Grandes temas del curso

I. Traductores de bajo nivel

II. Traductores de alto nivel

III. Sistemas Operativos

IV. Herramientas de Configuración, arranque y operación de los sistemas de cómputo

II.- Traductores de Alto Nivel

(interpretes y compiladores)

Ingrid Rovelo Wegener

Objetivo del Capítulo II

I. Revisar el contexto de los Traductores.

II. Identificar y reconocer a los traductores de alto nivel.

III. Conocer las características y como trabajan los traductores de alto nivel

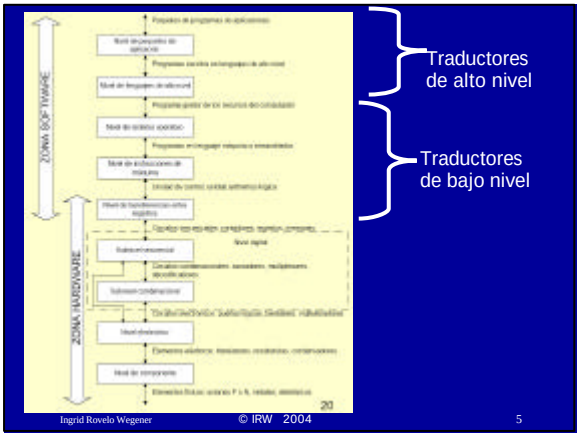
IV. Conocer las fases y el proceso de cada etapa de compilación.

V. Conocer las aplicaciones de los traductores de Alto nivel.

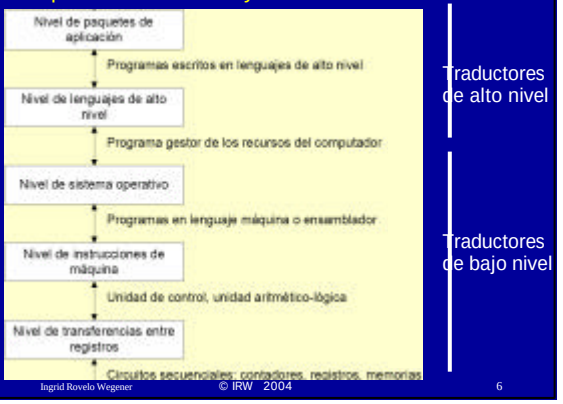
Contenido

II. Traductores de alto nivel

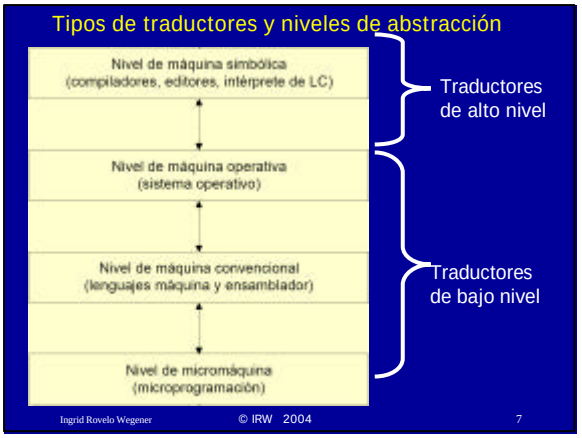
- Revisión de los tipos de traductores y niveles de abstracción
- Conceptos relacionados con los traductores
- Historia de los traductores
- Conceptos de traductores
- Traductores y computadoras simuladas por Software
- Aplicaciones de traductores de alto nivel.



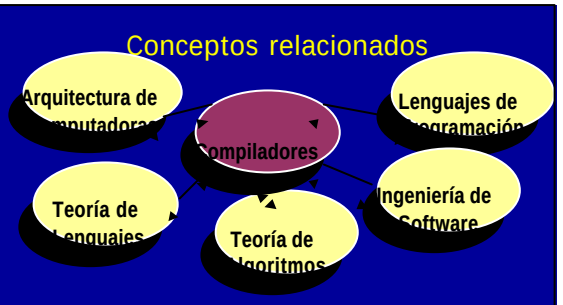
Tipos de traductores y niveles de abstracción



Tipos de traductores y niveles de abstracción



Conceptos relacionados



Con algunas técnicas básicas de escritura de compiladores se pueden construir traductores para una gran variedad de lenguajes y máquinas

Historia de los traductores

- 1946, se desarrolla el primer ordenador digital (lenguaje de máquina)
- 1950, John Backus dirige una investigación en IBM en un lenguaje algebraico
- 1954, se comienza a desarrollar FORTRAN
- 1957, FORTRAN se utiliza en la IBM modelo 704
- Surge el concepto traductor

Historia de los traductores

- El primer compilador de FORTRAN tardó 18 años-persona en realizarse.
- FORTRAN era dependiente de la máquina.
- Paralelamente al desarrollo de FORTRAN en América, en Europa surge una corriente que pretende que los lenguajes fuesen independientes de la máquina, esta corriente estaba influida por los trabajos sobre GLC de Chomsky.

Historia de los traductores

- Surge un grupo Europeo encabezado por F.L. Bauer, en la que participó ACM y John Backus.
  - De este grupo surge un informe que define un Lenguaje Algebraico Internacional, publicado en Zurich en 1958.
- 1969, aparece Algol 60.
- Junto con los lenguajes también la técnica de los compiladores avanza.

Historia de los traductores

- 1958, Strong y otros proponen una solución al problema de que un compilador fuera portable, y esta era dividir al compilador en dos fases “front end” (analiza el programa fuente) y “back end” (genera código objeto para la máquina objeto).
- El puente de unión era un lenguaje intermedio denominado UNCOL (no funcionó)
- 1959, Rabin y Scott proponen el empleo de AFD y AFN para el reconocimiento lexicográfico de los lenguajes

Historia de los traductores

- Aparece BNF (Backus-1960, Naur-1963, Knuth-1964) como una guía para el desarrollo del análisis sintáctico.
- 1959, Sheridan describe un método de parsing de FORTRAN para introducir paréntesis en una expresión.
- En los 60's se desarrollan diversos métodos de parsers ascendentes y descendentes

Historia de los traductores

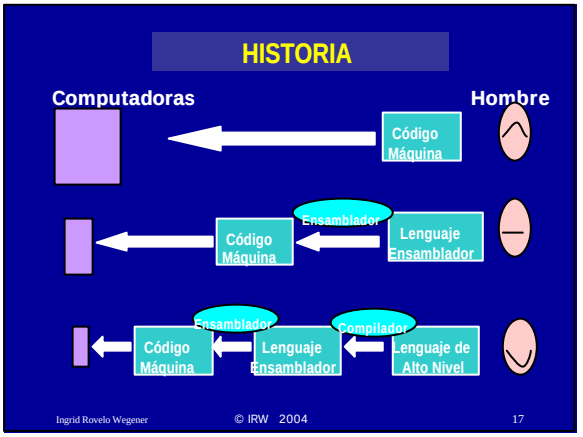
- Floyd más adelante introduce la técnica de precedencia de operadores y uso de funciones de precedencia.
- 1961, se usa por primera vez un parsing descendente recursivo.
- En los 60's se estudia el paso de parámetros por nombre, valor y referencia y se incluyen los procedimientos recursivos para Algol 60.

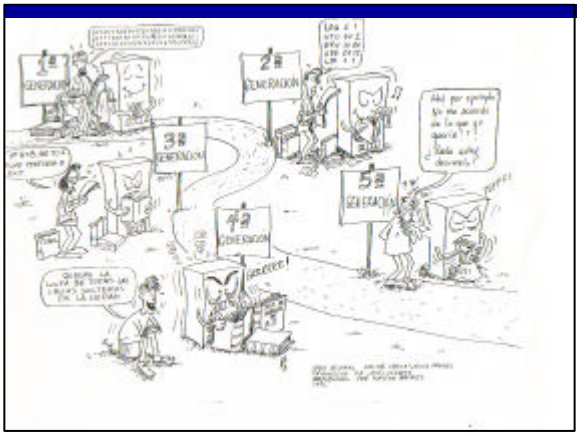
Historia de los traductores

- Se desarrolla la localización dinámica de datos.
- 1968, se estudia y definen las GLC, los parsers predictivos y la eliminación de recursividad izquierda.
- 1975, aparece LEX generador automático de analizadores léxicos a partir de expresiones regulares bajo UNIX.

Historia de los traductores

- A mitad de los 70's Johnson crea YACC para UNIX (generador de analizadores sintácticos).
- Ahora un compilador de divide en varias varias.
- El último lenguaje de programación de amplia aceptación es JAVA (es interpretado).





Cómo darle instrucciones a una computadora

- ¿Por qué no con lenguajes naturales?
  - ¿Español?
  - **"Abre las compuertas, Hal."**
  - **"Lo siento Dave, temo que no puedo hacer eso".**
- **Lenguajes Naturales:**
  La misma expresión describe muchas acciones posibles
  - **Ambiguas**
- **Usamos un lenguaje de programación**
  - **Ejemplos: Java, C, C++, Pascal, BASIC, Scheme**

¿Qué es un lenguaje de programación?

