Tema 1: Introducción a los lenguajes de programación

Sesión 2: Lenguajes de Programación

Referencias

- Tema 1 de apuntes
- Introducción capítulo 1 SICP (Building Abstractions with Procedures)
- Capítulo 1.2 PLP (The Programming Language Spectrum)
- Capítulo 1.3 PLP (Why Study Programming Languages)
- Capítulo 1.4 PLP (Compilation and Interpretation)

Indice

- Historia de los lenguajes de programación
- Elementos de los lenguajes de programación
- Abstracción
- Paradigmas de programación
- Compiladores e intérpretes
- ¿Por qué estudiar lenguajes de programación?

Historia de los lenguajes de programación

- Al comienzo sólo existía el código máquina (años 40). No existían los lenguajes de programación.

- Código máquina: secuencia de bits que controlan directamente un procesador. Muy tedioso. 55 89 e5 53 83 ec 04 f0 e8 31...

- Ensamblador: Abreviaturas mnemotécnicas para expresar operaciones. El traductor abreviatura-instrucción código máquina es el ensamblador. subl, pushl, movl, ...

- Necesidad de un lenguaje independiente de la máquina (años 50).

Historia de los lenguajes de programación

- A finales de los años 50 surgieron los primeros lenguajes de programación
- FORTRAN fue el primer lenguaje de programación. Desarrollado por un equipo de IBM dirigido por John Backus en 1956.
- Cita de John Backus:

> *Much of my work has come from being lazy. I didn't like writing programs, and so, when I was working on the IBM 701, writing programs for computing missile trajectories, I started work on a programming system to make it easier to write programs.*

Historia de los lenguajes de programación

- Desde 1954 hasta la actualidad se han documentado más de 2500 lenguajes de programación
- Árbol genealógico de lenguajes de programación
- Torre de Babel de Éric Lévenez
- Lenguajes más influyentes

Aspectos que provocan la evolución de los LP

- Recursos y tipos de ordenadores
- Aplicaciones y necesidades de los usuarios
- Nuevos métodos de programación
- Estudios teóricos
- Estandarización

¿Por qué estudiar lenguajes de programación?

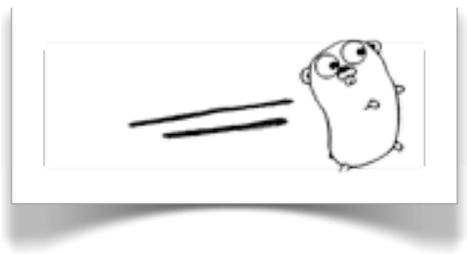
- Mejora el uso del lenguaje de programación
- Incrementa el vocabulario de los elementos de programación
- Permite una mejor elección del lenguaje de programación
- Mejora la habilidad para desarrollar programas efectivos y eficientes
- Facilita el aprendizaje de un nuevo lenguaje de programación
- Facilita el diseño de nuevos lenguajes de programación

Los LP en continua evolución

- Ruby:
  - Ruby, un lenguaje de programación ideado en 1993 por un joven japonés llamado Yukihiro Matsumoto
  - Lenguaje multi-paradigma interpretado y muy expresivo que actualmente se utiliza tanto para desarrollar aplicaciones web como videojuegos.
  - Proyecto vivo, cada año aparecen nuevas versiones

Los LP en continua evolución

- Scala:
  - Scala, diseñado en 2003 por el profesor alemán Martin Odersky
  - Respuesta a los problemas de los lenguajes tradicionales imperativos para manejar la concurrencia
  - Está implementado sobre Java y corre en la Máquina Virtual Java
- Go, el nuevo lenguaje de programación de Google



  - Una mezcla de C y Python que intenta conseguir un lenguaje de programación de sistemas muy eficiente, expresivo y también

Definición de lenguaje de programación

- Definición de la Encyclopedia of Computer Science

A programming language is a set of characters, rules for combining them, and rules specifying their effects when executed by a computer, which have the following four characteristics:
- It requires no knowledge of machine code on the part of the user
- It has machine independence
- Is translated into machine language
- Employs a notation that is closer to that of the specific problem being solved than is machine code

- Definición de Abelson y Sussman

A powerful programming language is more than just a means for instructing a computer to perform tasks. The language also serves as a framework within which we organize our ideas about processes. Thus, when we describe a language, we should pay particular attention to the means that the language provides for combining simple ideas to form more complex ideas.

