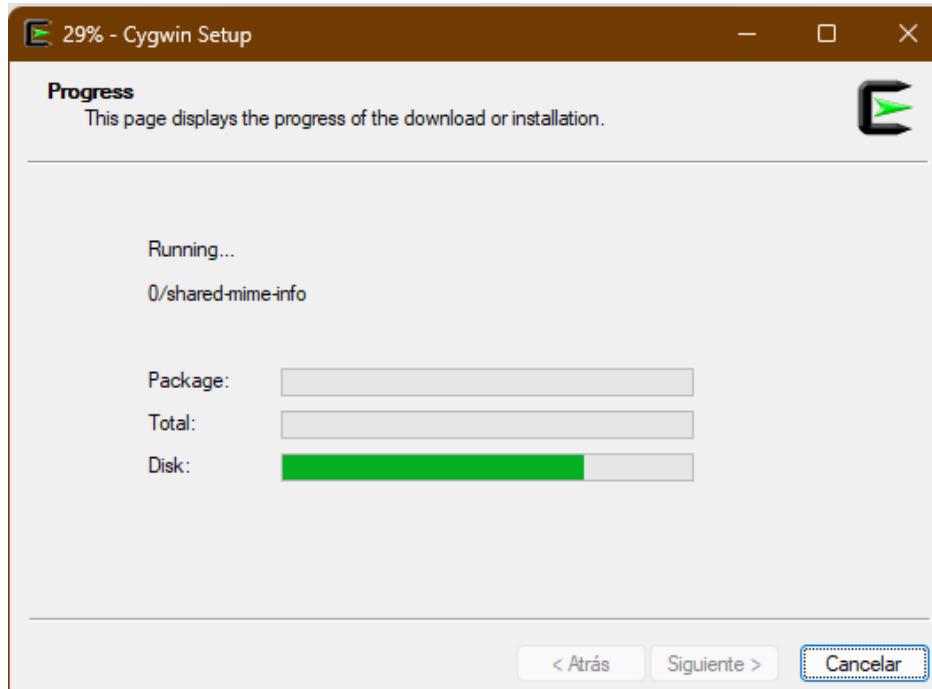
INSTITUTO TECNOLÓGICO DE CELAYA



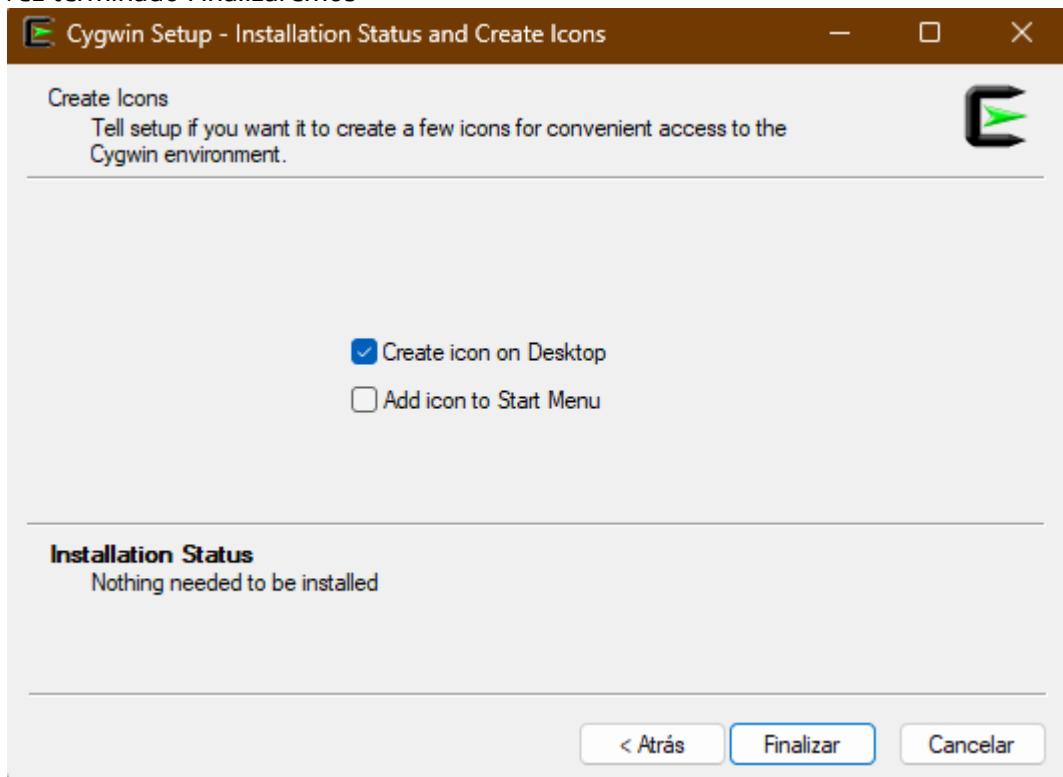
PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4

Una vez escogido todo damos en siguiente y se nos comenzara a descargar e instalar todos los componentes.



Una vez terminado Finalizaremos





TECNOLÓGICO
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II**

AUTOR: EQUIPO 4

Y Ya tendremos nuestro emulador de comandos.

The screenshot shows a terminal window with a dark background and light-colored text. At the top left is a green icon of a terminal window with a white 'E' and a tilde (~). To its right is the text 'crist@DESKTOP-3K8A2BV ~'. On the far right of the window are standard window control buttons for minimize, maximize, and close. The main body of the window is mostly blank, indicating no command has been entered yet.

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4**

5

DESARROLLO**¿CÓMO FUNCIONA EL GENERADOR DE CÓDIGO PARA ANALIZADORES SINTÁCTICOS YACC ?**

Como respuesta a la cuestión se empleará un ejercicio para su mejor comprensión, es una calculadora simple con la ayuda de yacc en donde se mostrarán los pasos que se siguieron para su elaboración.

Como primer paso debemos de hacer nuestro encabezado que indican al compilador qué bibliotecas y funciones externas se utilizan en el programa.

En este caso, se incluyen las bibliotecas estándar stdio.h y stdlib.h, ya que se necesitarán para las funciones printf y exit.

También se hace referencia a las funciones externas yylex() y yyerror(), que son proporcionadas por el analizador léxico que es Flex y por YACC, respectivamente.

```
% {  
#include <stdio.h>  
#include <stdlib.h>  
  
extern int yylex();  
void yyerror(char *msg);  
% }
```

Como siguiente paso la declaración de la Unión (%union). En esta sección, se define una unión llamada <f> que se utilizará para almacenar el valor numérico de los tokens.

Esto significa que los tokens marcados con <f> se considerarán valores de tipo float.

```
%union {  
    float f;  
}
```

Declaraciones de Tokens y Tipos (%token y %type). En esta parte, se especifica cómo se deben tratar los tokens en la gramática.

- %token <f> NUM indica que el token NUM se considera un número de punto flotante y se almacena en la unión <f>.
- %type <f> E T F especifica que los no terminales E, T y F se consideran valores de tipo float y se almacenarán en la unión <f>.

```
%token <f> NUM  
%type <f> E T F
```

En la siguiente sección debemos de generar nuestras reglas gramáticas con notación BNF. Cada regla define cómo se construyen las expresiones aritméticas y cómo se deben calcular.

Por ejemplo, la regla E : E '+' T indica que una expresión E puede ser una suma de E y T, y el valor se calcula sumando los valores de E y T.



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II**

AUTOR: EQUIPO 4

```
% %

S : E      {printf("%f\n", $1);}

;
E : E '+' T    {$$ = $1 + $3;}
| E '-' T    {$$ = $1 - $3;}
| T          {$$ = $1;}
;

T : T '*' F   {$$ = $1 * $3;}
| T '/' F   {$$ = $1 / $3;}
| F          {$$ = $1;}
;

F : '(' E ')'
| '-' F
| NUM        {$$ = $1;}
;

%
```

En el siguiente paso creamos la función de error, esta función se llama cuando YACC encuentra un error en la gramática o en los datos de entrada.

Imprime un mensaje de error en la salida estándar de error (stderr) y luego sale del programa con un código de error.

```
void yyerror(char *msg) {
    fprintf(stderr, "%s\n", msg);
    exit(1);
}
```

Por ultimo de nuestro archivo creamos la función main() que es la entrada principal del programa, llama a yyparse() para iniciar el análisis sintáctico de la entrada. Después de completar el análisis sintáctico, el programa termina con un código de salida cero si no se encontraron errores.

```
int main() {
    yyparse();
    return 0;
}
```

Una vez que tengamos nuestro archivo .y vamos a usar el comando byacc -d (nombre del archivo .y) Y nos creara y.tab.h que su función principal es proporcionar información y ayudar a la comunicación entre el analizador léxico y el analizador sintáctico, también nos creara y.tab.c que contendrá el código c que tendrá la función yyparse().

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4**

Ahora creamos nuestro archivo lex el cual se encarga de reconocer los tokens (componentes léxicos) en el código fuente de entrada y convertirlos en unidades reconocibles para el analizador sintáctico.

```
% {  
#include <stdio.h>  
#include <stdlib.h>  
#include "y.tab.h"  
% }  
  
%option noyywrap  
  
% %  
  
[0-9]+([.][0-9]+)?([eE][0-9]+)? { yyval.f = atof(yytext); return NUM; }  
[-+()*/] { return yytext[0]; }  
[ \t\f\v\n] { ; }  
  
% %
```

Incluimos "y.tab.h" es el archivo de encabezado generado por YACC y se utiliza para compartir información entre el analizador léxico y el analizador sintáctico.

%option noyywrap es una opción de configuración que indica a Lex que no debe generar la función yywrap(). Esta función a menudo se utiliza para habilitar el análisis de múltiples archivos de entrada, pero en este caso, se está deshabilitando, ya que la configuración del analizador se basa en un solo archivo de entrada.

Hacemos nuestras expresiones regulares que son para reconocer lo siguiente:

- [0-9]+([.][0-9]+)?([eE][0-9]+)?: Esta expresión regular reconoce números de punto flotante en notación decimal, que pueden incluir una parte decimal, una parte exponencial (opcional) y un exponente (opcional). Cuando se encuentra una coincidencia, el valor numérico se convierte a tipo float utilizando la función atof() y se almacena en yyval.f. Luego, se devuelve el token NUM.
- [-+()*/] Esta expresión regular reconoce los operadores aritméticos y los paréntesis. Cada uno de estos caracteres individuales se devuelve como un token.
- [\t\f\v\n]: Esta expresión regular reconoce caracteres de espacio en blanco, como espacios, tabuladores, retornos y saltos de línea. Se ignora cualquier espacio en blanco encontrado.

