Compiladores 24-2
Introducción

Lourdes del Carmen González Huesca
luglzhuesca@ciencias.unam.mx

Facultad de Ciencias, UNAM

31 enero 2024

Introducción

El estudio de lenguajes de programación es importante en Ciencias de la Computación:

- ✓ comprensión en el diseño de lenguajes y su implementación
- ✓ selección del lenguaje que sea mejor para resolver una tarea

Introducción

El estudio de lenguajes de programación es importante en Ciencias de la Computación:

- ✓ comprensión en el diseño de lenguajes y su implementación
- ✓ selección del lenguaje que sea mejor para resolver una tarea

Este estudio no involucra sólo los formalismos vistos en el curso de Lenguajes de Programación (sintaxis de lenguajes, diseño de sistema de tipos, manejo de variables, estilos de reducción, etc.) sino también la forma en que será evaluado un programa es decir las herramientas que permiten pasar del alto nivel al bajo nivel:

- entender y hacer buen uso de *debuggers*, ligadores, ensambladores
- comprender algunas características de los estilos de lenguajes como el manejo de estructuras de control, subrutinas o variables
- comprender otros detalles como los análisis sintácticos, optimizaciones y representaciones de código

Introducción

Un compilador **traduce** un lenguaje de alto nivel (adaptado a los seres humanos) en un lenguaje de bajo nivel (diseñado para ser ejecutado "eficientemente" por una computadora):

- ★ traducciones mediante **fases** bien definidas en donde cada una de ellas recaba o compila información útil para las fases siguientes;
- ★ en caso de que alguna fase no acepte el código de entrada se señalarán los **errores encontrados**;
- ★ Cada una de las transiciones debe **preservar el significado del programa fuente**, escrito en un lenguaje (imperativo), en un programa objeto, escrito en un lenguaje de bajo nivel, para finalmente ser ejecutado por un procesador designado.

Introducción

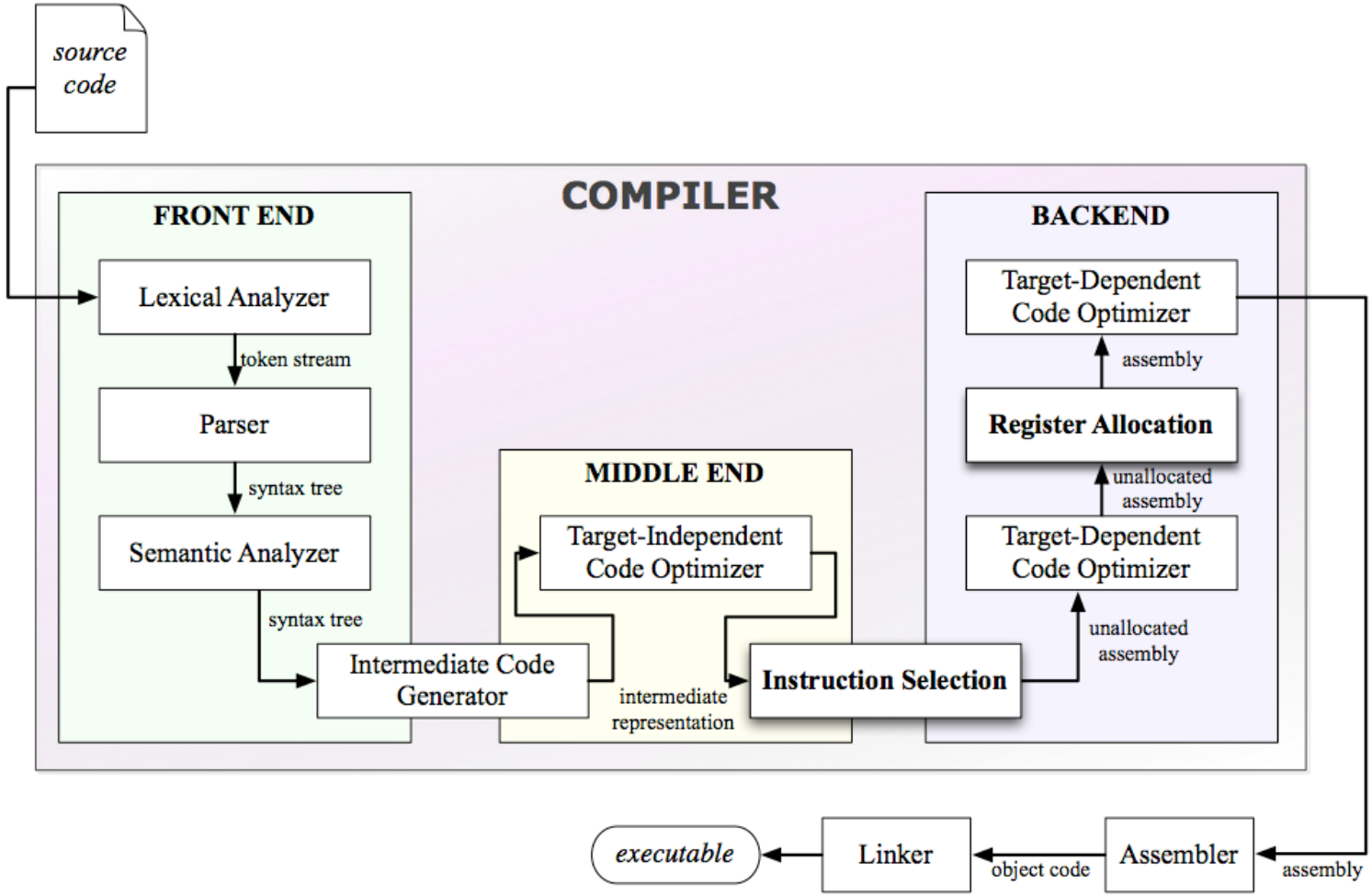
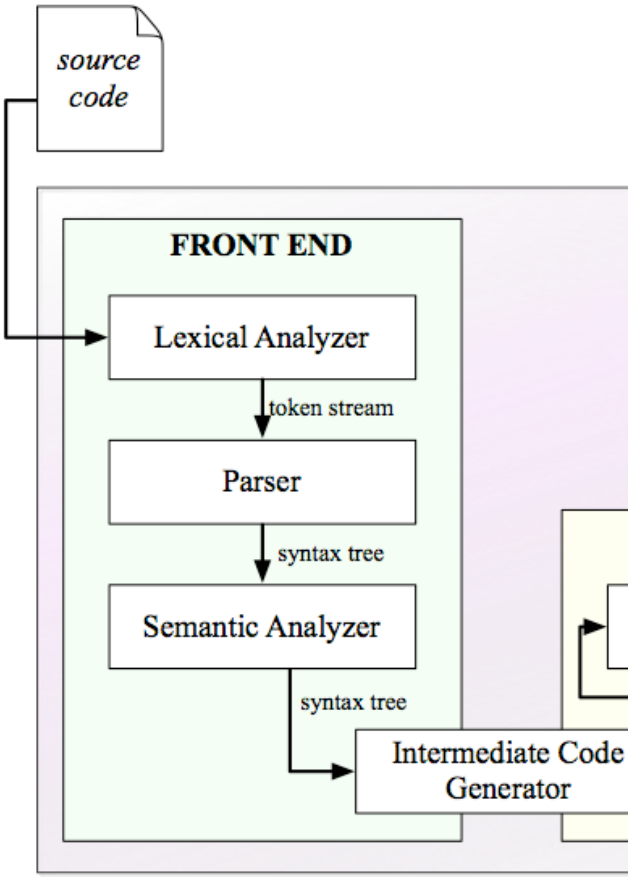