- ¿Cómo le damos instrucciones a una computadora?
- ¿Cómo hacemos que la computadora lleve a cabo las instrucciones eficientemente?
- Pueden ser usados para describir cualquier acción
- Hay muchas formas de describir la misma acción

Dilema del lenguaje de Programación

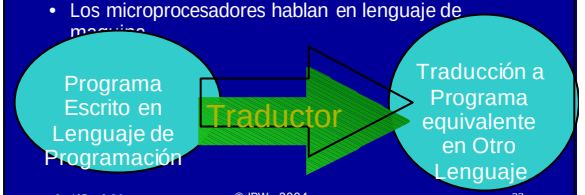
- Programa de Computadora Almacenado.
- ¿Como indicar a la computadora qué hacer?
  - Necesidad de un programa que la computadora pueda ejecutar.
  - Debe ser escrito en lenguaje de máquina.
- Es improductivo codificar en el lenguaje máquina.
- Diseñar un lenguaje de alto nivel.
- Implementar un lenguaje de alto nivel.
  - Alternativa 1: Intérprete
  - Alternativa 2: Compilador

La necesidad de traducir

- Escribimos en lenguajes de alto nivel
  - Código fuente de Programación
  - Lenguaje Script (Scripting language)
- La computadora ejecuta en lenguaje de bajo nivel
- **?** Requiere ser traducido
- Los Compiladores tradicionales son especificados dependiendo del tipo de arquitectura del procesador (CPU).

Cómo instruir a la computadora

- Escribimos un programa usando un lenguaje de programación
  - Descripción abstracta de alto nivel
  - El empleo de lenguajes simbólicos obliga a utilizar un traductor del lenguaje elegido al lenguaje máquina.
- Los microprocesadores hablan en lenguaje de máquina



Traductor

Un Traductor es un programa que
lee un programa en un lenguaje y
lo traduce a un programa equivalente en otro lenguaje,
y además informa al usuario sobre
la presencia de errores en el programa de entrada

Aparente complejidad- Tareas básicas

Cómo instruir a la computadora

- Input: lenguaje de programación de alto nivel
- Output: instrucciones de assembler de bajo nivel

- traductor tiene que:
  - leer y entender el programa
  - precisamente determinar que acciones se requieren
  - encontrar cómo llevar a cabo esas acciones
  - instruir a la computadora a lleva a cabo las acciones

Generalidades

- Proceso de ejecución (interpretación)
  - La CPU es un intérprete
- Proceso de interpretación
  - Un programa (intérprete) ejecuta otro programa
- Proceso de compilación
  - Un programa (compilador) traduce otro programa a código máquina
- Proceso de precompilación
  - Un programa (precompilador) traduce otro programa a otra representación más fácil de interpretar

¿Cómo se debe de desarrollar un programa?

- Generalmente, el código fuente es creado a través de un ciclo conocido como

  Editar-Compilar-Ejecutar

- El programador teclea el código (editar),

- lo compila, corrige los errores que encuentre y lo ejecuta.

Traductores y computadoras simuladas por Software

I. Tipos de errores

II. Traducción

Preprocesador
Compilador
Ensamblador
Cargador

III. Simulación de software

Tipos de Errores

- Errores de Sintaxis
  - Uso incorrecto del conjunto de términos y operadores clave
  - El Compilador o intérprete normalmente los detecta
- Errores al Tiempo de ejecución
  - Se encuentran en la ejecución
  - Errores en la lógica
  - El Compilador normalmente no los detecta

Tipos de Errores

1. A veces algunos errores ocultan otros
2. Un error puede provocar una avalancha de errores que se solucionan con el primero

Criterios a seguir a la hora de manejar errores

1. Pararse al detectar el primer error (conveniente para un compilador interactivo)
2. Detectar todos los errores de una pasada (conveniente para un compilador de línea)

Código Fuente

- Tipos de Código Fuente
  - Secuencia de caracteres ASCII, EBCDIC, Unicode, etc.
  - Dibujo vectorial (DFD).
  - Posiciones fijas en una hoja (fila, columna determinan la categoría sintáctica -RPG-)
  - Opciones de menú / Selecciones en formulario (Visual Basic, C++, etc.)

Código Fuente

- Distintos tipos de Lenguajes
  - Bajo nivel
    - Hexadecimal
    - Ensamblador / Macroensamblador
  - Lenguajes de Alto Nivel
    - Especializados / Generales
    - Imperativos, funcionales, lógicos
    - Orientados a objetos

Modelo de análisis y síntesis

- En la compilación se distinguen dos partes:
  - Análisis,
  - Síntesis.
- *Análisis*: divide al programa fuente en sus elementos y crea una representación interna del programa fuente.
- En el análisis se determinan las operaciones que implica el programa fuente y se registran en un árbol, llamado árbol sintáctico, donde cada nodo representa una operación y los hijos son los argumentos.

HERRAMIENTAS QUE REALIZAN ANÁLISIS

- **EDITORES DE ESTRUCTURAS**
- **IMPRESORAS ESTÉTICAS**
- **VERIFICADORES ESTÁTICOS**
- **INTÉRPRETES**
- **FORMADORES DE TEXTO**
- **COMPILADORES DE CIRCUITOS DE SILICIO**
- **INTÉRPRETES DE CONSULTAS (predicados)**

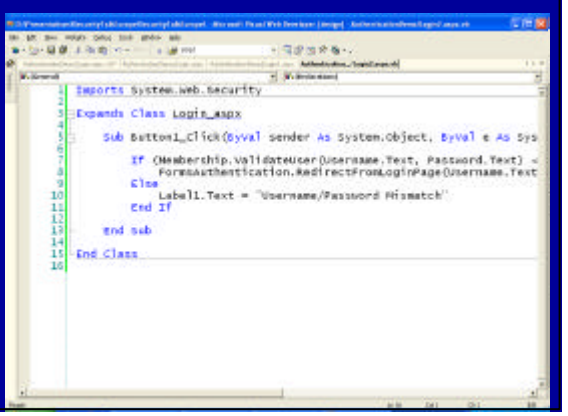
Muchas herramientas de software que manipulan programas fuente realizan primero algún tipo de análisis:

Editores de estructuras: for, while, correspondencia de paréntesis.

Impresoras estéticas: color para comentarios, indentación, anidamiento.

Verificadores estáticos: partes que no se ejecutan, variables que se usan antes de ser definidas.

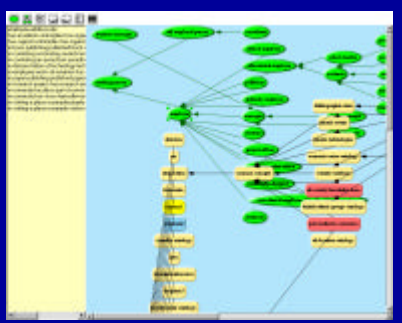
Interpretes: En lugar de producir un programa objeto como resultado de una traducción, un interprete realiza las operaciones que implica el programa fuente.



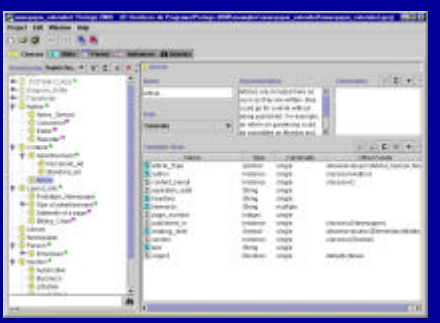
Editores (Reggie)

Editores (Reggie)

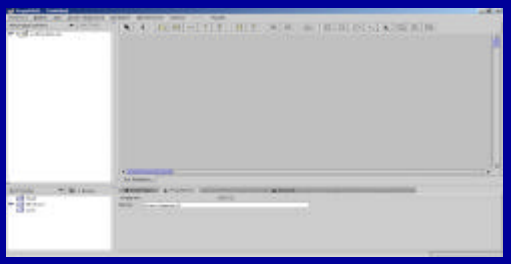
Editores (Webonto)



Editores (Protégé)