Cuando tengamos nuestro archivo lex usaremos el comando lex con nuestro nombre de archivo para producir el código fuente C para nuestro analizador léxico.



TECNOLÓGICO
NACIONAL DE MEXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4

```
crist@DESKTOP-3K8A2BV ~  
$ lex calc.1
```

lex.yy.c

Entonces por último tomaremos nuestro código fuente para nuestro analizador sintáctico y para nuestro lex y ejecutarlo a través del compilador para producir nuestra calculadora.

```
crist@DESKTOP-3K8A2BV ~  
$ cc lex.yy.c y.tab.c -o calc
```

Ejecutamos nuestra sencilla calculadora, poniendo nuestras operaciones aritméticas y ejecutando con Ctrl+d.

```
crist@DESKTOP-3K8A2BV ~  
$ ./calc  
22+56  
78.000000
```

```
crist@DESKTOP-3K8A2BV ~  
$ ./calc  
67+34*8  
339.000000
```

Manda excepciones cuando hay un error.

```
crist@DESKTOP-3K8A2BV ~  
$ ./calc  
acdf+56  
syntax error  
acdf
```

```
crist@DESKTOP-3K8A2BV ~  
$ ./calc  
136.56+89+/*+  
syntax error
```

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4****¿CÓMO FUNCIONA EL GENERADOR DE CÓDIGO PARA ANALIZADORES SINTÁCTICOS
BISON?**

El fuente de Bison se convierte en una función en C llamada yyparse. Aquí describimos las convenciones de interfaz de yyparse y las otras funciones que éste necesita usar. Se debe de tener en cuenta que el analizador utiliza muchos identificadores en C comenzando con 'yy' e 'YY' para propósito interno. Si utiliza tales identificadores (a parte de aquellos descritos en el manual) en una acción o en código C adicional en el archivo de la gramática, es probable que se encuentre con problemas. La Función del Analizador yyparse Se llama a la función yyparse para hacer que el análisis comience. Esta función lee tokens, ejecuta acciones, y por último retorna cuando se encuentre con el final del fichero o un error de sintaxis del que no puede recuperarse. Usted puede también escribir acciones que ordenen a yyparse retornar inmediatamente sin leer más allá. El valor devuelto por yyparse es 0 si el análisis tuvo éxito (el retorno se debe al final del fichero). El valor es 1 si el análisis falló (el retorno es debido a un error de sintaxis). La función del analizador léxico, yylex, reconoce tokens desde el flujo de entrada y se los devuelve al analizador. Bison no crea esta función automáticamente; se debe de escribir de manera que yyparse pueda llamarla. En programas simples, yylex se define a menudo al final del archivo de la gramática de Bison. En programas un poco más complejos, lo habitual es crear un programa en Flex que genere automáticamente esta función y enlazar Flex y Bison.

Aquí un ejemplo *paso a paso* de un pequeño analizador con Bison y Flex que reconoce Strings y números.

Como primer paso debemos de hacer nuestro encabezado que indican al compilador qué bibliotecas y funciones externas se utilizan en el programa.

En este caso, se incluyen las bibliotecas estándar stdio.h y stdlib.h, ya que se necesitarán para las funciones printf y exit.

También se hace referencia a las funciones externas yylex() y yyerror(), que son proporcionadas por el analizador léxico que es Flex y por YACC, respectivamente.

```
% {  
#include <stdio.h>  
#include <stdlib.h>  
  
extern int yylex();  
void yyerror(char *msg);  
% }
```

se definen los tokens que serán reconocidos por el analizador léxico. Se han definido tres tokens: STRING, NUM, OTHER, y SEMICOLON. Estos tokens representan elementos léxicos del lenguaje que se analizará.

```
%token STRING NUM OTHER SEMICOLON
```

se especifica el tipo de datos que se asocia a cada token. Por ejemplo, <name> se asocia al token STRING, y <number> se asocia al token NUM. Esto permite que Bison conozca el tipo de datos que debe esperar cuando se procesa un token.

```
%type <name> STRING  
%type <number> NUM
```

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II****AUTOR: EQUIPO 4**

Se define una unión que contiene dos miembros: name, que es un arreglo de caracteres de tamaño 20, y number, que es un entero. Esta unión se utiliza para almacenar el valor de los tokens STRING y NUM cuando se reconocen.

```
%union{
    char name[20];
    int number;
}
```

Se definen las reglas de producción que describen la estructura sintáctica del lenguaje que se está analizando.

La regla prog es el símbolo inicial de la gramática y solo contiene una referencia a stmts. La regla stmts define una lista de declaraciones separadas por SEMICOLON (pueden ser vacías).

La regla stmt reconoce los tokens STRING, NUM y OTHER, y realiza acciones de impresión en función del tipo de token reconocido.

```
% %

prog:
    stmts
    ;
stmts:
    | stmt SEMICOLON stmts

stmt:STRING      {printf("Haz ingresado el String - %s", $1);}
    | NUM        {printf("Haz ingresado el numero - %d", $1);}
    | OTHER
    ;
% %
```

La función yyerror se llama cuando se encuentra un error durante el análisis sintáctico y se utiliza para imprimir mensajes de error en la salida estándar de error (stderr).

```
void yyerror(char *msg)
{
    fprintf(stderr, "%s\n", msg);
    exit(1);
}
```

La función principal main inicia el análisis sintáctico llamando a yyparse(), que es una función generada automáticamente por Bison. Después de completar el análisis, el programa retorna 0 para indicar una finalización exitosa.

```
int main()
{
    yyparse();
    return 0;
}
```

Al tener nuestro archivo.y usamos el comando bison -d (nombre del archivo.y), si queremos que sea compatible con yacc debemos usar bison -dy (nombre del archivo.y). Nos creara (nombre del archivo.y).tab.h que su función principal es proporcionar información y ayudar a la comunicación entre el

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4**

análizador léxico y el analizador sintáctico, también nos creara (nombre del archivo .y).tab.c que contendrá el código c que tendrá la función yyparse().

```
crist@DESKTOP-3K8A2BV ~/análisisM
$ bison -d test.y
```

```
test.tab.c  test.tab.h
```

Ahora creamos nuestro archivo .l que contendrá nuestro análisis léxico

```
% {

#include <stdio.h>
#include <string.h>
#include "test.tab.h"
void showError();
% }

numbers  ([0-9])+
alpha    ([a-zA-Z])+
%%

{alpha}     {sscanf(yytext, "%s", yyval.name); return (STRING);}
{numbers}   {yyval.number = atoi(yytext); return (NUM);}
";"        {return (SEMICOLON);}
.          {showError(); return(OTHER);}

%%

void showError(){
    printf("Otra entrada no reconocida");
}
int yywrap(){
    return 1;
}
```

- Sección de Encabezado (%{ ... %}):

En esta sección, se incluyen las bibliotecas necesarias mediante #include.

Se declara una función showError() que se utilizará para mostrar mensajes de error cuando se encuentre una entrada no reconocida.

Se incluye "test.tab.h", que es el archivo de encabezado generado por Bison y se utiliza para compartir información entre el analizador léxico y el analizador sintáctico.

Definición de Expresiones Regulares (numbers y alpha):

Se definen dos expresiones regulares: numbers para reconocer secuencias de dígitos y alpha para reconocer letras mayúsculas o minúsculas. Estas expresiones regulares se utilizan más adelante en las reglas léxicas.



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II

AUTOR: EQUIPO 4

- Sección de Reglas Léxicas (% ... %):

En esta sección, se definen las reglas léxicas que describen cómo se reconocen y procesan los tokens en el flujo de entrada.

Reglas Léxicas ({alpha}, {numbers}, ";", .):

Cada regla léxica consta de una expresión regular y una acción asociada. Cuando se encuentra una coincidencia entre el patrón y el texto de entrada, se ejecuta la acción correspondiente.

La primera regla {alpha} reconoce secuencias de letras y utiliza sscanf para copiar el texto coincidente (yytext) en yyval.name. Luego, devuelve el token STRING al analizador sintáctico.

La segunda regla {numbers} reconoce secuencias de dígitos y utiliza atoi para convertirlas en un número entero. El número se almacena en yyval.number, y se devuelve el token NUM.

La tercera regla reconoce un punto y coma ";" y devuelve el token SEMICOLON.

La última regla . es una regla de comodín que se ejecuta cuando no se ha reconocido ningún token anterior. Llama a la función showError() y devuelve el token OTHER.

- Función showError():

Esta función se llama cuando se encuentra una entrada no reconocida y muestra un mensaje de error en la salida estándar.

- Función yywrap():

Esta función se llama cuando se alcanza el final del archivo de entrada. En este caso, siempre devuelve 1 para indicar que no hay más entrada para analizar.

Ahora como siguiente paso y ya teniendo el archivo .l Usamos el comando lex (nombre del archivo .l) para producir el código fuente C para nuestro analizador léxico.

```
crist@DESKTOP-3K8A2BV ~/análisisM
$ lex test.l
```

lex.yy.c

Entonces por último tomaremos nuestro código fuente para nuestro analizador sintáctico y para nuestro lex y ejecutarlo a través del compilador para producir nuestro analizador.

```
crist@DESKTOP-3K8A2BV ~/análisisM
$ cc lex.yy.c test.tab.c -o test
```



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4

Al crear nuestro ejecutable hacemos varias pruebas para ver que exitosamente reconoce lo que se esperaba, también mostrando un error de sintaxis.

```
crist@DESKTOP-3K8A2BV ~/análisisM
$ ./test
hola;
Haz ingresado el String - hola
adios;
Haz ingresado el String - adios
12345;
Haz ingresado el numero - 12345
;
syntax error
```

Mostramos un error más al poner símbolos que no pertenecen a nuestro alfabeto.

```
crist@DESKTOP-3K8A2BV ~/análisisM
$ ./test
*****
syntax error
Otra entrada no reconocidaOtra entrada no reconocida
```

¿CUÁLES SON LOS CARACTERÍSTICAS Y LAS FUNCIONES DE ESTOS DOS ANALIZADORES SINTÁCTICOS?

