



DEPARTAMENTO DE SISTEMAS COMPUTACIONALES E INFORMÁTICA

ASUNTO: SOLICITUD DE ACTIVIDADES

Celaya, Guanajuato, 27 / septiembre / 2022

LENGUAJES Y AUTÓMATAS II
DOCENTE DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ
SEMESTRE AGOSTO-DICIEMBRE 2022

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 PROponER DE MANERA PROFESIONAL LOS TEMAS PROPUESTOS EN LA ESTRUCTURA TEMÁTICA DE ESTA ASIGNATURA.

3. ANÁLISIS SINTÁCTICO.

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.

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.





EDUCACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO

Instituto Tecnológico de Celaya
Departamento de Sistemas y Computación

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 LOS 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.



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





EDUCACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO

Instituto Tecnológico de Celaya
Departamento de Sistemas y Computación

- 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.



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 DIRECTRICES 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.**





EDUCACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO

Instituto Tecnológico de Celaya
Departamento de Sistemas y Computación

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 , 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 CONSECUТИVO 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.

2022-09-27_TNM_CELAYA_LAI-A_A5_EQUIPO_99_9999999_PEREZ_PEREZ_JUAN_AGO-DIC22.PDF

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

SI EL TRABAJO SE SOLICITÓ INDIVIDUALMENTE.

2022-09-27_TNM_CELAYA_LAI-A_A5_9999999_PEREZ_PEREZ_JUAN_AGO-DIC22.PDF



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





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.



D M A
14/10/22

INSTITUTO TECNOLÓGICO NACIONAL DE MÉXICO EN CELAYA

LENGUAJES Y AUTÓMATASTII

Equipo: 3

Docente: Ricardo González González

Integrantes:

Ramírez García Marco Isaias (19030260)
García Ramírez Luis David (19030263)
Pérez Cabrera José Eduardo (19030985)

A 06 de Octubre de 2022, Celaya, Gto.

Introducción

Durante el desarrollo de los temas por ver se abordaron temas técnicos sobre la teoría que rodea el funcionamiento de un código que realizareí los análisis Léxico y sintáctico con ayuda del lenguaje C y las dependencias de Software libre Flex y GNU Bison que evolucionan las extensiones ".l" y ".y" que son acrónimos de Lex y Yacc, las primeras herramientas de analizadores pero de carácter privativo por la empresa Unix, por eso se hace una adopción libre. Primero que nada se abordaron los temas teóricos para entender los procesos a bajo nivel que se hacen con las estructuras de datos, y los recorridos con árboles de análisis sintáctico gramáticos y sus diferentes clasificaciones en las que se realizaba el análisis con los pasos y metodologías correspondientes, todo esto con el fin de poder plasmar el conocimiento de las evaluaciones con las GIC en las estructuras de datos pero ahora siendo implementadas en código, una vez comprendidos los temas y visto ejemplos realizaremos la codificación y pruebas de un proyecto real en un sistema operativo o novios.

Precedencia de Operadores

Como bien sabemos la programación se basa en gran parte sobre los matemáticos y sus distintas ramas que la conforman, en los alfabetos usados para desarrollar formulas o ecuaciones nos encontramos con múltiples operadores los cuales se encargan de realizar distintas operaciones para desarrollar un problema y poder llegar a una solución, para dicho desarrollo se tienen que tener en cuenta varios factores que afectan a la forma y estructura que se deberá seguir para llegar a un resultado cuantitativo, Un factor clave para el proceso es la precedencia de los operadores involucrados ya que los operadores están clasificados y categorizados con base en la jerarquía de precedencia que se le haya dado a cada operador involucrado,

podemos tener

$$Ej1 = 7 + 8 * 9$$

con esto, operaciones casi

$$79$$

mayor jerarquía ↑

identicas pero

$$(7 + 8) * 9$$

$$135$$

↑ mayor jerarquía ↑

Los operadores desde el punto de vista matemático son fundamentales para hacer entender el orden en el que se resuelve un problema específico que muchas de las veces está representado a manera de ecuación, sin embargo cuando los traducimos dichos operadores a un lenguaje de programación el contexto para usar los símbolos que representan a los operadores se enriquecen al darles nuevas funcionalidades, aunque cabe mencionar que la interpretación y funcionalidad puede representar distintas acciones en un lenguaje de programación. Recordando un poco la definición de un lenguaje formal recordaremos que los lenguajes en su estructura no permitirán la existencia de ambigüedades, es decir no se le podrán dar diferentes interpretaciones a algo que ya fue declarado previamente para la construcción del lenguaje, los operadores son palabras reservadas que dependiendo el lenguaje y la necesidad pueden representar acciones distintas incluso sin estar pegados a las reglas de las matemáticas.

= Igualación

== Comparación

+ Suma

++ Incremento

Para entender los operadores hay que tener en cuenta que existen diferentes tipos de operadores, cada uno con sus operaciones y funcionalidades específicas. Los diferentes tipos de operadores más comunes en los lenguajes de programación son:

- Operadores Completos

Utilizados para acceder a elementos cuyos tipos de datos son complejos como lo podría ser una matriz o alguna otra estructura de datos.

- Operadores Aritméticos

Son los operadores más comunes debido a su extenso uso en las matemáticas. Se usan para calcular el valor de dos o más cifras numéricas o incluso cambiar los signos de un número con la opción de establecer positivos y negativos dependiendo la operación hecha.

- Operadores de Cadena

Estos operadores sirven para la concatenación de cadenas u objetos, esta es una diferencia muy marcada con otros tipos de datos como los numéricos al trabajar con distintos contenidos, es decir se marea la diferencia entre valores cualitativos y cuantitativos.

• Operadores de Comparación

Son aquellos que comparan dos o más expresiones para tomar una decisión y devolver un resultado booleano que representara el cumplimiento (true) o rechazo (false) de una condición.

Ejemplos concretos de operadores de Comparación (estos ejemplos pueden variar en la sintaxis dependiendo el lenguaje de programación en donde se apliquen):

Operación	Aplicación de Símbolo	Definición / uso
$a == b$	Igualación	a igual que b
$a != b$	Distinción	a diferente de b
$a > b$	Mayor que	a mayor que b
$a >= b$	Mayor O igual	a mayor o igual b
$a < b$	Menor que	a menor que b
$a <= b$	Menor O igual	a menor o igual b

Operación	Aplicación de Símbolo	Definición / uso
-----------	-----------------------	------------------

• Operadores Lógicos

Se usan para combinar dos o más valores booleanos y obtener un resultado dependiendo lo que sea evaluado.

