



INSTITUTO TECNOLÓGICO NACIONAL DE MÉXICO EN CELAYA

Materia : Lenguajes y Autómatas II
ACTIVIDAD 2

Profesor: ISC. Ricardo González
González

Alumnos:

Isaac Salvador Bravo Estrada
2003048

Guillermo Peasland Aguilar 20030737

María del Carmen Chávez Patiño
20030296

Fecha de entrega: 27 de Agosto
de 2024

EQUIPO NO.3



DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

Celaya, Guanajuato, 22 / agosto / 2024

ASUNTO: **SOLICITUD DE ACTIVIDADES**

LENGUAJES Y AUTÓMATAS II

DOCENTE DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ

SEMESTRE AGOSTO-DICIEMBRE 2024

ACTIVIDAD 2 (VALOR 23 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.

1. INTRODUCCIÓN AL DISEÑO DE LENGUAJE DE PROGRAMACIÓN.

- A. ELABORE A MANO, UNA EXCELENTE REDACCIÓN EN LA CUAL EXPLIQUE CON MUCHA CLARIDAD LOS SIGUIENTES TEMAS :
1. ¿ QUÉ ES UN LENGUAJE MÁQUINA ?
 2. ¿ QUÉ ES UN LENGUAJE ENSAMBLADOR ?
 3. ¿ QUÉ ES UN TRADUCTOR DE LENGUAJE ENSAMBLADOR Y QUÉ PRODUCE ?
 4. ¿ CUÁL ES LA RELACIÓN QUE EXISTE ENTRE EL LENGUAJE MÁQUINA Y EL LENGUAJE ENSAMBLADOR ?
 5. ¿ CUÁLES SON LAS CARACTERÍSTICAS TANTO DEL LENGUAJE MÁQUINA COMO DEL LENGUAJE ENSAMBLADOR ?

NOTA : AL DESARROLLAR ESTE PUNTO, NO DEBE PENSAR TRANSCRIBIR CONTENIDO, SINO EN INVESTIGAR Y DESARROLLAR CON LA SUFICIENTE PROFUNDIDAD SU REDACCIÓN. RECUERDE QUE LA CALIDAD Y ORIGINALIDAD DE SU TRABAJO SERÁ EVALUADA PARA ASIGNAR SU CALIFICACIÓN EN ESTE EXAMEN.

- B. DEL SIGUIENTE MATERIAL REALICE LO SIGUIENTE:



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





1. DEFINA A MANO Y CON SUS PROPIAS PALABRAS QUÉ ES UN COMPILEADOR Y ESTABLEZCA SUS CARACTERÍSTICAS.
2. DEFINA A MANO Y CON SUS PROPIAS PALABRAS QUÉ ES UN INTÉRPRETE Y ESTABLEZCA SUS CARACTERÍSTICAS.
3. ANALICE EL VIDEO Y EXTRAIGA TODOS LOS CONCEPTOS IMPORTANTES E INTEGRE ÉSTOS EN UN CUADRO COMPARATIVO EN EL CUAL DEBERÁ AÑADIR ILUSTRACIONES DE SU PROPIA AUTORÍA.
4. ELABORE UN CUESTIONARIO CON 10 PREGUNTAS PLANTEADAS DE FORMA INTELIGENTE Y CON UN ALTO RETO COGNITIVO PARA LOS PUNTOS ANTERIORES.

FUENTE A CONSULTAR : [COMPILEADORES E INTÉRPRETES](#)

C. PERSONAL ACERCA DEL SIGUIENTE CONCEPTO :

- ¿ QUÉ SON LOS GENERADORES DE CÓDIGO PARA COMPILEADORES ?

ESTE PUNTO DEBE ABARCAR:

- ¿ QUÉ SON ?
- ¿ PARA QUÉ SIRVEN ?
- ¿ CUÁL ES SU IMPORTANCIA EN LA ASIGNATURA DE LAII ?
- ¿ CÓMO SE USAN ÉSTOS ?

DESPUÉS REDACTE A MANO LA DEFINICIÓN Y LAS CARACTERÍSTICAS DE LO QUE REPRESENTA EL ACRÓNIMO YACC EN EL ÁMBITO DEL DISEÑO DE LENGUAJES DE PROGRAMACIÓN.

D. REVISE LOS SIGUIENTES DOS VIDEOS Y HAGA LO QUE A CONTINUACIÓN SE INDICA.

- [PART 01: TUTORIAL ON LEX/YACC](#)
- [PART 02: TUTORIAL ON LEX/YACC](#)

DE AMBOS VIDEOS, AHORA REDACTE UN RESUMEN DETALLADO Y A MANO DE LO EXPUESTO EN ÉSTOS.

POR ÚLTIMO EXPLIQUE CÓMO SE DEBE TRABAJAR EN YACC PROponiendo UN EJEMPLO PRÁCTICO DE SU UTILIDAD EN ESTA ASIGNATURA.

SI LO CONSIDERA PERTINENTE, EL SIGUIENTE MATERIAL LE PUEDE SER DE UTILIDAD Y APOYO.

FUENTE A CONSULTAR : [YACC - EL COMPILEADOR-COMPILEADOR](#)





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, SUPONDRÍA 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	: LAII, AC MÁS EL GRUPO (-A, -B, -C)
DOCUMENTO	: A1-ACTIVIDAD 1, P1-PRACTICA 1, R1-REPORTE 1, TI-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.

2024-08-22_TNM_CELAYA_LAII-A_A2_EQUIPO_99_9999999_PEREZ_PEREZ_JUAN_AGO-DIC24.PDF

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

SI EL TRABAJO SE SOLICITÓ INDIVIDUALMENTE.

2023-08-22_TNM_CELAYA_LAII-A_A2_9999999_PEREZ_PEREZ_JUAN_AGO-DIC24.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.





Introducción al diseño de lenguaje de programación.

1. ¿Qué es el lenguaje máquina?

El lenguaje máquina es el sistema de instrucciones más básico y elemental que puede entender y ejecutar un procesador de computadora. Este lenguaje se compone exclusivamente de números binarios (ceros y unos), que representan las operaciones básicas de la CPU que debe realizar, como sumar, restar, mover datos de un lugar a otro, o realizar saltos condicionales.

Cada instrucción en lenguaje máquina se corresponde directamente con una operación específica del procesador, y su ejecución ocurre a nivel de hardware, lo que significa que no hay intermediarios entre la instrucción y su ejecución. Debido a su cercanía con el hardware, el lenguaje máquina es extremadamente eficiente en términos de ejecución. Sin embargo, es también extremadamente difícil de leer, escribir y mantener para los seres humanos, ya que este carece de abstracción y requiere un conocimiento profundo de la arquitectura del procesador.

Por ejemplo: una instrucción en lenguaje máquina podría verse como una secuencia de bits como '10110000 100100001', donde cada bit o conjunto de bits tiene un significado específico para el procesador, como un código de operación y los registros involucrados.

En resumen, el lenguaje máquina es la forma más directa de comunicación entre un programa y el hardware de una computadora, esencial para el funcionamiento básico de todos los sistemas informáticos, pero a su vez, es de difícil manejo para los programadores humanos. Es por esa razón que se han desarrollado lenguajes de programación de más alto nivel, que, aunque menos eficientes en términos de ejecución, permiten a los programadores escribir código de manera más intuitiva y menos propensa a errores.





2: ¿Qué es un lenguaje ensamblador?

El único lenguaje que entienden los microcontroladores es el código máquina formado por ceros y unos del sistema binario.

El lenguaje ensamblador expresa las instrucciones de una forma más natural al hombre a la vez que muy cercana al microcontrolador, ya que cada una de esas instrucciones se corresponde con otra en código máquina.

El lenguaje ensamblador trabaja con nemáticos, que son grupos de caracteres alfanuméricos que simbolizan las órdenes o tareas a realizar.