Los analizadores sintácticos Yacc y Bison son herramientas de software que generan código para el análisis sintáctico de un lenguaje de programación a partir de una gramática libre de contexto . Estas herramientas se usan a menudo junto con un generador de analizador léxico, como Lex o Flex, para crear compiladores o intérpretes .

Las características y funciones principales de estos analizadores son:

- Generan analizadores descendentes, es decir, que construyen el árbol sintáctico desde la raíz hasta las hojas, siguiendo las reglas de producción de la gramática.
- Permiten el uso de gramáticas de propósito general, es decir, que no están restringidas a un tipo específico de lenguaje o sintaxis.
- Permiten la utilización de atributos tanto sintetizados como heredados durante la construcción del árbol sintáctico, es decir, que pueden asociar información adicional a los símbolos de la gramática para facilitar el análisis semántico o la generación de código.
- Las especificaciones léxicas y sintácticas se ubican en un solo archivo, lo que simplifica la organización y el mantenimiento del código.
- Generan el código para el analizador sintáctico en el lenguaje de programación C, lo que facilita su integración con otras herramientas o aplicaciones.



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II**

AUTOR: EQUIPO 4

➤ Yacc:

-Generación de Analizadores Sintácticos: Yacc es una herramienta que genera analizadores sintácticos (parsers) a partir de una especificación de gramática formal. Estos analizadores se utilizan para analizar la estructura sintáctica de programas escritos en un lenguaje.

-Gramática Formal: Yacc trabaja con gramáticas formales que se definen mediante reglas de producción. Estas reglas especifican la sintaxis del lenguaje que se está analizando.

-Reglas de Producción: En Yacc, se definen reglas de producción que indican cómo se construye la estructura sintáctica del lenguaje. Cada regla consiste en un símbolo no terminal que se reemplaza por una secuencia de símbolos terminales y no terminales.

-Generación de Código en C: Yacc genera analizadores sintácticos en el lenguaje de programación C. Estos analizadores se utilizan comúnmente como parte de un compilador para generar código intermedio o código objeto.

-Integración con Lex: Yacc se utiliza en conjunto con Lex (o herramientas similares) para construir analizadores léxicos. Lex se encarga de reconocer tokens en el flujo de entrada antes de que Yacc realice el análisis sintáctico.

➤ Bison:

-Generación de Analizadores Sintácticos: Bison es una herramienta similar a Yacc que cumple la misma función: generar analizadores sintácticos a partir de una especificación de gramática.

-Compatibilidad con Yacc: Bison se desarrolló como una alternativa compatible con Yacc. Por lo tanto, las especificaciones de gramática escritas para Yacc suelen funcionar con Bison con pocos cambios.

-Mejoras y Extensiones: Bison ha introducido características adicionales y extensiones que no están presentes en Yacc. Esto incluye un mejor manejo de mensajes de error y mayor flexibilidad en la definición de gramáticas.

-Licencia de Software Libre: Bison se distribuye bajo una licencia de software libre (GNU General Public License o GPL), lo que lo hace más atractivo para proyectos de código abierto y comerciales.

-Mantenimiento Activo: Bison ha sido mantenido activamente por la comunidad de código abierto y ha recibido actualizaciones y correcciones de errores más recientes que Yacc.

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II****AUTOR: EQUIPO 4****6****BITÁCORA DE INCIDENCIAS**

Fecha	Problema encontrado	Solución
06/10/2023	No se ejecutaba correctamente el error	Declarar al inicio del archivo lex el error void yyerror(char *msg);
08/10/2023	Salía un error porque esperaba muchos archivos de entrada.	Declaramos en el archivo lex %option noyywrap

7**OBSERVACIONES**

Sería útil agregar comentarios descriptivos en el archivo YACC para explicar la gramática y el propósito de cada regla. Esto facilitaría la comprensión para cualquiera que revise el código.

Si se está trabajando con una gramática más compleja o con operadores de diferentes prioridades, considera agregar reglas que establezcan claramente el orden de evaluación. Esto evitará ambigüedades.

Agregar reglas para el manejo de errores sintácticos podría ser beneficioso. Esto incluye la detección de expresiones mal formadas y la emisión de mensajes de error útiles.

Cuando una gramática resulta “demasiado ambigua” Bison dará errores. Definir cuando una gramática es “demasiado ambigua” queda fuera del alcance de esta introducción. Una posible solución a algunos tipos de ambigüedad es el uso adecuado de las declaraciones de precedencia de operadores. Si surgen otros conflictos, es útil invocar a Bison con la opción -v, lo que generará un archivo de salida y.output (si se usa la opción -y) donde se puede mirar qué reglas son las problemáticas (aquellas donde aparezca la palabra conflict).

8**CONCLUSIÓN**

Tanto Yacc como Bison son herramientas que generan analizadores sintácticos a partir de especificaciones de gramáticas formales. Aunque Yacc es una herramienta más antigua y tiene restricciones de licencia, Bison es una alternativa más moderna y flexible con una licencia de software libre. La elección entre Yacc y Bison dependerá de las necesidades y restricciones específicas del proyecto.

9**REFERENCIAS**

Compiladores: Principios, técnicas y herramientas. Segunda Edición Aho, Lam, Sethi, Ullman Addison – Wesley, Pearson Educación, México 2008

Diseño de compiladores. A. Garrido, J. Iñesta, F. Moreno y J. Pérez. 2002. Edita Universidad de Alicante

De la Cruz. M(2012) Compiladores. recuperado de
https://arantxa.ii.uam.es/~mdlcruz/docencia/compiladores/2002_2003/compiladores_02_03_yacc_bison.pdf

Resendiz, C(2015) Herramienta Flex para el análisis lexicográfico de lenguajes formales recuperado de <https://core.ac.uk/download/76002179.pdf>

Bibliografía

- ✓ Morales, J. I. S. (s/f). Operadores. Ugr.es. Recuperado el 5 de octubre de 2023, de <https://ccia.ugr.es/~jfv/ed1/c/cdrom/cap2/cap26.htm>
- ✓ 2.3.9 Precedencia de Operadores. (s/f). Edu.mx. Recuperado el 5 de octubre de 2023, de http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro40/239_precedencia_de_operadores.html
- ✓ 3.3.- Operadores y Expresiones. (s/f). Slideshare.net. Recuperado el 5 de octubre de 2023, de <https://es.slideshare.net/yennysalazar1402/33-operadores-y-expresiones>
- ✓ (S/f). Uhu.es. Recuperado el 7 de octubre de 2023, de https://www.uhu.es/francisco.moreno/gii_pl/docs/Tema_3.pdf
- ✓ (S/f-b). Uma.es. Recuperado el 7 de octubre de 2023, de https://ocw.uma.es/pluginfile.php/1021/mod_resource/content/0/Capitulo_3.pdf
- ✓ (S/f). Cartagena99.com. Recuperado el 7 de octubre de 2023, de https://www.cartagena99.com/recursos/alumnos/apuntes/ININF2_M4_U4_T3.pdf
- ✓ Sintáctico, T. 3. –. (s/f). Procesadores de lenguaje. Uah.es. Recuperado el 8 de octubre de 2023, de http://www.cc.uah.es/ie/docencia/ProcesadoresDeLenguaje/ProcesadoresDelLenguajeTema3ParteII_1pagina.pdf
- ✓ (S/f-b). Uma.es. Recuperado el 9 de octubre de 2023, de <http://www.lcc.uma.es/~galvez/ftp/tci/tictema5.pdf>
- ✓ (S/f-b). Cartagena99.com. Recuperado el 9 de octubre de 2023, de https://www.cartagena99.com/recursos/alumnos/apuntes/ININF2_M4_U5_T2.pdf
- ✓ 3.1.1 Manejo de los Errores Sintácticos. (s/f). Edu.mx. Recuperado el 9 de octubre de 2023, de http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/311_manejo_de_los_errores_sintacticos.html
- ✓ 3.1.2 Estrategias de Recuperación de Errores. (s/f). Edu.mx. Recuperado el 9 de octubre de 2023, de http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/312_estrategias_de_recuperacion_de_errores.html
- ✓ de Autómatas y lenguajes formales Federico Simmross Wattenberg, T. (s/f). El generador de analizadores sintácticos yacc. Uva.es. Recuperado el 9 de octubre de 2023, de <https://www.infor.uva.es/~mluisa/talf/docs/lab0/L8.pdf>

- ✓ Universidad de Guanajuato. (2022, julio 16). Clase digital 4. Herramientas de software: Yacc/Bison. Recursos Educativos Abiertos; Sistema Universitario de Multimodalidad Educativo (SUME) - Universidad de Guanajuato. <https://blogs.ugto.mx/rea/clase-digital-4-herramientas-desoftware-yacc-bison>

[Link a Video](#)

VIDEO GENERAL DE ACTIVIDAD 5

<https://drive.google.com/file/d/1nfoZwqlWnLzRv7QvHVfKlimJz2eab5ny/view?usp=sharing>

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II****AUTOR: EQUIPO 4***Arias**Fern**Díaz*

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

PRACTICA No.	NOMBRE DE LA PRACTICA	FECHA DE ENTREGA
3	Avance: Planteamiento formal del proyecto semestral	02 de Octubre, 2023

1	INTRODUCCION
La creación de un lenguaje de programación propio es un proceso apasionante y desafiante que implica la concepción y diseño de un conjunto único de reglas y sintaxis para comunicarse con una máquina. Este acto de ingeniería lingüística es mucho más que simplemente establecer palabras clave y estructuras de código; representa una oportunidad para la innovación, la resolución de problemas y la expresión personal. En este viaje, los programadores se embarcan en la tarea de moldear un medio de comunicación entre humanos y computadoras, con el objetivo de lograr eficiencia, flexibilidad y un enfoque específico para satisfacer sus necesidades o metas únicas. La creación de un lenguaje de programación propio, a menudo, implica un equilibrio delicado entre la simplicidad y la potencia, y puede servir como una expresión de la visión y la creatividad de quienes se aventuran en este desafío. En esta exploración, examinaremos las razones para crear un lenguaje de programación personalizado, los beneficios que puede aportar y las consideraciones clave que deben tenerse en cuenta en este apasionante viaje hacia la creación lingüística en el mundo de la programación.	

2	OBJETIVO
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.	

3	MATERIALES NECESARIOS
<ul style="list-style-type: none">Conocimiento en programación y teoría de lenguajesHerramientas de desarrolloDocumentaciónConjunto de reglas (gramática)Compilador o interpretePruebas y depuraciónHardwareTiempo y esfuerzoColaboración	

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4**

4

MARCO TEÓRICO**Planteamiento del proyecto final**

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:

Innovación: 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.