Ejemplos concretos de operadores lógicos (pueden variar dependiendo la sintaxis usada)

• Conjunción

• AND
• &&

• Disyunción

• OR
• ||

• Negación

• NOT
• <>

• Analizador Sintáctico Descendente

Para la compilación de un código es necesario realizar un proceso de análisis el cual está dividido en tres partes, el análisis léxico que se basa en validar que el código esté escrito con base en el alfabeto establecido, después del análisis léxico se encuentra el análisis sintáctico gramatical, específicamente se aplicara un análisis sintáctico descendente, para cumplir con el análisis se necesitan seguir una serie de reglas y pasos ya que consiste en evaluar gramáticas independientes de contexto donde las entradas serán procesadas de izquierda a derecha, para que la gramática pueda ser procesada puede que sea necesario realizar algunas operaciones sobre la gramática para cumplir las características de una gramática LL de tipo 1, las GIC permiten la recursividad lo cual significa que puede ser un algoritmo más complejo por las condiciones, por lo que una de las características es que se debe de eliminar la recursividad por la izquierda ya que por la recursión se puede provocar un bucle infinito.

~~1.1~~

Por lo tanto para hacer el ASD se tienen que cumplir con las siguientes dos condiciones:

~ Eliminar Ambigüedades

Las ambigüedades se presentan cuando se le pueden asignar diferentes interpretaciones a las producciones realizadas en la gramática, por ello en el caso de tener una gramática con reglas de producción que permitan ambigüedades, se deberá reescribir nuevamente toda la gramática con el fin de que se eliminen todas las ambigüedades.

~ Eliminar la recursividad por la Izquierda

Para cumplir con esta restricción se tiene que ser muy cuidadoso para entender la derivación la cual debe cumplir con la regla de $A \rightarrow Aa$ teniendo en cuenta que las letras indican valores NO terminales y se espera que se elimine el valor NO terminal de la regla de producción para que de esta manera se evite que haga un ciclo infinito de declaración sobre valores NO terminales en el producto final.

Existen múltiples maneras de adaptar una serie de producciones que forman cuadros terminales, las gráficas como los **árboles de desarrollo** ayudan a facilitar la vista de un desarrollo panorámico del desarrollo de las producciones.

Se hace uso de estructuras de datos como lo pueden ser las pilas para llevar un orden en el proceso al usar los valores acumulados, con ellos se pueden guardar recorridos de árboles o **clásificarse** operandos de operadores para realizar operaciones aritméticas.

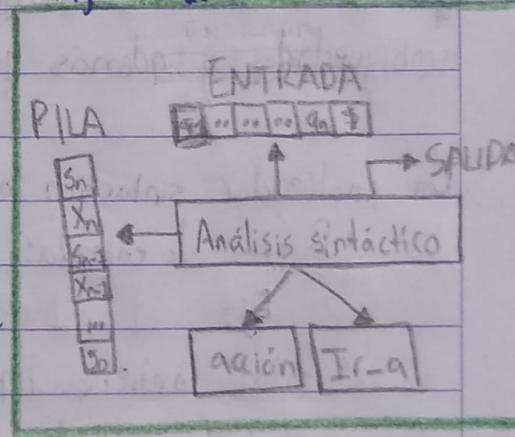
Con este análisis se pretende que lo que se vaya a evaluar se encuentre delimitado entre "**\$ Contenido \$**" en donde se busca reducir el contenido cada vez más, permitiendo así ver de manera detallada cada implementación en la pila de datos y como el contenido delimitado se simplificará de tal manera que como resultado se necesite adquirir únicamente como **Sólida** los delimitadores ya que de haberse realizado correctamente el análisis no debería encontrarse nada ya que todo se canceló (solo de la pila).

3.4.2. Analizador sintáctico ascendente (CLR, LALR)

La diferencia entre LALR y LR tiene que ver con el generador de tablas. Los generadores de analizadores sintáticos LR realizan un seguimiento de todas las posibles reducciones de estados específicos y su conjunto de precisión termina con estados, en los que cada reducción está asociada al contexto izquierdo. Los generadores de analizadores sintáticos LALR están puestos a combinar estados para generar reducciones sin conflicto, esto produce un número considerable menor de estados. Esto nos deja una diferencia muy en claro, los analizadores LR pueden analizar un conjunto de lenguajes más grande que los analizadores LALR pero generando tablas mucho más grandes.

Diagrama del analizador

El analizador sintáctico LR es el más utilizado en la construcción de compiladores, una ventaja de los analizadores LR es que la mayoría de los lenguajes de programación se pueden construir con este tipo de analizadores, es por eso por lo que se convierte en el más utilizado. Este analizador es muy eficiente ya que detecta los errores al momento que son detectados.



El analizador sintáctico ascendente es un tipo de analizador el cuál a diferencia del analizador sintáctico descendente, este genera un **árbol de análisis sintáctico** pero comenzando de distinta manera, este árbol empieza construyéndose de las hojas a la raíz con el fin de que se reduzca la cadena de entrada lo mayor posible, en otras palabras, reducir w al símbolo inicial.

Esta técnica de análisis se puede utilizar para analizar más a profundidad las gramáticas independientes del contexto. Este tipo de analizador sintáctico se utiliza comúnmente cuando algunos problemas no se pueden resolver de forma descendente ya que, la ambigüedad es difícil deuitar.

Árbol de análisis:

Es una representación gráfica de una derivación en el orden de las producciones.

El analizador lleva por nombre análisis sintáctico LR(k), el cual la letra "L" es por la entrada de left-to-right (traducido al español "izquierda a derecha"), la letra "R" significa rightmost derivation (traducido al español "derivación por la derecha"), la letra "K" es por el número de símbolos de entrada los cuales sirven para tomar decisiones del análisis sintáctico. Para que el analizador sintáctico ascendente utilice es necesario que no presente ambigüedades y además que también tenga el valor de K más pequeño.

Un analizador sintáctico ascendente LR cuenta con los siguientes 3 componentes:

- Buffer de entrada: Es la cadena que se va a analizar y procesar.
Ej: id = 1 + contador;
- Pilas: Contiene información de los elementos registrados, como los símbolos de la gramática terminales y no terminales.
Ej: Pila de análisis Sintáctico

1 \$0 3 \$OE1

2 \$On2 4 \$OE1+3.

- Tabla de análisis: Se divide en 2 partes, la tabla de acción y la tabla de Ira.

- Tabla de acción: Esta tabla contiene los estados del autómata y en las columnas se encuentran los símbolos terminales de la gramática.

Ejemplo tabla de acción:

Estado	Entrada				
	if	else	otro	\$	
0	s4		s3		
1				aceptar	
2		r1		m1	

- Tabla de Ir a: Esta tabla contiene los mismos datos que la tabla de acción pero a diferencia que esta contiene los símbolos de estado para saber cuál es el siguiente estado después que el analizador realice una reducción de la cadena.

Ejemplo tabla Ir a:

Ir a	
5	1
1	2
5	2

Características

- Se pueden construir analizadores sintácticos LR para reconocer todos los lenguajes de programación.
- El método de análisis sintáctico LR es el método más conocido, esto por su eficiencia, desplazamiento y reducción sin retroceso.
- Las gramáticas que se analizan en el analizador sintáctico LR es un superconjunto de las gramáticas de los analizadores predictivos.
- Se generan estructuras llamadas árbol de análisis o de derivación, los cuales son para representar el análisis.