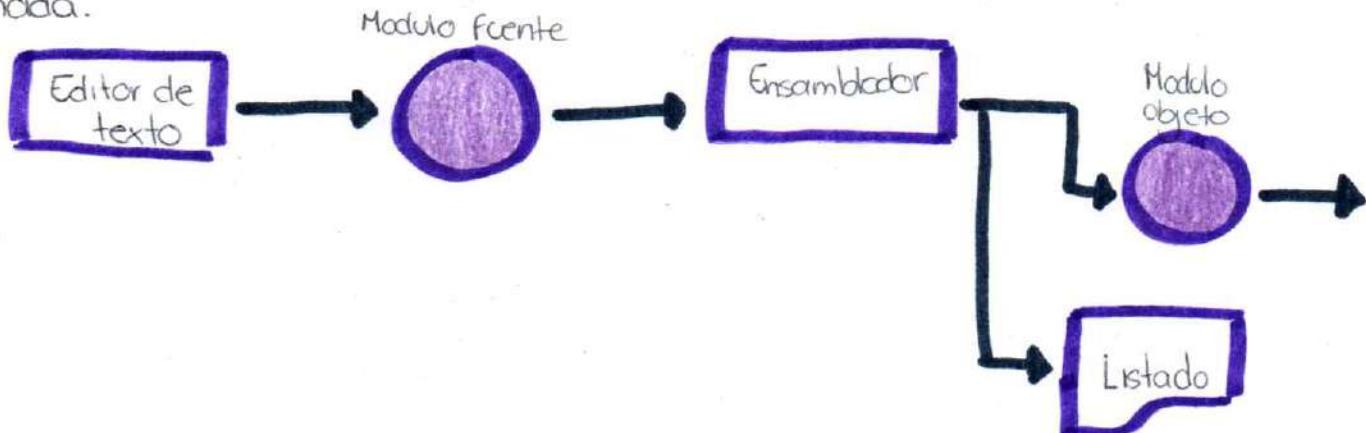
La traducción de los nemáticos a código máquina entendible por el microcontrolador la lleva a cabo un programa ensamblador. El programa escrito en lenguaje ensamblador se denomina código fuente (*.asm). El programa ensamblador proporciona a partir de este fichero el correspondiente código máquina, que suele tener la extensión *.hex.

Campo de código

Instrucciones: son aquellos nemáticos que son convertidos por el ensamblador en código máquina que puede ejecutar el núcleo del microcontrolador. En la gama media (PIC16xxx) cada nemático se convierte en una palabra en la memoria de programa.

Directivas: Pseudoinstrucciones que controlan el proceso de ensamblado del programa, pero son parte del código. Son indicaciones al programa ensamblador de cómo tiene que generar el código máquina.

Macros: Secuencia de nemáticos que pueden insertarse en el código fuente del ensamblador de una manera abreviada mediante una simple llamada.



El lenguaje ensamblador es una herramienta poderosa para los programadores que necesitan maximizar el rendimiento del hardware y tienen un conocimiento profundo de la arquitectura del procesador. Aunque es menos accesible que otros lenguajes de programación debido a su complejidad y especificidad.

< ¿Qué es un traductor de lenguaje ensamblador y >

Al programar, nosotros solamente usamos lenguajes de alto nivel, esto debido a que son mas sencillos de usar y entender para nosotros como humanos al tener instrucciones faciles de recordar e interpretar por nosotros.

Estas instrucciones son lo que llamamos código, pero para que una computadora ejecute lo que le pedimos necesita leer las instrucciones en su propio lenguaje, es decir, lenguaje maquina.

El encargado de este proceso de llevar el código que escribimos a lenguaje maquina no es otra mas que el traductor o mejor conocido como ensamblador.

La principal tarea del ensamblador es convertir el código del lenguaje de programación usado a lenguaje máquina para que así las instrucciones que hemos escrito sean ejecutadas por la computadora.

Pero hay que aclarar que un ensamblador solo traduce programas escritos en lenguaje ensamblador a lenguaje máquina, mientras que el encargado de traducir código de lenguajes de programación de alta nivel son otra tipo de traductores conocido como compilador, y estos tienen la particularidad de mostrar errores dentro del código.

¿Cuál es la relación que existe entre el lenguaje máquina y el lenguaje ensamblador?

El lenguaje máquina y el ensamblador se encuentran estrechamente interconectados debido a que en esencia el ensamblador es una representación simbólica del lenguaje máquina.

El lenguaje máquina es el nivel más bajo en cuanto a las instrucciones que puede entender una computadora, este hace uso de instrucciones en forma de cadenas binarias (0 y 1) las cuales se pueden ejecutar directamente y realizan operaciones fundamentales como mover datos entre registros, operaciones aritméticas y controlar el flujo de un programa.

Mas sin embargo, dado que las instrucciones son cadenas binarias, si no se cuenta con el conocimiento adecuado son imposibles de entender por el humano.

Por otro lado, el diseño del lenguaje ensamblador está hecho para que los humanos podamos comprender mejor las instrucciones.

El lenguaje ensamblador hace uso de mnemónicas y nombres de registros para representar las instrucciones, y cada mnemónico es correspondiente a una instrucción en lenguaje máquina.

Por ejemplo, la instrucción MOV AX, 5 indica que el procesador mover el valor 5 al registro AX y aquí trabaja el ensamblador, el cual es un traductor de ensamblador a lenguaje máquina.

Con todo lo anterior se puede afirmar que su relación es directa, donde cada instrucción en lenguaje ensamblador tiene una única instrucción en lenguaje máquina y viceversa.

Esto da como resultado que en caso de querer crear un programa que interactúe de forma directa con hardware se use el lenguaje ensamblador, pero esto no quita que el lenguaje ensamblador siga siendo complejo ya que se sigue tratando con aspectos a muy bajo nivel como la manipulación de registros.

Pero aun así el lenguaje ensamblador sigue siendo la mejor opción a usar en escenarios donde se requiere de un control preciso del hardware.

CARACTERÍSTICAS DEL LENGUAJE MÁQUINA

- 1- **CÓDIGO BINARIO:** Está compuesto exclusivamente por secuencias de Bits, normalmente en formato binario (0s y 1s).
- 2- **DEPENDENCIAS DE HARDWARE:** Específico para cada arquitectura de procesador (x86, ARM, MIPS, etc).
- 3- **OPERACIONES BÁSICAS:** Las instrucciones en el lenguaje máquina realizan operaciones muy básicas y primitivas, como operaciones aritméticas, lógicas, y de control de flujo.
- 4- **ALTA VELOCIDAD DE EJECUCIÓN:** Dado que las instrucciones del lenguaje máquina se ejecutan directamente por el hardware, la velocidad de ejecución es muy alta.
- 5- **DIFÍCIL LEGIBILIDAD Y MANTENIMIENTO:** Es extremadamente difícil de leer, escribir y mantener por humanos debido a su naturaleza binaria y falta de abstracción.

CARACTERÍSTICAS DEL LENGUAJE ENSAMBLADOR.

- 1- **MNEMÓNICOS EN LUGAR DE BINARIOS:** Utiliza mnemónicos (abreviaturas simbólicas) en lugar de secuencias de bits para representación de instrucciones.
- 2- **RELACIÓN DIRECTA CON EL HARDWARE:** Cada instrucción en el lenguaje ensamblador es más legible y manejable que el lenguaje máquina.
- 3- **MAYOR LEGIBILIDAD:** Aunque todavía es un lenguaje de bajo nivel, el ensamblador es más legible y manejable que el lenguaje máquina.
- 4- **NECESITA ENSAMBLADOR:** Para ejecutar un programa escrito en lenguaje ensamblador, se debe pasar a través de un ensamblador, una herramienta que traduce el código de ensamblador al correspondiente código máquina.