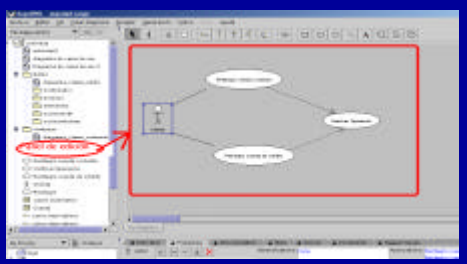


2. Interfaz



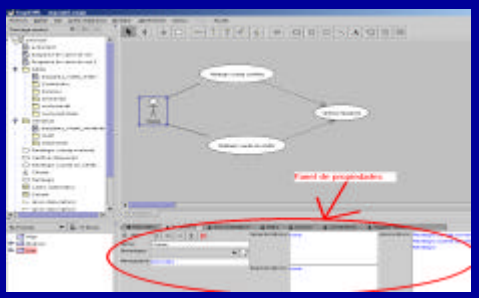
2.1 Panel de navegación

2.2 Panel de edición



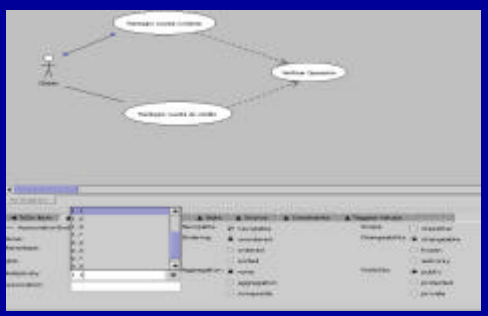
2.3 Panel de tareas pendientes

2.4 Panel de propiedades

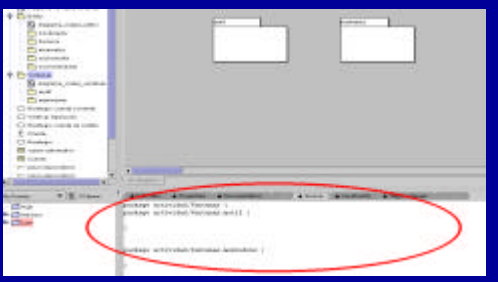


2.4 Panel de Propiedades
Ejemplo de propiedades

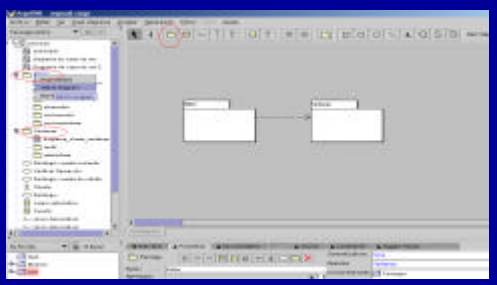
2.4 Panel de Propiedades
Ejemplo de propiedades



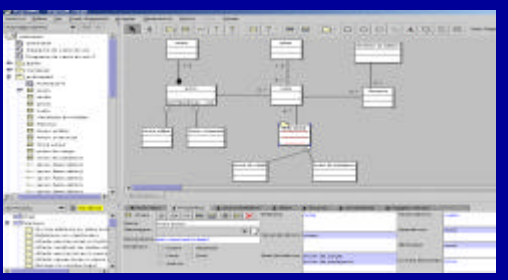
2.4 Panel de propiedades
Ejemplo de generación de código



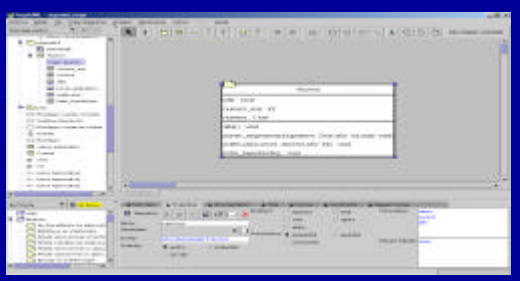
3.1 Diagrama de clases



3.1 Diagrama de clases



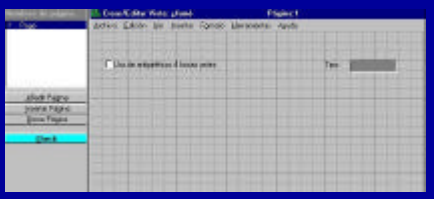
Atributos y métodos de clase



3.2 Diagrama de casos de uso

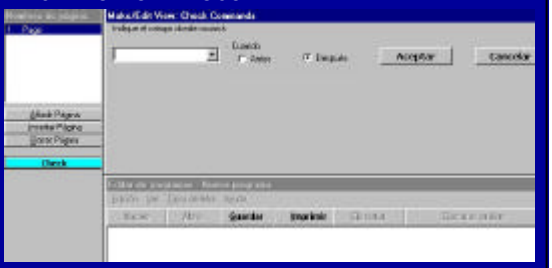
Crear Saltos Condicionales

- Para usar la orden GOTO pulsar el botón CHECK en la parte izquierda de la pantalla de VISTA.



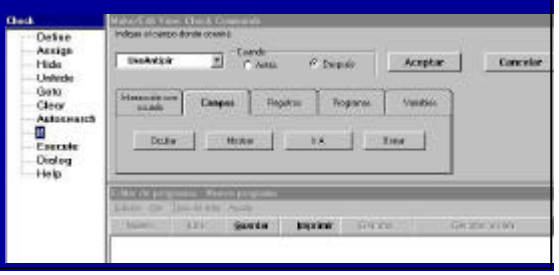
Crear Saltos Condicionados

- Seleccionar un campo de la lista INDIQUE EL CAMPO DONDE OCURRIRÁ.



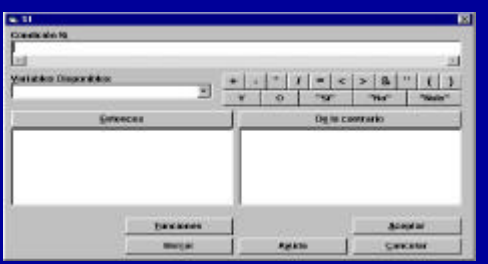
Crear Saltos Condicionados

- Seleccionar de la lista de comandos de la izquierda If



Crear Saltos Condicionados

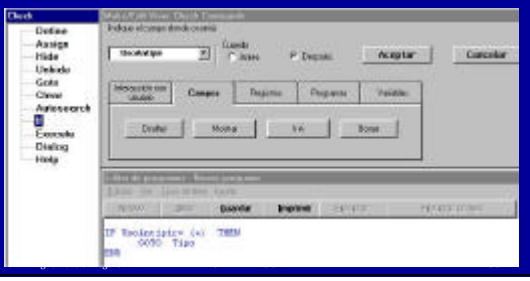
- Aparecerá otra ventana,



Crear Saltos Condicionados

Crear Saltos Condicionados

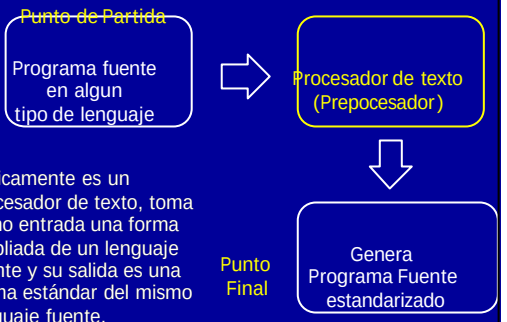
- Noten que en la zona inferior del Editor de programas aparece la SINTAXIS en azul.



Modelo de análisis y síntesis

- Existen otras aplicaciones que también utilizan el análisis:
  - *Formadores de texto:* Cualquier procesadores de textos que use texto estructurado por párrafos figuras, etc.
  - *Compiladores de circuitos de silicio:* Un compilador de circuitos de silicio tiene un lenguaje fuente similar o idéntico a un lenguaje de programación convencional, pero las variables representan señales lógicas (1 o 0) y la salida es el diseño de un circuito en un lenguaje apropiado.
  - *Intérpretes de consultas.* Traduce un conjunto de operadores relacionales y boléanos a órdenes para buscar en una base de datos.
  - *Síntesis*: construye el programa objeto a partir de la representación intermedia.

Preprocesador



Básicamente es un procesador de texto, toma como entrada una forma ampliada de un lenguaje fuente y su salida es una forma estándar del mismo lenguaje fuente.

Funcionamiento del ensamblador

Un ensamblador traduce un archivo con sentencias en lenguaje ensamblador a un archivo de instrucciones máquina y datos binarios.

- Traducción en dos pasadas:
  - Primera pasada:
    - Calcula las posiciones de memoria que corresponden a los nombres simbólicos que aparecen en el programa para que sean conocidas cuando de traduzcan las instrucciones. Crea tabla de símbolos.
  - Segunda pasada:
    - Traduce cada sentencia del lenguaje ensamblador al combinar los equivalentes numéricos de los códigos de operación, especificadores de registros y rótulos de la tabla de símbolos en una instrucción legal.

El ensamblador como traductor

Punto de Partida

Programa fuente en lenguaje ensamblador

Traducor de Ensamblador

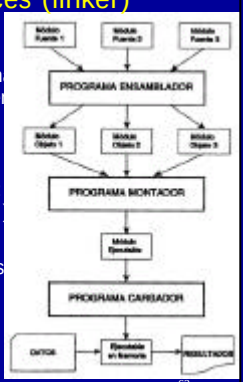
Genera Programa Objeto en algún otro lenguaje Lenguaje de Máquina

Es un traductor cuyo lenguaje objeto es también alguna variedad de lenguaje máquina, pero cuyo lenguaje fuente, un lenguaje ensamblador, constituye en gran medida una representación simbólica del código de máquina objeto.

Punto Final

El montador de enlaces (linker)

- El ensamblador procesa cada archivo de un programa de form individual. Solamente se conoce las direcciones de las etiquetas (rótulos) locales.
- Necesidad de otra herramienta: El montador de enlaces (o linker
- Este proceso consiste en unir los módulos objeto (archivos de programa) en un único módulo ejecutable.



El montador de enlaces (linker)

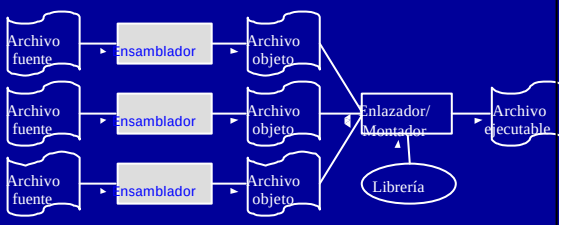
- Sus funciones son:
  - Combina una colección de archivos objetos y librerías (opcional) en otro archivo ejecutable al resolver los rótulos (etiquetas) externos.
  - El ensamblador asiste al montador suministrándole una tabla de símbolos o rótulos (etiquetas) y referencias no resueltas.
  - Resolver las referencias entre elementos de varios programas o módulos (referencias cruzadas).
  - Asignar direcciones a los diferentes módulos o etiquetas del programa.
  - Generar un archivo conteniendo el programa ejecutable.

