INDUSTRIAL DATA 2001, (4) 1 : 71-74

# LENGUAJES DE PROGRAMACIÓN: CONCEPTOS Y PARADIGMAS

Edgar Ruíz L.*


## RESUMEN

El artículo presenta los conceptos que rigen a los lenguajes de programación y los paradigmas de la programación y cómo éstos influyen en el desarrollo del software.


**Palabras Clave**: Lenguaje ensamblador, paradigma, lenguaje orientado a objeto.


## ABSTRACT

The article presents the concepts that govern around the programming languages and the paradigms of the programming and the influence in the developme of the software.


**Key words :** Language assembler, paradigm, language guided to object.

## INTRODUCCIÓN

Los primeros lenguajes de programación de alto nivel se diseñaron durante los años 1950. Desde entonces los lenguajes de programación han sido una fascinante y prolífica área de estudio para los científicos de la computación y los ingenieros.

El estudio de los lenguajes de programación, es llamado a *veces lingüística de la programación*, por analogía con la lingüística de los lenguajes naturales. La analogía se basa en el hecho en que ambos; lenguajes naturales y lenguajes de programación, poseen *sintáxis* (forma) y *semántica* (significado). La analogía no puede tomarse en todo el contexto. Los lenguajes de programación no pueden ser comparados con los lenguajes naturales en términos de su rango de expresividad y subjetividad. Por otro lado, un lenguaje natural no es más ni menos que un grupo de personas que hablan y escriben, así que la lingüística natural está restringida al análisis de los lenguajes existentes; mientras que los lenguajes de programación son concienzudamente diseñados y se pueden implementar en computadoras.

## CONCEPTOS BÁSICOS

Cada lenguaje de programación es una creación y como tal ha sido cuidadosamente diseñado. Algunos lenguajes han sido diseñados por personas únicas, como por ejemplo Pascal. Otros, han sido diseñados por un grupo grande de personas, tales como PL/I y Ada. La experiencia sugiere que aquellos lenguajes diseñados por personas únicas o grupos pequeños, tienden a ser más compactos y coherentes que aquellos lenguajes diseñados por grandes grupos.

Un lenguaje de programación, digno de su nombre, debe reunir ciertos requisitos.

El lenguaje de programación debe ser *universal*. Es decir, cualquier problema debe tener una solución que puede ser programada en el lenguaje y dicha solución ser implementada en cualquier computador. Este requisito es uno de los más fuertes y pocos lenguajes lo poseen. Se dice que cualquier lenguaje en el cual pueden definirse funciones recursivas se considera universal. De otro lado, un lenguaje sin recursión ni iteración no puede ser universal. Existen ciertos lenguajes de aplicación que no son universales, pero si podrían ser razonablemente descritos así mismos, como *lenguajes de programación*.

El lenguaje de programación debe ser implementable en una computadora, es decir; debe ser posible ejecutar un programa en términos del lenguaje en cualquier máquina. La notación matemática generalmente no es implementable porque en su notación es posible formular problemas que no pueden ser resueltos por cualquier computador. Los lenguajes naturales tampoco son implementables por razones totalmente diferentes: ellos son tan imprecisos y tienden a ser muy ambiguos.

* Ingeniero Industrial. Instituto de Investigación. Facultad de Ingeniería Industrial. UNMSM.
E-mail :d260045@unmsm.edu.pe

## SINTAXIS Y SEMÁNTICA

Cada lenguaje tiene *syntáxis* y *semántica*:

- La *syntáxis* de un lenguaje de programación está relacionada con la *forma* de los programas, por ejemplo, como es que las expresiones, comandos, declaraciones, etc. son puestos juntos en un programa.
- La *semántica* de un lenguaje de programación está relacionada con el *significado* de los programas; por ejemplo, cómo ellos se comportarán cuando se ejecutan en una computadora.

