

# Introducción

Pedro O. Pérez M., PhD.

Desarrollo de aplicaciones avanzadas de ciencias computacionales

Módulo 3: Compiladores

Tecnológico de Monterrey

*pperezm@tec.mx*

03-2025

## ① 1.1 - 1.3

1.1 Procesadores de lenguajes

1.2 La estructura de un compilador

1.3 La evolución de los lenguajes de programación

## ② 1.4 - 1.6

1.4 La ciencia de construir un compilador

1.5 Aplicaciones de la tecnología de los compiladores

# ¿Qué es un compilador?

Un compilador es una herramienta que traduce software escrito en un lenguaje a otro. Para realizar esta traducción, la herramienta debe ser capaz de comprender el lenguaje original (sintaxis, semántica). Además, necesita comprender el lenguaje final. Finalmente, debe tener un esquema que le permita hacer la traducción.

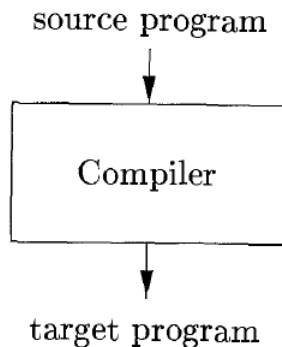


Figure 1.1: A compiler

Un interprete es otro tipo de herramienta común. En vez de producir un programa destino (o traducción), un interprete ejecuta directamente las operaciones especificadas en el programa fuente con las entradas dadas por el usuario.

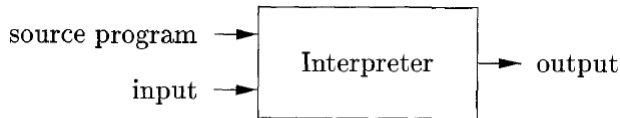


Figure 1.3: An interpreter

Un código destino producido por un compilador es usualmente más rápido que una código que está siendo interpretado. Sin embargo, un interprete puede, usualmente, dar mejor diagnóstico de errores que un compilador, porque ejecutar el código fuente instrucción por instrucción.

Pero, además tenemos esquemas híbridos...

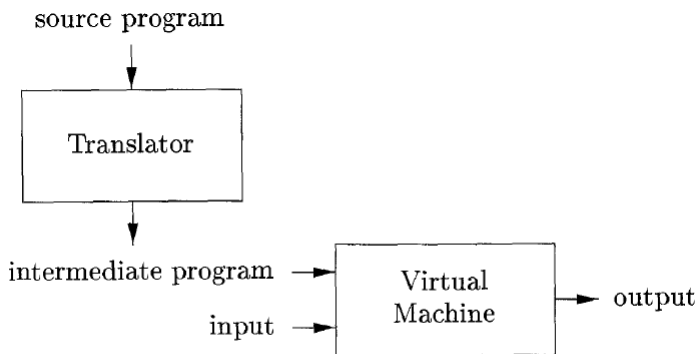


Figure 1.4: A hybrid compiler

# ¿Cómo es el proceso de generación de un código fuente?

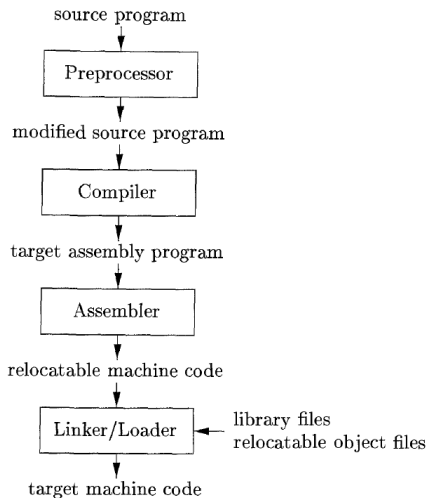
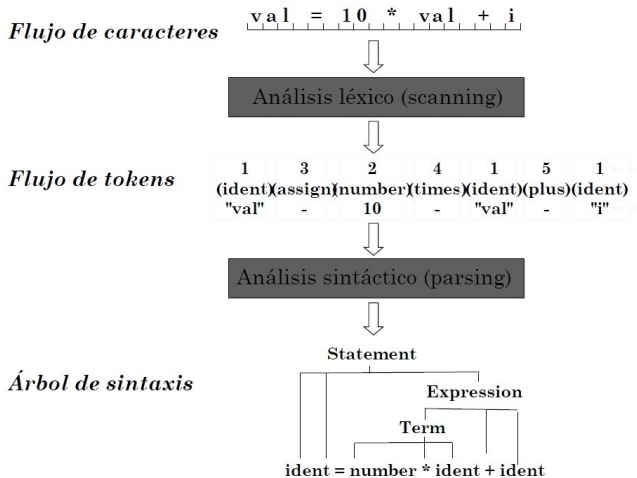