El montador de enlaces (linker)

Los montadores pueden generar código de dos tipos:

- Código absoluto: en este caso, el montador asignará direcciones absolutas a las instrucciones y los datos.
- Código reubicable: el montador generará un módulo que puede ser ubicado en diferentes direcciones de memoria. Las referencias a memoria (instrucciones y datos) serán relativas.

El montador de enlaces (linker)



CARGA Y EJECUCIÓN DEL PROGRAMA

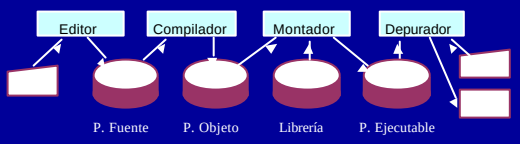
Consiste en la transferencia del programa ejecutable a la memoria del computador desde el archivo en disco, y en el posterior lanzamiento de su ejecución. La herramienta utilizada es el cargador.

- Este programa pertenece al sistema operativo.

El cargador puede ser de dos tipos:

- Cargador binario absoluto: necesario cuando el montador genera código absoluto. En este caso, el cargador toma el programa ejecutable y lo carga en las direcciones de memoria especificadas en el mismo, que son todas absolutas.
- Cargador reubicador: necesario cuando el montador genera código reubicable. En este caso, el cargador redefine las direcciones relativas presentes en el programa ejecutable cuando procede a cargarlo en memoria.

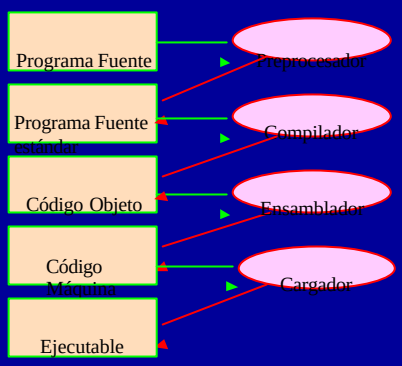
Entorno (ambiente) de programación



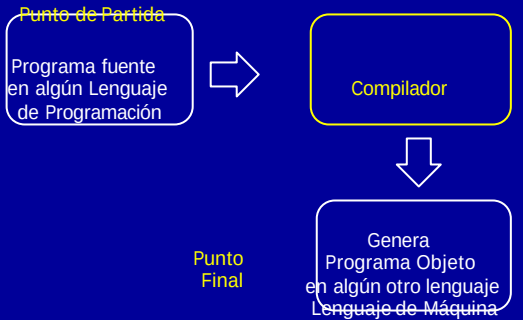
El Cargador es un traductor cuya entrada es un lenguaje objeto y la salida es un programa en lenguaje máquina relocalizable.

TIPOS DE SISTEMAS DE COMPILACIÓN

- **ENSAMBLADOR**
  Traducen programas escritos en lenguaje ensamblador a código máquina
- **COMPILADOR**
  Traducen programas escritos en lenguaje de alto nivel a código intermedio o a código máquina
- **INTERPRETE**
  No genera código objeto, analiza y ejecuta directamente cada proposición del Programa Fuente (PF)
- **PREPROCESADOR**
  Sustituyen macros, incluyen archivos o extensión del lenguaje.



La Compilación como traductor



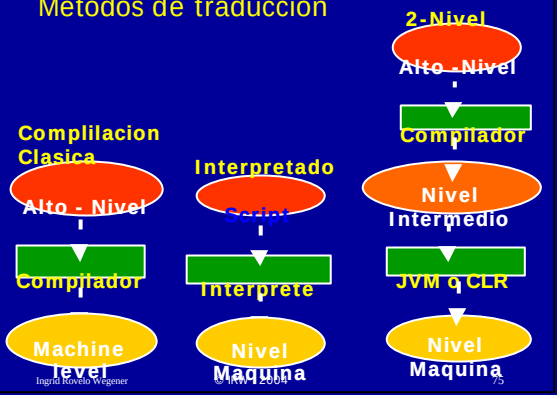
Compilación Clásica

- El compilador traduce el código fuente al código objeto a nivel máquina.
  - Se traduce una sola vez, se ejecuta muchas veces.
  - Salida: instruccions en lenguaje de máquina en la forma .EXE y .dlls .
  - Veloz ? porque la computadora ejecuta código de máquina y no tiene que traducirlo al mismo tiempo que ejecuta.

Interpretes

- Los Interpretes convierten el código a lenguaje de maquina línea por línea solo lo hace antes de la ejecucion del programa.
- Traduce cada vez que se ejecuta el programa
- Es mas lento porque tiene que traducir para cada uso (ejecucŏn)
- En un principio debido a la escasez de memoria se utilizaban más los intérpretes, ahora se usan más los compiladores (a excepción de JAVA).

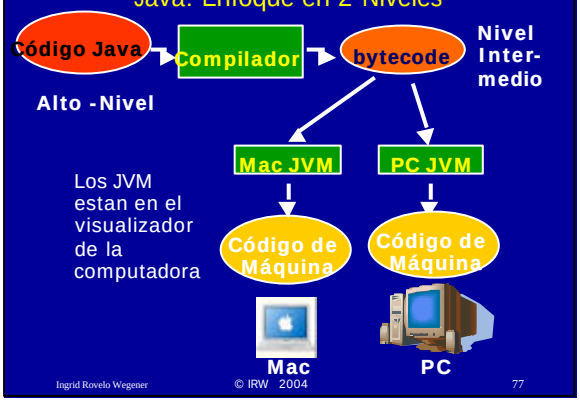
Metodos de traduccion



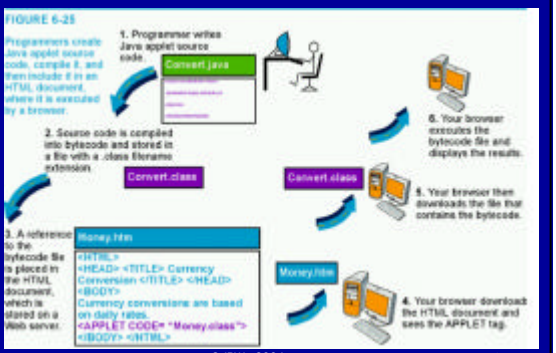
Métodos de Traducción

Tipo	Ejemplos	Cómo
Compilación Clásica	Pascal, C	Archivo completo
Interprete	HTML, Java Script	Línea por línea
Nivel 2	Java, Visual Basic.Net	Archivo completo, pero en dos estapas

Java: Enfoque en 2-Niveles



Creación y uso de los Applets de Java



Comparación de Java Applet y Script

Java Applets	JavaScript
Must be compiled into bytecode	Not compiled
Stored as a separate file	Entire script embedded within the HTML document
Referenced using the <APPLET> or <OBJECT> HTML tags	Identified by the <SCRIPT> HTML tag
Requires the browser to use a Java Virtual Machine	Requires a JavaScript interpreter
Programmers and Web page authors must use Java programming tools	No special programming tools required

Evaluacion de Scripts y Applets

- Aplicacion Cliente : El visualizador Web con interpreter o JVM
- Ventajas
  - Facil de actualizar en el servidor
    - Los Scripts y Applets son distribuidos desde el servidor al cliente en el tiempo de ejecucion
    - Independencia de la Platforma
- Desventajas
  - Lento rendimiento, el ancho de banda del Internet afecta el rendimiento , no son permitidos los applets a traves de algunos firewalls (muros contrafuegos)

Dimensión de un Compilador

- Reducción a subproblemas
- Teoría (Simplificar las soluciones)
  - Fib(n) = if n<2 then 1 else Fib(n-1)+Fib(n-2)
- Metodología (Implementación de programas grandes)

Ejemplo (programa de entrada)

```
int expr(int n)
{
        int d;
        d = 4 * n * n * (n + 1) * (n + 1);
        return d;
}
```


Ejemplo (Assembler de salida)

```
        lda $30,-32($30)
        stq $26,0($30)
        stq $15,8($30)
        bis $30,$30,$15
        bis $16,$16,$1
        stl $1,16($15)
        lds $f1,16($15)
        sts $f1,24($15)
        ldl $5,24($15)
        bis $5,$5,$2
        s4addq $2,0,$3
        ldl $4,16($15)
        mull $4,$3,$2
        ldl $3,16($15)
        addq $3,1,$4
        mull $2,$4,$2
        ldl $3,16($15)
        addq $3,1,$4
        mull $2,$4,$2
        stl $2,20($15)
        ldl $0,20($15)
        br $31,$33
$33:
        bis $15,$15,$30
        ldq $26,0($30)
        ldq $15,8($30)
        addq $30,32,$30
        ret $31,($26),1
```