Referencias

Proceso de traducción de los lenguajes de programación. (s. f.). DesarrolloWeb.com.
<https://desarrolloweb.com/articulos/2387.php>

Marker, G. (2022, 12 noviembre). *Lenguaje de máquina*. Tecnología + Informática.
<https://www.tecnologia-informatica.com/lenguaje-de-maquina/>

de Expertos en Ciencia y Tecnología, E. (2016, octubre 27). Conociendo el lenguaje de máquina. *VIU España*. <https://www.universidadviu.com/es/actualidad/nuestros-expertos/conociendo-el-lenguaje-de-maquina>

(S/f-a). Unioviedo.es. Recuperado el 27 de agosto de 2024, de
<https://www.unioviedo.es/ate/alberto/TEMA3-Ensamblador.pdf>

¿Qué es el lenguaje ensamblador (ASM)? (2023, diciembre 5). *Educaopen.com*.
<https://www.educaopen.com/digital-lab/blog/software/lenguaje-ensamblador>

Marker, G. (2022, noviembre 12). *Lenguaje de máquina*. Tecnología + Informática; Tecnología+Informatica. <https://www.tecnologia-informatica.com/lenguaje-de-maquina/>

DEFINICIÓN Y CARACTERÍSTICAS DE UN COMPILADOR

Definición: Un compilador es una herramienta fundamental en el desarrollo de software, cuyo propósito principal es traducir un programa escrito en un lenguaje de programación de alto nivel (por ejemplo, C, C++, Java) a un lenguaje de bajo nivel, específicamente código máquina o ensamblador. Este código de bajo nivel es lo que una computadora entiende y puede ejecutar directamente. A diferencia de los lenguajes de alto nivel, que son más cercanos al lenguaje humano y fáciles de entender para los programadores, el código máquina es una secuencia de instrucciones binarias que el procesador de la computadora puede ejecutar de manera eficiente.

El proceso de compilación no es un simple acto de traducción. Implica múltiples fases, cada una con su propósito específico. Estas fases incluyen:

ANÁLISIS LÉXICO:

En esta primera fase, el compilador toma el código fuente y lo convierte en una serie de tokens. Estos tokens son las unidades mínimas de significado en el código, como palabras clave, operadores y símbolos. El análisis léxico es crucial para la estructura del código.

ANÁLISIS SINTÁCTICO:

También conocido como "parsing", esta fase toma los tokens generados por el análisis léxico y los organiza en una estructura sintáctica conocida como árbol de sintaxis abstracta (AST). El AST representa la estructura lógica del código, basada en las reglas gramaticales del lenguaje de programación.

ANÁLISIS SEMÁNTICO:

En esta fase, el compilador verifica que el código siga las reglas del lenguaje en términos de tipos de datos, uso de variables, y otras restricciones semánticas. Esto asegura que las operaciones realizadas en el código tengan sentido en el contexto del programa.

GENERACIÓN DE CÓDIGO INTERMEDIO:

El compilador traduce el AST en un código intermedio que es independiente del hardware. Este código intermedio es una representación más cercana al lenguaje máquina, pero todavía no está optimizado.

OPTIMIZACIONES:

Esta fase se encarga de mejorar el código intermedio para hacerlo más eficiente. Las optimizaciones pueden incluir la eliminación de código redundante, la reducción del uso de memoria, y la velocidad de ejecución del código final.

GENERACION DE CÓDIGO:

Finalmente, el código intermedio optimizado se traduce en código máquina específico para la arquitectura del sistema en el que se ejecutara el programa.

ENLAZADO:

En algunos casos, el compilador también realiza la fase de enlace, donde el código máquina generado se combina con otras bibliotecas y módulos necesarios para producir un archivo ejecutable.

CARACTERÍSTICAS DEL COMPILADOR.

TRADUCCIÓN COMPLETA Y EFICIENTE:

Un compilador traduce todo el código fuente en una sola pasada, generando un archivo ejecutable. Esta característica lo diferencia de un intérprete, que traducen y ejecutan el código línea por línea. La traducción completa permite al compilador aplicar optimizaciones que mejoran significativamente la eficiencia del programa en términos de tiempo de ejecución y uso de recursos.

OPTIMIZACIÓN DE CÓDIGO:

Los compiladores modernos incluyen técnicas avanzadas de optimización, que pueden transformar el código fuente en un código máquina más eficiente. Esto puede incluir la inlining de funciones, la desenrollado de bucles, la eliminación de código muerto, y más.

DETECCION TEMPRANA DE ERRORES:

Una de las grandes ventajas de los compiladores es que detectan errores en el código fuente antes de que el programa sea ejecutado. Esto incluye errores sintácticos, como la falta de un punto y coma, así como errores semánticos, como operaciones ilegales entre distintos tipos de datos.

GENERACION DE CODIGO INTERMEDIO

Algunos compiladores generan un código intermedio que es independiente de la arquitectura del hardware. Lo que facilita la portabilidad y la optimización del programa.

DEPENDENCIA DEL SISTEMA:

Los programas compilados están diseñados para ejecutarse en una arquitectura específica. Esto significa que el código generado por el compilador está optimizado para el hardware particular en el que se va a ejecutar.

DEFINICIÓN Y CARACTERISTICAS DE UN INTÉPRETE

DEFINICIÓN: Un intérprete es otro tipo de herramienta para la ejecución de programas, pero con una enfoque diferente al de un compilador. En lugar de traducir todo el programa a código máquina antes de la ejecución, un intérprete traduce y ejecuta el código fuente línea por línea en tiempo real. Esto significa que el código se procesa de manera secuencial, lo que permite una retroalimentación inmediata al programador.

A diferencia de los compiladores, los intérpretes no generan un archivo ejecutable inmediatamente. En su lugar, el código fuente debe ser interpretado cada vez que se ejecuta. Esto puede resultar en una ejecución más lenta, ya que la traducción se realiza durante la ejecución, pero ofrece ventajas en términos de flexibilidad y facilidad de depuración.

CARACTERÍSTICAS DEL INTÉPRETE

EJECUCIÓN INMEDIATA:

Una de las principales ventajas de un intérprete es su capacidad para ejecutar código inmediatamente después de que se escribe. No es necesario compilar el código en un archivo ejecutable, lo que permite una retroalimentación rápida durante el desarrollo.

FLEXIBILIDAD:

Dado que el código se traduce y ejecuta línea por línea, los programadores pueden modificar el código sobre la marcha y ver los resultados de inmediato. Esta flexibilidad es invaluable durante la depuración y el desarrollo de aplicaciones, permitiendo ajustes rápidos sin la necesidad de recompilar el código.

INTERACTIVIDAD:

Los intérpretes permiten la ejecución interactiva de código, lo que significa que los programadores pueden introducir y ejecutar comandos uno a uno en un entorno interactivo.

PORTABILIDAD:

A diferencia de los compiladores, los intérpretes no generan código específico para una arquitectura de hardware en particular. Esto significa que el mismo código fuente puede generarse y ejecutarse en diferentes sistemas sin modificación.

MENOR EFICIENCIA EN LA EJECUCIÓN:

Debido a que la traducción del código se realiza durante la ejecución, los programas interpretados suelen ser más lentos que los compilados. Esto es porque cada línea de código debe ser traducida cada vez que se ejecuta, lo que añade una sobrecarga significativa en comparación con el código ya compilado.

DEPENDENCIA DEL INTÉRPRETE

Los programas interpretados requieren que el intérprete esté disponible en el sistema en donde se ejecutan. Esto puede ser una ventaja en términos de portabilidad, pero también puede ser una desventaja si el intérprete no está disponible o es incompatible con el sistema.

ANALISIS DEL VÍDEO Y CUADRO COMPARATIVO:

ANALISIS DEL VÍDEO: El video discute las diferencias entre compiladores e interpretes, se pueden identificar varios conceptos clave que son cruciales para entender como funcionan estos dos tipos de herramientas. Algunos de los conceptos importantes incluyen:

Proceso De Compilación:

Se describe en detalle las fases del proceso de compilación, desde el análisis léxico hasta la generación de código máquina.

