Lenguajes de Programación, 2019-2
Nota de clase 1
Historia de los lenguajes de programación

Karla Ramírez Pulido Manuel Soto Romero Javier Enríquez Mendoza

7 de enero de 2019
Facultad de Ciencias, UNAM

Los lenguajes de programación en la actualidad proveen diversos mecanismos que facilitan al programador la realización de múltiples tareas con un nivel de abstracción lo más alto posible, estos lenguajes de programación son llamados de alto nivel, sin embargo, esto no siempre fue así. Anteriormente la programación de los dispositivos de cómputo era una tarea tediosa que hacia que la labor de programar fuera compleja y que pocas personas dominaban.

En esta nota se expone la historia de los lenguajes de programación desde los fundamentos teóricos, pasando por la programación de dispositivos de cómputo antiguos hasta llegar a los lenguajes de alto nivel que son usados hoy en día por cientos de programadores.

1.1 Los fundamentos

Antes de que existieran los lenguajes de programación se definieron algunos fundamentos que dieron origen no sólo a los lenguajes de programación, sino los conceptos teóricos que dieron origen a lo que conocermos hoy en día como computadora así como teorías que permiten decidir qué es computable o no , todos estos fundamentos han sido de utilidad al crear a los lenguajes de programación como se les conoce actualmente. A continuación se listan algunos de estos fundamentos [1]:

Lenguajes formales

Al igual que las matemáticas, los lenguajes de programación son lenguajes formales, lo cual permite expresar ideas (programas) de forma concreta, breve y precisa. Al ser lenguajes formales, es posible construir modelos de los mismos, lo cual permite razonar sobre lenguajes de programación, manipularlos, demostrar propiedades de los mismos, y obtener nuevos resultados a partir de otros ya conocidos. Lo cual permite a su vez encontrar respuestas a problemas sobre dichos modelos y generar extensiones del conocimiento [8].

Ada Lovelace y la Máquina Analítica

En 1834 Charles Babbage inició el diseño de un dispositivo de cómputo al que llamó *Máquina Analítica*, esta máquina sería capaz de calcular el valor numérico de cualquier fórmula o función, dada una entrada, en donde el usuario podía especificar el método de solución, la forma en que trabajaría la máquina sería completamente automática [5].

Ada Lovelace, conocida como la primera programadora fue principal colaboradora del trabajo de Babbage. Ada escribió un programa para calcular los números de Bernoulli, este programa fue descrito en un diagrama que exponía paso a paso cómo debía la máquina realizar el cálculo, esto es, un programa descrito en un lenguaje entendible por la máquina para ser ejecutado.

El Cálculo Lambda

El Cálculo Lambda[1], es una de las teorías más importantes de los lenguajes de programación, fue introducido por Alonzo Church en los años 1930 con el fin de formalizar el concepto de computabilidad. Se dice que es el lenguaje de programación universal más pequeño del mundo y consiste de un lenguaje formal para describir términos y definir funciones, usando una única regla de transformación que permite sustituir variables llamada β-reducción [3].

Máquinas de Turing

Las máquinas de Turing son un modelo abstracto de cómputo propuesto por Alan Turing en 1936, que al igual que el Cálculo Lambda, proveen una definición formal sobre lo que es o no computable. Es una máquina de estados finita compuesta por una cinta infinita que incluye símbolos de un alfabeto finito (denotado por Σ), cuenta con una cabeza lectora que puede leer o escribir en dicha cinta. A partir del símbolo que se está leyendo y el estado actual, una Máquina de Turing escribe un nuevo símbolo en la posición actual y se mueve hacia la izquierda o derecha de acuerdo a lo especificado por una *función de transición*. La función de transición es el "programa" que especifica cómo se comporta la máquina, algo similar a los lenguajes de programación actuales [4].

1.2 Lenguajes ensambladores

En su inicios, programar una computadora electrónica, como la ENIAC (*Electronic Numerical Integrator And Computer)* consistía en conectar y desconectar cables para configurar el hardware, lo cual era una tarea tediosa y que pocos podían realizar. Con la llamada *Arquitectura de von Neumann* se resuelve esto al tener un conjunto de instrucciones almacenadas en la memoria de la computadora y ejecutando las mismas en la Unidad Central de Proceso (CPU, en inglés)[6]. Sin embargo, estas instrucciones debían escribirse en un lenguaje que entendiera la computadora, como es el sistema binario. De esta forma, una instrucción en lenguaje máquina luce de la siguiente manera [7]:

0110 1001 1010 1011

Cada parte de estas instrucciones indican a la computadora qué acción realizar, por ejemplo, los primeros cuatro bits en la instrucción anterior (0110 de izquierda a derecha), podrían indicar que se trata de una suma, los siguientes 8 bits podrían representar los valores a sumar y los últimos cuatro bits (1011) indican el registro de memoria en que se almacenará el resultado de la suma. Por supuesto, esto varía entre las diferentes arquitecturas que existen de computadoras.

Este tipo de instrucciones conforman un lenguaje llamado *lenguaje de máquina*, sin embargo, este lenguaje no resulta ser un lenguaje tan claro y fácil de leer por lo humanos. Debido a esto, se crearon los *lenguajes ensambladores* que representaban un mapeo entre etiquetas llamadas mnemotécnicos y el lenguaje de máquina. Por ejemplo, una operación de suma, representada por el código 0110, se convierte en la instrucción `ADD` (suma). La siguiente línea de código en lenguaje ensamblador representa la instrucción en lenguaje máquina 0110 1001 1010 1011 (Sumar la variable `M` con la `N` y almacenar el resultado en el registro `P`) [7]:

[1] En inglés λ-calculus.

```
ADD M, N, P
```

A pesar de que este tipo de instrucciones forman lenguajes de programación con una sintaxis y semántica bien definida, tienen la desventaja, (al igual que el lenguaje de máquina) de que dependen de la arquitectura en particular. Hoy en día, este tipo de lenguajes tienen sus aplicaciones muy reducidas y se centran principalmente en control de procesos y de dispositivos electrónicos [7].