Ejemplo análisis sintáctico ascendente LR:

Reducción Regla

(b)+b

(T)+b T→b

(A)+b A→T

T+b T→(A)

A+b T→b

A A→A+b

S S→A

Análisis sintáctico ascendente LALR.

El análisis llamado LALR (Look Ahead LR) es el método más utilizado cuando se habla de los generadores automáticos como CUP o Yacc. El análisis LALR se basa en el análisis LR, simplificando el tamaño del autómata unificando todos los estados por medio de principios del análisis.

Este método consigue reducir el número de estados pero un punto en contra de este método, es que no todas las gramáticas del análisis LR puede ser reconocidas con un analizador LALR, provocando conflictos como reducción/reducción en este tipo de gramáticas.

El análisis LALR cuenta igual que el análisis LR con 3 componentes: Buffer de entrada, pila y la tabla de análisis, la tabla de análisis se construye de igual forma que en el análisis LR, por medio de la tabla de acción y de Ira, esta tabla nos debe quedar al final más pequeña que la tabla de LR, pues como ya se mencionó el análisis LALR consigue reducir el número de estados, es una buena práctica comparar las tablas de los dos tipos de análisis y notar diferencias.

3.5. Diseño y administración de una tabla de símbolos.

La tabla de símbolos se crea durante la fase de análisis léxico pero durante el proceso del analizador pueden sufrir algunas modificaciones, así obteniendo una nueva tabla de símbolos, es común que se agreguen valores de tipo y significado para el análisis sintáctico.

Generalmente se agregan valores de tipo con su significado para el análisis sintáctico.

Ejemplo tabla de símbolos

NOMBRE	TIPO	DIRECCIÓN
:	í	:
I	Entero	0258
J	Entero	0259
K	Entero	0210
:	:	,

Características

- Almacena los nombres de todas las entidades de forma estructurada en un solo lugar.
- Sirve para la verificación de la creación de entidades evitando que haya 2 entidades con el mismo nombre y evitando la ambigüedad.
- Comprueba el tipo de elemento para así comprobar que el código sea semanticamente correcto.
- Para determinar el alcance de los elementos también es utilizada esta tabla.

Las tablas de símbolos pueden ser integradas de distintas formas dependiendo del tamaño de la información que se necesite almacenar, para información

pequeña, la información puede ser implementada por medio de una lista desordenada, en cambio, si una información es más grande se puede implementar la tabla de símbolos de las siguientes formas:

- Lineal (ordenadas o desordenadas).
- Árbol de búsqueda binaria.
- Tabla Hash.

Cuál sea la manera de implementar la tabla de símbolos, se debe implementar 2 acciones para la manipulación de la información, la acción para insertar y la acción para buscar, la acción de insertar para agregar nuevos símbolos y llenando nuestra tabla de símbolos con nuevos elementos, y la acción de buscar cuando el análisis ocupe la información como para verificar al momento de agregar elementos que la información no se duplique.

3.6. Manejo de errores sintácticos y su recuperación.

Un punto importante que hay que tener en cuenta primero que todo al utilizar un analizador sintáctico, es que ningún método de recuperación de errores resuelve todos los problemas, siempre se trata de hacer que el método sea lo más que se pueda de eficiente pero siempre se van a presentar anomalías.

Los errores sintácticos son aquellos errores que tienen distribución errónea de los elementos de una frase, por ejemplo, una expresión aritmética con sus parentesis mal equilibrados o también olvidarse de colocar el famoso punto y coma (;) al final de una instrucción.

Ejemplo de error sintáctico

```
public static void m
{
    n
    return suma;
}
```

La mayoría de los errores se encuentra en la fase del análisis sintáctico, el manejador de errores cuando detecta un error debe informar inmediatamente la presencia del error, informando claramente cuál es el error, la exactitud donde se encuentra y que tipo de error es, estos parámetros son importantes que se muestren al usuario ya que, para el análisis es muy fácil saber toda esta información pero para el usuario hay que gestionar la información para que la pueda entender y así pensar en una solución para el problema. Otro punto importante que debe tener un manejador de errores es que no se debe retrasar el procesamiento de los programas, su análisis de errores debe ser lo más rápido posible sin afectar la eficiencia del análisis para no causar demora del tiempo y hacer pensar al usuario que se está tardando porque se presentó un error cuando no es así, además que el tiempo del usuario debe ser nuestra prioridad.

Las especificaciones actuales de los lenguajes de programación no describen cómo es que un compilador debe responder a la presencia de errores, la respuesta debe ser construida por el diseñador del compilador. Desde un principio se debe pensar en la estructura del manejador de errores para simplificar tanto como la estructura del compilador como la respuesta de los errores.

Los errores que puede presentar un programa son muy diversos además de solo los errores sintácticos, por ejemplo, los errores pueden ser:

- Léxicos: Escribir mal las palabras, etc.
- Semánticos: Aplicar caracteres a otro tipo de datos.
- Lógicos: Los problemas relacionados a la mala implementación del código por parte del usuario.

Compilador: Es un programa que traduce un código escrito en un lenguaje de programación a otro lenguaje para ser reconocido por la máquina.

Una razón del surgimiento de errores es por la naturaleza sintáctica o también cuándo la cadena de componentes que proviene del análisis léxico desobedece las reglas gramaticales que definen al lenguaje de programación. Otra razón también es la precisión de los métodos de los análisis sintácticos, que pueden detectar la presencia de errores de una forma muy eficiente pero la detección exacta de la presencia de errores semánticos y lógicos son muy difíciles de detectar al momento de la compilación.

Existen varias estrategias para la recuperación de errores:

- Modo pánico.
- Nivel de frase.
- Producciones de error.
- Corrección global.

Modo pánico.

Este tipo de recuperación es el más sencillo de utilizar, el analizador sintáctico recorre todos los componentes hasta encontrar un carácter que le informe que debe de parar, mejor conocido como carácter de sincronización, los cuales son como el punto (.), punto y coma (;), entre otros más caracteres. Estos caracteres de sincronización deben de ser definidos por el diseñador del compilador a la hora de la construcción y deben ser seleccionados para que se adesen al lenguaje fuente.

Aunque la recuperación de errores por modo pánico omite una gran cantidad de entrada sin comprobar, de existencia de errores extras, esta estrategia tiene la ventaja de la sencillez y está garantizado contra cadenas infinitas. Esta recuperación se utiliza en situaciones en donde son raros los errores en una misma entrada.

Recuperación de errores a nivel de frase.

La recuperación de errores a nivel de frase se utiliza para la corrección de caracteres adyacentes, esto utilizando operaciones como la inserción, eliminación o intercambio de caracteres. Esta recuperación no es muy utilizada en la mayoría de los problemas ya que, tiene muchas deficiencias en su utilización.