Imagen tomada del curso de F. Pfenning https://www.cs.cmu.edu/~fp/courses/15411-f14/

Fases del front-end

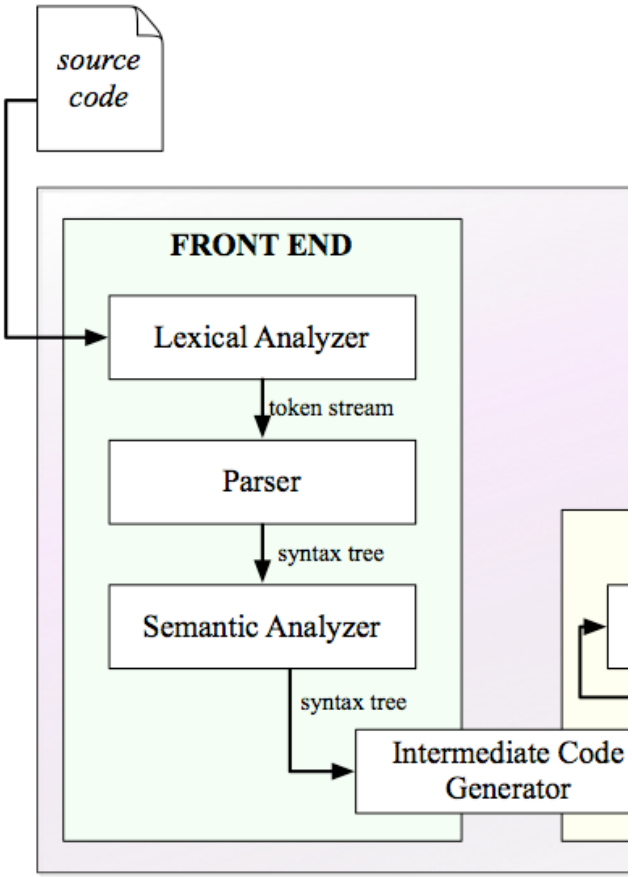
parte sintáctica



- **Análisis Léxico**
  Primer fase de revisión del código fuente para detectar posibles errores de escritura.