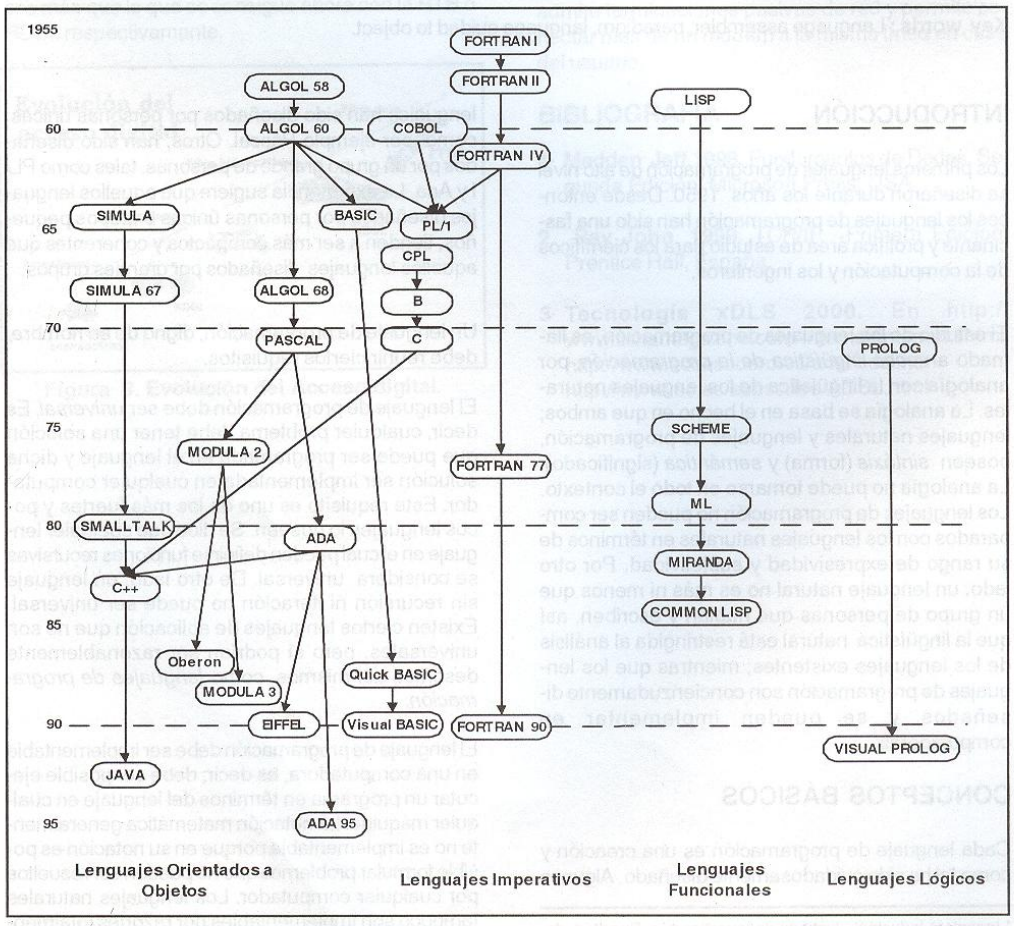
La *syntáxis* de un lenguaje influye en cómo los programas son *escritos* por el programador, *leídos* por otro programador y *traducidos* por el computador. La *semántica* de un lenguaje determina como los programas son *compuestos* por el programador, *entendidos* por otros programadores e *interpretados* por el computador. La *syntáxis* es importante; pero la semántica es más importante aún.

## ENFOQUE HISTÓRICO

Los lenguajes de programación de hoy son el producto de un desarrollo que se inició en los 1950´s. Numerosos conceptos de lenguajes han sido inventados, examinados e implementados en sucesivos lenguajes. Con muy pocas excepciones, el diseño de cada lenguaje ha sido fuertemente influenciado por la experiencia con los lenguajes iniciales. Los lenguajes de hoy no son el producto *final* del desarrollo del diseño del lenguaje; nuevos conceptos y paradigmas están siendo desarrollados y el escenario de los lenguajes de programación de los próximos diez años podría ser un poco diferente al de hoy.

