Lenguajes de Programación I

Introducción

Ernesto Hernández-Novich
<emhn@usb.ve>

Universidad "Simón Bolívar"

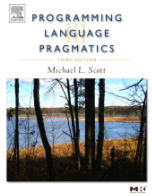
La Materia

Enero-Marzo 2016

- Ernesto Hernández-Novich <emhn@usb.ve> – `@iamemhn` / `#ci3641`
- Lun y Mié AUL-214 9:30-11:30 VET –
  aprecio su puntualidad y el silencio de sus dispositivos.
- Consultas por e-mail (pongan `[CI3641]` en Asunto) –
  protip: el enlace en esta lámina es "mágico".
- Consultas en persona, antes de clase, MYS-220C –
  después de clase previa cita (no vengo todos los días).
- Tres (3) exámenes escritos
  - Primer Parcial (25 %) el 2016-01-27 (Miércoles Semana III)
  - Segundo Parcial (35 %) el 2016-02-22 (Lunes Semana VIII)
  - Tercer Parcial (40 %) TBD (Lunes Semana XII)

http://www.ldc.usb.ve/~emhn/cursos/ci3641/201509

Bibliografía



Michael L. Scott.
Programming Language Pragmatics
Morgan Kaufmann Publishers, Inc.
San Francisco, California
ISBN: 978-0-12-374514-9

- Uso la tercera edición ("el blanco") – la segunda ("el negro") sirve.
- Estaré revisando las láminas durante el trimestre.
- Cada clase trata de mostrar ejemplos de los conceptos –
  si leyeron el libro *antes*, mejor para ustedes.

Preparación para los exámenes

It's deja vu, all over again – Yogi Berra

- El libro es **denso** e **intenso**.
- Aprender de memoria los conceptos es inútil.
- Tendrán que escribir programas concretos –
  siempre terminan siendo programas cortísimos.
  - Haskell, C, C++, Scheme y Prolog suelen aparecer –
    es irrelevante que Ud. no esté inscrito en CI-3661
  - En la mayoría de las clases hay ejemplos de código –
    protip: hacerlos funcionar y modificarlos.
- ¡Estudie las soluciones a los exámenes de cursos previos!

¿Por qué es importante esta materia?

- ¿Cuál es tu rol ante los lenguajes?
  - Si eres programador, eres un mero *usuario* de lenguajes.
  - Si eres un *implantador* la materia es Traductores – eventualmente la cadena de Lenguajes...
  - Esta materia apoya al *diseñador* de lenguajes.
- "¡Pero yo no soy diseñador de lenguajes!" – en ese caso, comprender el contenido:
  - Permite *aprender nuevos lenguajes* con facilidad.
  - Permite *comprender y explotar las cosas oscuras* de los lenguajes.
  - Ayuda a *escoger la mejor alternativa* para cada problema.
  - Contribuye a mejorar el uso (y abuso) de *entornos de programación*.
  - Induce a *imitar* un lenguaje dentro de otro.

Historia (1940s)

- En el principio sólo había ceros y unos (y a veces, sólo ceros).
- Programar era el mismo arte que ahora, pero conectando cables.
- Luego se programaba en lenguaje de máquina directamente, pasando muy rápidamente al lenguaje ensamblable (como mucho con macros).
- La programación estaba *centrada en la máquina*
  - Muy bajo nivel.
  - Intimo conocimiento del hardware.
  - La portabilidad no era importante.
  - La comprensión del trabajo de otros era difícil.

Historia (1950s)

- Se desarrolla el primer lenguaje de alto nivel (FORTRAN).
- Le siguen lenguajes como LISP y Algol.
  - LISP marcó *profundamente* la evolución de los lenguajes tanto por el nicho en qué nació, como por el paradigma funcional y la "auto-referencia".
  - Algol marcó *profundamente* la evoluación de los lenguajes por servir de base al estilo imperativo Von Neumann.
- Traducción del lenguaje de alto nivel a lenguaje de máquina usando los primeros compiladores.
  - Construidos a mano *enteramente* – nada de herramientas automáticas para análisis léxico-sintáctico.
  - Monolíticos – estrechamente ligados a la plataforma.