Fases del front-end

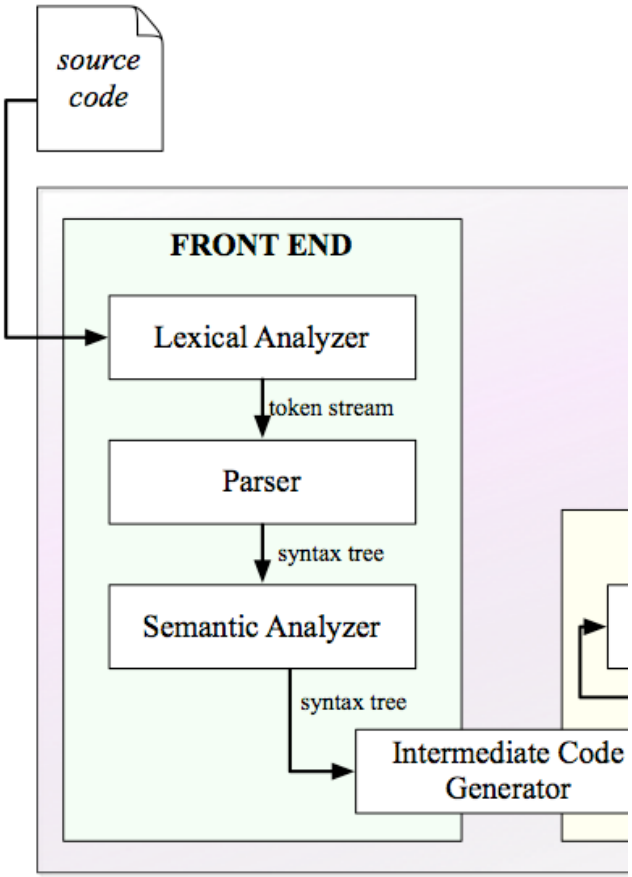
parte sintáctica



- **Análisis Léxico**
  Primer fase de revisión del código fuente para detectar posibles errores de escritura.
  Identificar las partes sintácticas mediante la clasificación de **tokens** generando una secuencia de símbolos significativos.
  (Limpieza de caracteres innecesarios)

Fases del front-end

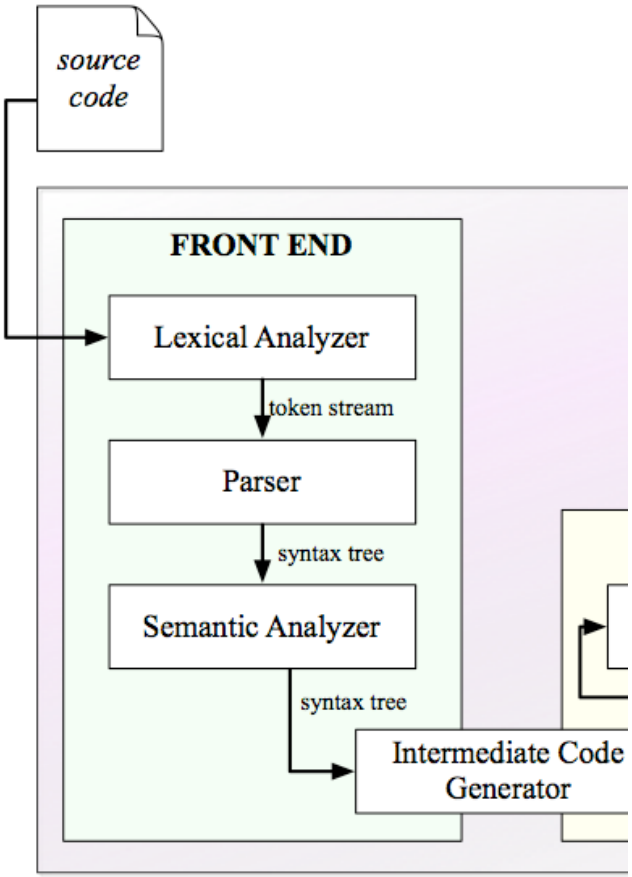
parte sintáctica



- **Análisis Léxico**
  Primer fase de revisión del código fuente para detectar posibles errores de escritura.
  Identificar las partes sintácticas mediante la clasificación de **tokens** generando una secuencia de símbolos significativos.
  (Limpieza de caracteres innecesarios)
- **Análisis Sintáctico**
  El código es analizado bajo la estructura del lenguaje de alto nivel para obtener una representación de alto nivel mediante un *parse tree*.

Fases del front-end

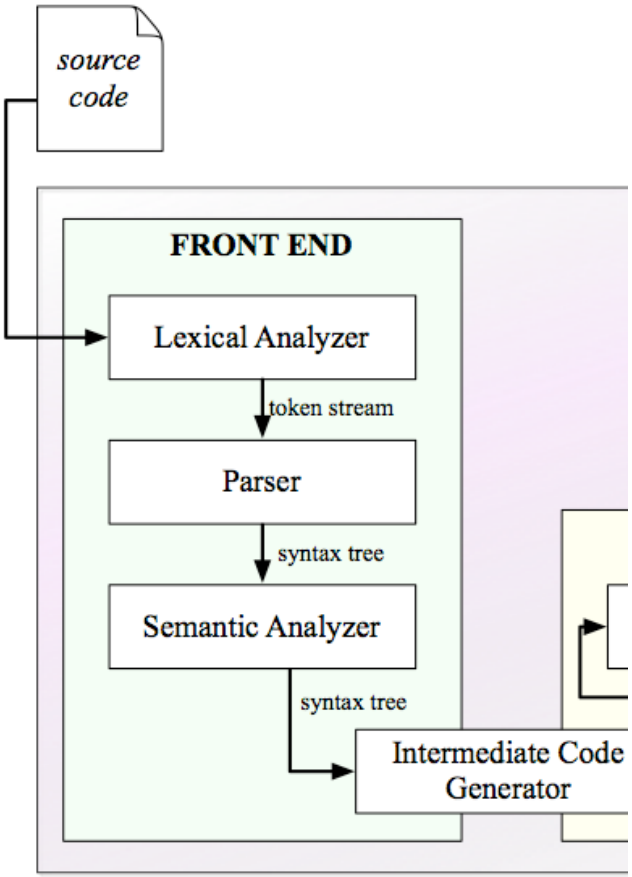
parte semántica



- **Análisis Semántico**
  Análisis dependientes del contexto como la verificación de tipos y esta información se incluye en el arb́ol de la fase anterior.