Proceso De Interpretación:

Explica como un interprete ejecuta el código fuente línea por línea, y cómo esta ejecución directa afecta el rendimiento y la flexibilidad del desarrollo de software.

Uso En Diferentes Contextos:

Otro punto relevante es como se utilizan en distintos contextos. Por ejemplo los compiladores son comunes en aplicaciones de alto rendimiento como videojuegos o software de sistemas, mientras que los interpretes son más comunes en scripts de automatización, desarrollo web y entornos educativos.

CUADRO COMPARATIVO

ASPECTO	COMPILADOR	INTÉPRETE
Proceso de Traducción	Traduce todo el código fuente a código máquina antes de su ejecución.	Traduce y ejecuta el código fuente línea por línea, en tiempo real
Velocidad de ejecución	Alta, ya que el código está en lenguaje máquina, lo que elimina la traducción en tiempo real.	Baja, ya que la traducción ocurre durante la ejecución, lo que añade una Sobrecarga.
Detección de errores	Se detectan antes de la ejecución, durante el proceso de compilación.	Los errores se detectan durante la ejecución, línea por línea
Flexibilidad.	Menor flexibilidad, ya que requiere recompilar el código para cada cambio.	Mayor flexibilidad, permite modificar y ejecutar el código inmediatamente sin recompilar.
Portabilidad	El código compilado es específico para una arquitectura de hardware, lo que puede limitar la portabilidad.	El mismo código fuente puede ejecutarse en diferentes sistemas sin modificación, siempre que el intérprete esté disponible.
Uso de Recursos	Requiere más recursos en la fase de compilación, pero menos durante la ejecución.	Requiere menos recursos para traducir el código, pero más durante la ejecución debido a la Sobrecarga de interpretación.

10
11
12

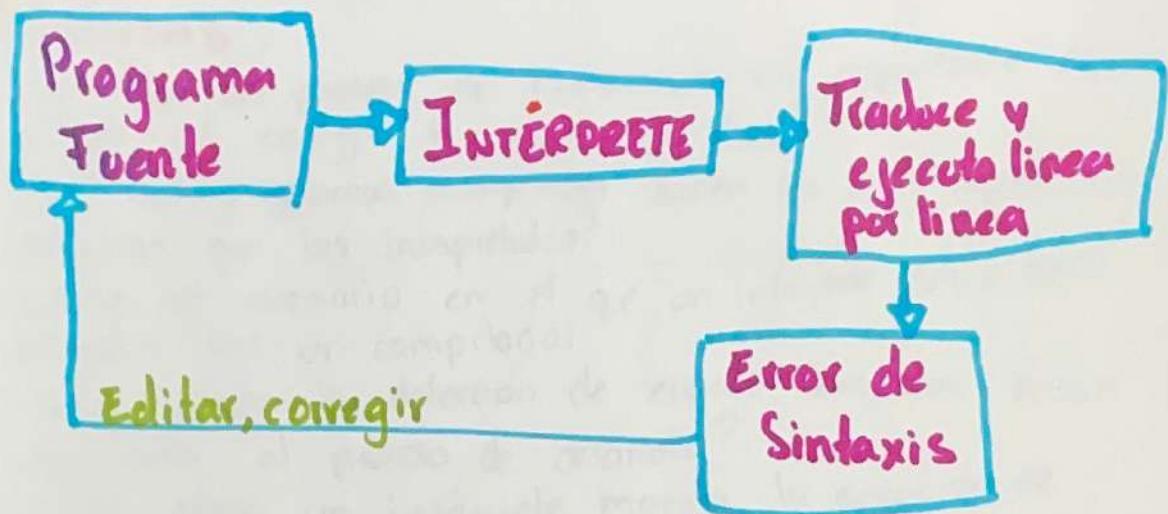


DIAGRAMA DE UN INTÉPRETE

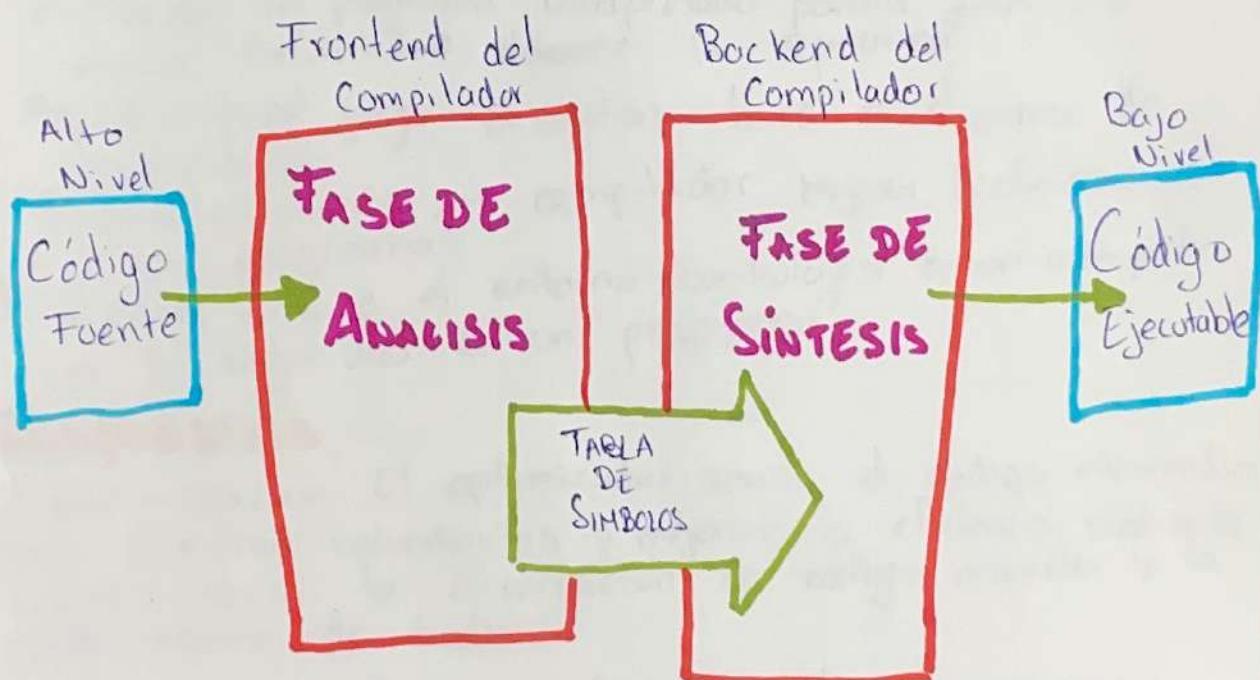


DIAGRAMA DE UN COMPILADOR

PREGUNTAS.

- 1= ¿Qué fase del proceso de compilación es responsable de optimizar el código y cómo lo hace?
- 2= ¿Por qué los programas compilados suelen ser más rápidos en ejecución que los interpretados?
- 3= Describe un escenario en el que un intérprete sería más ventajoso que un compilador.
- 4= ¿Cómo afecta la detección de errores temprana en un compilador al proceso de desarrollo?
- 5= Explica cómo un intérprete maneja la ejecución de código en comparación con un compilador.
- 6= ¿Cuál es la diferencia clave en la forma en que un compilador y un intérprete manejan la portabilidad del código?
- 7= ¿Por qué un programa interpretado podría tener una mayor flexibilidad durante la depuración?
- 8= ¿Qué papel juega el análisis léxico en el proceso de compilación?
- 9= ¿Cómo podría un compilador mejorar la eficiencia de un programa?
- 10= ¿Cómo influye el entorno controlado de un intérprete en la seguridad de un programa?

RESPUESTAS

- 1= Optimización: El optimizador revisa el código intermedio para eliminar redundancias y mejorar la eficiencia, mediante técnicas como la eliminación de código muerto y la optimización de bucles.
- 2= Eficiencia de Programas Compilados: Los programas compilados son más rápidos porque ya están en código máquina, eliminando la necesidad de traducción en tiempo real.

