SUMA 22
junio 1996, pp. 33-42

El software matemático y los lenguajes de programación

Jose F. Quesada Moreno

ARTÍCULOS

Se lleva a cabo un análisis de los lenguajes de programación desde el punto de vista de sus relaciones con el software matemático. Para ello se comienza con una definición bastante flexible de software matemático, para continuar con un análisis metodológico de los lenguajes de programación, estudiando los paradigmas imperativo, funcional, la programación lógica y la orientación a objetos. Por último se realiza un estudio histórico de los lenguajes de programación, así como de los lenguajes de programación más adecuados para la implementación de algoritmos matemáticos.

El software matemático: objeto y método de estudio

El objetivo de este artículo es el estudio del software matemático desde el punto de vista de los lenguajes de programación. Aunque aún estamos inmersos en la corriente de crecimiento exponencial que caracteriza a las tecnologías de la información, lo que podría cuestionar una visión crítica acerca del mismo proceso de desarrollo, creo que la interconexión entre las ciencias de la computación y las matemáticas es lo suficientemente notoria como para ser expuesta sin temor a una grave equivocación.

Es algo más que una simple curiosidad el que los primeros ordenadores estuviesen dirigidos, en lo fundamental, hacia la automatización de determinados cómputos matemáticos. Esta concepción «matematizadora» se ha mantenido durante todos estos años y dan cuenta de su permanencia algunos datos bastante significativos. En supercomputación, el ámbito de máxima potencia computacional, se mide la potencia de una plataforma en FLOPS[1] o en unidades más sofisticadas que integran el denominado ámbito de los *benchmarks,* que en su conjunto, y a pesar de sus diferencias, tienen un fuerte componente de cómputo matemático. Otro dato de interés se puede desprender de la consideración del conjunto de microinstrucciones implementadas a nivel hardware en la mayoría de los procesadores comerciales, donde aparece como un apartado, en ninguna medida secundario, el conjunto de operaciones matemáticas. También se puede mencionar la comercialización de coprocesadores matemáticos, o la enorme importancia que para determinados tipos de problemas están asumiendo los procesadores vectoriales.

La interconexión mencionada no se ha producido únicamente en el plano teórico, sino que se ha visto reflejada

en una ingente cantidad de utilizaciones concretas de los recursos informáticos para la resolución de problemas matemáticos, tanto de las diferentes ramas de la matemática pura como de las múltiples áreas de aprovechamiento de la matemática aplicada.

Así pues, a la especificación inicial según la cual el objetivo propuesto sería el estudio del software matemático, habría que añadir el que este campo se puede caracterizar por la profusión de investigación y desarrollo, por lo que se hace necesario el discutir, a modo de estructuración conceptual, el objeto y el método de estudio en torno a los que girará nuestro análisis.

Para centrar el tema de estudio, asumiendo siempre como objetivos la brevedad y la economía conceptual, creo que la siguiente definición es bastante adecuada, al menos como hipótesis de trabajo: se entenderá por *software matemático* toda implementación computacional de algoritmos matemáticos.

Cabe destacar dos ideas de la definición anterior: en primer lugar, la noción de *algoritmo*, y más específicamente de algoritmo matemático, nos retrotrae hasta la existencia de un mecanismo formalizado y general para la resolución de un tipo dado de problemas. Las especificaciones formales del concepto de algoritmo conforman en sí mismas un campo amplio y apasionante; no obstante, para el problema que nos ocupa será suficiente considerar que un algoritmo es un conjunto de reglas mediante las que se especifica una serie de estructuras de datos y mecanismos u operadores para su manipulación, de tal forma que su ejecución, en un tiempo finito, permite obtener una o más estructuras nuevas de datos, cuyo contenido será la solución para el problema considerado.

La segunda idea a destacar de la definición de software matemático es la noción de *implementación computacional*. Es evidente que sólo se podrá hablar de software matemático cuando el algoritmo matemático pueda ser reescrito de forma tal que sea procesable por un computador, entendiendo un computador como una máquina, de propósito general, diseñada específicamente para el procesamiento de símbolos (lo que recuerda, aunque sólo en su base conceptual, la noción de *máquina de Turing*).

La definición presentada para el software matemático es intencionadamente genérica, es decir, no lo es por casualidad, sino porque se ha considerado que la imposición de otras restricciones, aunque permitiría perfilar con mayor precisión el campo, podría excluir áreas de verdadero interés.

Estas últimas consideraciones nos llevan hasta el problema del método de estudio. Existen varias formas de abordar el estudio de este campo caracterizado por la existencia de miles de entornos. En nuestro caso se ha optado por un enfoque analítico y sistemático, en el que, a través de una serie de esquemas clasificatorios, se ordena la mayor parte del software, herramientas, entornos, paquetes, etc., disponibles.

...se entenderá por software matemático *toda implementación computacional de algoritmos matemáticos.*

Un primer marco de clasificación nos permite delinear tres grandes áreas:

a) Lenguajes de programación.

b) Librerías matemáticas.

c) Sistemas de álgebra computacional.

El presente artículo se centrará en la primera de las áreas: los lenguajes de programación.

Los lenguajes de programación y el software matemático

Teniendo en cuenta el enfoque presentado, el primer tema que se debe abordar es el relativo a los lenguajes de programación, y ello por dos razones fundamentales.