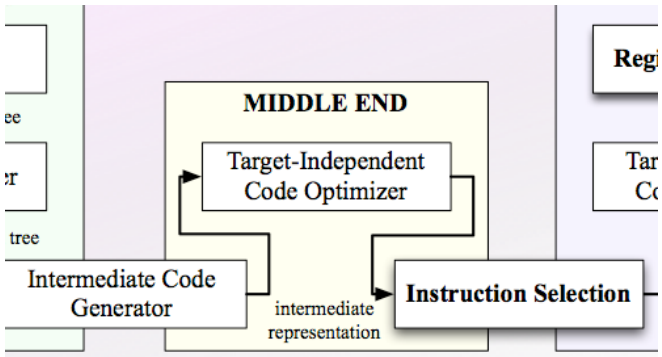
Fases del front-end

parte semántica



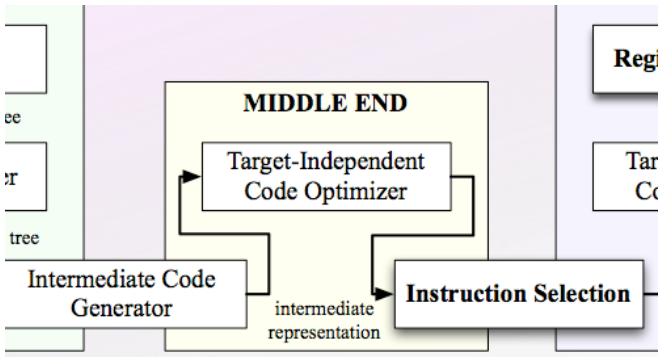
- **Análisis Semántico**
  Análisis dependientes del contexto como la verificación de tipos y esta información se incluye en el arbol de la fase anterior.
- **Generación de código intermedio**
  En esta etapa se considera la tabla de símbolos para clasificar los identificadores, su tipo, estructura y alcance como complemento al árbol que representa el código fuente.

Middle-end



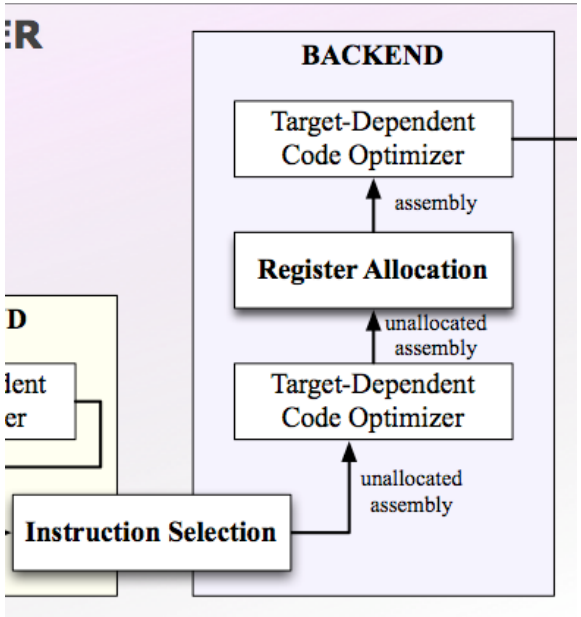
- **Forma intermedia modificada**
  A la representación obtenida anteriormente se le hacen anotaciones (atributos en nodos) para mejorar el código independiente de la máquina.

Middle-end



- **Forma intermedia modificada**
  A la representación obtenida anteriormente se le hacen anotaciones (atributos en nodos) para mejorar el código independiente de la máquina.
  Permite que se realicen optimizaciones intermedias.
  - árbol de sintaxis abstracta
  - gráficas de control de flujo
  - código de 3 direcciones

Back-end



- **Selección de instrucciones**
  Esta fase se encarga de traducir una código intermedio en un código usando un lenguaje muy cercano a ensamblador.
- **Análisis de control del flujo**
  Obtención de la gráfica de control de flujo (idendificación de estructuras de control y bloques para ordenar la ejecución del código).
  Se analiza el tiempo de vida de las variables para optimizar el uso de registros.
- **Asignación de registros**
  Esta fase es la más cercana a la generación de código final o código ensamblador.
  Manejo de la memoria en un nivel bajo lo cual puede o no depender de la máquina en donde se ejecutará el programa.