Ejecución eficiente de las acciones

- Mapeo de alto a bajo nivel
  - Mapeo simple de un programa a assembler produce ejecución ineficiente
  - Más alto el nivel de abstracción ➡ más ineficiente
- Si no es eficiente
  - Abstracciones de alto nivel son inútiles
- Necesitamos
  - proveer un nivel alto de abstracción
  - con rendimiento equivalente a si usáramos assembler

Ejemplo (Assembler de salida)

No optimizado

```
        lda $30,-32($30)
        stq $26,0($30)
        stq $15,8($30)
        bis $30,$30,$15
        bis $16,$16,$1
        stl $1,16($15)
        lds $f1,16($15)
        sts $f1,24($15)
        ldl $5,24($15)
        bis $5,$5,$2
        s4addq $2,0,$3
        ldl $4,16($15)
        mull $4,$3,$2
        ldl $3,16($15)
        addq $3,1,$4
        mull $2,$4,$2
        ldl $3,16($15)
        addq $3,1,$4
        mull $2,$4,$2
        stl $2,20($15)
        ldl $0,20($15)
        br $31,$33
$33:
        bis $15,$15,$30
        ldq $26,0($30)
        ldq $15,8($30)
        addq $30,32,$30
        ret $31,($26),1
```

Optimizado

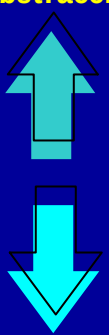
```
        s4addq $16,0,$0
        mull $16,$0,$0
        addq $16,1,$16
        mull $0,$16,$0
        mull $0,$16,$0
        ret $31,($26),1
```


Compiladores optimizan código para...

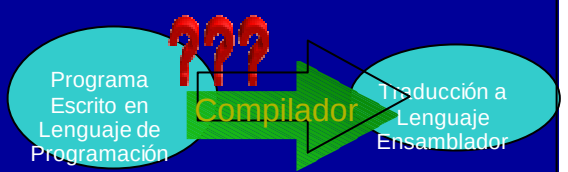
- Velocidad / Rendimiento
- Tamaño de código
- Consumo de Poder
- Compilación rápida / eficiente
- Seguridad / Confiabilidad
- Debugging (depuración)

Compiladores ayudan a incrementar el nivel de abstracción

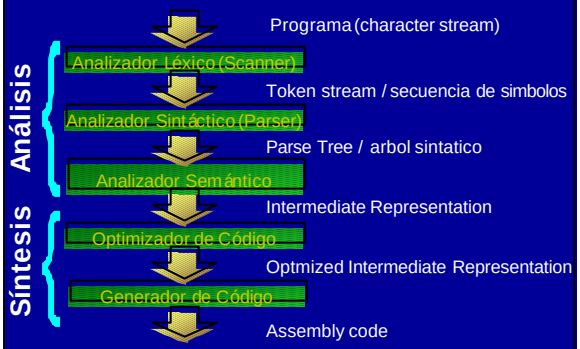
- Lenguajes de Programación
  - Desde C a lenguajes OO con garbage collection
  - definiciones aún más abstractas
- Microprocesadores
  - De simple CISC a RISC a VLIW a ...



Anatomía de un compilador



Anatomía de un compilador



Punto de partida

Lenguaje imperativo estándar (Java, C, C++)

- Estado
  - Variables,
  - Estructuras,
  - Arreglos
- Computación
  - Expresiones (aritméticas, logicas, etc.)
  - Instrucciones de asignación
  - Control del flujo (condicionales, ciclos)
  - Procedimientos

Punto Final – Ensamblador (PCSPIM)

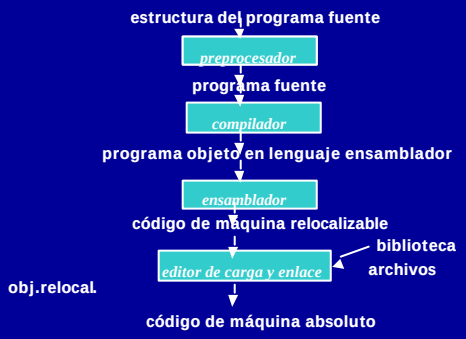
- Memoria (direccionamiento de 32 bit, direccionamiento del byte)
  - Banco de registros de **Enteros**, constituido por:
    - 32 Registros de 32 bits de propósito general para operaciones con enteros.
    - 2 Registros especiales de 32 bits HI y LO.
- Códigos de Condición
  - Indican resultados de operaciones con números enteros
  - Utilizadas en intrucciones ramificadas (branch)

Punto final – Computación (PCSPIM)

- ld <addr>,<reg>
- st <reg>, <addr>
- <binary op> <src1>, <src2>, <dst>
- <comp op> <src1>, <src2>
- <branch op> <address>
  - Condicional
  - Incondicional

Etiqueta	Instrucción	Operandos		Comentarios
		Destino	Fuentes	
demo:	add	$t1,	$s1, $v0	# $t1= $s1 + $v0
sumai:	addi	$t5,	$s2, 5	# $t5= $s2 + 5
carga:	lw	$s3,	100($zero)	# $s3= M[100+$zero] # $zero = 0 siempre!

Sistema Para Procesamiento de un Lenguaje



Verificación Del Funcionamiento Del Programa

- En este paso se produce la comprobación de los efectos causados por la ejecución de las instrucciones en las variables y el estado de la máquina.
- Esto puede efectuarse directamente, ejecutando el programa sin más, o bien a través de un depurador.

Verificación Del Funcionamiento Del Programa

Un depurador es una herramienta que permite efectuar las siguientes acciones:

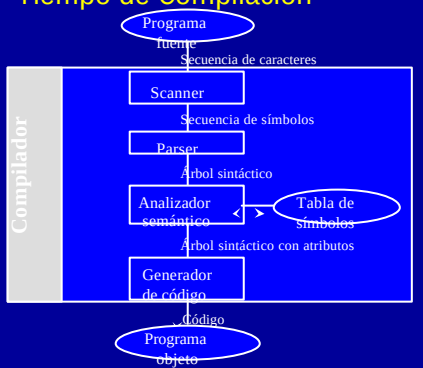
- Cargar programas en la memoria del computador.
- Ejecutar programas paso a paso o de forma continua.
- Establecer puntos de ruptura para detener la ejecución del programa en lugares determinados.
- Ver los contenidos de los registros y la memoria, y modificar sus contenidos en cualquier instante durante la ejecución de los programas.

Desarrollo cruzado de programas

A veces los programas no se desarrollan en la propia máquina objetivo en la que van a ser ejecutados, sino en máquinas más potentes en las que se dispone de las herramientas necesarias para desarrollar y probar dichos programas.

- En este caso, la máquina en la que se desarrolla el programa dispone de:
  - Un ensamblador cruzado que genera código objeto para la máquina objetivo.
  - Un montador de enlaces capaz de generar programas ejecutables para la máquina objetivo.
  - La posibilidad de conectarse a la máquina objetivo, o un simulador que permita emularla y que funcione en la propia máquina en la que se realiza el desarrollo.

Tiempo de Compilación



Compiladores

Programas que el compilador necesita para obtener un programa ejecutable

- Preprocesador
- Ligador
- Cargador
- Depurador
- Ensamblador

Que es lo que un compilador compila?

Source language:	**Target language:**
Programming language	**Machine language**
C++	codigo ensamblador MIPS
Programming language	**Abstract machine**
Java Java	Bytecode
Programming language	**Programming language (source-to-source)**
C++	C

Que es lo que un compilador compila?

Domain specific language	**Application language**
LaTeX	HTML
Data base language (SQL)	Data base system calls

Application generator :

Domain specific language	**Programming language**
SIM Toolkit	language Java

Some languages are **interpreted** rather than compiled :

Lisp, Prolog, Script languages like PHP, JavaScript, Perl

Comparación Intérpretes

↓Edición:
- **↓ Tiene su propio editor, corrige los errores de sintaxis.**

↓Ejecución:
- **↓ Si se realiza una modificación en el código fuente, se puede probar automáticamente.**
- **↓ Ante una nueva corrida del programa, el mismo debe ser interpretado nuevamente.**

↓Ocupación de memoria:
- **↓ Carga el programa a interpretar en lenguaje fuente.**

↓Rapidez:
- **↓ El programa fuente se debe traducir cada vez que se ejecuta y por ello el tiempo de ejecución será la suma de tiempo de la traducción de la instrucción a código máquina y de la subsiguiente ejecución.**

Comparación Compiladores

↓Edición:
- **↓No tiene su propio editor, no corrige los errores de sintaxis, salvo algunos desarrollos nuevos que sí lo hacen.**
- **↓Se puede trabajar en forma de intérprete para el desarrollo del programa y después compilar.**

↓Ejecución:
- **↓Si se realiza una modificación en el código fuente, se debe volver a compilar.**

↓Ocupación de memoria:
- **↓Solo carga el código objeto (programa compilado).**

↓Rapidez:
- **↓El tiempo de ejecución sólo es el tiempo que implica la propia ejecución del programa compilado en código máquina**

Ventajas

Compilar vs a interpretar
- Se compila una vez, se ejecuta n veces
- En ciclos, la compilación genera código equivalente, interpretándolo se traduce tantas veces una línea como veces se repite el ciclo
- El compilador tiene un visión global del programa

Intérprete vs el compilador
- Un intérprete necesita menos memoria que un compilador.
- Permiten una mayor interactividad con el código en tiempo de desarrollo.