Características de los lenguajes de programación

- Define un proceso que se ejecuta en un computador
- Es de alto nivel, cercano a los problemas que se quieren resolver (abstracción)
- Permite construir nuevas abstracciones que se adapten al dominio que se programa

Elementos de los lenguajes de programación

- Para Abelson y Sussman, todos los lenguajes de progamación permiten combinar ideas simples en ideas más complejas mediante los siguientes tres mecanismos

  - **Expresiones primitivas** que representan las entidades más simples del lenguaje

  - **Mecanismos de combinación** con los que se construyen elementos compuestos a partir de elementos más simples

  - **Mecanismos de abstracción** con los que dar nombre a los elementos compuestos y manipularlos como unidades

Los lenguajes son para las personas

Abstraccion

- Una misión fundamental de los lenguajes de programación es proporcionar herramientas que sirvan para construir abstracciones
- Abstracciones: sirven para tratar la complejidad del mundo real
- Existen abstracciones propias de la computación: listas, árboles, grafos, tablas hash...

Paradigmas de programación

- Un paradigma define un conjunto de reglas, patrones y estilos de programación que son usados por un grupo de lenguajes de programación
    - Paradigma funcional
    - Paradigma lógico
    - Paradigma imperativo o procedural
    - Paradigma orientado a objetos

Paradigma funcional

- La computación se realiza mediante la evaluación de expresiones
- Definición de funciones
- Funciones como datos primitivos
- Valores sin efectos laterales, no existe la asignación
- Programación declarativa
- Lenguajes: LISP, Scheme, Haskell, Scala,

```
(define (factorial x)
   (if (= x 0)
      1
      (* x (factorial (- x 1)))))

(factorial 8)
40320
(factorial 30)
265252859812191058636308480000000
```

Paradigma lógico

- Definición de reglas
- Unificación como elemento de computación
- Programación declarativa
- Lenguajes: Prolog, Mercury, Oz.

```
padrede('juan', 'maria'). % juan es padre de maria
padrede('pablo', 'juan'). % pablo es padre de juan
padrede('pablo', 'marcela').
padrede('carlos', 'debora').

hijode(A,B) :- padrede(B,A).
abuelode(A,B) :-  padrede(A,C), padrede(C,B).
hermanode(A,B) :- padrede(C,A) , padrede(C,B), A \== B.

familiarde(A,B) :- padrede(A,B).
familiarde(A,B) :- hijode(A,B).
familiarde(A,B) :- hermanode(A,B).

?- hermanode('juan', 'marcela').
yes
?- hermanode('carlos', 'juan').
no
?- abuelode('pablo', 'maria').
yes
?- abuelode('maria', 'pablo').
no
```

Paradigma imperativo

- Definición de procedimientos
- Definición de tipos de datos
- Chequeo de tipos en tiempo de compilación
- Cambio de estado de variables
- Pasos de ejecución de un proceso

```
type
   tDimension = 1..100;
   eMatriz(f,c: tDimension) = array [1..f,1..c] of real;

   tRango = record
      f,c: tDimension value 1;
   end;

   tpMatriz = ^eMatriz;


procedure EscribirMatriz(var m: tpMatriz);
var filas,col : integer;
begin
   for filas := 1 to m^.f do begin
      for col := 1 to m^.c do
         write(m^[filas,col]:7:2);
      writeln(resultado);
      writeln(resultado)
     end;
end;
```

Paradigma orientado a objetos

- Definición de clases y herencia
- Objetos como abstracción de datos y procedimientos
- Polimorfismo y chequeo de tipos en tiempo de ejecución
- Ejemplo en Java

```
public class Bicicleta {
    public int marcha;
    public int velocidad;

    public Bicicleta(int velocidadInicial, int marchaInicial) {
        marcha = marchaInicial;
        velocidad = velocidadInicial;
    }

    public void setMarcha(int nuevoValor) {
        marcha = nuevoValor;
    }

    public void frenar(int decremento) {
        velocidad -= decremento;
    }

    public void acelerar(int incremento) {
        velocidad += incremento;
    }
}

public class MountainBike extends Bicicleta {
    public int alturaSillin;

    public MountainBike(int alturaInicial, int velocidadInicial, int
marchaInicial) {
        super(velocidadInicial, marchaInicial);
        alturaSillin = alturaInicial;
    }

    public void setAltura(int nuevoValor) {
        alturaSillin = nuevoValor;
    }
}

public class Excursion {

        public static void main(String[] args) {
           MountainBike miBicicleta = new MoutainBike(10,10,3);
           miBicicleta.acelerar(10);
           miBicicleta.setMarcha(4);
           miBicicleta.frenar(10);
        }
}
```