El presente...

- La cantidad de lenguajes es abrumadora.
  - ¿Cuántos lenguajes conoces?
  - ¿Cuántos lenguajes *dominas*?
- ¿Por qué tal diversidad?
  - Evolución (encontramos mejores formas de hacer las cosas).
  - Propósitos especiales (dominio del problema).
  - Preferencias personales (¡Perl rules! ¡Haskell conquers!).

¿Por qué un lenguaje se hace popular?

- Poder expresivo – expresar cosas complejas con sencillez.
- Facilidad de uso para el novato – "se parece a...".
- Facilidad de implantación, en particular Software Libre.
- Calidad de las herramientas (compiladores, IDEs).
  - *Debuggers* – porque usar banderas es *mainstream*.
  - *Profilers* – ¿cuál parte es la más lenta? ¿cuál más usada?.
  - *Test harness* – ¿cómo puedo probar lo probable y lo improbable?
- Calidad del ecosistema (herramientas y comunidad) – ¿cuántas librerías están disponibles? ¿dónde? ¿son buenas?
- Inercia y dinero – "a nadie han botado por usar Java..."

¿Cómo expresar el problema?

- Imperativos.
  - Enfasis en *cómo* hacer las cosas.
  - Von-Neumann (Fortran, Pascal, C, Basic).
  - Orientados a Objetos (C++, Smalltalk, Eiffel, Java).
- Declarativos.
  - Enfasis en *qué* hacer.
  - Funcionales (LISP/Scheme, Haskell, ML, Erlang).
  - Flujo de Datos (Id, Val, hojas de cálculo).
  - Lógicos (Prolog, Datalog, SQL).
- Mixtos.
  - Combinan características de ambos grupos.
  - Perl, Python, Ruby.

Lenguajes dentro de lenguajes

Cuando un lenguaje engendra y hospeda otro

- Domain Specific Languages (DSL)
  - Haskore/Euterpea (expresión musical dentro de Haskell).
  - Perligata (escribir programas Perl en latín).
  - Unix Shell Scripting.
  - R (probabilidades y estadística).
  - LaTeX (creación de documentos — están viendo una "corrida")
- Lenguajes embebibles.
  - Lua (usado en WoW y Half Life 2).
  - EMACS Lisp.
  - Vim + (Perl, Python, Ruby).
  - AutoLISP (LISP dentro de AutoCAD).
  - Perl y R dentro de PostgreSQL.

¿Cómo pasa de código fuente a resultados?

- Compilados.
- Interpretados.
- Máquinas virtuales (el punto medio).

Solía ser una clasificación "dura",
pero los tiempos cambian...

Compilación pura

- Código fuente procesado por un compilador, que genera un programa objeto ejecutable en lenguaje de máquina.
- Programa objeto ejecutado de manera directa y autónoma sobre el procesador, procesando sus entradas y produciendo sus salidas.
- Tienden a producir programas con mayor velocidad de ejecución, a un costo de procesamiento previo.
- C/C++, Fortran, Haskell.

Interpretación pura

- Código fuente procesado por un interpretador, que controla la evolución del programa paso a paso. El interpretador ejecuta directamente sobre el procesador anfitrión.
- Cada línea o instrucción del programa es leída por el interpretador, analizada y ejecutada, posiblemente utilizando entradas y produciendo salidas.
- Tienden a producir programas con menor velocidad de ejecución, pero con mucha más flexibilidad en el ciclo de desarrollo y diagnósticos.
- Shell Unix, LISP, Prolog y Smalltalk.
- Haskell (wait, what...?)

Máquinas virtuales

Lo mejor de ambos mundos... massomeno...