Tipos de Compiladores

- Existen gran variedad de lenguajes fuente y lenguajes objeto. Un lenguaje objeto puede ser otro lenguaje de programación o un lenguaje máquina de cualquier computador, pudiendo tener éste uno o varios procesadores.
- Los compiladores se pueden clasificar según varias categorías:
  - De una pasada o múltiples pasadas,
  - De carga o ejecución,
  - De depuración o de optimización.

Explorando el comportamiento de un compilador

- Ya iniciaron con la entrada del programa en ensamblador
- Compilarlo un programa fuente en lenguaje de alto Nivel
- Intentar igualar el código fuente con el código generado.
- El estado de Translación
  - Variables (global, local, parámetros)
  - Estructuras y arreglos
- Translación Computacional
  - Evaluación y asignación de Expresiones
  - La construcción del Flujo de control
  - Las llamadas a rutinas y el regreso de éstas (Procedure call and return)

HOY.... Y A FUTURO

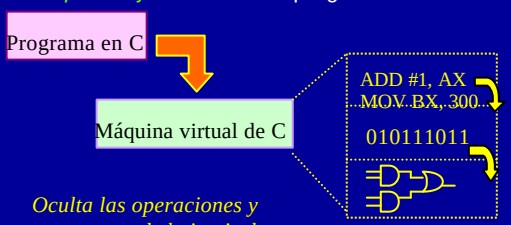
- El Diseño de un compilador surge como resultado de:
  - Desarrollo de un nuevo lenguaje de programación
  - Adición de extensiones a los ya existentes
  - Explotación de las características del hardware
- A futuro:
  - Extensión para el cómputo paralelo y distribuido
  - Explotación de características multimedia (MMX)

Máquina Virtual

- Simulan una funcionalidad no nativa del entorno
- Por medio de intérpretes y sistemas basados en reglas

Máquina Virtual

- Son las estructuras de datos y algoritmos de un lenguaje que se emplean durante el *tiempo de ejecución* de un programa.

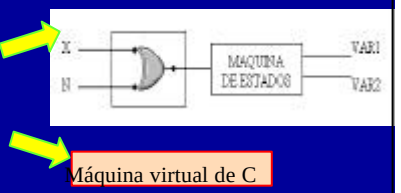


Oculta las operaciones y estructuras de bajo nivel

Relación entre Lenguaje y Máquina Virtual

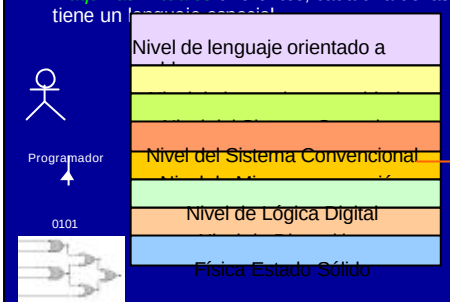
```
Código en C

if (x==n) {
  var1 = 1;
  var2 = 0;

} else {
     var1 = 0;
     var2 = 1;
  }
var1 = 0;
var2 = 0;
```

- Una máquina define un lenguaje
- Un lenguaje define una máquina



Jerarquía de Máquinas Virtuales

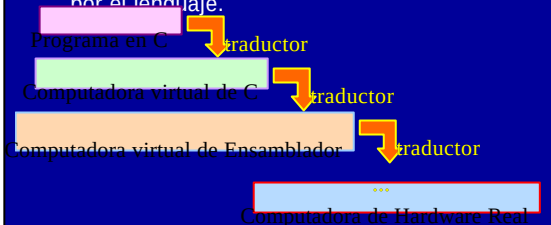
- Una computadora con *n* niveles puede verse como *n máquinas virtuales* diferentes, cada una de las cuales tiene un lenguaje especial



Los RISC no cuentan con una descripción de microinstrucciones

Traductor para la Máquina Virtual

- Se debe suministrar un *traductor* para traducir programas de usuario al lenguaje de máquina de la computadora virtual definida por el lenguaje.



Jerarquía de Máquinas Virtuales

- Cada nivel representa una abstracción con objetos y operaciones diferentes
- Cada nivel esta construido sobre su predecesor
- Si se quiere escribir programas para la máquina virtual del nivel n, no se necesita conocer los interpretes ni los traductores de los niveles de abajo.

Jerarquía de Máquinas Virtuales

- *Las computadoras están diseñadas como una serie de niveles*
- *Para diseñar nuevos niveles, se necesita conocer todos.*

La Máquina Virtual de Java

¿Por qué están de moda los Bytecodes

Con el lenguaje de programación Java, Puedes

"escribir una vez, y ejecutar en cualquier parte".

La Máquina Virtual de Java

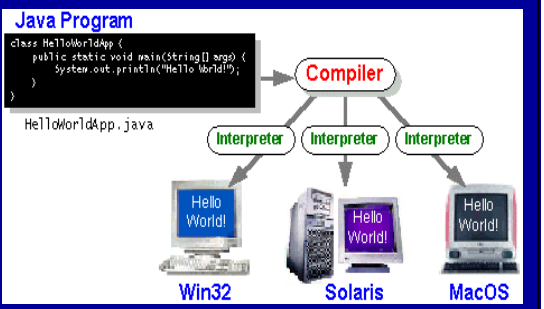
Esto significa que cuando se compila un programa, no se generan instrucciones para una plataforma específica.

En su lugar, se generan bytecodes Java, que son instrucciones para la Máquina Virtual Java (Java VM). Si tu plataforma- sea Windows, UNIX, MacOS o un navegador de internet-- tiene la Java VM, podrá entender los bytecodes.

¿Cómo es el proceso en Java?

- A diferencia de otros lenguajes de alto nivel, Java ofrece una extraña combinación entre compiladores e interpretes.
- Primero, el código fuente es **compilado** a un código intermedio, conocido como **bytecode**.
- Segundo, para que un programa pueda ser ejecutado, el **bytecode** debe ser **interpretado** y traducido a lenguaje maquina.

La Máquina Virtual de Java



Compilación

Antes

- Una computadora no tenía memoria suficiente
- Se tuvo que dividir al compilador en fases
- Cada fase leía un archivo y producía otro

Actualmente

- Se tiene memoria suficiente
- El tamaño del archivo ejecutable es relativamente pequeño
- Se han reducido el número de pasadas y el número de archivos que se tienen que leer y escribir

Compilación

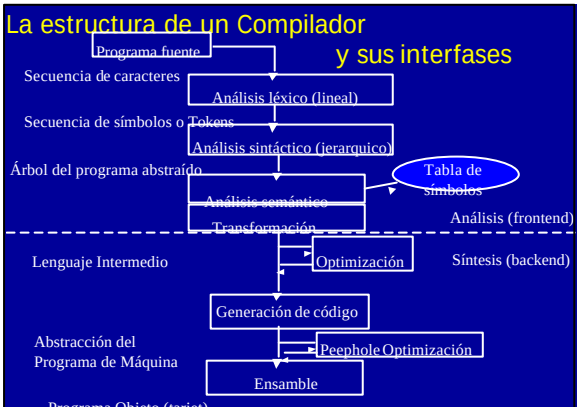
Front end

- Dependiente del lenguaje fuente
- Independiente de la máquina objeto para la que se va a generar código

Back end

- Independiente del lenguaje objeto
- Dependiente del lenguaje objeto

La estructura de un Compilador y sus interfases



Partes de la compilación

- ANÁLISIS (Etapa Inicial):
  Divide al Programa fuente en sus elementos componentes y crea una representación intermedia.
  Se determinan las operaciones y se registran en una estructura de árbol (ej. árbol sintáctico)
- SÍNTESIS (Etapa Final):
  Construye el Programa Objeto deseado a partir de la representación Intermedia (requiere técnicas más especializadas)

Análisis Lexicográfico y Sintáctico

- Gramáticas y Lenguajes de Programación
  - ¿Cómo describir las formas / combinaciones válidas del programa fuente?
  - Gramáticas: Definición. Producciones
  - Distintos tipos de gramáticas (Chomsky):
    - Tipo 0 Generales
    - Tipo 1 Sensitivas al contexto
    - Tipo 2 Libres de contexto
    - Tipo 3 Regulares

Análisis Lexicográfico y Sintáctico

- Partición en subproblemas: Tres niveles de gramáticas.
  - Lexicográfica
  - Sintáctica
  - Semántica

Análisis del programa fuente

- El análisis consta de tres fases:
  - *Análisis lineal (o léxico-* **Exploración- Scanner***) :* se lee la cadena de caracteres de izquierda a derecha agrupando componentes léxicos, que son secuencias de caracteres que tienen un significado como agrupación; es decir, Palabra (Token): palabras en que puede ser particionada una frase determinada.
  - *Análisis jerárquico (o sintáctico-* **Parser***)* : en el que los caracteres o componentes léxicos se agrupan jerárquicamente en colecciones anidadas. con un significado colectivo (frases gramaticales que por lo general se representan mediante árboles sintácticos)
  - *Análisis semántico*: se realizan comprobaciones para asegurar que los componentes se ajustan de un modo significativo.