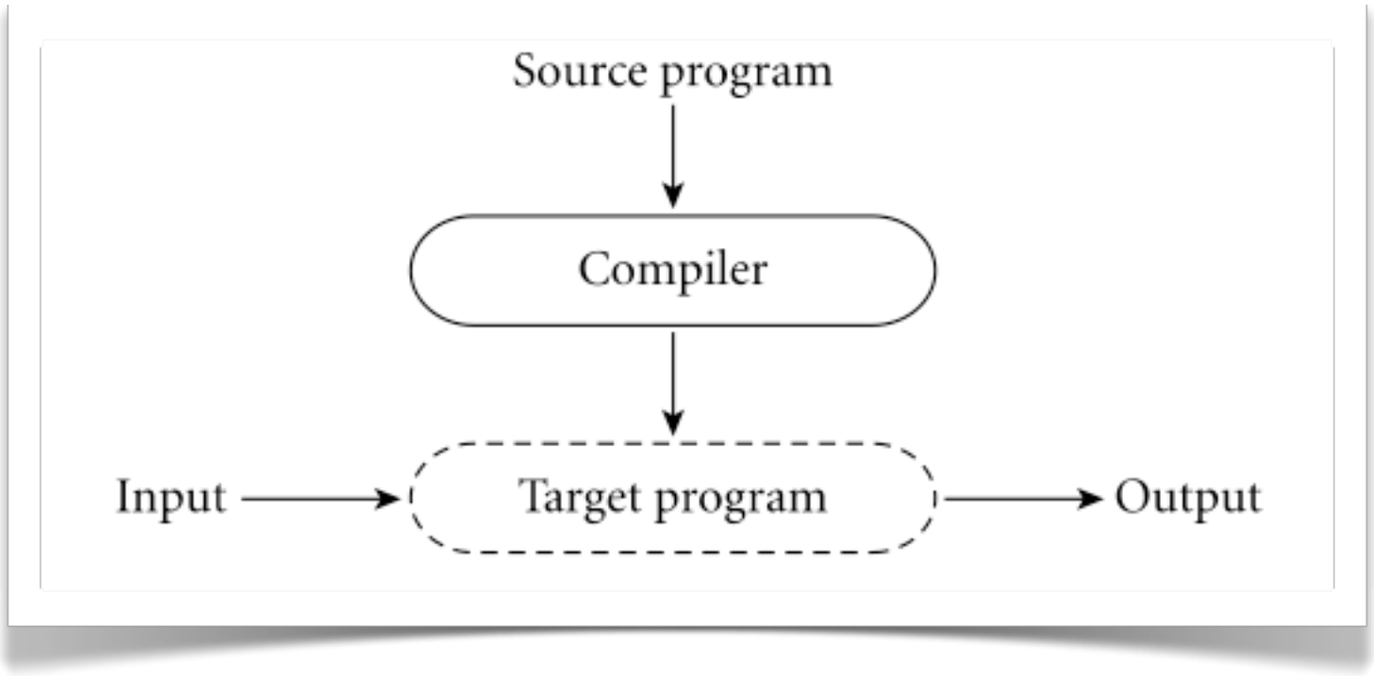
Compiladores e intérpretes

- Existen una gran variedad de estrategias para conseguir que un programa se ejecute en un ordenador

- Todas se basan en los "meta-programas" (compiladores, intérpretes, etc.) cuyos datos de entrada son el código fuente de otros programas.