Eficiencia y productividad: 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.

Educación y enseñanza: 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.

Especialización: 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.

Propiedad intelectual: 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 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.



TECNOLÓGICO
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4

5

DESARROLLO

6

BITÁCORA DE INCIDENCIAS

Fecha

Problema encontrado

Solución

7

OBSERVACIONES

8

CONCLUSIÓN

Ajor

Fern

Díaz



TECNOLÓGICO
NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II**

AUTOR: EQUIPO 4

9

REFERENCIAS

Ajor

Fern

Díaz

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II****AUTOR: EQUIPO 4**

Aitor
Fern
Dafy

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

PRACTICA No.	NOMBRE DE LA PRACTICA	FECHA DE ENTREGA
6	2023-10-09 : TAREA EN EQUIPO	09 de octubre, 2023

1	INTRODUCCION
<p>ASML, una destacada empresa neerlandesa, se encuentra en la vanguardia de la fabricación de equipos de litografía utilizados en la producción de semiconductores. Fundada en 1984 y con su sede en Veldhoven, Países Bajos, ASML despliega tecnología avanzada para la impresión de patrones microscópicos en obleas de silicio, una actividad esencial en la creación de chips de computadora y otros dispositivos electrónicos de vanguardia. La empresa juega un papel crítico en la cadena de suministro de la industria de semiconductores, abasteciendo sus equipos a empresas líderes en tecnología de todo el mundo, entre las que se cuentan nombres como Intel, TSMC, Samsung y otros prominentes fabricantes de chips.</p> <p>Los microchips, también conocidos como circuitos integrados o simplemente "chips", representan un hito en la electrónica moderna. Estos componentes electrónicos concentran una vasta cantidad de circuitos electrónicos miniaturizados en una única placa de silicio o sustrato semiconductor. Estos circuitos son ingeniosamente diseñados para cumplir funciones específicas, tales como el procesamiento de datos, el almacenamiento de información o el control de hardware.</p> <p>Estos elementos son cruciales en la operatividad de una amplia variedad de dispositivos, desde computadoras y teléfonos inteligentes hasta electrodomésticos, automóviles, dispositivos médicos y una gama interminable de productos electrónicos. La verdadera maravilla radica en que, a pesar de su tamaño diminuto, los microchips son capaces de llevar a cabo tareas complejas en espacios sumamente reducidos y consumir mucha menos energía que los componentes electrónicos tradicionales.</p>	

2	OBJETIVO
<p>Visualizar un video informativo sobre ASML que proporcione una visión detallada de la empresa, sus cifras financieras, su desarrollo tecnológico y su impacto en la industria de semiconductores, con el propósito de adquirir un entendimiento completo de su importancia en el mercado y su contribución al avance de la electrónica moderna</p>	

3	MATERIALES NECESARIOS
<ul style="list-style-type: none">• Conocimiento en programación y teoría de lenguajes• Herramientas de desarrollo• Documentación• Conjunto de reglas (gramática)• Compilador o interprete	

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4**

- Pruebas y depuración
- Hardware
- Tiempo y esfuerzo
- Colaboración

4

MARCO TEÓRICO

ASML es una empresa neerlandesa líder en el campo de la fabricación de equipos de litografía utilizados en la producción de semiconductores. La empresa fue fundada en 1984 y tiene su sede en Veldhoven, Países Bajos. ASML fabrica equipos de litografía avanzados que se utilizan para imprimir patrones microscópicos en obleas de silicio, lo que es fundamental en la producción de chips de computadora y otros dispositivos electrónicos. La compañía es esencial en la cadena de suministro de la industria de semiconductores y suministra sus equipos a empresas líderes en tecnología de todo el mundo, como Intel, TSMC, Samsung y otros fabricantes de chips.

Un microchip, también conocido como circuito integrado o chip, es un componente electrónico que contiene una gran cantidad de circuitos electrónicos miniaturizados en una única placa de silicio o sustrato semiconductor. Estos circuitos están diseñados para realizar funciones específicas, como procesamiento de datos, almacenamiento de información o control de hardware.

Los microchips son fundamentales en la electrónica moderna y se utilizan en una amplia variedad de dispositivos, desde computadoras y teléfonos inteligentes hasta electrodomésticos, automóviles, dispositivos médicos y muchos otros productos electrónicos.

Los microchips permiten la miniaturización y la eficiencia en la electrónica, ya que pueden realizar tareas complejas en un espacio muy pequeño y consumir menos energía que los componentes electrónicos tradicionales.

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4**

5

DESARROLLO**Por qué ASML tiene el Monopolio Más COLOSAL de la Historia****Primeros 5 minutos**

Esta máquina de litografía ha revolucionado el mundo tal y como lo conocemos. Es del tamaño de un autobús, pero es capaz de fabricar los chips más avanzados del mundo, logrando imprimir transistores 10,000 veces más pequeños que un pelo humano y lo más sorprendente es que estas máquinas las fabrica una sola compañía en todo el mundo.



ASML es la encargada de fabricar esta potente máquina. Esta es la historia de una compañía que desde sus inicios en un cobertizo ha logrado romper la barrera de la física y de la ingeniería, cargando sobre sus hombros el progreso del ser humano.

El profesor Chris comenta que ASML tiene el monopolio en la fabricación de las máquinas de litografía EUV.



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA

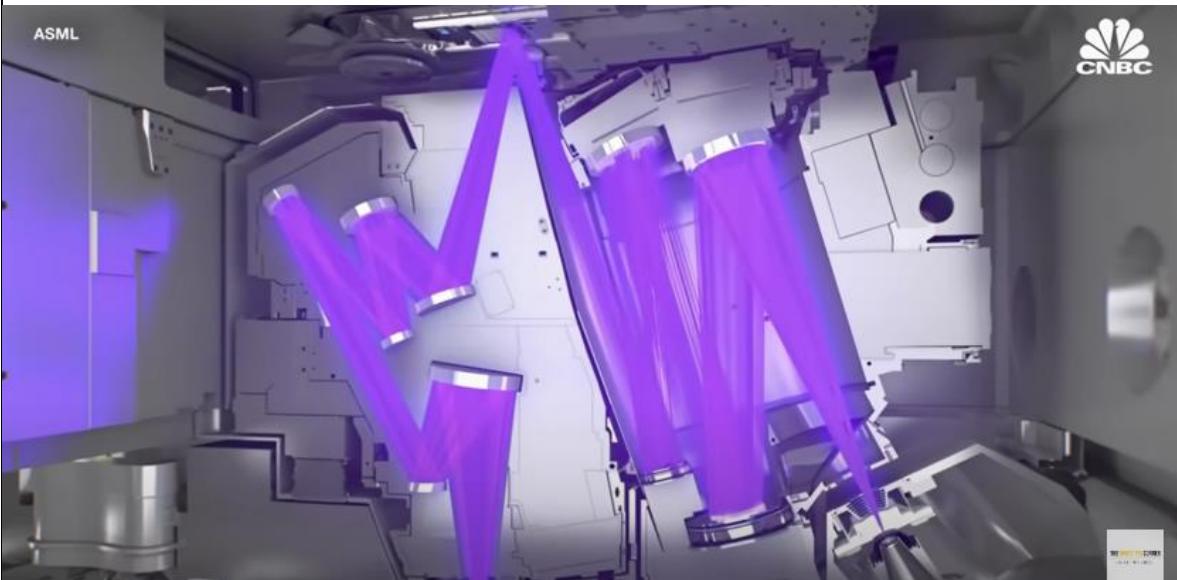


PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

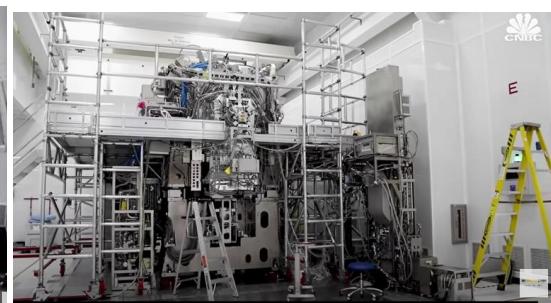
AUTOR: EQUIPO 4



Estas máquinas son necesarias para fabricar los chips más avanzados que usamos hoy en día.

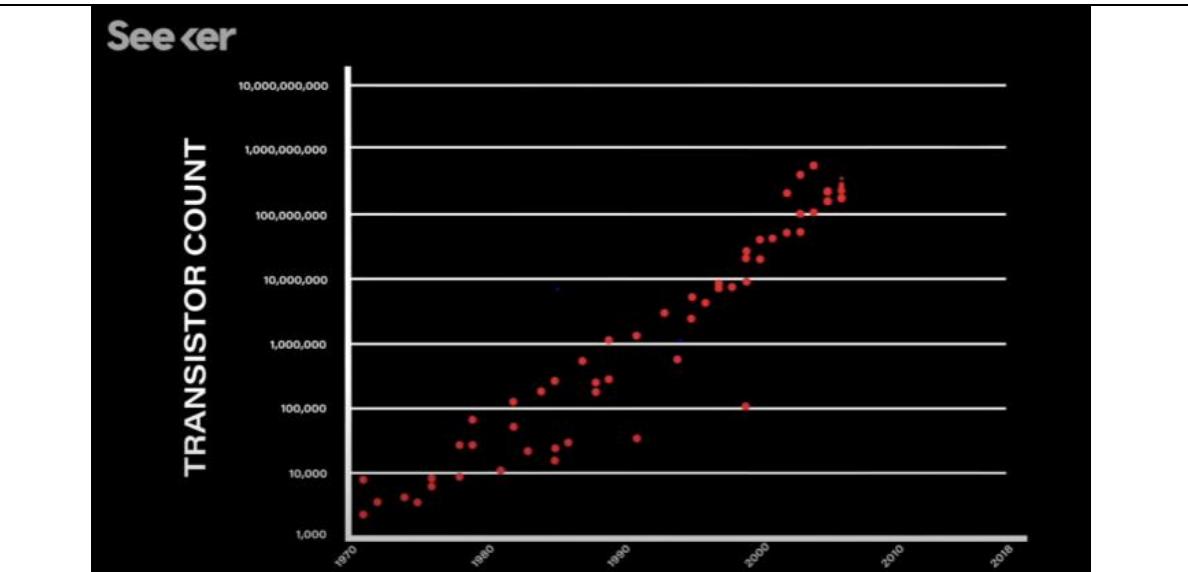


ASML es una de las organizaciones más increíbles, del mundo, que las máquinas que fabrican son de los artefactos más complejos jamás creados.