Un ejemplo de la corrección de este tipo de recuperación es sustituir una coma (;) por un punto y coma (;), suprimir un punto, insertar un punto o una coma faltante. La elección de la corrección queda en manos del diseñador del compilador. Algo importante que hay que recalcar es que se debe evitar sustituciones que dirigan a lazos infinitos ya que crearía errores en este tipo de recuperación.

Recuperación de errores por producciones de error.

Este tipo de recuperación trata de la generación de gramáticas para así producir producciones de error, trata de encontrar un fin a cada una de las reglas gramaticales para después generar el error y así seguir con el proceso de recuperación. Estas producciones de error sirven para indicar más fácilmente la construcción errónea reconocida en la entrada.

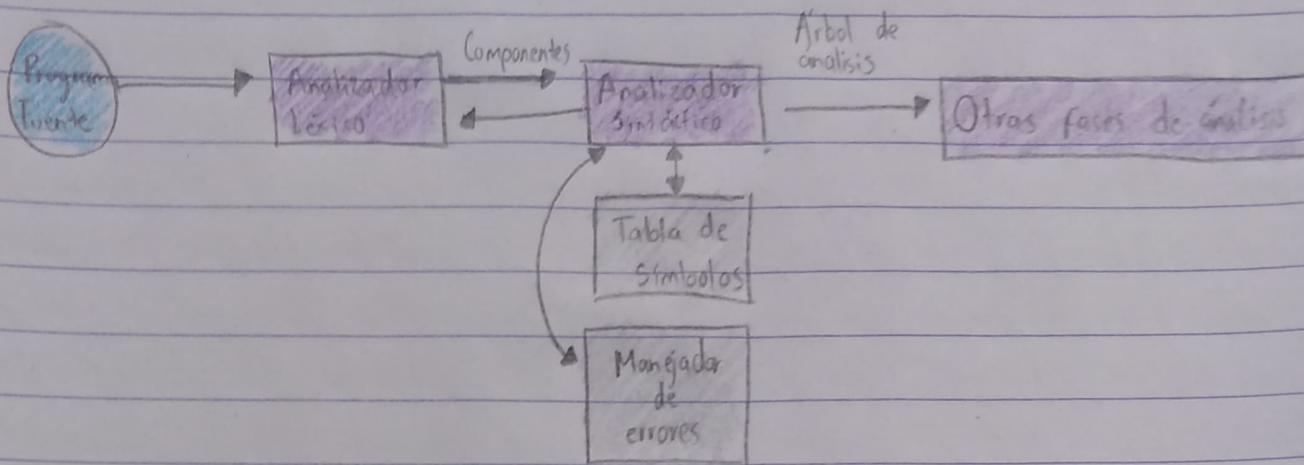
Recuperación de errores por corrección global.

La recuperación por corrección global trata de hacer el mínimo de cambios para que sea posible procesar una entrada, esto nos daría como resultado menores costos para realizar cambios y obtener un resultado correcto, pero también nos llevaría más tiempo y espacio llevar a cabo esta estrategia porque tienen que ser bien pensadas para no generar ambigüedades.

Dada una cadena de entrada incorrecta y la gramática, este tipo de recuperación

ración encuentra un árbol de análisis sintáctico para una cadena relacionada, y las inserciones, supresiones y modificaciones de los componentes necesarios para transformar la cadena errónea sean los mínimos posibles.

¿Cuáles son los objetivos y funciones de un analizador sintáctico?



El principal objetivo del analizador sintáctico es detectar los errores de tipo sintáctico, mostrando al usuario un mensaje para que el usuario resuelva el error y se vuelva a evaluar para saber si la cadena pasa a la siguiente fase de análisis.

Primero el programa fuente pasa a ser evaluado en el analizador léxico, el cuál es el encargado de evaluar y pasar los componentes del programa para su evaluación en el análisis sintáctico, el cuál con ayuda de una tabla de símbolos y un manejador de errores arroja un resultado, en este caso representado por un árbol de análisis sintáctico, el cuál nos sirve para ver los símbolos que contiene la parte evaluada y así pase a otra parte del análisis evitando que el programa no sea ejecutado o sea ejecutado conteniendo errores.

La tabla de símbolos se utiliza para poder comparar los elementos del programa para comprobar que estén sintácticamente correctos, mientras que el manejador de errores se utiliza para controlar la información que se mostrará en los errores.

3.7 Generadores de Código para analizadores Sintácticos: YACC, BISON.

YACC.

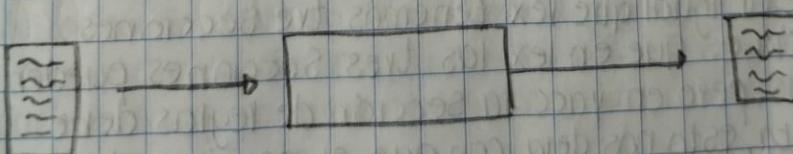
Funcionamiento.

En actividades anteriores habíamos visto acerca del funcionamiento de lex, en esta ocasión hablaremos de su homólogo en Sintaxis, Yacc. Este como tal no es un analizador, más bien, es un generador de analizadores.

Partiendo de un archivo en yacc se generará un archivo con código fuente en lenguaje C que contiene el analizador sintáctico.

Yacc por si sólo no puede funcionar, ya que este necesita de un analizador léxico, dicho de otra forma, el código fuente en C que yacc genera, contiene llamadas a una función `yylex()` que debe estar definida y el mismo tiene que retornar el tipo de lexema que ha sido encontrado.