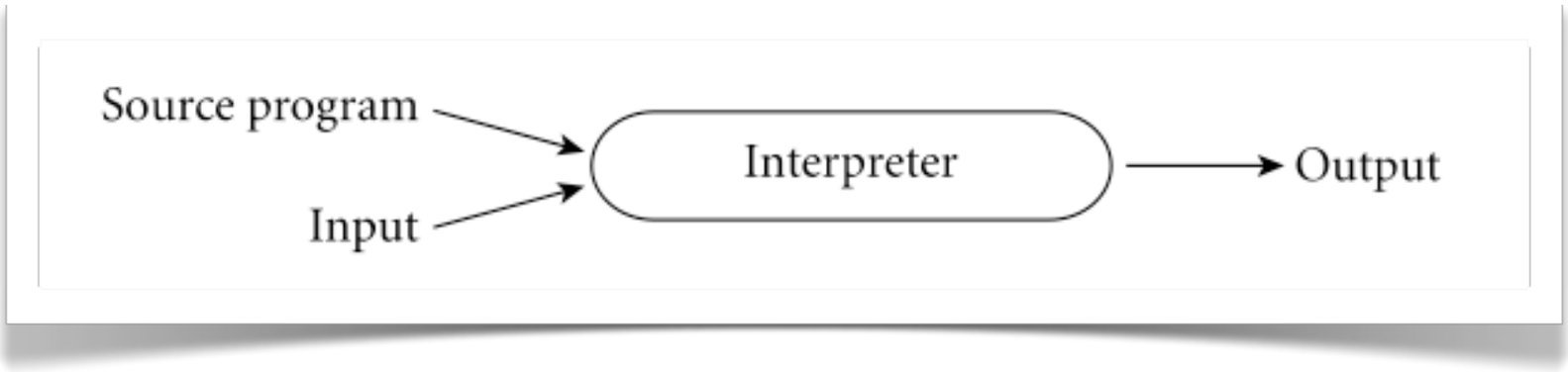
Compilación

- Ejemplos: C, C++
- Diferentes momentos en la vida de un programa: tiempo de compilación y tiempo de ejecución
- Mayor eficiencia



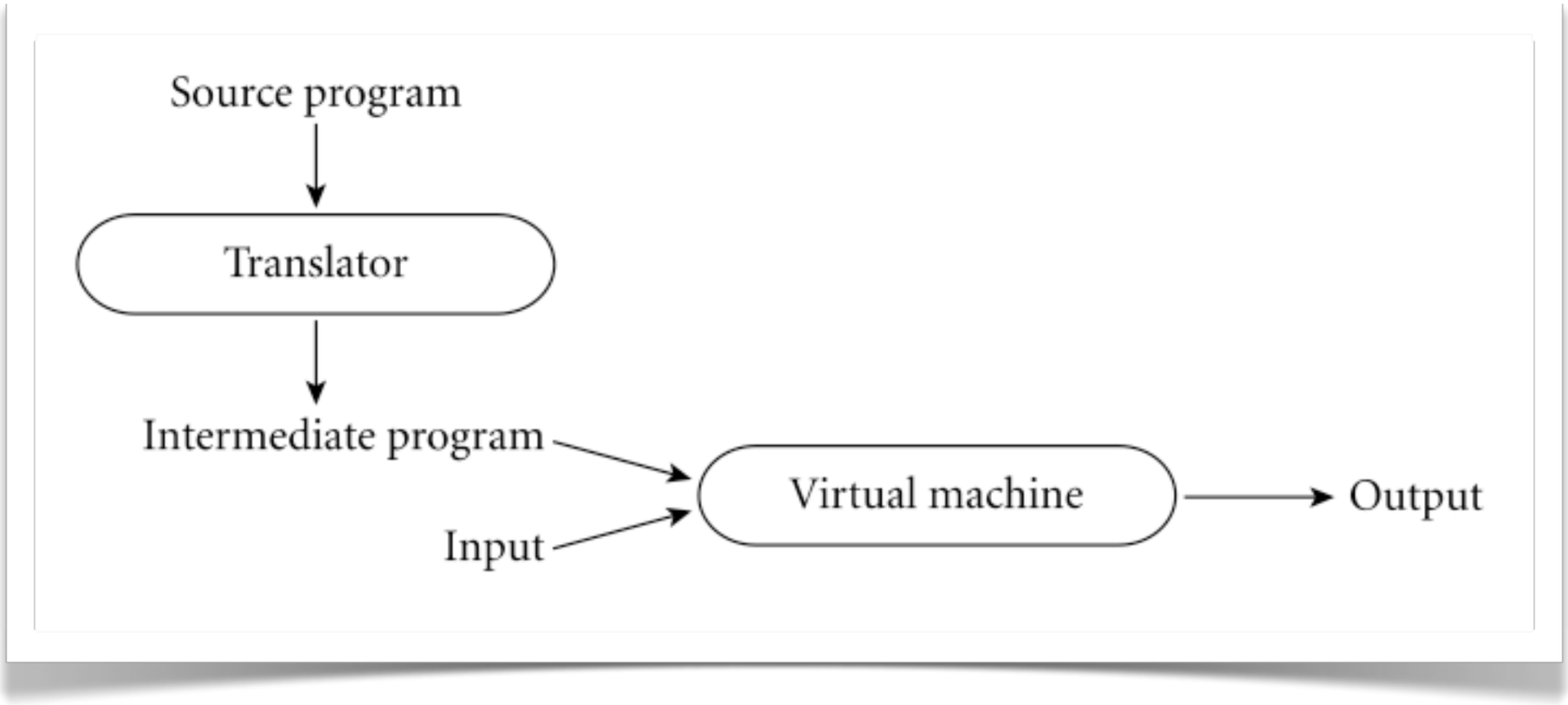
Interpretación

- Ejemplos: BASIC, LISP, Scheme, Python, Ruby
- No hay diferencia entre el tiempo de compilación y el tiempo de ejecución
- Mayor flexibilidad: el código se puede construir y ejecutar "on the fly" (funciones lambda o clousures)