Introducción

clasificación de lenguajes de programación

Los lenguajes de programación se clasifican de diferentes formas y de acuerdo a diferentes factores:
por la expresividad, por su aparición histórica, por el poder de abstracción y propósito, moda y uso, características semánticas, etc.
Estas condiciones también los subclasifican en otras muchas ramas.

https://www.infoq.com/articles/programming-language-trends-2019/

Introducción

clasificación de lenguajes de programación

Los lenguajes de programación se clasifican de diferentes formas y de acuerdo a diferentes factores:
por la expresividad, por su aparición histórica, por el poder de abstracción y propósito, moda y uso, características semánticas, etc.
Estas condiciones también los subclasifican en otras muchas ramas.

https://www.infoq.com/articles/programming-language-trends-2019/

Alto nivel

- propósito específico & multi-estilo (quinta generación)
- propósito general y manejo especializado (cuarta generación)
- imperativos, declarativos, etc. independientes de máquina (tercera generación)

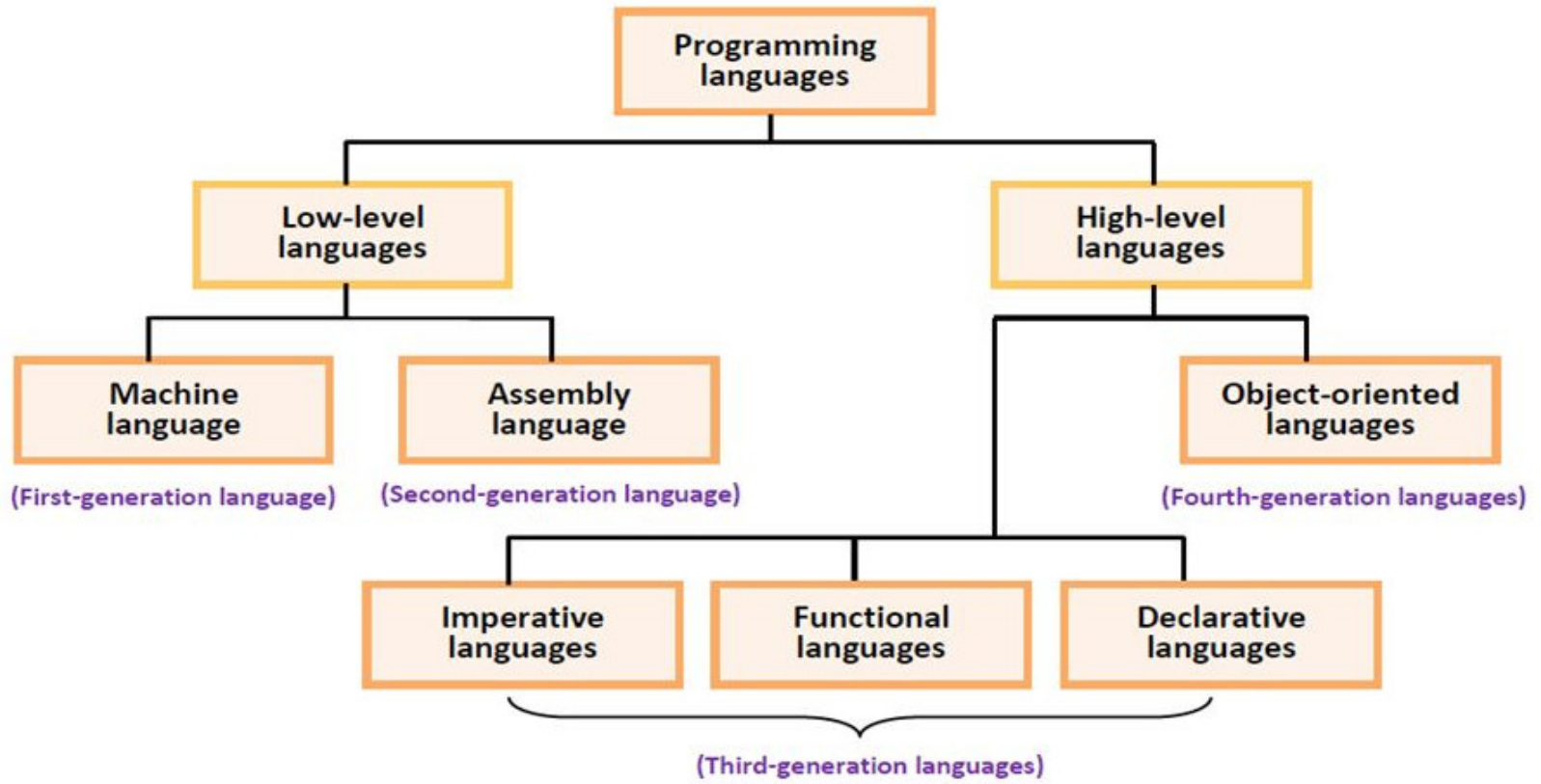
Bajo nivel

- ensamblador (segunda generación)
- lenguaje máquina (primera generación)