- Código fuente procesado por un traductor, produce una representación intermedia del programa de un nivel más bajo (*bytecode*), como si se tratara de una máquina ideal con operaciones abstractas.
- Código intermedio es interpretado por una máquina virtual que implanta esas operaciones abstractas, y que utiliza las entradas o produce salidas.
- Beneficios en ambos extremos (desarrollo vs. ejecución)
  - Si el lenguaje intermedio es expresivo.
  - Si el traductor aplica majoras astutas al código.
  - Si la máquina virtual está bien implantada.
- Perl, Ruby, Python y Java son ejemplos.
- Parrot, a VM to rule them all...

Preprocesadores

- Aplicable a compilación e interpretación.
- Expande macros – mera sustitución de cadenas.
- Posible enriquecer código fuente incluyendo código fuente adicional.
- Posible incluir o eliminar código fuente condicionalmente – la maldición del soporte a múltiples plataformas y configuraciones.
- Generalmente separado (cpp, el preprocesador de C) – algunos intrínsecos (*source filter* Perl, Template Haskell).

Librerías y cargadores

Link-loader* y *dynaloaders

- Compilación separada o por partes.
- Librería – colección de rutinas previamente desarrolladas y posiblemente optimizadas.
- Código fuente hace referencia a las variables y rutinas provistas.
  - Compilación pura – enlazador resuelve esas dependencias estáticamente durante la construcción del programa objeto.
  - Interpretción pura – interpretador agrega las librerías durante la ejecución y resuelve las dependencias dinámicamente.
- Con la infraestructura adecuada, el programa puede *diferir* el enlazado hasta el último momento
  - Para enlazar sólo el código efectivamente empleado.
  - Para seleccionar versiones diferentes de la librería.
- ¿Y si el "programa" es una gran librería y lo reemplazo a tiempo de ejecución? *Non-stop updates* como en Erlang.

Compilación a ensamblable

. . . o a otro engendro "útil"

- Compilador genera lenguaje ensamblable del procesador anfitrión en lugar de generar código objeto directamente.
  - *Debugging* – "¿qué demonios hace el compilador?"
  - Flojera – si ya tengo un ensamblador, no reinvento la rueda.
  - LLVM – virtualsambladoptimizado... ¡ponicornios!
- **Traducción de fuente a fuente** – ¿Por qué no generar otro lenguaje de alto nivel?
  - SQL embebido en C – cambiar el SQL por llamadas a librerías.
  - Compiladores para LISP o Prolog.
  - Así funcionaban las primeras versiones de C++.
  - Haskell genera Core, que se optimiza para C o código de máquina.

Entonces, ¿compilado o interpretado?

- Existen máquinas virtuales para lenguajes "tradicionalmente" interpretados – WAM para Prolog.
- Algunas máquinas virtuales compilan partes del programa en *ejecución* "justo a tiempo" (**Just In Time** compilation) para aprovechar mejor el procesador.
- Haskell hace las tres cosas.
- Algunos interpretadores ponen el interpretador al alcance del programador – "evaluar" una cadena (como en LISP, Prolog o Perl).
- Algunas máquinas virtuales permiten "descompilar" el código ejecutable hasta el fuente – Perl.

Quiero tener el lenguaje Foo

Bootstrapping

- Cuando los hombres eran hombres, escribían compiladores a mano. . .
  - Escribir un compilador para Foo en ensamblable – Muy difícil.
  - Escribir un interpretador para Foo en ensamblable – Doloroso.
  - Teniendo el interpretador para Foo corriendo en la máquina, escribo un compilador para Foo en Foo.
  - ¡Corro el compilador para Foo en el interpretador para Foo, y le doy como entrada el compilador para Foo!
- Técnica similar hizo popular a Pascal. Niklaus Wirth distribuía:
  - Compilador para Pascal escrito en Pascal que generaba P-código.
  - Compilador para Pascal traducido a P-código.
  - Interpretador de P-código escrito en Pascal.
  - Basta traducir este interpretador a algún lenguaje local.

Inception before it was cool. . .

Bibliografía

- [*Scott*]
  - Secciones 1.1 a 1.5.
  - Leer la sección 1.6 – de eso se trata Traductores y la cadena.
  - Ejercicios y Exploraciones