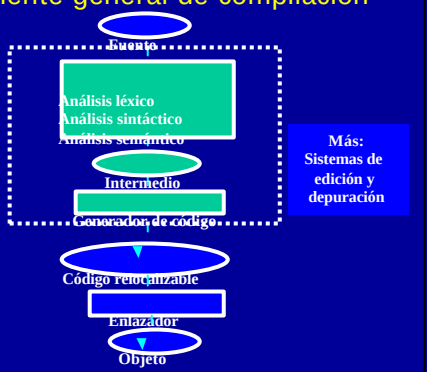
Niveles de Análisis

- Análisis Lexicográfico (Scanner)
  - Expresiones regulares
  - secuencia de caracteres => secuencia de símbolos
- Análisis Sintáctico (Parser)
  - Gramáticas libres del contexto
  - Descubrir la estructura gramatical del programa
  - Construir el árbol sintáctico
- Análisis Semántico
  - Validar la semántica del programa (información sensitiva al contexto)
  - Formalización (gramática de atributos)
  - Generación / utilización de la información semántica
  - Tabla de Símbolos

Errores y Generación de código

- Control de errores
  - Sólo detectar el 1er. error
  - Detectar todos los errores, pero no demasiados
    - Sincronización después del error
    - Corrección de errores
- Generación de código
  - Formalización (semántica interpretativa)
  - En un solo paso
  - Mediante un código intermedio
    - Con optimización local
    - Con optimización global
    - Con optimización dependiente de máquina

Un ambiente general de compilación



Análisis del Programa Fuente

Análisis Lineal: la cadena de caracteres que constituye el programa fuente se lee de izquierda a derecha y se agrupa en **Componentes Léxicos**, que son secuencias de caracteres que tienen un significado colectivo.

posicion := inicial + velocidad * 60

b) Análisis sintáctico (reglas recursivas)

- Las construcciones léxicas no requieren recursión (ej. Reconocer un identificador) mientras que las sintácticas suelen requerirlas (ej. Emparejamiento de paréntesis o Begin-End)
- La estructura jerárquica de un programa normalmente se expresa mediante reglas recursivas

  Exp :: ident | nro

  Exp :: Exp + Exp | Exp * Exp | (Exp)
- Las gramáticas libres de contexto (GLC) son una formalización de reglas recursivas que pueden guiar el análisis sintáctico

EJEMPLO DE ANÁLISIS:
posicion := inicial + velocidad * 60

a) Componentes léxicos:

1. El identificador ***posicion***
2. El símbolo de asignación ***:=***
3. El identificador ***inicial***
4. El signo de suma: ***+***
5. El identificador ***velocidad***
6. El signo de multiplicación: *******
7. El número ***60***

Los identificadores o nombres reconocidos se organizan en una *tabla de símbolos* que se usará en los pasos siguientes

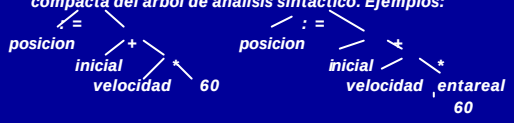
posicion := inicial + velocidad * 60

b) Análisis sintáctico (árbol de analis. sint.)

posicion := inicial + velocidad * 60

b) Análisis semántico

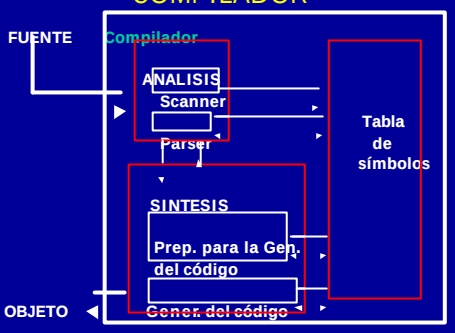
- *Significado de una unidad gramatical, interpretación*
- *Traducir la entrada a una forma de representación intermedia*
- *Análisis y verificación de tipos*
- *Utiliza la estructura jerárquica del Análisis sintáctico*
- *El árbol sintáctico permite una representación interna compacta del árbol de análisis sintáctico. Ejemplos:*



FASES DE UN COMPILADOR

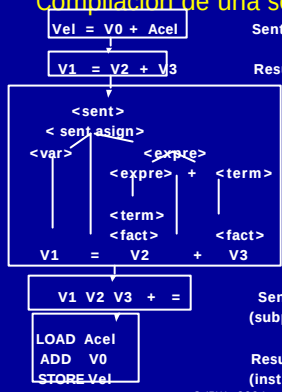
Cada fase transforma al PF de una representación a otra

ESQUEMA DE BLOQUES DE UN COMPILADOR



ESTRUCTURA FUNCIONAL DE UN COMPILADOR (de una pasada)

Compilación de una sentencia ejemplo



Sentencia fuente a compilar

Resultado del EXPLORADOR

RECONOCEDOR:

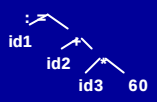
Análisis sintáctico:
la sentencia es correcta

Sentencia en notación polaca
(subproducto del reconocedor)

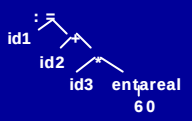
Resultado del GEN. DE CÓDIGO
(instrucciones para máquina)

posicion := inicial + velocidad * 60

- A. Lex: id1 : = id2 + id3 * 60
- A. Sint:



- A. Seman:



ADMINISTRADOR DE LA TABLA DE SÍMBOLOS

- **TABLA DE SÍMBOLOS**
  Estructura de datos que contiene un registro por cada identificador, con los campos para los atributos:
  -- Información sobre la memoria asignada
  -- tipo
  -- ámbito
  -- Si es nombre de procedimiento (número, tipo
  y método de paso de cada argumento)
- Permite encontrar rápidamente cada ID y almacenar o consultar datos de ese registro
- En el Análisis Léxico se detectan los ID y se introducen en la Tabla de Símbolos
- Las fases restantes introducen información sobre los ID y después la utilizan

DETECCIÓN E INFORMACIÓN DE ERRORES

Cada fase puede encontrar errores y debe tratarlo para continuar con la Compilación, permitiendo detectar más errores

Las fases de Analisis Sintáctico y Semántico manejan la mayoría de los errores

En el Anáilsis Semántico se detectan errores donde la estructura sintáctica es correcta pero no tiene significado la operación
(Por. ej. sumar dos ID , donde uno es el nombre de una matriz y el otro un nombre de procedimiento)

GENERACIÓN DE CÓDIGO INTERMEDIO

- Se genera una representación intermedia explícita del PF
- La representación intermedia es como un programa para una máquina abstracta
- Esta representación debe ser fácil de producir y de traducir al programa objeto
- Una de ellas es el "código intermedio de 3 direcciones" (Cada posición de memoria puede actuar como registro) (Cada instrucción tiene como máximo 3 operandos)

Ejemplo t1 := entareal (60)
t2 := id3 + t1
t3 := id2 + t2
id1 := t3

OPTIMIZACIÓN DE CÓDIGO

- Trata de mejorar el código intermedio para que resulte un código de máquina más rápido de ejecutar
- En el ejemplo: t1 : = id3 * 60.0
  id1 : = id2 + t1
  La conversión a real se hace en compilación
  No necesita t2 ni t3.
- **Compiladores optimadores** : La fase de optimación ocupa una parte significativa del tiempo del compilador
- Hay optimaciones sencillas que mejoran el tiempo de ejecución del programa sin retardar mucho la compilación

GENERACIÓN DE CÓDIGO

- La fase final genera código objeto (en general código de máquina recolalizable o código ensamblador)
- Se seleccionan las posiciones de memoria para las variables usadas por el programa .
- Se traduce cada una de las instrucciones intermedias a una secuencia de instrucciones de máquina
- Un aspecto decisivo es la asignación de variables a registros.
- En el ejemplo, utilizando los registros 1 y 2:

```
MOVF   id3, R2
MULF   %  60.0, R2
MOVF   id2, R1
ADDF   R2, R1
MOVF   R1, id1
```


PROGRAMAS RELACIONADOS CON UN COMPILADOR

- **PREPROCESADORES** (producen la entrada para un comp.)
  Procesamiento de Macros
  Inclusión de archivos
  Preprocesadores " racionales" (*estruct. de control)*
  Extensiones a lenguajes (*bases de datos)*
- **ENSAMBLADORES**
  Producen código ensamblador que se pasa a un ensamblador para su procesamiento (versión mnemotécnica del código de máquina: nombres de operaciones y nombres de direcciones de memoria)
- **ENSAMBLADO DE DOS PASADAS** (lecturas del archivo IN)
  Primera: Identificadores - Tabla de símbolos
  Segunda: Traduce códigos de operaciones e identificadores
  El resultado es código de maquina relocalizable
- **CARGADORES Y EDITORES DE ENLACE**
  Modifica las direcciones relocalizables y ubica en memoria.
  Forma un solo prog. desde varios archivos relocalizables

AGRUPAMIENTO DE FASES EN LA IMPLEMENTACION

- **ETAPA INICIAL Y ETAPA FINAL**

  Inicial : Fases que dependen del lenguaje fuente Hasta cierta optimación

  Final : Partes que dependen de la maq. objeto y del leng. intermedio

- **PASADAS**

  Se agrupan las actividades de varias fases en una misma pasada (lectura de un archivo de entrada y escritura de un archivo de salida)

- **REDUCCION DEL NUMERO DE PASADAS**

  Pocas pasadas --> Varias fases dentro de una pasada --> Prog. completo en memoria en representación intermedia Fusión de código intermedio y objeto: " backpatching"

HERRAMIENTAS PARA CONSTRUCCIÓN DE COMPIL.