https://en.ppt-online.org/234073

Introducción

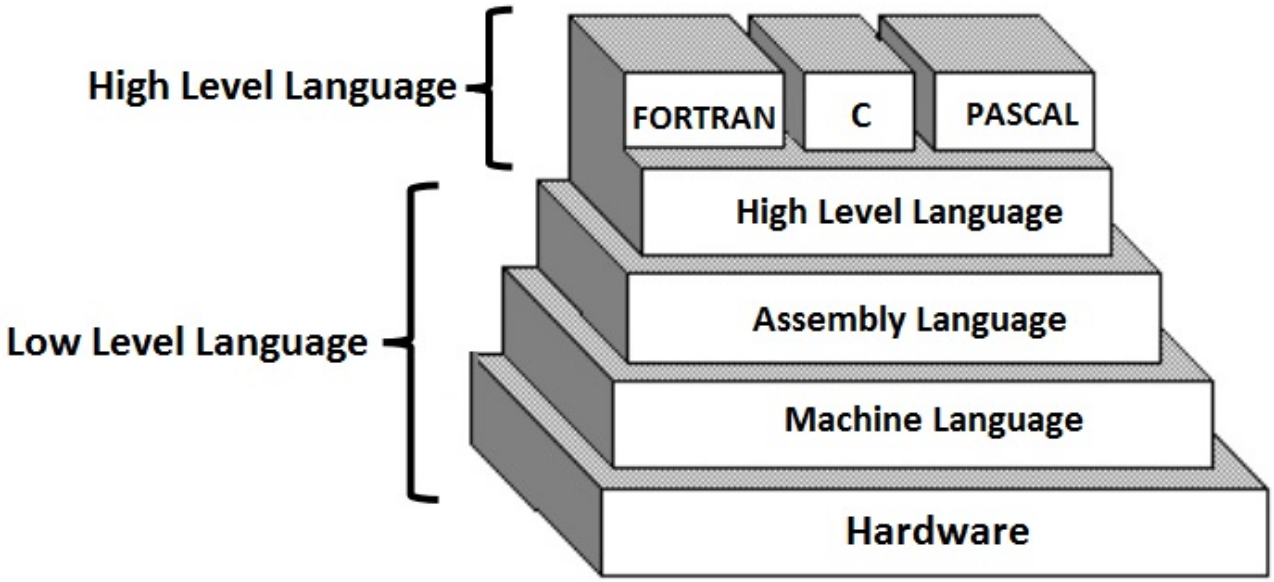
clasificación de lenguajes



Computer Science Course 2301, presentación disponible en SlidePlayer.