- 3- Ventajas del intérprete: En el desarrollo de scripts o en la educación, donde la flexibilidad y la corrección rápida son clave, un intérprete es más útil.
- 4- Detección de errores Temprana: Permite al desarrollador conseguir y corregir errores antes de la ejecución, lo que ahorra tiempo y evita la ejecución de código defectuoso.
- 5- Manejo de ejecución: Un intérprete ejecuta código directamente sin generar un archivo ejecutable, traduciéndolo línea por línea.
- 6- Portabilidad del Código: Un compilador genera código específico para un sistema, mientras que un intérprete puede ejecutar el mismo código en diferentes sistemas sin modificación.
- 7- Flexibilidad en la Depuración: La ejecución línea por línea permite identificar y corregir errores en tiempo real, sin necesidad de recompilar.
- 8- Análisis Léxico: Es la fase que convierte el código fuente en una secuencia de tokens, facilitando la comprensión del código por parte del compilador.
- 9- Eficiencia del programa: Al optimizar el código, un compilador puede reducir el uso de recursos y acelerar la ejecución del programa.
- 10- Seguridad del intérprete: El entorno controlado minimiza los riesgos de ejecutar código malicioso, ya que el intérprete maneja directamente la ejecución.

Referencias

- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). Compilers: Principles, techniques, and tools (2nd ed.). Pearson Education.
- Sebesta, R. W. (2011). Concepts of programming languages (10th ed.). Addison-Wesley.
- Grune, D., Bal, H. E., Jacobs, C. J. H., & Langendoen, K. G. (2012). Modern compiler design (2nd ed.). Springer.
- Python Software Foundation. (2023). Python 3 documentation. <https://docs.python.org/3/>
- Wirth, N. (1976). Algorithms + data structures = programs. Prentice-Hall.

Generadores de código para compiladores

Son componentes que forman parte de un pase fundamental durante la compilación, su objetivo es el de transformar el código intermedio que fue generado previamente por el análisis sintáctico y semántico.

El generador de código toma las representaciones abstractas de un programa para convertirlas en instrucciones entendibles para el hardware y este las ejecute.

¿Para qué sirven?

Como ya se mencionó previamente, son importantes para la traducción de el código de un lenguaje de programación de alto nivel a instrucciones entendibles y ejecutables por la máquina.

Son importantes ya que así los programadores pueden hacer uso de lenguajes de alto nivel y accesibles como C++, Java o Python, haciendo que así se traduzca estos lenguajes y ejecutados eficientemente.

Además, el generador de código optimizar el código para mejorar su rendimiento, haciendo al programa final lo mas eficiente posible.

¿Cuál es su importancia en la asignatura de LAII?

Su relevancia radica en que como se dijo al comienzo de la asignatura se busca desarrollar un lenguaje de programación y parte de este proceso requiere conocer como funcionan los compiladores para el posterior diseño y construcción de uno, entonces al conocer como operan los generadores de código y como realizan la traducción de lenguajes de alto nivel a instrucciones de bajo nivel permite que podamos implementar un compilador adecuado.

¿Cómo se usan?

El uso de generadores de código implica las siguientes fases para la transformación de código fuente a instrucciones que la máquina pueda entender.

Entrada Cuando el código pasa por el análisis léxico, sintáctico y semántico, se crea una representación intermedia (IR). La IR forma la forma de una estructura como el árbol sintáctico abstracto para representar el programa de la forma más cercana a la ejecución en máquinas.

Selección de instrucciones El generador de código selecciona la mejor combinación de instrucciones máquina para replicar eficientemente la lógica del código fuente.

Optimización de código Se realizan optimizaciones para mejorar el rendimiento del programador, desde la eliminación de instrucciones redundantes, reordenación para minimizar la latencia de la CPU y mejorar en el uso de cache.

Emisión del código El código es traducido para ser ejecutado por el procesador, este se encuentra en binario o ensamblador para ser ensamblado en binario.

< YACC >

Yet Another Compiler-Compiler por sus siglas, se trata de un herramienta usada en el diseño de lenguajes de programación para la generación de analizadores sintácticos.

También se les conoce como parsers y se encargan de analizar la estructura gramatical de un lenguaje de programación.

Características:

- Generación automática de parsers: YACC permite especificar la gramática de un lenguaje de programación en formato normal, después en automático generar el código para un parser el cual puede analizar y validar la sintaxis del lenguaje.
- Manejo de ambigüedades: YACC tiene mecanismos capaces de resolver ambigüedades en la gramática. Se permite definir prioridades y asociaciones para operadores y gestar al parser.
- Integración con Lex: se suele usar en conjunto a Lex, Lex se encarga del analisis léxico y YACC el analisis sintáctico. Unidas facilitan el desarrollo de un compilador.

- **Extensibilidad y Flexibilidad:** con YACC se puede extender y modificar la gramática mientras el lenguaje evoluciona, estos cambios pueden ser fácilmente incorporados al parser.
- **Salida en código C:** El parser que genera YACC está escrito en C, esto lo hace eficiente y portable, así como útil para la creación de compiladores multiplataforma.

Conclusion

El comprender el funcionamiento de los generadores de código, su importancia y aplicación nos permite lograr un mejor desarrollo de un lenguaje de programación, esto apoyado al uso de YACC permitirán un compilador eficiente aprovechándose de todos los beneficios de YACC.

Referencias

Diseño de Compiladores I YACC: Yet Another Compiler-Compiler. (s/f). Edu.ar. Recuperado el 1 de septiembre de 2023, de
https://users.exa.unicen.edu.ar/catedras/compila1/index_archivos/Herramientas/Yacc.pdf ✓
Diseno de
compildores(2008)https://users.exa.unicen.edu.ar/catedras/compila1/index_archivos/Introduccion.pdf



Part 01 · Tutorial on lex / Yacc

El tutorial comienza con una introducción a lex, una herramienta fundamental en el análisis léxico, la primera fase es el proceso de compilación. Esta fase convierte el código fuente en una secuencia de tokens, luego se pasan al analizador sintáctico, que construye un árbol de análisis sintáctico, y eventualmente el generador de código, que produce el código objeto, o código máquina.

¿Qué es Lex?

Lex es un generador de escáneres (scanners) o analizadores léxicos. Su función es tomar una descripción de expresiones regulares y las acciones asociadas a estas, escritas en lenguaje C, y generar automáticamente un escáner, que se guarda en un archivo llamado "lex.yy.c", procesa el código fuente, identifica los tokens según las expresiones regulares, y ejecuta las acciones definidas.

Estructura de un archivo lex

Un archivo lex se estructura en dos partes:

1- Parte opcional: Puede contener definiciones de reemplazos de texto, ajustes de tablas internas de lex y código C global que se utilizará en las acciones. Esta sección termina con el símbolo '%`'.

2- Expresiones Regulares y acciones: Aquí se enumeran las expresiones regulares seguidas de las acciones en C que deben ejecutarse cuando se reconoce un patrón. Cada patrón puede tener una única acción (una instrucción en C) o un bloque de acciones (múltiples instrucciones encerradas entre '{}').

3- Parte opcional: Puede incluir código C adicional que se utilizará en el programa principal que incorpora el escáner generado).

Ejemplo simple con Lex

Se presentó un ejemplo donde el archivo Lex detecta la cadena "hello world", en lugar de imprimirla, devuelve "good bye". Este archivo Lex define dos patrones: uno para "hello world" que imprime "good bye", y otro que ignora cualquier otro carácter. El proceso incluye la compilación del archivo Lex para generar el escáner, la compilación de este escáner con el programa principal en C, y finalmente, la ejecución del programa que corresponde de acuerdo con los patrones definidos.

Integración del Escáner en un Programa en C.

En una demostración más avanzada, el tutorial muestra cómo integrar el escáner generado por lex en un programa C que procesa un archivo de configuración. Este archivo contiene pares de nombre - valor, como "DBType : MySQL", que el escáner identifica y procesa.