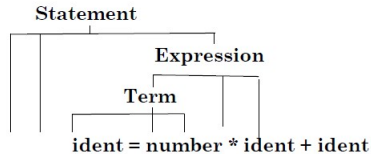
Figure 1.5: A language-processing system

## Estructura de un compilador





### Árbol de sintaxis



## Análisis semántico (type checking)



Árbol de Sintaxis, tabla de símbolos



# Optimización



## Generación de código



```
ld.i4.s 10
ldloc.1
mul
...
```

### Representación intermedia

*Código de máquina*

# Herramientas de construcción de compiladores

Existen herramientas especializadas definir e implementar componentes específicos:

- *Parser generators*: Automáticamente produce analizadores léxicos a partir de una descripción gramatical de un lenguaje de programación.
- *Scanner generators*: Producen analizadores léxicos a partir de la expresiones regulares que definen los “tokens” de un lenguaje de programación.
- *Syntax-directed translation engines*: Producen colecciones de rutinas para recorrer un árbol de análisis sintáctico para la generación de código intermedio.
- *Code-generator generators*: Producen generadores de códigos a partir de una colección de reglas de traducción de código intermedio a código fuente.

- *Data-flow analysis engine*: Facilitan la recuperación de información acerca de cómo los valores son transmitidos de una a otra parte del programa. El análisis de flujo de datos es una parte importante de la optimización de código.
- *Compiler-construction toolkits*: Proveen de una conjunto de rutinas para la construcción de las varias fases de un compilador.

## Instrucciones:

- De manera individual, lee el documento que se encuentra en el ejercicio 1.1 que se encuentra en Canvas.
- Genera un breve resumen de las ideas que ahí se describen, envíalo como evidencia de esta actividad y discútelo en clase.

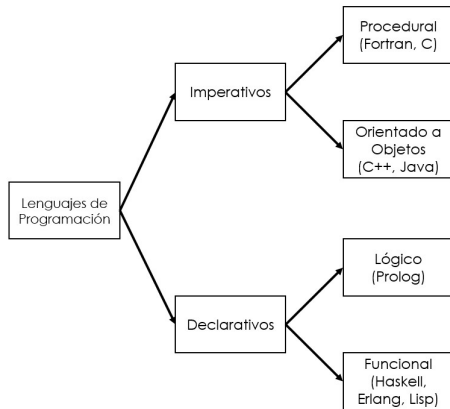
# La evolución de los lenguajes de programación

## Generaciones

- Primera generación:
  - Entendido directamente por las computadoras.
  - Utilizan instrucciones del procesador.
  - Dependientes del procesador.
  - En código binario.
- Segunda generación:
  - Siguen siendo dependientes del procesador.
  - Utilizan mnemónicos para representar instrucciones del procesador.
  - Más fáciles de recordar y leer.

- Tercera generación:
  - Independientes del procesador.
  - Utilizan variables y secuencias.
  - Incluyen ramas y ciclos.
- Cuarta generación:
  - Independientes del procesador.
  - Utilizan el llenado de formularios.
  - Gráficos asistidos por computadoras.
- Quinta generación:
  - Dependiente del procesador.
  - Utiliza técnicas de IA.
  - La computadora obtiene inferencias del código.

# Paradigmas



### Instrucciones:

- En parejas,
- Discután cuáles de los siguientes términos: imperativo, declarativo, von Neumann, orientado a objetos, funcional, tercera generación, cuarta generación, scripting se pueden aplicar a los siguientes lenguajes: C, C++, COBOL, Fortran, Java, Lisp, Scala, Perl, Python, Rust, Go.
- Escribe un reporte con los resultados. De forma individual, sube el reporte generado a Canvas en la sección de Tareas.



# La ciencia de construir un compilador

Un buen compilador es un microcosmos del área de las ciencias computacionales: hace uso práctico de algoritmos codiciosos (asignación de recursos), técnicas de búsqueda heurística (programación de listas), algoritmos de grafos (eliminación de código muerto), programación dinámica (selección de instrucciones), autómatas finitos y push-down (escaneo y análisis), entre muchos otros temas. En otras palabras, trabajar en el diseño y programación de un compilador proporciona experiencia práctica en ingeniería de software que es muy difícil de obtener en sistemas más pequeños y menos complejos.

# Aplicaciones de la tecnología de los compiladores

- Implementación de lenguajes de programación de alto nivel.
- Optimizaciones para arquitecturas computacionales:
  - Paralelismo a nivel de instrucciones.
  - Jerarquía de memorias.
- Diseño de nuevas arquitecturas computacionales: Desde que los lenguajes de alto nivel son la norma, el desempeño de los sistemas computacionales está determinado no solo por su velocidad sino, también, por lo bien que los compiladores explotan sus características.
- Traducción de programas: traducción binaria (convertir código máquina a otro), hardware sintético (VHDL), interpretes de consultas a base de datos.