Introducción

clasificación de lenguajes

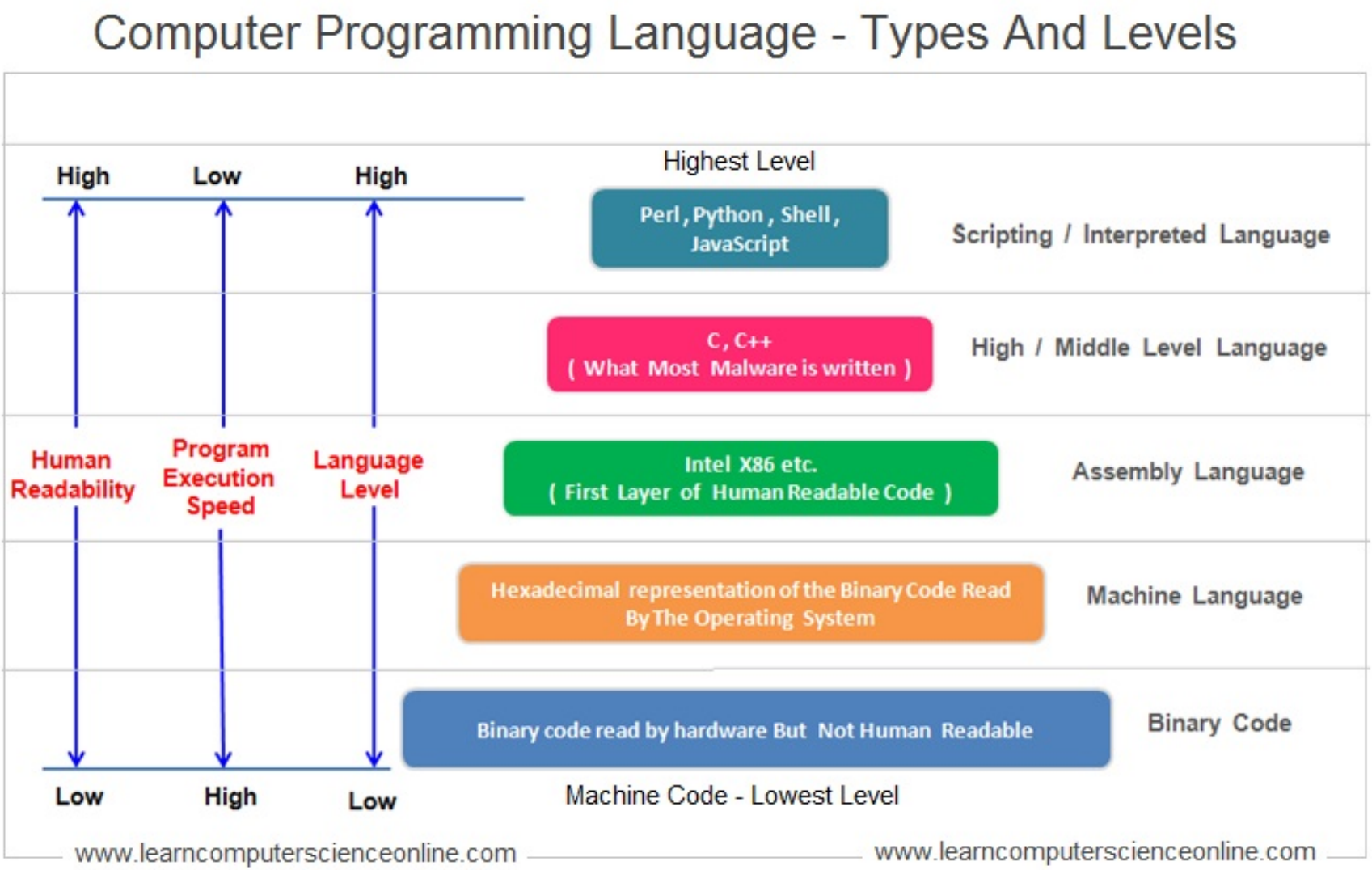


Computer Language and its Types

https://informationq.com/computer-language-and-its-types/

Introducción

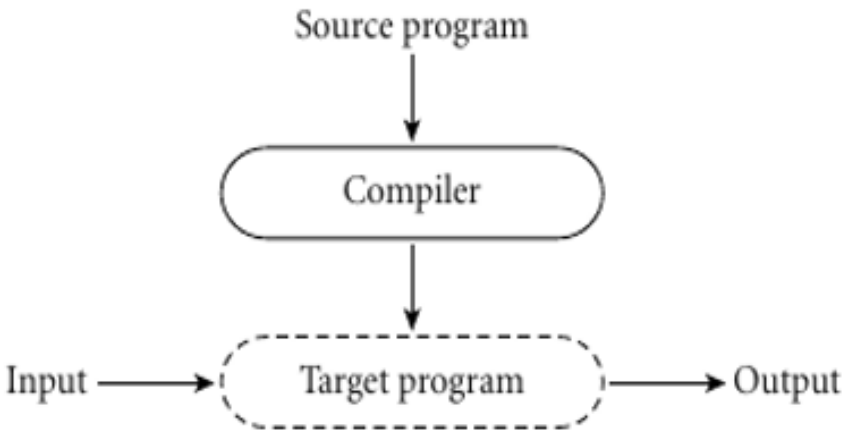
clasificación de lenguajes



https://www.learncomputerscienceonline.com/computer-programming/

Ejecución de un programa

compilación

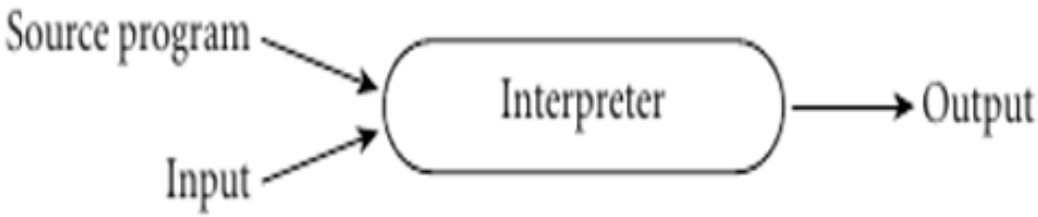


El compilador traduce el programa fuente en uno **equivalente** de bajo nivel y se pasa al sistema operativo para ejecutarse mediante un archivo que contiene el código objeto, así este código puede ejecutarse muchas veces con diferentes entradas sin necesidad de procesarse nuevamente.

Ejecución de un programa

interpretación

Otra forma de implementación de lenguajes es la interpretación:



El proceso de interpretación es más simple pero debe de realizarse cada vez que se ejecuta el programa con diferentes entradas.

Ejecución de un programa

Los compiladores e intérpretes no trabajan solos, se asisten de otras herramientas:

- ensambladores
- *debuggers*
- preprocesadores
- ligadores
- editores e IDEs

para facilitar el desarrollo y ejecución de programas.

Compilador vs Intérprete

- ★ La compilación es un proceso complejo, como se puede observar por las fases de transformación de código, pero se realiza una vez (cada vez que se compila) agilizando su uso múltiples veces para diferentes datos de entrada.
- ★ El intérprete no genera código objeto ni traducciones intermedias.
- ★ El código compilado es generalemente más eficiente.

Compilador vs Intérprete

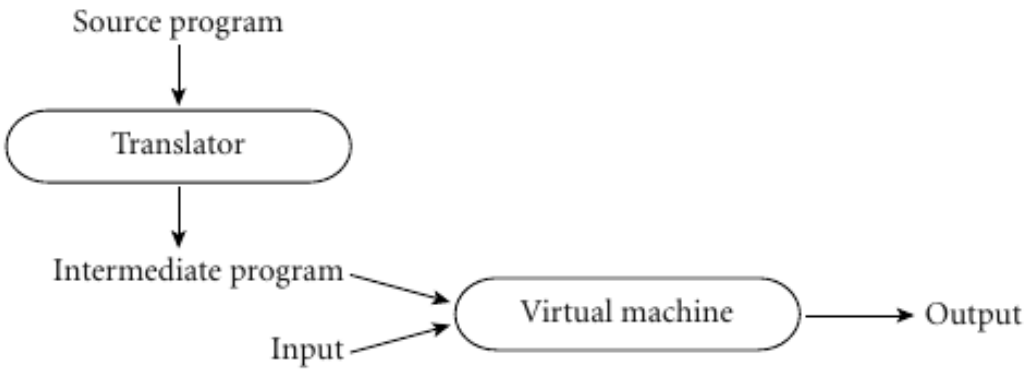
- ★ La compilación es un proceso complejo, como se puede observar por las fases de transformación de código, pero se realiza una vez (cada vez que se compila) agilizando su uso múltiples veces para diferentes datos de entrada.
- ★ El intérprete no genera código objeto ni traducciones intermedias.
- ★ El código compilado es generalemente más eficiente.