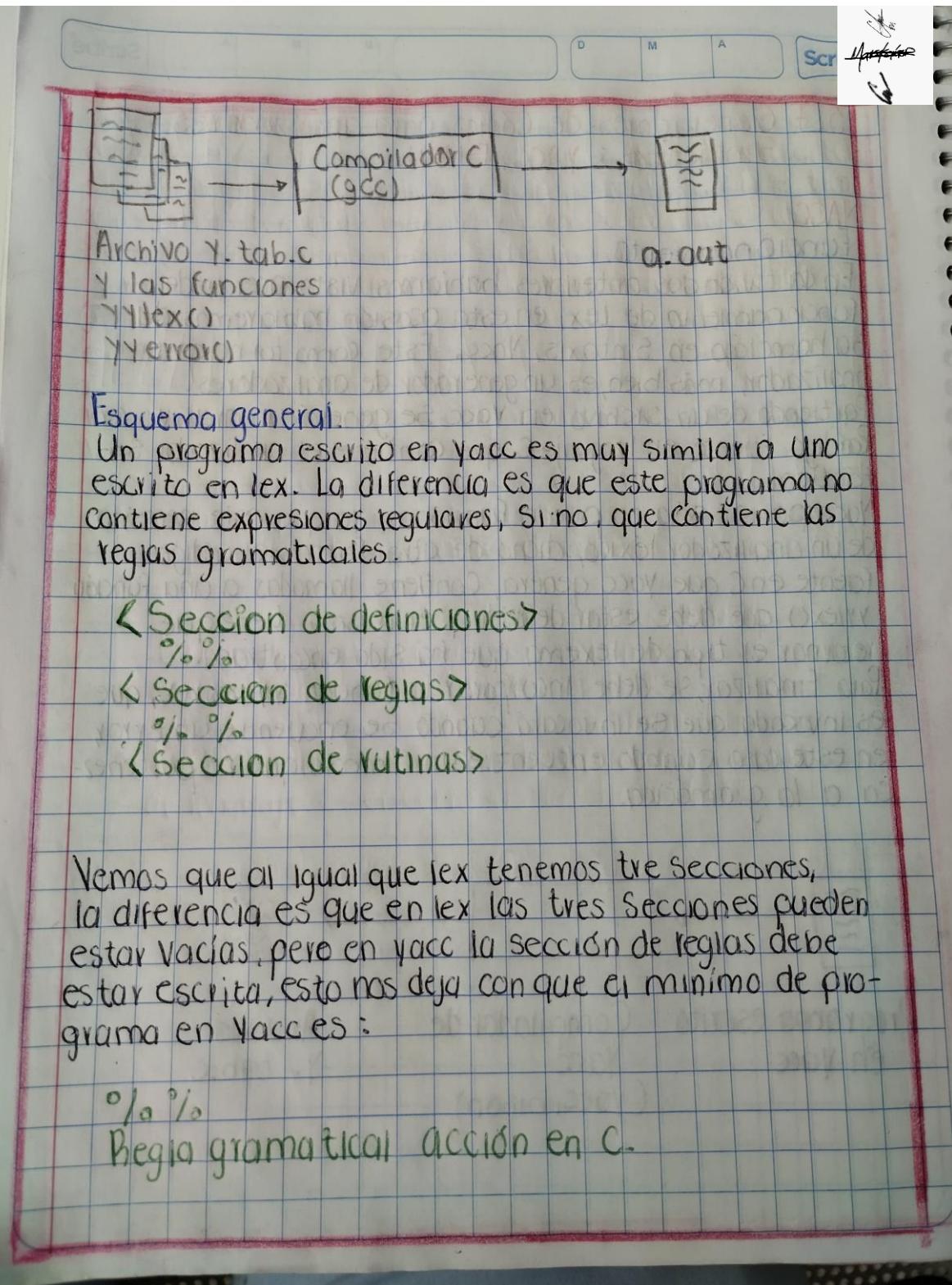
Para finalizar, se debe incorporar la función `yyerror()`, que es invocada que se invocará cuando se encuentre un error, en este caso cuando encuentre un símbolo que no pertenezca a la gramática.



Programa escrito
en yacc

Compilador de
Yacc
(yacc, bison)

Archivo
y.tab.c



~~LEX~~

La sección de declaraciones como ocurría con lex puede incluir varias cosas, en lex se utilizaban para declarar expresiones regulares, en yacc es usada para declarar los símbolos terminales de la gramática mediante la directriz %token. Todo lo que no sea símbolo terminal es considerado como símbolo NO TERMINAL.

% token IF, ELSE, FOR, WHILE

La sección de reglas, como su nombre lo dice contiene las reglas que deberá seguir la gramática. Componentes es una combinación de terminales y no terminales.

no-terminal : Componentes {acciones en C}

La sección de rutinas tiene la misma función que la de lex, yacc no define por defecto la función main(), yylex() o yyerror(), así que aquí debemos de incluirlas.

Yacc genera la función yyparse() que en si contiene el analizador sintáctico, la función principal de la función es servir como una máquina de estados, esto para reducir el archivo de entrada al símbolo inicial de la gramática.

Si todo marcha correctamente, yacc volverá sin error, en caso contrario se invocará la función yyerror().

Sección de reglas.

Una regla en yacc especifica la gramática. La estructura de una regla es la siguiente, en la parte izquierda se tiene un símbolo terminal, seguido de los componentes, es decir la combinación de símbolos terminales y no terminales y por último cuando yacc detecte que la regla se cumple correctamente deberá ejecutar acciones en C.

Simbolo - result : Componentes acción en C.

- El simbolo resultado debe estar situado en la primera posición de la linea; no debe tener ningún espacio en blanco antes de este.
- Los componentes son una combinación de terminales y no terminales, deben estar separados por un espacio en blanco.
- Las acciones en C son instrucciones nativas de lenguaje C encerradas en llaves {}.

Bison

Es un programa que genera un analizador sintáctico, este lenguaje es de propósito general, lo que quiere decir que usado para un único propósito, el proyecto Bison es un GNU disponible para todos los sistemas operativos. Al igual que Yacc, este no puede usarse por sí solo necesita de un generador de análisis léxico, en este caso deflex aunque se puede con cualquier otro analizador léxico.

Bison fue creado por Robert P. Corbett, está basado en C, la función de Bison es transformar una gramática libre del contexto a un programa en C, C++ o Java que se encargará de hacer un análisis sintáctico.

Es usado para crear analizadores en muchos lenguajes, se utiliza para hacer varios proyectos, desde simples calculadoras revisando la sintaxis de las operaciones, hasta realizar programas más complejos. Se recomienda que para utilizar Bison se tenga algo de experiencia con sintaxis para escribir gramáticas.



~~class~~
~~for~~

Bison es compatible con Yacc, las gramáticas que funcionan en Yacc funcionan correctamente para Bison, sin tener que hacer algún tipo de modificación, esto quiere decir que si usted está familiarizado con Yacc también lo estará con Bison.

Como se mencionó anteriormente este programa fue escrito por Robert Corbett pero Richard Stallman hizo que el mismo fuera compatible con Yacc y por último Wilfred Hansen proveniente de la Universidad de Carnegie Mellon añadió Soporte para literales multicaracter y otras características.

~~Análisis~~ ~~descendente~~ → ¿Cómo funcionan los analizadores descendentes?

Un analizador descendente cumple con una serie de características que lo constituyen para evaluar una expresión. Cuando se hablan de analizadores sintácticos debemos tener en cuenta que se pueden clasificar en ascendentes y descendentes, estas diferencias entre análisis son muy notorias ya que cada una establece reglas de construcción muy específicas que hace que el proceso si bien pueden tener similitudes la forma en la que se desarrollan los análisis es literalmente opuesta uno del otro.

Las reglas que componen un analizador descendente son las siguientes:

- Se busca una derivación por la izquierda para una cadena de entrada
- Para ser evaluado de manera gráfica por un árbol de análisis sintáctico tenemos que para este método se construirá paulatinamente comenzando desde la Raíz y creando los nodos conforme se hacen las producciones.

- Se deberá eliminar la recursividad de la izquierda para evitar ciclos infinitos con valores no terminales, una gramática es recursiva por la izquierda si tiene un terminal A tal que existe una derivación $A \rightarrow Aa$, esto no es permitido por lo que se pueden usar métodos para eliminar la recursividad por ejemplo;

$$A \rightarrow Aab$$

Se puede sustituir por

$$A \rightarrow b A'$$

$$A' \rightarrow a A' \mid E$$

En este caso se pueden seguir generando las cadenas pero ya no son recursivas por la izquierda, con esto evitamos que se generen ciclos infinitos.