El programa C, al usar el escáner, puede reconocer diferentes tokens como tipos de bases de datos, nombres de tablas y puertos, y realizar validaciones y acciones según el contenido del archivo de configuración.

Conclusion :

Este tutorial introduce a Lex como una herramienta poderosa para automatizar el análisis léxico en el proceso de compilación, destacando su capacidad para simplificar la creación de escáneres que, de otro modo, serían difíciles de escribir a mano para lenguajes más complejos. La integración de Lex en programas en C facilita el procesamiento de datos de entrada estructurados, permitiendo a los programadores centrarse en la lógica del negocio en lugar de la gestión manual de tokens.



Part 02: Tutorial on lex/yacc.

Objetivo del tutorial: El tutorial se enfoca en el uso de 'yacc' (también conocido como 'bison') para análisis sintáctico y semántico en el procesamiento de lenguajes, junto con 'lex' para el análisis léxico.

2: Proceso de análisis:

- **Análisis léxico con 'lex':** 'lex' se utiliza para el análisis léxico, donde se definen expresiones regulares para identificar tokens en un archivo fuente. Estos tokens son secuencias de caracteres que representan unidades significativas del lenguaje.
- **Análisis sintáctico con 'yacc':** 'yacc' toma los tokens generados por 'lex' y realiza el análisis sintáctico. Construye un árbol de análisis a partir de estos tokens según la gramática definida. Esto permite realizar el procesamiento semántico o interpretar el lenguaje.

3: Estructura de Archivos 'yacc' y 'lex':

- **Declaraciones C:** Definiciones y declaraciones necesarias para el procesamiento, como estructuras de datos y funciones auxiliares.
- **Reglas gramaticales:** Definición de la gramática del lenguaje en forma de producciones. Estas reglas indican cómo se deben combinar los tokens para formar expresiones válidas del lenguaje.
- **Acciones:** Código C asociado a cada regla gramatical. Estas acciones especifican el procesamiento que se debe realizar cuando se reconoce una producción de la gramática.
 - Archivo 'yacc' ('.y'): Se compone de tres secciones.
 - Archivo 'lex' ('.l'): Define cómo se deben reconocer los tokens en el texto de entrada. Utiliza expresiones regulares para identificar tokens y las acciones correspondientes para cada token.

4: Ejemplo Práctico - Procesador de Lenguaje Simple:

- Se presenta un ejemplo de un procesador de lenguaje que realiza cálculos aritméticos. El archivo 'yacc' define la gramática para expresiones aritméticas y las acciones para calcular y mostrar resultados. El archivo 'lex' se encarga de identificar tokens de entrada (números, operadores, etc).
- El procesador soporta operaciones básicas como asignaciones, sumas y restas, y permite imprimir resultados y salir del programa.

5: Compilación y ejecución:

- **Generación de archivos:** Primero se usa 'yacc' para generar dos archivos: un archivo de encabezado ('.h') y un módulo ('.c') que contiene el parser. Luego se usa 'lex' para generar un archivo C que maneja la identificación de tokens.



• **Compilación:** Los archivos generados se compilan juntas con un compilador C ('gcc') para crear un ejecutable que puede interpretar el lenguaje definido en los archivos 'yacc' y 'lex'.

• **Ejecución:** El ejecutable resultante procesa las entradas conforme a la gramática y las acciones definidas, permitiendo realizar cálculos y mostrar resultados según las reglas establecidas.

Conclusión:

El tutorial sobre 'lex' y 'yacc' proporciona una guía fundamental para entender y aplicar estas herramientas en la construcción de procesadores de lenguaje. A través de un enfoque práctico y ejemplos detallados, se demuestra cómo 'lex' se encarga del análisis léxico, identificando tokens en la entrada, mientras que 'yacc' realiza el análisis sintáctico, estructurando estos tokens conforme a una gramática definida para realizar un proceso más complejo.



INSTITUTO TECNOLÓGICO DE CELAYA

PARACTICA DE LABORATORIO-
LENGUAJES Y AUTOMATAS 2

AUTORES: MARÍA DEL CARMEN CHÁVEZ PATIÑO,
ISAAC SALVADOR BRAVO ESTRADA Y GUILLERMO PEASLAND AGUILAR

CARRERA	NOMBRE DE LA ASIGNATURA
INGENIERIA EN SISTEMAS COMPUTACIONALES	LENGUAJES Y AUTOMATAS 2

PRACTICA NO.	NOMBRE DE LA PRACTICA	DURACION(HORAS)
1	EJEMPLO SOBRE EL USO DE YACC	2 horas

1	INTRODUCCIÓN
En "Lenguajes y Autómatas 2", se profundiza en los conceptos avanzados de teoría de autómatas, gramáticas formales, lenguajes formales, y técnicas de análisis sintáctico. Se estudian herramientas que permiten el análisis de lenguajes más complejos y la implementación de compiladores e intérpretes. YACC (Yet Another Compiler Compiler) es una herramienta que permite generar analizadores sintácticos basados en una gramática libre de contexto. Se utiliza comúnmente para la construcción de compiladores y traductores de lenguajes de programación, ayudando a definir cómo deben interpretarse las estructuras de un lenguaje.	

2	OBJETIVO (COMPETENCIA)
El objetivo de aprender a usar YACC es desarrollar la capacidad de diseñar analizadores sintácticos que permitan procesar y entender lenguajes formales. Los estudiantes deben ser capaces de crear herramientas que traduzcan código fuente a representaciones intermedias o ejecutables, usando una gramática formal para definir las reglas del lenguaje.	

3	FUNDAMENTO
YACC se basa en la teoría de lenguajes formales y gramáticas libres de contexto. En YACC, una gramática se describe mediante un conjunto de reglas de producción, que especifican cómo se pueden combinar los tokens para formar expresiones válidas en el lenguaje de entrada.	

4	REQUISITOS BASICOS
Para trabajar con YACC, es necesario tener conocimientos en: Gramáticas libres de contexto: Comprender cómo se estructuran y definen las gramáticas. Lenguaje C o C++: YACC genera código en C, por lo que es necesario poder integrar y entender este código.	

Lex: Herramienta que se utiliza junto con YACC para el análisis léxico, es decir, para identificar los tokens de la entrada.

5

DESARROLLO

Cómo se debe trabajar en YACC:

Definir la gramática: La primera parte del archivo YACC contiene la definición de tokens y reglas de producción que describen la estructura del lenguaje. Estos tokens suelen ser identificados por un analizador léxico como Lex.

Escribir las acciones: Para cada regla de producción, se definen acciones que se ejecutan cuando esa regla es reconocida. Estas acciones suelen manipular los valores de los tokens para realizar cálculos o construir estructuras de datos.

Compilación: El archivo YACC se compila para generar un analizador sintáctico en C. Este código se integra con el código del analizador léxico y se compila para producir un ejecutable.

Ejecución: El ejecutable generado puede tomar una entrada, analizarla conforme a las reglas definidas y ejecutar las acciones asociadas, permitiendo el procesamiento del lenguaje.