  Una clasificación de lenguajes está dada por la forma en que se abstrae, se procesa y se ejecuta un programa:

  - lenguajes interpretados eg. Basic, COBOL, Ruby, Python, etc.
  - lenguajes compilados a un lenguaje intermedio y luego interpretados eg. Java, OCaml, Scala, etc.
  - lenguajes compilados a algún otro lenguaje de alto nivel
  - lenguajes compilados al vuelo
  - lenguajes compilados eg. C, Pascal, etc.

Intérprete y Compilador

Muchos lenguajes son implementados usando una combinación de las ventajas que ofrecen los intérpretes y los compiladores al usar una máquina virtual:

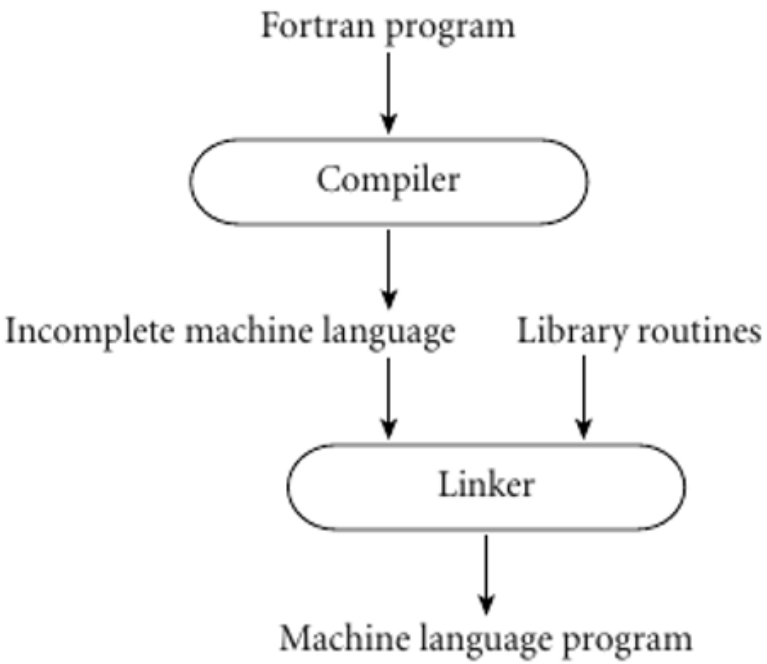


- si el traductor o preprocesador es sencillo se puede decir que es interpretado sino, será compilado
- la máquina virtual es compleja y puede indicar que el lenguaje es interpretado
- las fases de análisis y primeras transformaciones son las que caracterizan la compilación

Ligador

Al finalizar la compilación se obtiene un código objeto que será usado para la ejecución.
El uso de métodos o definiciones de las **bibliotecas** del lenguaje (*libraries*) es incorporado por el ligador o *linker* antes de ejecutar el código.

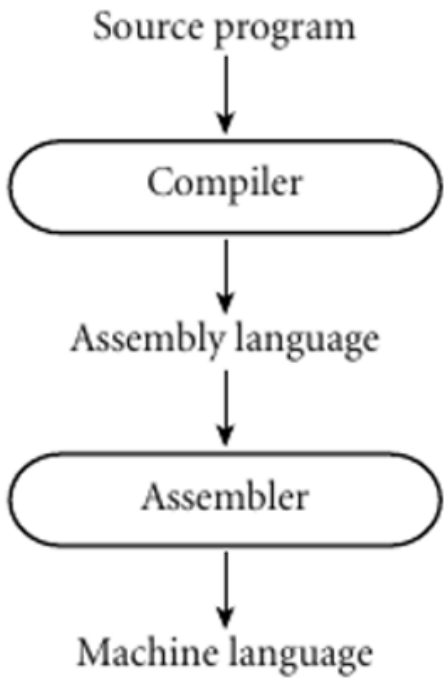
Por ejemplo, la implementación de Fortran:



Ensamblador

Un lenguaje ensamblador es cercano a las instrucciones de una máquina en particular y cuyas expresiones son identificadores de operaciones y los operandos que serán usados.

Un ensamblador finalmente traduce al language máquina, la última traducción usando macros para abreviar secuencias de instrucciones.



Referencias

Imágenes tomadas del libro [5].

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman.
Compilers, Principles, Techniques and Tools.
Pearson Education Inc., Second edition, 2007.

[2] H. R. Nielson and F. Nielson.
Semantics with Applications: An Appetizer (Undergraduate Topics in Computer Science).
Springer-Verlag, Berlin, Heidelberg, 2007.

[3] F. Pfenning.
Notas del curso (15-411) Compiler Design.
https://www.cs.cmu.edu/~fp/courses/15411-f14/, 2014.

[4] B. C. Pierce.
Types and Programming Languages.
The MIT Press, 1st edition, 2002.

[5] M. L. Scott.
Programming Language Pragmatics.
Morgan-Kauffman Publishers, Third edition, 2009.

[6] Y. Su and S. Y. Yan.
Principles of Compilers, A New Approach to Compilers Including the Algebraic Method.
Springer-Verlag, Berlin Heidelberg, 2011.

[7] L. Torczon and K. Cooper.
Engineering A Compiler.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2011.

[8] S. Zdancewic.
Notas del curso (CIS 341) - Compilers, Universidad de Pennsylvania, Estados Unidos.
https://www.cis.upenn.edu/~cis341/current/, 2018.