## PARADIGMAS DE LA PROGRAMACIÓN

Para que una computadora realice una tarea, debe programársela para que lo haga colocando en la memoria principal un algoritmo apropiado el cual es expresado en lenguaje máquina. En los inicios de




**La figura 1. Muestra la genealogía de los principales lenguajes de programación.**

la programación, esta tarea era onerosa por lo laborioso y difícil de diseñar cada algoritmo (sin contar los errores en que se podría incurrir). El gran paso se dió cuando se empezó a dar *mnemónicos* a los diversos códigos de operación y a los operandos del lenguaje de máquina. Con esto, los programadores pudieron aumentar considerablemente la comprensibilidad de las secuencias de instrucciones máquina. Por ejemplo, la siguiente rutina, empleando el método mnemónico:

```
CRG      R2,    TARIFA
CRG      R3,    HORAST
MULTI R0 R2 R3
ALM      R0     PAGO
STOP
```

Se puede interpretar como: cargar en el registro R2 al valor de TARIFA, cargar en el registro R3 el valor de HORAST, en la tercera sentencia, MULTI R0, R2,R3 significa multiplique el contenido de R2 por R3 y póngalo en R0. A este tipo de lenguaje de programación se convino en llamarlo *lenguaje ensamblador* debido a que justamente un programa llamado *ensamblador* se encargaba de traducir estos mnemónicos a una forma más compatible con la máquina. (Se le llamó ensamblador porque su tarea era ensamblar instrucciones en lenguaje máquina a partir de los códigos de operación y operandos obtenidos al traducir nombre mnemónicos e identificadores). Al lenguaje ensamblador se le conoce también como lenguaje de bajo nivel. Una desventaja importante del lenguaje ensamblador es el ser dependiente de la máquina, es decir si, se cambia la máquina, cambia el programa ensamblador.

Al estudio de los lenguajes en cuanto al enfoque del proceso de programación se le denomina *paradigmas de la programación*, entendiéndose el término paradigma como la forma de *ver* y *hacer* los programas. Bajo este enfoque se tienen cuatro paradigmas los cuales son:

- paradigma por procedimientos o paradigma imperativo
- paradigma declarativo
- paradigma funcional
- paradigma orientado a objetos

El paradigma por procedimientos, es tal vez el más conocido y utilizado en el proceso de programación, donde los programas se desarrollan a través de procedimientos. Pascal C y BASIC son tres de los lenguajes imperativos más importantes. La palabra latina *imperare* significa "dar instrucciones". El paradigma se inició al principio del año 1950 cuando los diseñadores reconocieron que las variables y los comandos o instrucciones de asignación constituían una simple pero útil abstracción del acceso a memoria y actualización del conjunto de instrucciones máquina. Debido a la estrecha relación con la arquitectura de la máquina, los lenguajes de programación imperativa pueden ser implementados muy eficientemente, al menos en principio.

El paradigma imperativo aún tiene cierto dominio en la actualidad. Una buena parte del software actual ha sido desarrollado y escrito en lenguajes imperativos. La gran mayoría de programadores profesionales son principalmente o exclusivamente programadores imperativos (Hay que añadir que los paradigmas de la programación concurrente y orientada al objeto son en realidad sub-paradigmas de la programación imperativa, así que sus adeptos también son programadores imperativos).

El paradigma declarativo o paradigma de programación lógica se basa en el hecho que un programa implementa una relación antes que una correspondencia. Debido a que las relaciones son mas generales que las correspondencias (identificador – dirección de memoria), la programación lógica es potencialmente de más alto nivel que la programación funcional o la imperativa. El lenguaje más popular enmarcado dentro de este paradigma es el lenguaje PROLOG. El auge del paradigma declarativo se debe a que el área de la lógica formal de las matemáticas ofrece un sencillo algoritmo de resolución de problemas adecuado para usarse en un sistema de programación declarativo de propósito general.