- El modelo para desarrollar el análisis con sus componentes es el siguiente

Entrada

Pila

EXPRESION

← Delimitadores

A
B
C
D
\$

← Programa de
análisis
Sintáctico

"Evaluaciones"

Salida

\$ #

Scribe

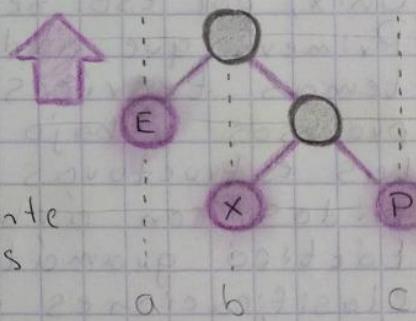
¿Cómo funcionan los analizadores ascendentes?

Como ya se mencionó previamente el analizador ascendente forma parte de uno de los dos tipos de analizador sintáctico junto con el descendente.

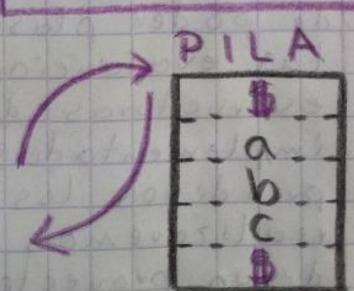
Ambos analizadores tienen sus peculiaridades ya que la manera en la que se desarrollan son totalmente opuestas al realizar los recorridos siendo que ambos comparten las mismas reglas pero por ejemplo la manera en la que se desarrollaría un árbol de análisis sintáctico, como ya se mencionó en los analizadores descendentes, estos para desarrollar su solución comienzan desde la raíz para crear las ramificaciones y hacer las funciones necesarias para que a partir de las producciones se generen las distintas ramas que contienen los nodos que conformaban la cadena y generaron por último los nodos hojas que forman parte de los nodos sin hijos, lo cual nos indica por naturaleza la obtención de los valores terminales de una cadena y nos asegura que se terminan las iteraciones en esa ramificación.

~~Nota~~
Las reglas que se deben seguir si bien son más cortas puede resultar en casos más complejos teniendo en cuenta los siguientes factores:

- Para la construcción de nuestro árbol de análisis sintáctico la cadena de entrada que estaba siendo evaluada comenzaría a realizar el análisis desde los nodos hojas, que son aquellos que no tienen hijos y por consiguiente al ser el último proceso de iteración deberán contener únicamente **símbolos terminales** para que al final la pila de datos en donde se almacenan los registros temporales al final de las iteraciones únicamente cuente con los símbolos delimitantes **\$**.



- La dirección en la que se realizan las derivaciones es por la derecha y en sentido **inverso** buscando con esto simplificar la cadena para poder llegar a la **raíz**.



CÓMO FUNCIONA LOS ANALIZADORES SINTÁCTICOS YACC

Como ya vimos en este mismo trabajo el analizador sintáctico Yacc provee de una herramienta general para analizar estructuralmente una entrada. Recordemos la estructura de los programas escritos en yacc.

- Un conjunto de reglas que describen los elementos de entrada.
- Un código que cuando reconoce que se cumple una regla ejecuta un conjunto de instrucciones.
- Una o más subrutinas para examinar la entrada.

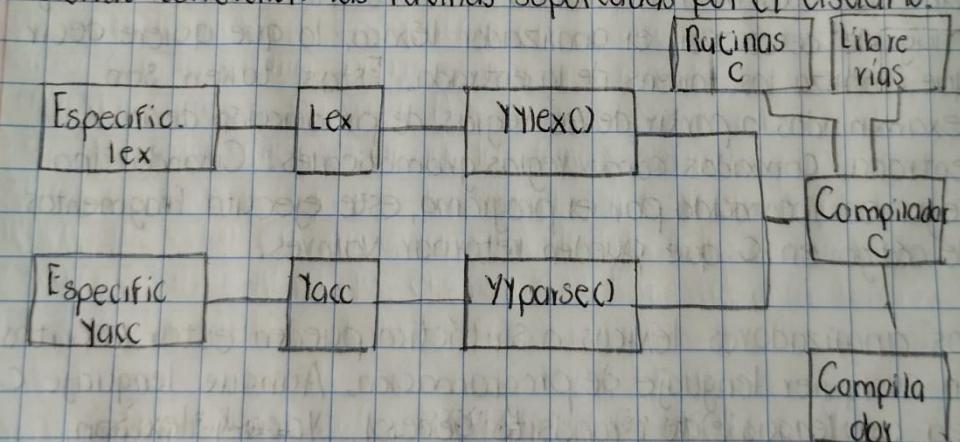
Yacc a partir de una función en C convierte la especificación que examina la entrada de datos. Esta función trabaja muy de cerca con el analizador léxico, lo que quiere decir que analiza los tokens de la entrada. Estos token son examinados a partir de las reglas de producción de la entrada, conocidas como reglas gramaticales. Cuando una regla es reconocida por el programa, este ejecuta fragmentos de código en C que pueden retornar valores.

Los analizadores léxicos o sintácticos pueden estar escritos en cualquier lenguaje de programación. Aunque lenguaje C sea un lenguaje de propósito general, Yacc y lex son más flexibles y por lo tanto más fáciles de usar.

Lex genera un código C para un analizador léxico a diferencia de Yacc que genera un código para un parser.

Tanto Lex y Yacc se alimentan de un archivo de especificaciones que es más fácil de entender y de leer. La extensión de un archivo Lex y Yacc Son .l y .y respectivamente. La salida de estos dos analizadores es un código nativo de lenguaje C. Yacc Crea una rutina llamada `Yyparse` en un archivo llamado `Y.tab.c`

Estas Subrutinas combinadas con el código fuente en C que es provisto por el usuario, que se encuentra en un archivo separado pero se puede utilizar en el archivo de especificaciones de Yacc. Las rutinas antes mencionadas deben ser compiladas, en casi todas las casas las librerías Yacc y Flex son las que se encargan del proceso de compilación. Estas librerías contienen las rutinas soportadas por el usuario.



Por lo general un programa puede basarse en una gramática libre de contexto. La notación más usada para representar una gramática libre de contexto es BNF. En Yacc se hace uso de esta notación.