Código Fuente:

```
%{  
#include <stdio.h>  
#include <string.h>  
#include "y.tab.h"  
%}  
  
/* Definición de las reglas léxicas para el análisis */  
%%  
[0-9]+          yyval.number=atoi(yytext);return NUMBER;  
/* Captura números enteros y los convierte a enteros, asignándolos a yyval.number */  
  
calentador      return TOKHEATER;  
/* Reconoce la palabra clave "calentador" y devuelve el token TOKHEATER */  
  
encender|apagar  yyval.number=!strcmp(yytext,"encender");return STATE;  
/* Reconoce "encender" o "apagar" y asigna 1 si es "encender" y 0 si es "apagar" a yyval.number */  
  
objetivo         return TOKTARGET;  
/* Reconoce la palabra clave "objetivo" y devuelve el token TOKTARGET */  
  
temperatura      return TOKTEMPERATURE;  
/* Reconoce la palabra clave "temperatura" y devuelve el token TOKTEMPERATURE */  
  
[a-zA-Z0-9]+      yyval.string=strdup(yytext);return WORD;  
/* Captura cualquier palabra o número y los asigna a yyval.string como un puntero a una cadena */  
  
\n                  /* Ignora los saltos de línea */  
[\t]+               /* Ignora espacios en blanco y tabulaciones */  
%%
```

```
/* Fin de las reglas léxicas */

%{
#include <stdio.h>
#include <string.h>

/* Declaraciones de funciones para evitar advertencias de compilación */
int yyparse(); // Declaración de yyparse
int yylex(); // Declaración de yylex

/* Función para manejar errores */
void yyerror(const char *str)
{
    fprintf(stderr, "error: %s\n", str);
}

/* Función que indica el final de la entrada */
int yywrap()
{
    return 1;
}

/* Función principal que inicia el análisis sintáctico */
int main() // Cambiado a int
{
    yyparse();
}

/* Variable global para almacenar el nombre del calentador seleccionado */
char *heater = "default";

%}

/* Declaración de tokens utilizados en el análisis sintáctico */
%token TOKHEATER TOKTARGET TOKTEMPERATURE

/* Definición de la unión utilizada para manejar diferentes tipos de valores */
%union
{
    int number;
    char *string;
}

/* Declaración de tokens con sus tipos asociados */
%token <number> STATE
%token <number> NUMBER
%token <string> WORD

%%
/* Inicio de las reglas gramaticales */

/* Definición de la estructura general de comandos */
commands:
    | commands command
    ;
```

```

/* Un comando puede ser vacío o una lista de comandos */

command:
heat_switch | target_set | heater_select
;
/* Un comando puede ser un cambio de estado de encendido, un ajuste de temperatura
o la selección de un calentador */

heat_switch:
TOKHEATER STATE
{
    if($2)
        printf("\tCalentador '%s' encendido\n", heater);
    else
        printf("\tCalentador '%s' apagado\n", heater);
}
;
/* Regla para cambiar el estado del calentador (encendido o apagado) */

target_set:
TOKTARGET TOKTEMPERATURE NUMBER
{
    printf("\tTemperatura del calentador '%s' ajustada a %d grados\n", heater,
$3);
}
;
/* Regla para ajustar la temperatura del calentador */

heater_select:
TOKHEATER WORD
{
    printf("\tCalentador seleccionado: '%s'\n", $2);
    heater=$2;
}
;
/* Regla para seleccionar un calentador específico */

%%
/* Fin de las reglas gramaticales */

```

Explicación:

Este programa es un ejemplo de un intérprete básico para controlar un sistema de calentadores a través de comandos de texto. Está construido utilizando herramientas de análisis léxico y sintáctico como Lex (Flex) y Yacc (Bison). El programa permite al usuario interactuar con un "calentador" simulando comandos como encender o apagar el calentador, ajustar su temperatura, o seleccionar un calentador específico.

Partes Principales del Programa:

1. Análisis Léxico (Archivo example4.l):

- **Propósito:** El archivo example4.l utiliza Flex para definir las reglas léxicas, es decir, cómo identificar y clasificar las diferentes palabras y símbolos en la entrada del usuario.
- **Funcionamiento:**

- **Números:** Se reconocen secuencias de dígitos ([0-9]+) y se convierten a enteros, asociándolos con el token NUMBER.
- **Palabras Clave:** Se identifican palabras específicas como calentador, encender, apagar, objetivo y temperatura, las cuales se transforman en tokens que el parser usará.
- **Palabras Generales:** Cualquier otra combinación de letras y números es tratada como un WORD, que es un token general para cadenas de texto.
- **Ignorar:** Se ignoran los espacios en blanco y las líneas en blanco, ya que no aportan significado al análisis.

2. Análisis Sintáctico (Archivo example4.y):

- **Propósito:** El archivo example4.y utiliza Bison para definir las reglas sintácticas, es decir, cómo se estructuran los comandos del usuario y qué acciones se toman en función de esos comandos.
- **Funcionamiento:**
 - **Tokens:** El parser recibe los tokens generados por el analizador léxico y los organiza en estructuras lógicas llamadas "comandos".
 - **Comandos:**
 - **Cambio de Estado (heat_switch):** Permite al usuario encender (encender) o apagar (apagar) el calentador seleccionado. El programa imprime un mensaje indicando si el calentador está encendido o apagado.
 - **Ajuste de Temperatura (target_set):** Permite al usuario establecer una temperatura objetivo para el calentador seleccionado. El programa imprime la temperatura ajustada.
 - **Selección de Calentador (heater_select):** Permite al usuario seleccionar un calentador específico por su nombre. El nombre se guarda y se utiliza en los otros comandos. El programa imprime qué calentador ha sido seleccionado.

3. Funcionamiento del Programa:

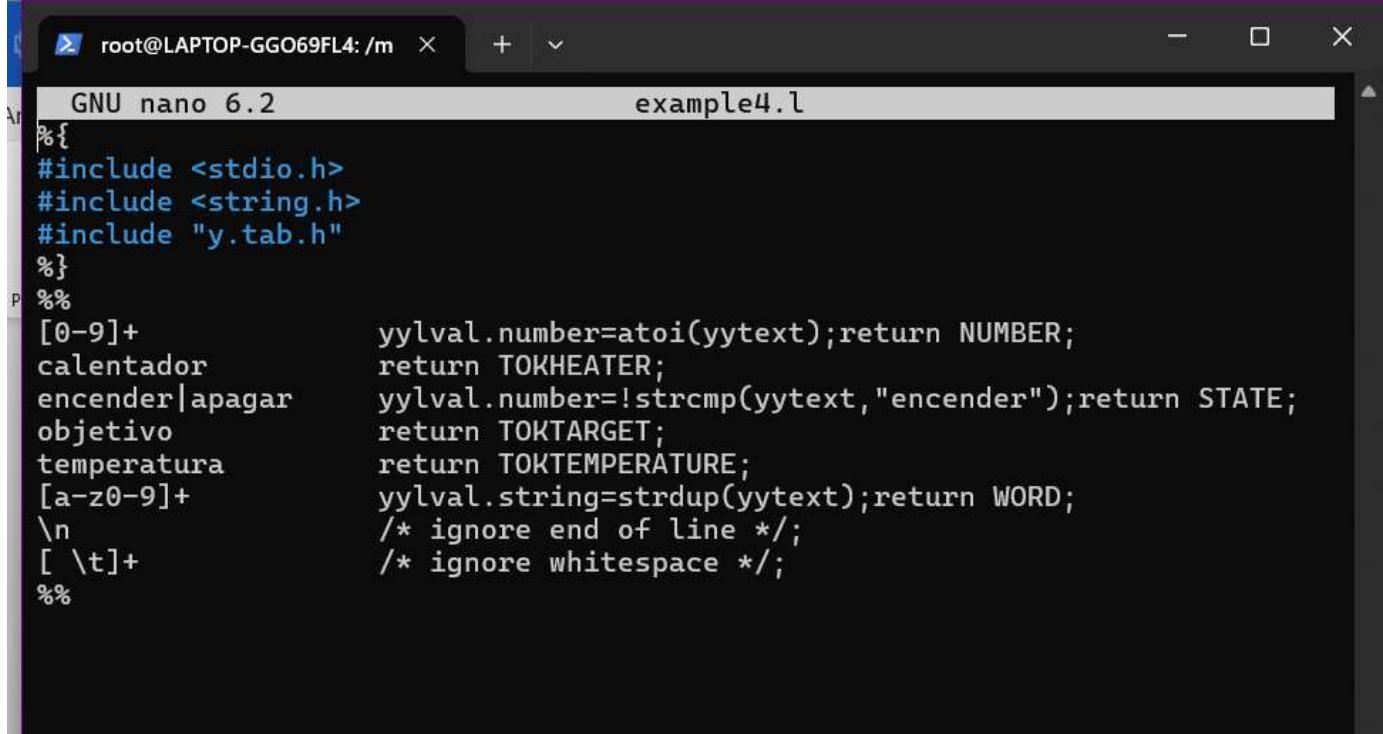
- **Inicio:** Cuando se ejecuta el programa, se inicializa con un calentador por defecto llamado "default".
- **Entrada del Usuario:** El usuario ingresa comandos como calentador encender, objetivo temperatura 75, o calentador sala. El programa analiza y ejecuta estos comandos, respondiendo con mensajes adecuados en español.
- **Flujo:** El flujo del programa es simple: el usuario introduce un comando, el lexer lo analiza, el parser lo organiza, y el programa ejecuta las acciones correspondientes.