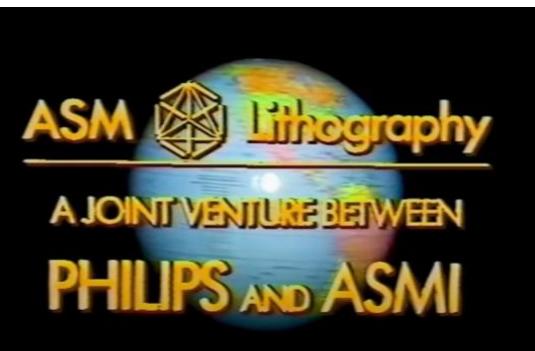


Como es posible saber en todo momento existen obstáculos para cualquier empresa y una de estas para ASML es que tiene que romper la Famosa Ley de Moore, ya que cada dos años se duplica el número de transistores de un chip.

Ajor
Fern
Daff

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4**

Todo comienza cuando unos científicos comienzan a reunirse para hacer realidad una visión. "La litografía es el futuro de la fabricación de los chips". En 1984 Philips y ASM se unen para formar la compañía Advance Semicontactor Material Liptography.



Esta era la época de los primeros móviles cuando las personas compraban su primer ordenador, cuando en 1985 lanzan la primera máquina que usaba rayos de luz de una forma muy precisa para imprimir diseños en silicio y fabricar chips y esta tecnología se conoce como "Fotolitografía".

TECNOLÓGICO
NACIONAL DE MÉXICO

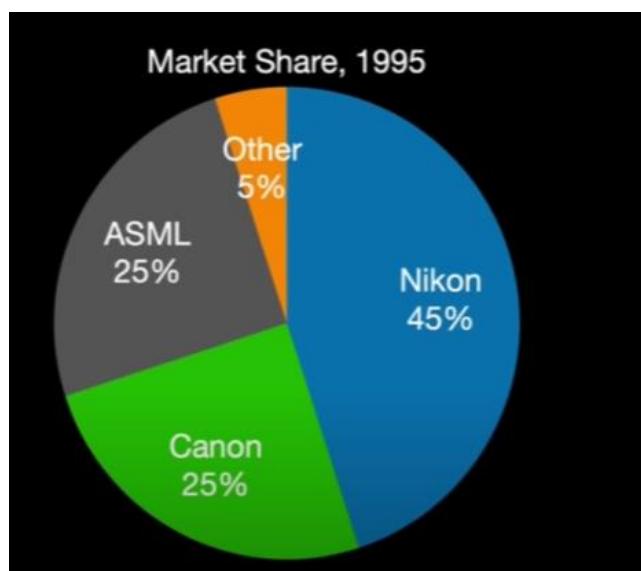
INSTITUTO TECNOLÓGICO DE CELAYA

PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4



ASML como Canon y Nikon decidieron apostarle a la litografía, en 1996 doce años después de su nacimiento, Nikon y Canon tienen el monopolio del mercado y a ASML se le acaba del dinero.



Sabían que debían tener un plan por lo que deciden invertir todo su dinero a una maquina llamada "TwinsCan" y la está funciono a la perfección ya que, al combinar el proceso de medición e impresión en una misma máquina, lograba imprimir chips 3 veces más rápido que las máquinas de la época.



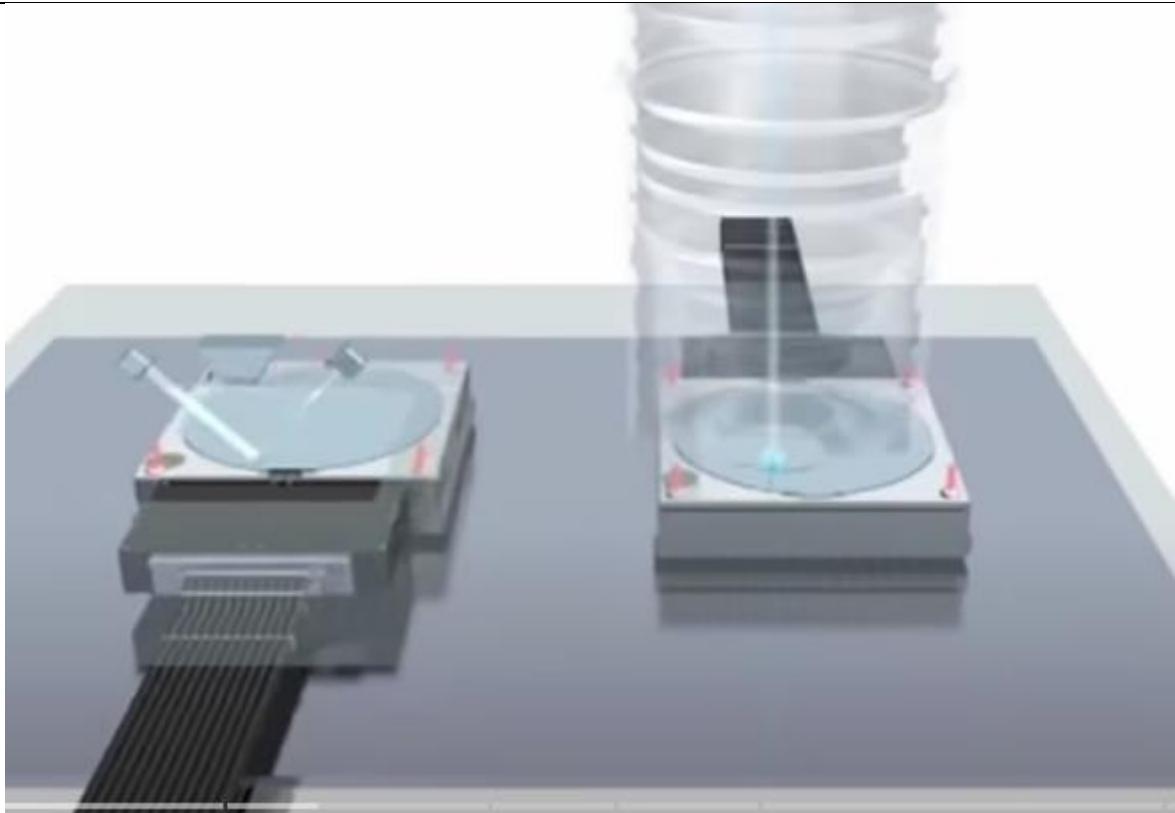
TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4



Con el paso del tiempo sabían que tenían que invertirles más a los chips ya que tanto Canon como Nikon no les podían seguir el ritmo por lo que para esto decidieron y sabían que el diseño del chip tenía que ser más pequeño y así sería más eficiente y se les ocurre utilizar una luz de una frecuencia mucho más alta a las que utilizaban en la maquina TwinsCan de 193 nanómetros.

Aitor
Fern
Dafy



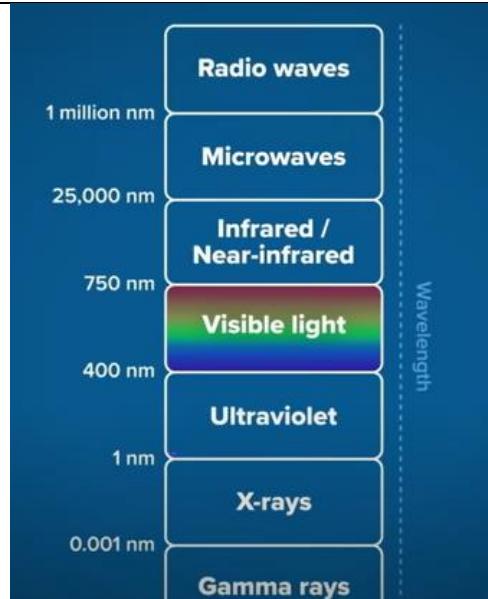
TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4



ASML es capaz de crear una luz de una longitud de onda de 13.5 nanómetros llamada "Extreme ultraviolet" o "EUV" con una longitud de onda 14 veces menor que la maquina TwinsCan y por ello puede imprimir chips con transistores que son 10,000 veces mas pequeño que un pelo humano.

Siguientes 5 minutos

Los microchips han sido una de las grandes innovaciones del siglo 20, esta tecnología esta diseñada con la intención de recibir muchas entradas y salidas de información en forma binaria, es decir con 0 y 1.

Aitor
Yerai
Dafne



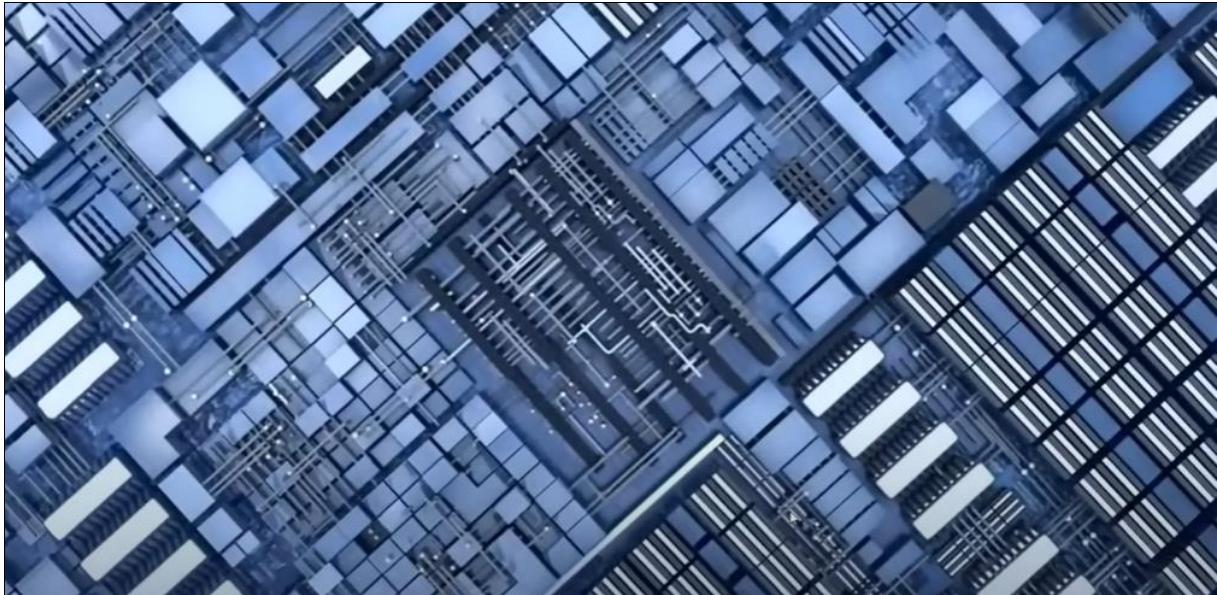
TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA

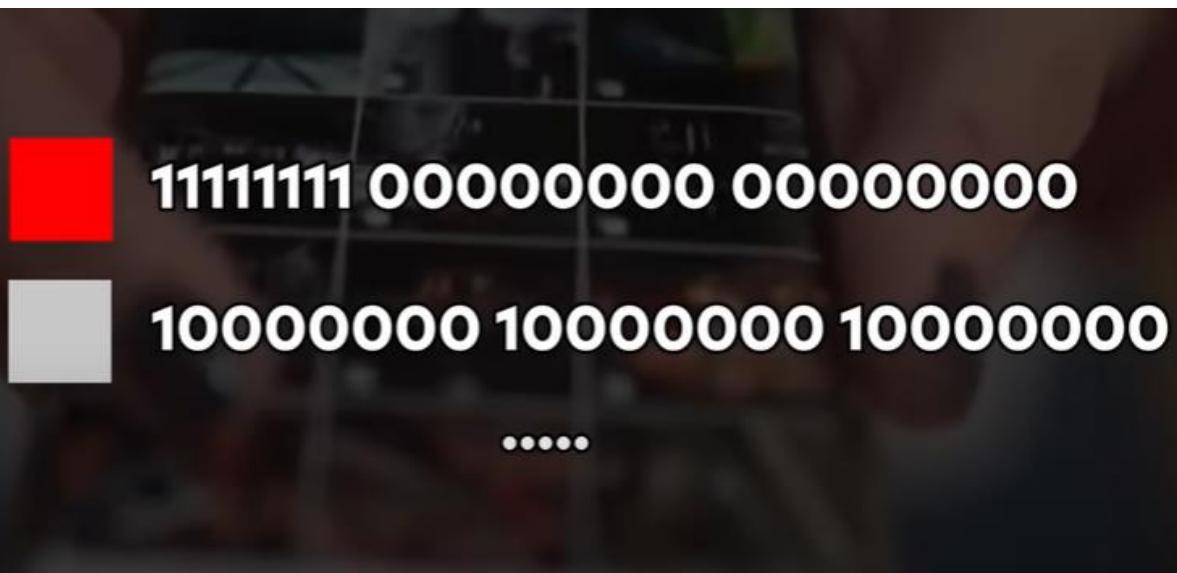


PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4



Estos ceros y unos están presentes en nuestra vida diaria en nuestros teléfonos, ya que desde cualquier letra, símbolo o hasta los colores de los pixeles de la pantalla esta conformado del sistema binario.



Acciones que se ven tan sencillas como leer el periódico en el celular necesitan miles de millones de sumas de estos ceros y unos para poder mostrar, entre más sumas el microchip haga por segundo más rápido va a ser.

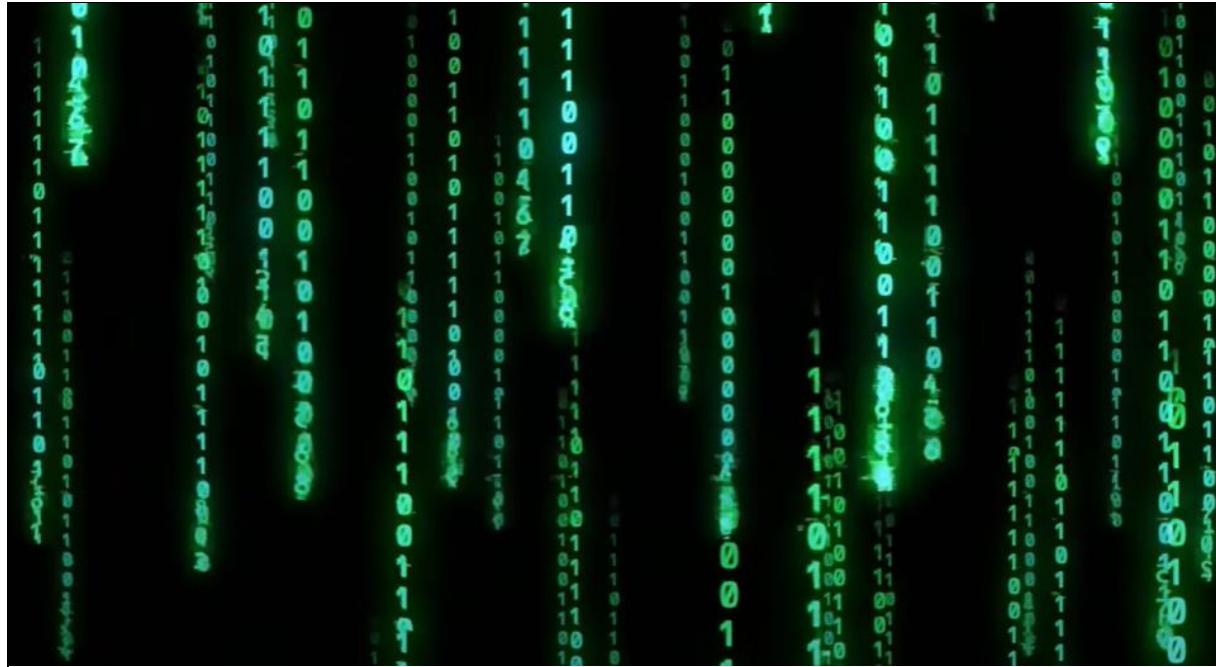
Aitor
Fern
Dafy

TECNOLÓGICO
NACIONAL DE MÉXICO

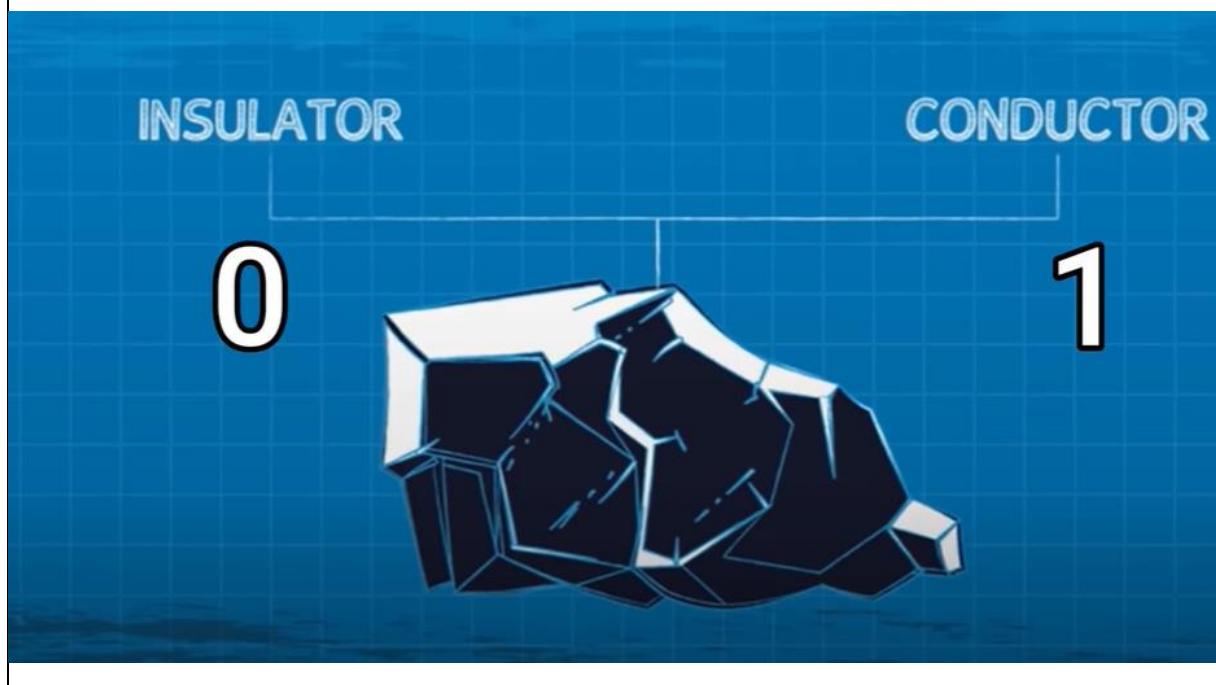
INSTITUTO TECNOLÓGICO DE CELAYA

PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4



Para crear los unos y ceros que necesita el microchip se necesita electricidad y la ayuda de los materiales semiconductores, el flujo de electricidad por si misma no nos da la posibilidad de crear los ceros y unos que necesitamos es por eso que recurrimos a varias herramientas como los semiconductores que nos ayudaran a tener o no flujo de electricidad que se interpretara con un 1 o un 0 respectivamente.