1.3 Lenguajes de alto nivel

Los lenguajes de programación de alto nivel, son los más utilizados en la actualidad, pues son diseñados con el fin de ser más entendibles para los humanos y porque los programas escritos en este tipo de lenguajes no dependen de ningún dispositivo en particular; esto es, que no dependen de la arquitectura de la máquina y como consecuencia pueden ser ejecutados en cualquier computadora a diferencia de los lenguajes máquina o ensambladores.

La Tabla 1.1 muestra algunos de los lenguajes de programación más utilizados hoy en día junto con información acerca de sus creadores y sus principales características.

Nombre	***Diseñador(es)***	***Año de creación***	***Paradigma(s)***	***Usos***
FORTRAN	John Backus	1954	Estructurado Orientado a Objetos	Aplicaciones científicas y de ingeniería; programas que evalúan el desempeño y posicionamiento de supercomputadoras.
COBOL	Comisión CODASYL	1950	Estructurado Orientado a Objetos	Aplicaciones comerciales que requieren de una manipulación precisa y eficiencia de grandes volúmenes de datos.
LISP	John McCarthy	1959-1960	Funcional	Representación de expresiones simbólicas y manejo de listas.
ALGOL	John Backus	1960	Estructurado	Utilizado principalmente en aplicaciones científicas.
SIMULA	Ole Johan Dahl / Kristen Nygaard	1965	Orientado a Objetos	Simulación de eventos simultáneos.
Pascal	Niklaus Wirth	1969	Estructurado	Se diseñó para la enseñanza de la programación estructura y fue popular en cursos universitarios durante varias décadas.
C	Dennis Ritchie	1972	Estructurado	Aplicaciones de escritorio, aplicaciones científicas, en particular simulaciones.

Nombre	***Diseñador(es)***	***Año de creación***	***Paradigma(s)***	***Usos***
Prolog	Alain Comerauer, Philippe Roussel	1972	Lógico	Inteligencia artificial, sistemas expertos, reconocimiento del lenguaje natural.
ML	Universidad de Edimburgo	1973	Funcional Estructurado	Principalmente académicos.
Scheme	Guy L. Steele	1975	Funcional Estructurado	Manejo de listas, definición de macros, aplicaciones de escritorio, verificación de programas.
Smalltalk	Alan Kay	1980	Orientado a Objetos	Lenguaje de propósito general, programas de escritorio y web principalmente.
C++	Bjarne Stroustrup	1983	Estructurado Orientado a Objetos	Programas de escritorio, programación web, graficación por computadora y animación.
Haskell	Universidad de Glasgow	1990	Funcional	Principalmente en la academia, sin embargo, se tienen algunas aplicaciones en la industria aeroespacial, de finanzas y algunas empresas de diseño de hardware.
Python	Guido van Rossum	1991	Orientado a Objetos Funcional	Acceso a bases de datos, servicios web, aplicaciones de escritorio y web, gráficos 3D, juegos.
PL/SQL	Oracle	1993	Estructurado	Manipulación de tablas y consultas dentro de una o más bases de datos.
Racket (PLT Scheme)	Grupo PLT	1994	Funcional Estructurado Orientado a objetos	Diseñado para la creación de nuevos lenguajes de programación y enseñanza de la programación.
Java	Sun Microsystems	1995	Orientado a Objetos	Aplicaciones empresariales, aplicaciones web, aplicaciones de escritorio, programación de dispositivos móviles.
Ruby	Yukihiro Matsumoto	1995	Orientado a Objetos	Principalmente en la programación web a través del *framework* Ruby on Rails.

Nombre	***Diseñador(es)***	***Año de creación***	***Paradigma(s)***	***Usos***
OCaml	Xavier Leroy	1996	Funcional Orientado a oObjetos	Aplicaciones de escritorio, web, dispositivos móviles, implementación del asistente de pruebas Coq.

Tabla: 1.1: *Lenguajes de alto nivel*

Referencias

[1] Samuel A. Rebelsky, *Programming Languages,* Notas de clase, Grinell College revisión 99S. Consultado el 20 de noviembre de 2018 [`http://www.math.grin.edu/~rebelsky/Courses/CS302/99S/Outlines/outline.02.html`]

[2] Favio E. Miranda, Elisa Viso, *Matemáticas Discretas*, Las Prensas de Ciencias, Segunda Edición, 2016.

[3] Raúl Rojas, *A Tutorial Introduction to the Lambda Calculus*, ArXiv, 2015.

[4] Bobby Kleinberg, Notas de clase, Cornell Universitu, revisión 2012sp. Consultado el 7 de enero de 2019. [`http://www.cs.cornell.edu/courses/cs4820/2012sp/handouts/turingm.pdf`]

[5] History of Computers, *The Analitical Engige of Charles Babbage,* Consultado el 7 de enero de 2019. [`https://history-computer.com/Babbage/AnalyticalEngine.html`]

[6] José Galaviz, *Organización y Arquitectura de Computadoras,* Notas de clase, Facultad de Ciencias UNAM, revisión 2015-2.

[7] Imelda Avalos, Introducción a la programación, Universidad Autónoma de Nayarit, Consultado el 7 de enero de 2019. [`http://correo.uan.edu.mx/~iavalos/introprog.htm#Lenguajes`]

[8] EcuRed, *Enciclopedia Cubana*. Consultado el 7 de enero de 2019. [`https://www.ecured.cu/`]