Ejemplo de Ejecución:

1. **Entrada:** calentador encender
 - **Salida:** Calentador 'default' encendido
2. **Entrada:** objetivo temperatura 75
 - **Salida:** Temperatura del calentador 'default' ajustada a 75 grados
3. **Entrada:** calentador sala

- **Salida:** Calentador seleccionado: 'sala'

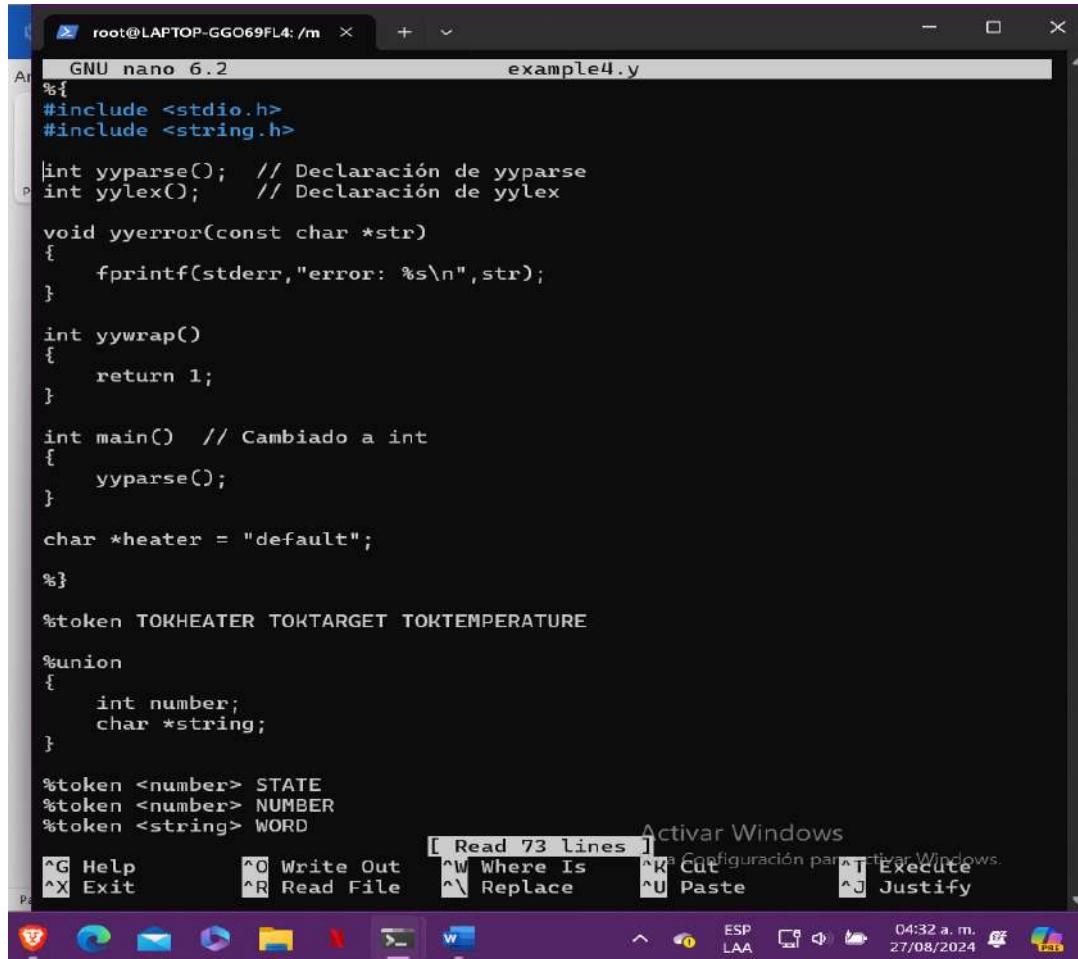
Capturas de Pantalla:



```

GNU nano 6.2                               example4.l
%{
#include <stdio.h>
#include <string.h>
#include "y.tab.h"
%}
%%
[0-9]+          yyval.number=atoi(yytext);return NUMBER;
calentador      return TOKHEATER;
encender|apagar yyval.number=!strcmp(yytext,"encender");return STATE;
objetivo        return TOKTARGET;
temperatura     return TOKTEMPERATURE;
[a-z0-9]+        yyval.string=strdup(yytext);return WORD;
\n              /* ignore end of line */;
[ \t]+           /* ignore whitespace */;
%%

```



```

GNU nano 6.2                               example4.y
%{
#include <stdio.h>
#include <string.h>

int yyparse(); // Declaración de yyparse
int yylex(); // Declaración de yylex

void yyerror(const char *str)
{
    fprintf(stderr,"error: %s\n",str);
}

int yywrap()
{
    return 1;
}

int main() // Cambiado a int
{
    yyparse();
}

char *heater = "default";
%}

%token TOKHEATER TOKTARGET TOKTEMPERATURE

%union
{
    int number;
    char *string;
}

%token <number> STATE
%token <number> NUMBER
%token <string> WORD

```

Activar Windows

[Read 73 Lines]

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute
 ^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify

04:32 a. m.
27/08/2024

```
root@LAPTOP-GGO69FL4:/mnt/c/Users/melis# nano example4.y
root@LAPTOP-GGO69FL4:/mnt/c/Users/melis# lex example4.l
root@LAPTOP-GGO69FL4:/mnt/c/Users/melis# yacc -d example4.y
root@LAPTOP-GGO69FL4:/mnt/c/Users/melis# cc lex.yy.c y.tab.c -o example4
root@LAPTOP-GGO69FL4:/mnt/c/Users/melis# ./example4
heat on
    Calentador 'default' encendido
target temperature 75
    Temperatura del calentador 'default' ajustada a 75 grados
```

6

ANEXOS

Bitácora de Incidencias:

Como es algo prácticamente nuevo tuvimos varios errores a la hora de estructurar el programa o también a la hora de solucionar ciertos percances, por lo cual se tuvo que dar una investigación un poco más detallada.

También uno que otro error al olvidar que se estaba trabajando con WSL y querer hacer acciones que hacemos normalmente o que se nos olvidaba de repente algunos comandos.

7

REFERENCIAS

Engelsma, J. [@JonathanEngelsma]. (s/f-a). Part 01: Tutorial on lex/yacc. Youtube. Recuperado el 27 de agosto de 2024, de <https://www.youtube.com/watch?v=54bo1qaHAfk>

Engelsma, J. [@JonathanEngelsma]. (s/f-b). Part 02: Tutorial on lex/yacc. Youtube. Recuperado el 27 de agosto de 2024, de https://www.youtube.com/watch?v=__-wUHG2rfM

Mi primer proyecto utilizando Yacc y Lex. (s/f). Erick Navarro. Recuperado el 27 de agosto de 2024, de <https://ericknavarro.io/2020/10/01/27-Mi-primer-proyecto-utilizando-Yacc-y-Lex/>

(S/f-a). Unioviedo.es. Recuperado el 27 de agosto de 2024, de <https://www.unioviedo.es/ate/alberto/TEMA3-Ensamblador.pdf>

(S/f-b). Uma.es. Recuperado el 27 de agosto de 2024, de <http://www.lcc.uma.es/~galvez/ftp/tci/TutorialYacc.pdf>