TECNOLÓGICO
NACIONAL DE MÉXICO

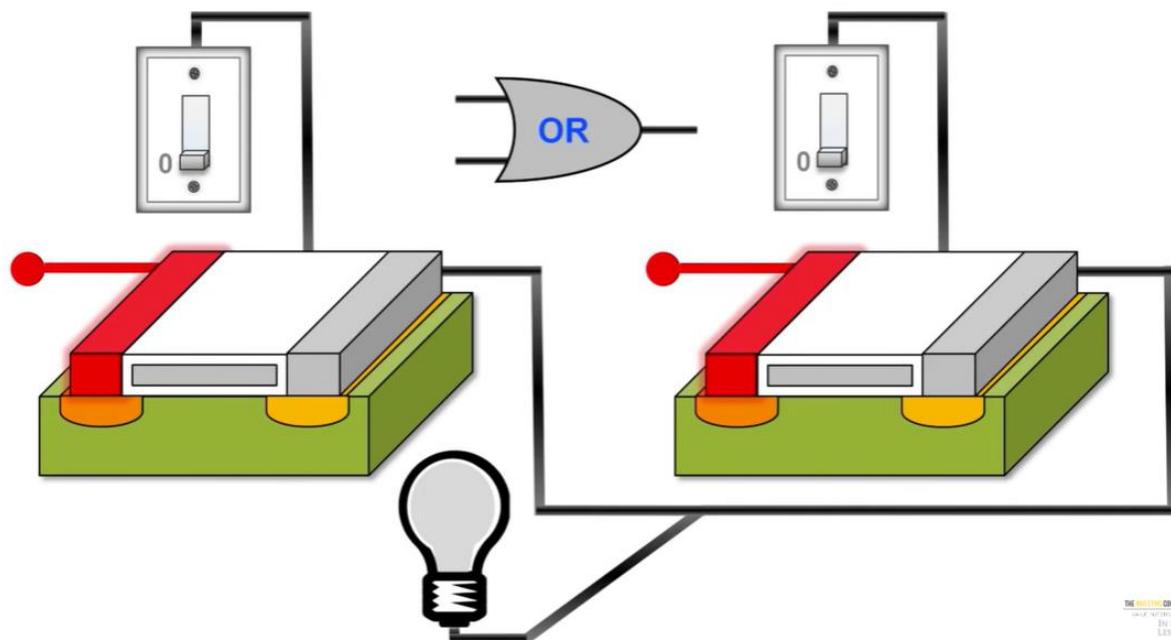
INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4

Debemos tener este flujo de ceros y unos de manera controlada, es aquí donde aparecen los transistores que nos ayudan a darle paso o no a la electricidad y así nos ayuda a cumplir la función de un interruptor.



Para hacer los microchips más rápido se necesitan cada vez más transistores y también cada vez más pequeños, aquí la maquina ASML entra en juego diseñando caminos para el flujo de ceros y unos.



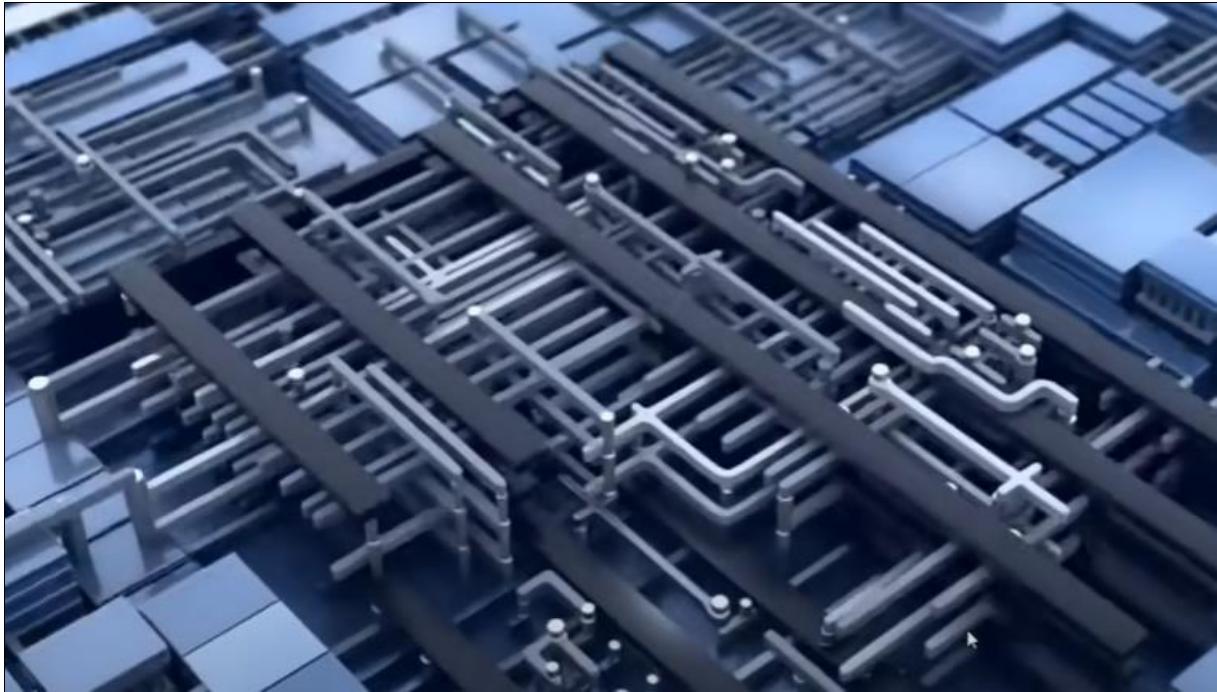
TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4



Para la fabricación de un chip comienza recolectando arena que es rica en silicio, está en combinación de otros átomos se hace un material semiconductor.



Después de hacer la purificación de la arena se hace un lingote de silicio, este se corta y se crean láminas denominadas Wafer.

Autor: *[Handwritten signatures]*



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA

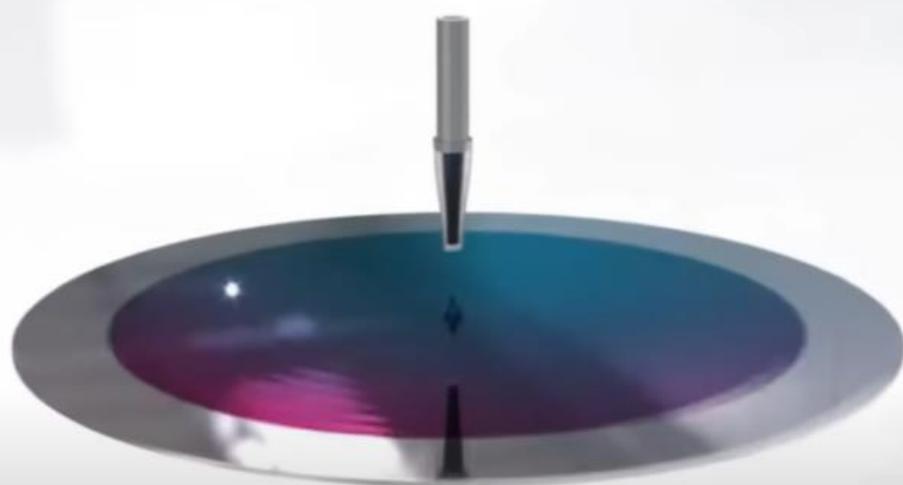


PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4



En el wafer se depositan diferentes materiales en cada capa, la primera es una para resistencia a la luz que cuando este recibe luz se endurece.



Alvaro
Fern
Dafne



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA

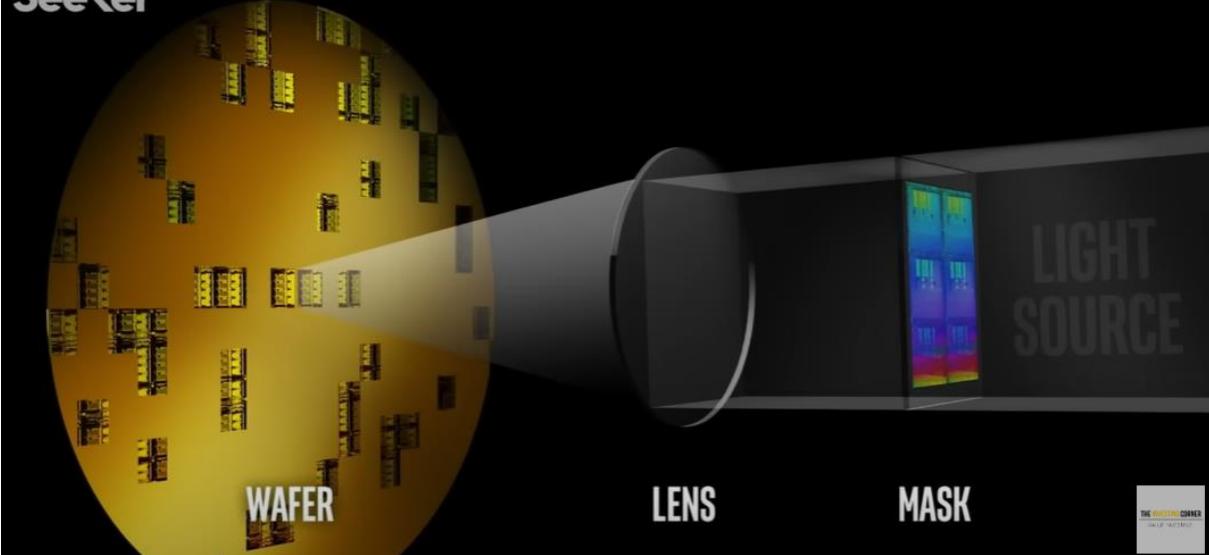


PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4

La luz debe de pasar por patrones en específico y este tipo de diseño se llama fotomáscara

Seeker



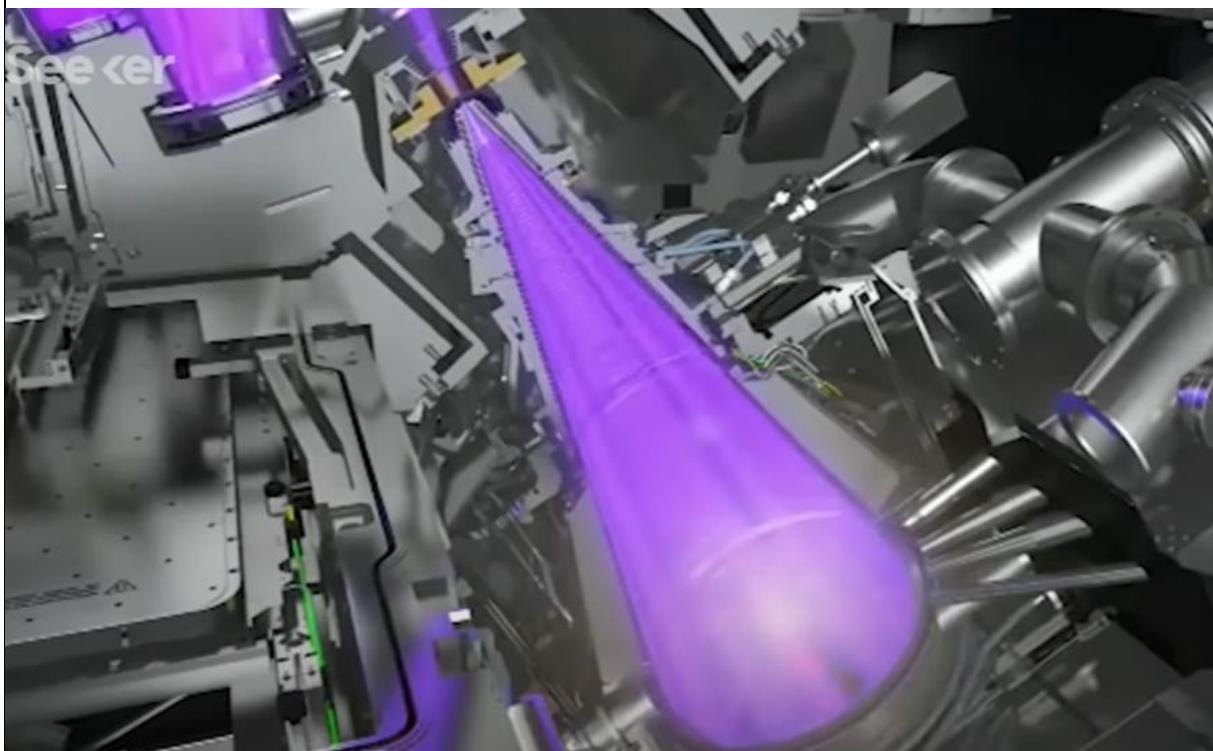
Después se la maquina de ASML aplica luz ultravioleta por la fotomascara la cual estará dejando caminos.