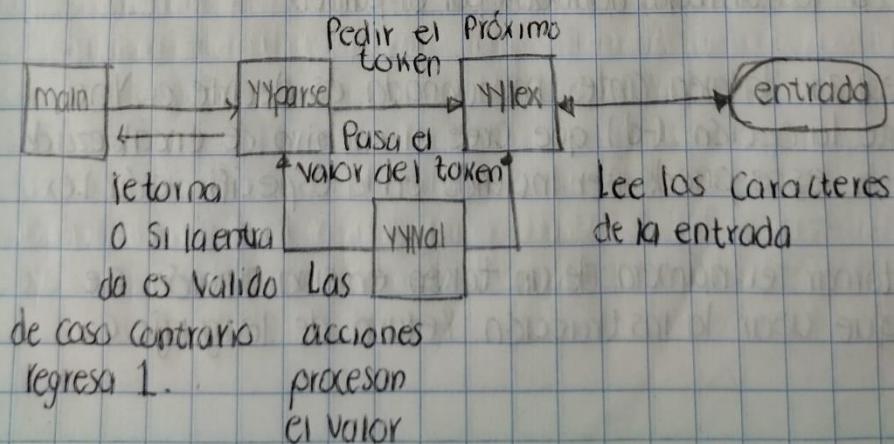
La forma para representar esta notación es de la siguiente manera, consta de un nombre de no terminal, del lado izquierdo, Seguida por su respectiva definición, del lado derecho tenemos los símbolos no terminales que apuntan a otras reglas o hacia otras terminales que corresponde a los tokens. Ejemplos:

lista \leftarrow objeto | lista objeto
Objeto \leftarrow Cadena | numero
Cadena \leftarrow texto | comentario | comando.

Interacción entre las rutinas Léxicas y Parsing.

La rutina main invoca a la función yyparse para determinar si la entrada es una entra valida o no lo es. Esta última rutina mencionada llama a otra Yylex() cada vez que necesite un token. Yylex() lee la entrada y para cada vez reconoce un token, regresa el número del mismo al parser. También se puede pasar el valor del token en forma de una variable externa yyval.

Por último, las rutinas pueden llamar a otras funciones que fueron definidas por el usuario en la sección de código.



Tokens.

La función principal de un analizador léxico es detectar un token y regresar el número correspondiente si ese token o de caso contrario regresar un valor indicando que el token no es válido.

Las especificaciones de lex consisten en sentencias de lenguaje C que como mencionamos retorna el token y su valor.

Quien se encarga de asignar un número a cada token es Yacc cuando este procesa los tokens declarados en las especificaciones. La sentencia llamada define es usada para definir los números de tokens.

#define NUMERO 257.

Las definiciones se encuentran en el archivo Y.tab.c junto con la rutina yyparse, hay que recordar que cada carácter contiene un valor ASCII cuyo valor abarca de 0 a 256, esto quiere decir que los tokens definidos por el usuario comienzan a partir de 257. El parser y el analizador tienen que estar en sincronía para identificar los mismos tokens, esto significa que el analizador léxico deberá tener acceso a los símbolos definidos por el parser.

Para realizar la tarea antes mencionada es decirle a Yacc, mediante la opción (-d) que cree el archivo de encabezado Y.tab.h, este puede ser incluido en la especificación lex.

Para retornar el número de un token en una acción se tiene que usar la instrucción return de lenguaje C.

Por ejemplo para retornar el valor de los token's que son reconocidas como número Se haría de la siguiente manera

$[0-9]^+$ { return NUMERO; }

Para enviar el valor del token al parse, se hace uso de una función llamada `YYtext`, que contiene la cadena de caracteres que serán reconocidas por la expresión regular. La variable `YYval` es sentada por Yacc para enviar el valor del token desde el analizador léxico hacia el parser. `YYval` es una variable de tipo entera y `YYtext` es una variable de tipo `Char`, por lo que se debe hacer un cast explícito para convertir el `Char` a su respectivo valor `int`. Por ejemplo, podemos usar la función `atoi`.

$[0-9]^+$ {

`YYval = atoi(YYtext)`

return NUMERO;

CÓMO FUNCIONAN LOS ANALIZADORES SINTÁCTICOS BISON

Por lo general los archivos Bison tienen la extensión .Y la cual describe una gramática. El archivo ejecutable que se genera indica si el archivo de entrada con el cual es alimentado Bison pertenece o no al lenguaje por el cual fue generado. Escribir una gramática en Bison es muy similar a yacc. Ejemplo:

① 1

declaraciones en C

② 3

declaraciones de Bison

③ 0

④ 0

⑤ 0

legías gramáticas

⑥ 0

Código C adicional.

Las declaraciones en lenguaje C pueden definir tipos y variables que serán utilizadas por las posteriores acciones. Se pueden utilizar algunos Comandos del micro procesador para definir macros que serán utilizadas allí o también se puede hacer uso de #include para importar librerías desde la cabecera.

En las declaraciones Bison se declaran los símbolos terminales y no terminales, es usado para definir la procedencia de operadores y el tipo de datos de los valores Semánticos de varios Símbolos.

[Handwritten notes: 'Símbolos terminales' and 'no terminales' with arrows pointing to the text]

Las reglas de producción de la gramática pueden llevar asociadas acciones, acciones en lenguaje C que son posteriormente ejecutadas cuando se encuentra que la regla se cumplió correctamente.

Por último, el Código C adicional puede tener fragmentos de código escrito en el mismo lenguaje. Por lo general esta parte es utilizada para realizar definiciones del analizador léxico `yylex`, más subrutinas son invocadas en las reglas gramaticales. Si se trata de un programa simple, el mismo puede ir aquí.

Símbolos terminales y no terminales.

Los símbolos terminales también son conocidos como tokens y tienen que ser declarados en la sección de definiciones. Por seguir un estándar estos suelen ir escritos en mayúsculas, mientras que los símbolos no terminales en minúsculas.

Los nombres de estos símbolos pueden letras y dígitos aunque estos últimos no pueden ir al principio; guiones y puntos. Estos últimos sólo tienen sentido en los símbolos no terminales.

Para escribir un símbolo terminal de la gramática hay tres maneras de hacerlo, aunque aquí sólo exemplificaremos dos:

- Un token se tiene que describir como un identificador, de igual manera que se hace en lenguaje C. Como se mencionó antes deben estar en mayúsculas. Se tiene que usar la declaración de `%token` para cada nombre.

- Token de carácter usan la misma Sintaxis usada en C para las constantes de un carácter, ejemplo, '-' es un tipo de token de carácter. Este tipo de tokens no necesitan ser declarados al menos que necesite especificar el tipo de datos de su valor semántico.

Sintaxis de las reglas gramáticas.

Cualquier regla gramatical escrita en Bison debe seguir la siguiente forma.

Resultado: Componentes ...

donde el resultado representa el símbolo no terminal que describe esta regla, Componentes son todos aquellos símbolos terminales y no terminales que estén unidos por la misma regla, ejemplo:

exp : exp '+' exp

Explicando el ejemplo anterior, tenemos el símbolo terminal llamado 'exp' que sigue la siguiente regla de producción $\text{exp} + \text{exp}$, donde '+' es un símbolo no terminal, esto quiere decir que podemos llamar a la misma regla, como si de recursión se tratase.

Los espacios en blanco no se toman en cuenta, pues son únicamente usados para delimitar a cada símbolo de otro.

En el medio de los componentes podemos encontrar acciones que nos pueden ayudar a determinar la Semántica de la regla. Una acción aparece de la siguiente forma:

¿Sentencias en C?