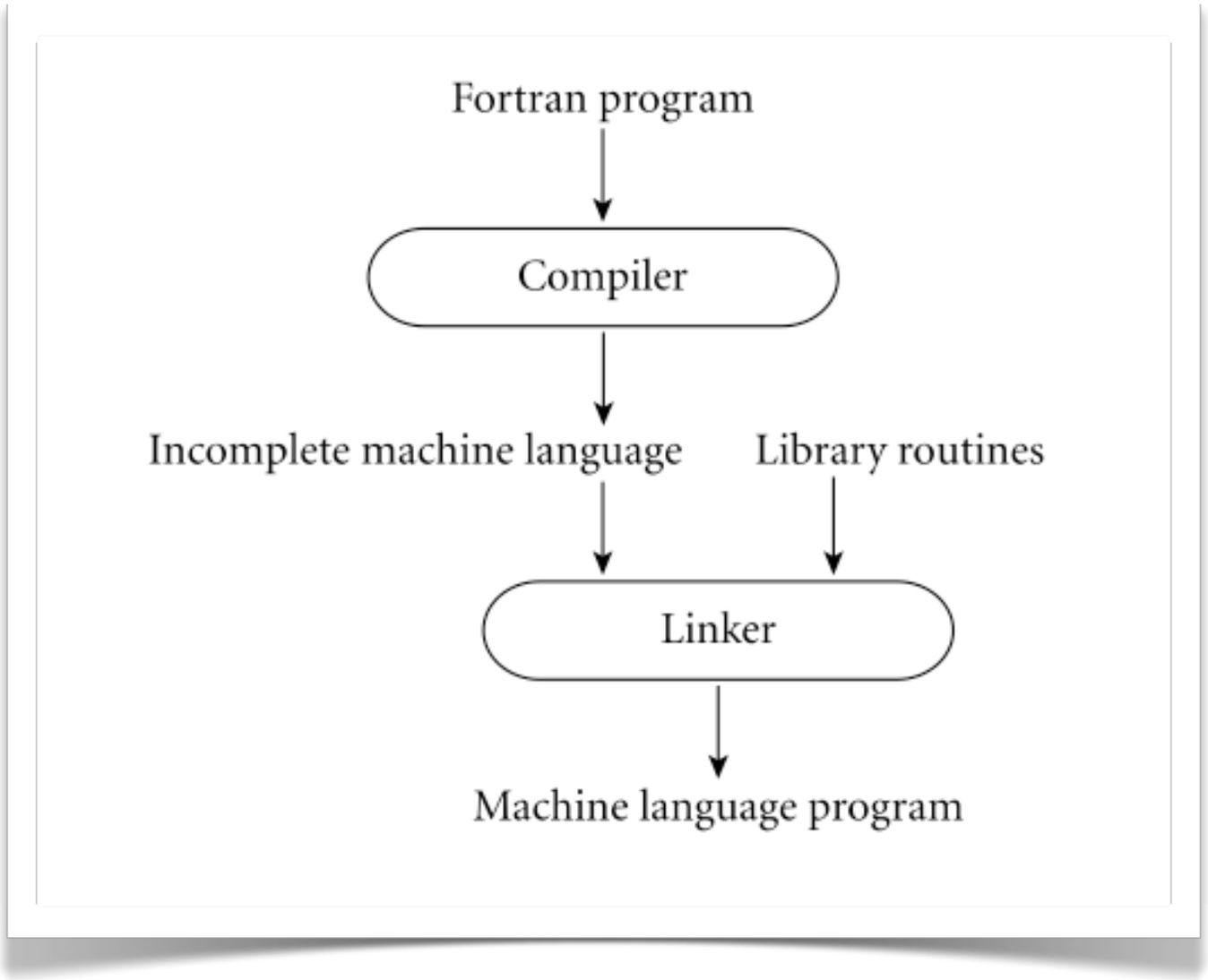


Ejecución en máquina virtual

- Ejemplos: Java, Scala



Enlazado de rutinas y librerías



Preprocesamiento

- El preprocesador analiza el código y sustituye macros. Ejemplo: C, C++. Scala hace algo parecido con Java.