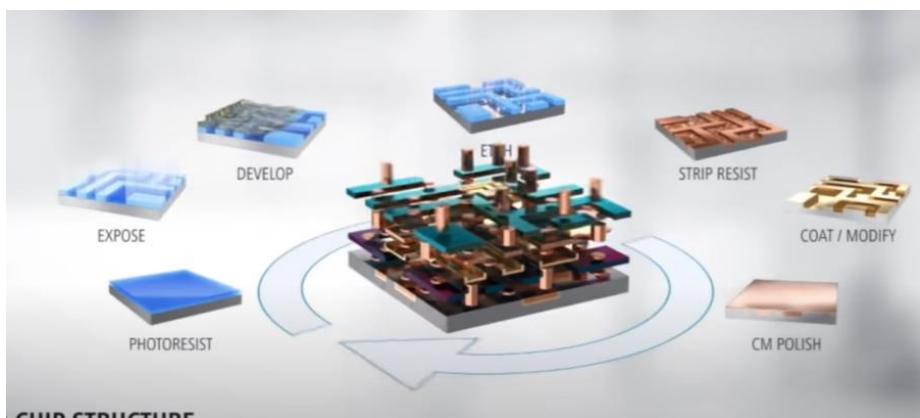
Aitor
Fern
Dafy

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4**

La máquina EUV para crear esta luz lanza muchas gotas de estaño que son interceptadas por un láser, hasta dos veces a una sola gota generando plasma generando así la luz ultravioleta.



Después de este proceso se aplican varias capas de materiales las cuales ayudan a eliminar materiales que no fueron expuestas a la luz ultravioleta, ionizar el silicio, agregar materiales foto resistentes y se agrega una capa de metal donde la electricidad fa a fluir y se pule. Este proceso se repite muchas veces para formas los pasadizos que se necesitan.





TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



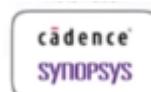
PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4

Últimos 5 minutos

Dentro de la industria de microchips hay una gran variedad de compañías que están implicadas con ASML, siendo compañías de tipo:

- Software



- Diseñadoras del chip



- Maquinaria de fases



Cabe destacar que un smartphone solo podrá evolucionar si ASML es capaz de fabricar procesadores más pequeños cada vez.

ASML siempre está un paso por delante de las compañías que podrían decirlo competirle, ya que en este año, ya tiene su máquina esplendor para el siguiente año 2024 llamada HIGH NA, valorada en 350 millones de euros

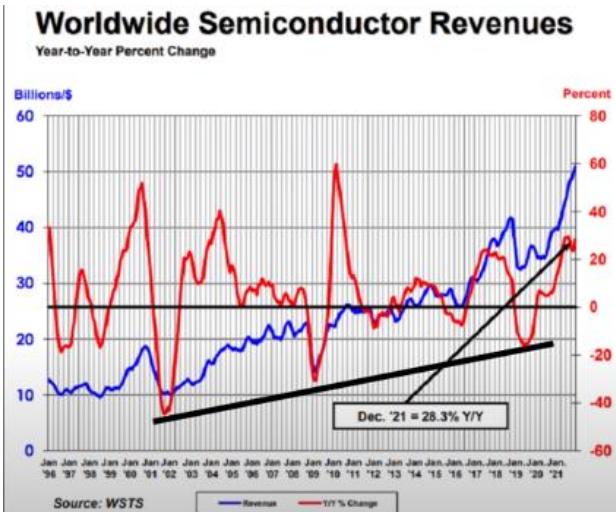
TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4**

Aitor
Fern
Dafy

Siendo así que TSMC anuncio que en 2025 comenzara a fabricar chips de 2nm.

ASML cuida su parte financiera a futuro, ya que solo pueden trabajar con ella, en el caso que una compañía quiera competir con la propia ASML, ya que la inexperiencia que puede ocasionar la fabricación de los competidores les puede salir mal el plan.

Dentro del sector de semiconductores, las recesiones son menos agresivas, ya que la demanda siempre irá por delante de la oferta, debido a el auge de chips y todos los aparatos electrónicos que utilizamos en el día a día.



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA

PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II

AUTOR: EQUIPO 4

Añadiendo la disminución de la oferta debido a que solo ASML es la fabricante de estos chips.

TSMC sabe que invirtiendo cuando la demanda esta abajo, entonces cuando se recupere esta oferta, recuperar toda la cuota de mercado de aquellas empresas que no hayan invertido cuando estaba baja.



Siendo así que ASML tiene el monopolio mas grande del mercado de la venta de microchips y lleno a duplicar sus cifras exponencialmente a lo largo de los años

	2016	2017	2018	2019	2020	2021	2022	2023e	2024e	2025e	2026e	2027e
Income Statement (EUR Mn)												
Net system sales	4.672,0	6.373,7	8.259,1	8.996,2	10.316,6	13.652,8	15.430,3	21.377,5	21.867,5	26.944,0	29.251,0	31.374,0
Net services and field option sales	2.122,8	2.678,1	2.684,0	2.823,6	3.061,9	4.050,2	5.743,1	5.743,1	6.432,5	7.204,1	7.824,6	8.717,0
Revenues	6.794,8	9.052,8	10.944,0	11.420,0	13.973,5	18.611,0	21.173,4	27.026,6	28.319,6	34.149,2	37.175,6	40.091,1
Cost of system sales	-2.468,2	-3.450,0	-4.141,2	-4.676,2	-8.169,3	-8.482,9	-7.843,3	-10.213,2	-10.068,2	-12.124,9	-13.183,0	-14.118,3
Cost of services and field option	-1.282,1	-1.517,1	-1.773,7	-1.884,0	-2.012,0	-2.319,1	-2.817,4	-2.841,8	-3.023,2	-3.241,9	-3.468,8	-3.748,3
Gross profit	3.044,5	4.078,7	5.020,1	5.279,8	6.797,2	9.809,0	10.512,7	14.165,6	15.228,3	18.781,5	20.525,8	22.224,4
R&D	-1.105,8	-1.259,7	-1.575,9	-1.968,5	-2.200,8	-2.547,0	-2.282,1	-3.512,7	-3.881,6	-4.439,3	-4.832,8	-5.211,8
SG&A	-374,8	-416,6	-488,0	-520,5	-544,9	-725,6	-809,6	-1.080,8	-1.132,6	-1.365,9	-1.487,0	-1.603,6
Other	93,8	95,6	0,0	0,0	213,7	0,0	0,0	0,0	0,0	0,0	0,0	0,0
EBIT	1.657,7	2.496,2	2.965,2	2.790,8	4.051,5	6.750,1	7.321,0	9.572	10.414	12.976	14.206	15.409
Finance income	71,7	72	13,5	11,6	9,4	10,0	16,2	0,0	0,0	0,0	0,0	0,0
Financial Expenses	-36,0	-57,5	-41,8	-36,6	-43,3	-54,6	-60,8	-42,6	-35,1	-36,1	-35,1	-35,1
Profit before tax	1.691,4	2.446,9	2.936,9	2.785,8	4.016,6	6.706,5	7.276,4	9.529	10.379	12.941	14.171	15.374
Income tax expense	-219,8	-310,7	-301,6	-191,7	-551,5	-1.021,4	-1.018,6	-1.572,4	-1.712,5	-2.135,3	-2.338,2	-2.636,7
Income after income taxes	1.471,9	2.136,2	2.585,3	2.574,1	3.465,1	5.684,1	6.287,8	7.967,1	8.866,4	10.805,9	11.832,7	12.837,2
Profit from investments in ass	-16,7	6,2	18,2	88,6	199,1	138,0	0,0	0,0	0,0	0,0	0,0	0,0
Net income	1.471,9	2.119,5	2.591,5	2.592,3	3.553,7	5.583,2	6.395,8	7.957,1	8.866,4	10.805,9	11.832,7	12.837,2
Diluted number of shares	427,7	431,6	426,4	421,6	419,1	410,4	390,0	382,0	386,1	380,4	374,7	369,0
Attributable diluted EPS	3,44	4,91	6,06	6,1	8,5	14,3	16,1	20,30	22,44	26,41	31,59	34,79
P/E	9.697,3	9.206,0	9.661,0	9.705,8	2.061,4	6.240,4	3.931,0	9.679,4	10.412,6	19.879,5	14.599,0	14.208,0

Y a su vez siendo capaces de generar retornos de capital encima del 50% y menos cíclica que el propio sector de semiconductores, permitiéndose así incrementar y mejorar la fabricación de estos chips.

TECNOLÓGICO
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y
AUTÓMATAS II****AUTOR: EQUIPO 4****6****BITÁCORA DE INCIDENCIAS**

Fecha	Problema encontrado	Solución

7**OBSERVACIONES****8****CONCLUSIÓN**

El mercado de microchips es altamente competitivo, y muchas empresas compiten en la fabricación y diseño de chips. Sin embargo, la tecnología de litografía avanzada de ASML ha sido un factor importante en la mejora y miniaturización de los chips, lo que le ha otorgado una posición destacada en la industria. En resumen, ASML tiene un monopolio de microchips y desempeña un papel crucial en la cadena de suministro de semiconductores.

9**REFERENCIAS**

- https://www.youtube.com/watch?v=zAYCfw_syFc