Por lo general solo hay una única acción que sigue a los componentes. Por lo que podemos escribir varias reglas por separado para el mismo resultado o también estas pueden unirse entre sí con la barra vertical ('|'), ejemplo:

Resultado: Componente- r_1 ...
| Componente- r_2 ...
... ;

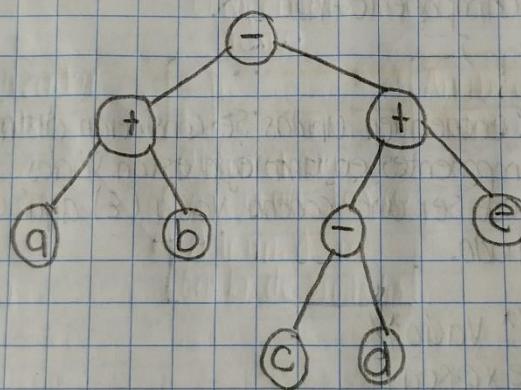
Incluso aunque se encuentren unidas se consideran distintas reglas. Si los componentes en una regla están vacíos significa que el resultado puede ser una cadena vacía (ϵ) aquí un ejemplo de lo antes mencionado.

expseq: /* Vacío */
| expseq1;

expseq1: exp1
expseq1 ',' exp
;

Para cuestiones de buenas prácticas de programación se usa el comentario vacío para señalar que se trata de una cadena vacía (*/* vacío */).

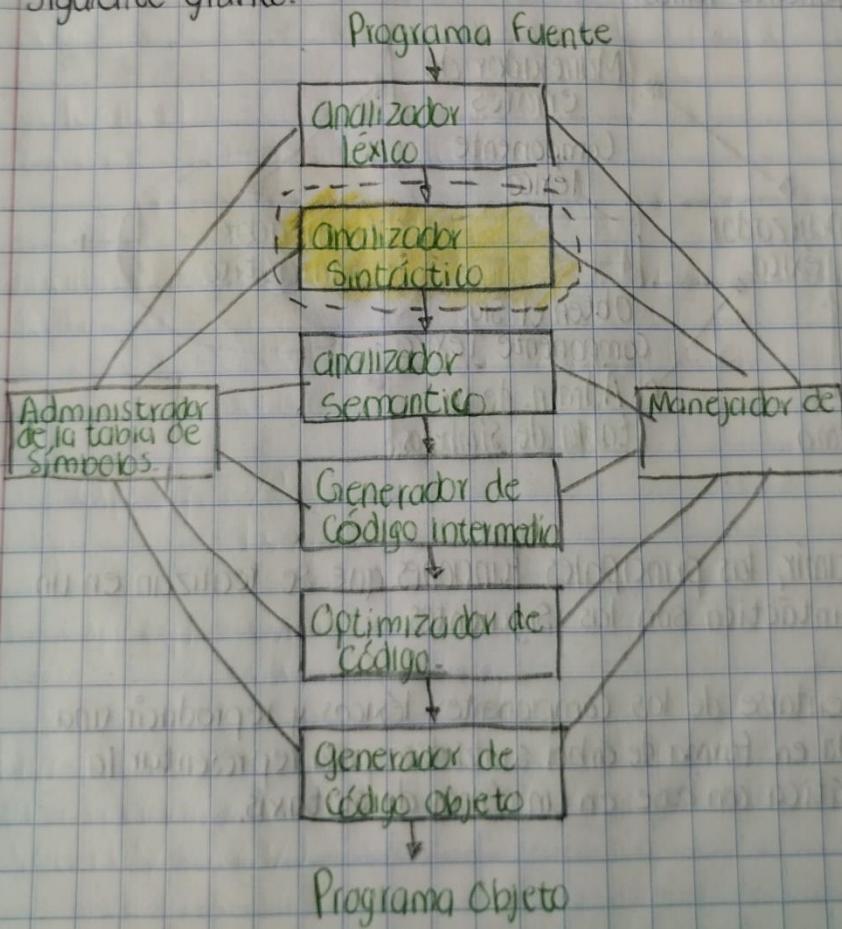
Se dice que una regla es recursiva cuando su no-terminal resultado aparece también de lado derecho. Como en programación una función se llama así misma. Por lo general en Bison todas las reglas gramaticales hacen uso de la recursión ya que es la única manera de definir Secuencias. Por ejemplo en Java se hace uso de la recursión cuando se crea un árbol binario ya que cada nodo puede tener dos hijos como máximo, esto de los árboles también es usado de cierta manera por los analizadores sintácticos, pues permiten ver si una cadena está escrita correctamente. Aquí un ejemplo de esto último.



Expresión : $(a+b)-((c-d)+e)$.

CARACTERÍSTICAS Y FUNCIONES DE LOS ANALIZADORES SINTÁCTICOS

La función principal de estos es más que Obvia, pues viene en su nombre, la cual es analizar que una cadena de entrada esté bien escrita Sintácticamente hablando. Aunque esta función sólo es parte de una lista de tareas que se deben realizar para llevar el proceso de compilación. Como se muestra en el siguiente gráfico.



Debemos notar que la Segunda Fase por la que pasa el Proceso de compilación es el análisis sintáctico. La principal función de estos analizadores es analizar la secuencia con la que fueron ingresados los tokens, todo esto con el fin de encontrar que se ha cumplido la regla gramatical correspondiente. Esto se logra gracias al trabajo conjunto que hace el analizador léxico y sintáctico. El analizador léxico recibe y lee todos los caracteres de entrada hasta que logra identificar el componente léxico.



Para resumir, las principales funciones que se realizan en un análisis sintáctico son las siguientes.

- Alimentarse de los componentes léxicos y reproducir una salida en forma de árbol sintáctico para representar la gramática con base en un árbol de sintaxis.

- Interactuar con la tabla de símbolos. Para mantener todos los símbolos presentes en la entrada.
- Chequear que los tipos de datos coincidan correctamente con el fin de evitar errores y pérdida de datos.
- Sirve para generar un código intermedio, ya sea para un MV o no, para realizar la compilación o interpretación de los archivos de la entrada.
- Informar sobre los errores de la entrada.

Evidencia del video.

<https://youtu.be/W2F00mRIJsM>

2018

D M A

S

Clase
~~Matemática~~
↓

Bibliografía.

- Análisis Sintáctico descendentes y ascendentes - b36269 (2015, noviembre). Sites.google.com. Recuperado 6 de octubre de 2022 de

<https://sites.google.com/site/b36269/automatas/resumenes/sintactico>

- Informatica LLC. (2018, marzo). Precedencia de operadores. docs.Informatica.com. Recuperado 6 de octubre de 2022 de

https://docs.informatica.com/es_es/data-integration/data-services/10-0-0/precedencia-de-operadores.html

- El generador de analizadores Sintácticos Yacc. (s. f.). Universidad de Valladolid. Recuperado 6 de octubre de 2022 de.

<https://www.infor.uva.es/~mluisa/talf/docs/lab0/L8.pdf>