Si la programación imperativa se caracteriza por el uso de variables, comandos y procedimientos, la programación funcional se caracteriza por el uso de expresiones y funciones. Un programa dentro del paradigma funcional, *es una función* o *un grupo de funciones* compuestas por funciones más simples estableciéndose que una función puede llamar a otra, o el resultado de una función puede ser usado como argumento de otra función. El lenguaje por excelencia ubicado dentro de este paradigma es el LISP. Por ejemplo si se desea obtener la nota promedio de un alumno podría construirse una función promedio la cual se obtendría a partir de otras funciones más simples: una (sumar) la cual obtiene la suma de las entradas de la lista, otra (contar) la cual cuenta el número de entradas de la lista y la tercera (dividir) que obtiene el cociente de los valores anteriores, su sintáxis será:

**(dividir (sumar notas) (contar notas))**

Obsérvese que la estructura anidada refleja el hecho de que la función *dividir* actúa sobre los resultados de *suma* y *contar*.

El paradigma orientado a objetos, se basa en los conceptos de *objetos* y *clases* de *objetos*. Un objeto es una variable equipada con un conjunto de operaciones que le pertenecen o están definidas para ellos. El paradigma orientado a objetos actualmente

es el paradigma más popular y día a día los programadores, estudiantes y profesionales tratan de tomar algún curso que tenga que ver con este paradigma, podría decirse, que programar orientado a objetos está de moda.

Alrededor de 1970 David Parnas planteó el *ocultamiento de la información* como una solución al problema de gerenciar grandes proyectos software. Su idea fue encapsular cada variable global en un módulo con un grupo de operaciones (al igual que los prodecimientos y las funciones) que permitan tener un acceso directo a la variable. Otros módulos pueden acceder a la variable sólo indirectamente, llamando a estas operaciones. Hoy se usa el término *objeto* para tales módulos o variables encapsuladas a sí mismas. Lenguajes imperativos como Pascal Y C han sido modificados (o añadidos) para que soporten el paradigma orientado a objetos para dar Delphi en el caso de Pascal y C++ en el caso de C.

Una de las bondades importantes de los lenguajes orientados a objetos es que las definiciones de los objetos pueden usarse una y otra vez para construir múltiples objetos con las mismas propiedades o modificarse para construir nuevos objetos con propiedades similares pero no exactamente iguales.

El lenguaje orientado a objetos por excelencia es Smaltalk desarrollado en Palo Alto Research Center durante los 1970´s.

Pero que es exactamente un lenguaje orientado a objetos? Los siguientes conceptos señalan las características generalmente aceptadas acerca de los lenguajes orientados a objetos.

- *Objetos* y *clases* son obviamente los conceptos fundamentales. Una clase es un conjunto de objetos que comparten las mismas operaciones.
- *Objetos* (o al menos referencia a objetos) deben ser valores de la clase base. Así, cualquier operación puede tomar un objeto como un argumento y puede devolver un objeto como resultado. De esta manera el concepto de clase de objetos está relacionado con el concepto de tipo de dato.
- *Herencia* es también vista como un concepto clave dentro del mundo de los objetos. En este contexto, la herencia es la habilidad para organizar las clases de objetos en una jerarquía de subclases y superclases y las operaciones dadas para una clase se pueden aplicar a los objetos de la subclase.

## CONCLUSIONES

La comprensión básica de los conceptos de los lenguajes de programación y los diferentes paradigmas son necesarios para todos los ingenieros de software, no tanto para los especialistas en un lenguaje de programación. Esto se debe a que los lenguajes de programación son una herramienta fundamental.

Los lenguajes de programación influencian notablemente la manera en que *pensamos* acerca del diseño y construcción del software y los algoritmos y estructuras de datos que utilicemos para desarrollar software.

## BIBLIOGRAFÍA

1· **BROOKSHEAR, GLEN J. 1985.** Introducción a las Ciencias de la Computación. Marquette University. 4ª Ed., Edit. Addison Wesley Iberoamericana, S.A. U.S.A.

2· **WATT, D. 1989.** Programming Languaje Concepts and Paradigms. Glasgow University, Uk. Edit. Prentice Hall U.K.

3. **SEBESTA, R. R. 1999.** Concepts of Programming Languajes. University of Colorado. 4ª Ed.Edit. Addison Wesley U.S.A.