- **SIST. DE AYUDA PARA ESCRIBIR COMPILADORES**

  Comp. de comp. / Generadores de comp. / Sist. generadores de traductores

- **HERRAMIENTAS GENERALES PARA EL DISEÑO AUTOMÁ-TICO DE COMPONENTES ESPECÍFICOS DE UN COMP.**

  Utilizan leng. específicos para especificar e imple - mentar la componente

  Ocultan detalles del algoritmo de generación

  Producen componentes que se pueden integrar al resto del compilador

Herramientas para construcción de C.

- **GENERADORES DE ANALIZADORES SINTACTICOS**

  Producen AS a partir de una Gramática Libre de Contexto Hoy esta es una de la fases más fáciles de aplicar

- **GENERADORES DE ANALIZADORES LEXICOS**

  Producen AL a partir de una especificación en Expres. Regulares. El AL resultante es un Autómata Finito

- **DISPOSITIVOS DE TRADUC. DIRIGIDA POR LA SINTAXIS**

  Producen grupos de rutinas que recorren el árbol de AS generando código intermedio

- **GENERADORES AUTOMÁTICOS DE CÓDIGO**

  Las proposiciones en cod. Int. se reemplazan por plan- tillas que representan secuencia de instruc. de máquina

- **DISPOSITIVOS PARA ANALISIS DE FLUJO DE DATOS**

  Inf. sobre como los valores se transmiten de una parte a otra del programa

Herramientas para construcción DE COMPIL.

Lex y YACC

Herramientas que nos permiten desarrollar componentes o la mayor parte de un compilador

Son un recurso invaluable para el profesional y el investigador

Existen paquetes freeware

Algunos tipos especiales de compiladores

- **COMPILE- LINK - GO**
  Se compilan segmentos por separado y luego se montan todos los objetos producidos en un módulo cargable listo
- **COMPILADOR DE VARIAS PASADAS**
  No es más lento. Ocupa poca memoria. Fácil de mantener
- **COMPILADOR INCREMENTAL (o interactivo)**
  Se pueden compilar solo las modificaciones
- **AUTOCOMPILADOR**
  Comp. escrito en el propio leng. que traduce. Portabilidad.
- **METACOMPILADOR**
  Programa al que se le especifica el lenguaje para el que se quiere un comp. y produce el comp. como resultado
- **DECOMPILADOR**
  Traduce de código máquina a leng. de alto nivel

El lenguaje y la herramienta

MODELO	LENGUAJE	CARACTERISTICAS
Compilado	Fortran, COBOL, C/C++, Pascal	Sintaxis rigurosa, velocidad y tamaño
Interpretado	Lisp,AWK, BASIC, SQL	Desempeño lento. Actividades no planea- das. Sintaxis relajadas
Pseudocompilado	Java	Transportabilidad absoluta, desempeño intermedio. Sintaxis rigurosa

Aspectos academicos y de investigacion

AREA	BENEFICIOS
Leng. de prog.	Principios para su desarrollo Herramientas para implementación
Inteligencia artificial	Interfases de reconocimiento de lenguaje natural
Sistemas operativos	Desarrollo de interfases de control y usuario final. Intérpretes de comandos (shells)
Diseño de interfaces	Desarrollo de interf. orientadas a comando y carácter. Voz o escritura
Administración de proyectos inform.	Selección de herramientas de desarrollo. Evaluación de costo y beneficios.

¿Qué es un analizador léxico?

Source Program Text → Tokens

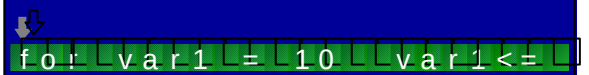
- ¿Qué hacemos en procesamiento de lenguaje natural?
- Primero tokenizamos
- Ejemplo:
  - Holacomoestantodos
- Se convierte en:
  - Hola como estan todos

¿Qué es un analizador léxico?

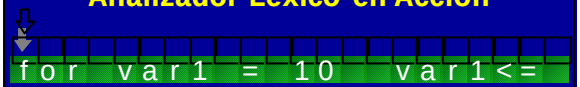
Source Program Text  Tokens

- Ejemplos de Tokens:
  - operadores = + - > ({ := == <>
  - keywords if while for int double
  - literales numéricos 43 6.035 -3.6e10 0x13f3a
  - literales de carácter 'a' '~' '\''
  - strings literales "6.983" "compiladores" "\"\""
- Ejemplos de no-tokens
  - espacios en blanco espacio(' ') tab('\t') eol('\n')
  - comentarios /* este no es un token */

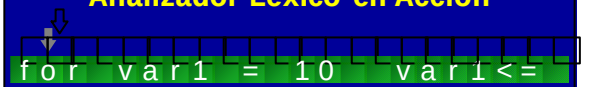
Analizador Léxico en Acción



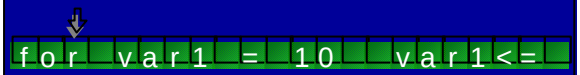
Analizador Léxico en Acción



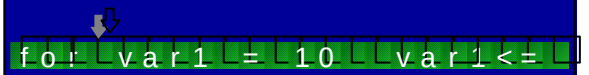
Analizador Léxico en Acción



Analizador Léxico en Acción



Analizador Léxico en Acción



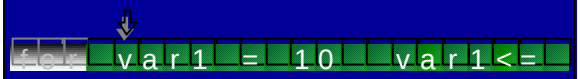
Analizador Léxico en Acción

for

Analizador Léxico en Acción

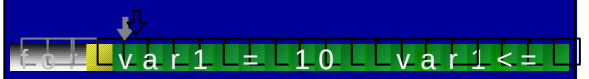
for

Analizador Léxico en Acción



for

Analizador Léxico en Acción



for

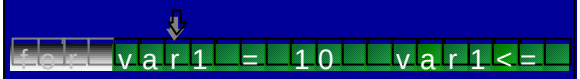
Analizador Léxico en Acción

for

Analizador Léxico en Acción

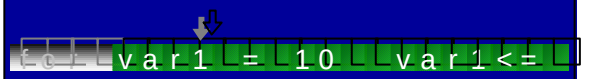
for

Analizador Léxico en Acción



for

Analizador Léxico en Acción



for

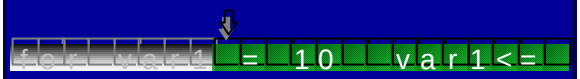
Analizador Léxico en Acción

for

Analizador Léxico en Acción

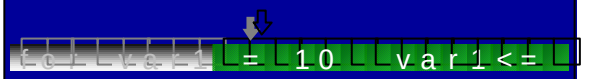
for ID(var1)

Analizador Léxico en Acción



for ID(var1)

Analizador Léxico en Acción

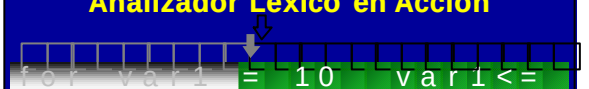


for ID(var1)

Analizador Léxico en Acción

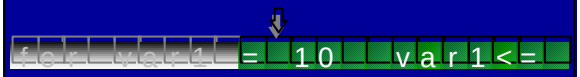
for ID(var1)

Analizador Léxico en Acción



for ID(var1)

Analizador Léxico en Acción



for ID(var1)

Analizador Léxico en Acción

for ID(var1) eq_op

Analizador Léxico en Acción

for ID(var1) eq_op

Analizador Léxico en Acción

for ID(var1) eq_op

Analizador Léxico en Acción

for ID(var1) eq_op

Analizador Léxico en Acción

for ID(var1) eq_op

Analizador Léxico en Acción

for ID(var1) eq_op

Analizador Léxico en Acción

for ID(var1) eq_op

Analizador Léxico en Acción

for ID(var1) eq_op Num(10)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10) ID(var1)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10) ID(var1)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10) ID(var1)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10) ID(var1)

Analizador Léxico en Acción

for ID(var1) eq_op Num(10) ID(var1) leq_op

Analizador Léxico en Acción

for ID(var1) eq_op Num(10) ID(var1) leq_op

Analizador Léxico en Acción

for ID(var1) eq_op Num(10) ID(var1) leq_op

Analizador Léxico en Acción

for ID(var1) eq_op Num(10) ID(var1) leq_op

Analizadores Léxicos deben...

- Particionar el texto del programa de entrada en una subsecuencia de caracteres correspondientes a tokens
- attacharle los atributos correspondientes a los tokens
- eliminar espacios en blanco y comentarios

¿Por qué esto no es trivial?

for ID(var1) eq_op Num(10) ID(var1) leq_op

- Separar precisamente el stream de texto en el stream correcto de tokens
  - ID("var1") no ID("var") Num(1)
  - ID("var1") leq_op no ID("var1<=")