En primer lugar, porque la implementación real de algoritmos matemáticos se debe hacer mediante algún lenguaje de programación. Siempre que alguien intente implementar cualquier algoritmo, desde los más básicos o elementales hasta los más sofisticados y complejos, deberá recurrir a un lenguaje, que es un concepto abstracto que engloba estructuras de datos, mecanismos o reglas lógicas que rigen la manipulación simbólica, estructuras de control del flujo del proceso, pero que es también un ente concreto formado por compiladores, editores, depuradores, etc. Además de esta razón, ocurre que el resto de los paquetes, librerías, entornos interactivos, sistemas de álgebra computacional, programas especializados para determinados ámbitos de la matemática aplicada, etc.; en general, todos los entornos más o menos sofisticados del software matemático, están programados en algún lenguaje, de forma que el conocimiento del lenguaje que le sirve de base puede ayudar a comprender de una forma más adecuada sus prestaciones, la sintaxis propia del entorno, e incluso nos puede per-

1 FLOPS (floating-point operations per second) se define como operaciones matemáticas elementales en punto flotante (usando decimales) por segundo.

mitir incorporar módulos propios o definidos por el usuario para abordar cuestiones no resueltas directamente por el paquete.

Otra razón, de cariz más general, es que el estudio de los lenguajes de programación permitirá conocer las técnicas básicas y fundamentales que se están empleando a nivel computacional y que sirven de soporte para comprender la evolución y las potencialidades del software que se está desarrollando y se desarrollará en los próximos años.

El resto del artículo se estructurará en torno a los siguientes temas: En la sección siguiente se presentará una breve descripción de lo que se entiende por lenguaje de programación, junto con una primera clasificación, muy esclarecedora, de los cuatro modelos o paradigmas básicos de la programación. Más adelante se ofrece una segunda clasificación de los lenguajes, en este caso dirigida por un motivo más historicista y que pretende dar cuenta de la evolución de los lenguajes a través de las así denominadas generaciones. A continuación se presentarán brevemente las fechas más características asociadas con los lenguajes más representativos, y por último, se abordará el estudio más técnico de las características de los lenguajes que han tenido, tienen o podrán tener mayor interés para el software matemático.

Los lenguajes de programación. Una clasificación metodológica

Partimos de la noción de *lenguaje* entendido de acuerdo con la tradición del racionalismo lingüístico antisolipsista representada por el *Tractatus logico-philosophicus* de Wittgenstein, según la cual un lenguaje es un conjunto de reglas que permiten la comunicación o transmisión de ideas, comunicación que es posible al ser las reglas compartidas y aceptadas por un grupo de individuos. La consecuencia fundamental de este planteamiento es la noción de convención en torno a un conjunto de reglas: un *formalismo*, para la comunicación de ideas, entendidas estas últimas en un sentido muy amplio, donde por supuesto caben los algoritmos.

...un lenguaje es un conjunto de reglas que permiten la comunicación o transmisión de ideas, comunicación que es posible al ser las reglas compartidas y aceptadas por un grupo de individuos.

Así pues, el concepto de lenguaje queda automáticamente ligado con otro no menos importante a este nivel como el de *gramática*, entendida esta última como el conjunto de reglas que especifican la sintaxis de un lenguaje. El conjunto de reglas que forman una gramática definen todas las construcciones válidas del lenguaje, por lo tanto, una gramática se puede contemplar formalmente como una definición intensiva de un lenguaje. Desde un punto de vista constructivo o de diseño, una gramática está formada por cuatro elementos básicos:

a) Un conjunto de *símbolos terminales*, que son las realizaciones atómicas (individuales) concretas que se combinan para formar construcciones del lenguaje.

b) Un conjunto de *símbolos no terminales*, que no forman parte del lenguaje en sí mismo, sino que representan elementos intermedios de cuya definición se encargan las reglas de producción, y que de alguna forma constituyen las abstracciones teóricas o conceptuales implicadas en el proceso de análisis de una entrada según las reglas, para determinar su gramaticalidad.

c) Un conjunto de *reglas de producción* que especifican formalmente las definiciones de los símbolos no terminales y que a nivel sintáctico determinan el proceso de análisis de la gramaticalidad de las construcciones de símbolos no terminales del lenguaje.

d) Un *símbolo meta*, perteneciente al conjunto de los símbolos no terminales, y que concebido como símbolo inicial de derivación sobre el que operan las reglas de producción determina el conjunto (finito o infinito) de fórmulas (combinaciones o secuencias de símbolos terminales del lenguaje) bien formadas.

La estructura de las reglas de producción determina el modelo gramatical del lenguaje. A este nivel es aún aceptada la clasificación de Chomsky que distingue entre gramáticas generales, sensibles al contexto, de contexto libre y regulares.

Esto nos permite establecer una primera especificación de lo que es un lenguaje de programación como en general un lenguaje, que, por consiguiente, tendrá asociada de forma biunívoca una gramática que determinará exhaustivamente la sintaxis del mismo y las fórmulas bien formadas que lo componen. Por tanto, en cuanto a su presentación formal un lenguaje de programación, o más exactamente su gramática, será una 4-tupla:

$$G = (T,N,P,I)^2$$

En esta definición el elemento determinante es el conjunto de reglas de producción o producciones, cuya estruc-

2 Donde T es el conjunto de símbolos terminales, N es el conjunto de los símbolos no terminales (y se cumple $T \cap N = \phi$), P es el conjunto de las reglas de producción e I es el símbolo inicial ($I \in N$).

tura determinará la complejidad gramatical del lenguaje. La mayoría de los lenguajes actuales poseen gramáticas incluidas dentro del tipo de contexto libre (CFG: *context free grammars*) dentro de la clasificación de Chomsky[3].

No obstante, los lenguajes de programación presentan ciertas características que los diferencian del resto de los lenguajes naturales, características que afectan fundamentalmente a los focos de la comunicación (intervención del computador), al contenido (programas, cuya estructura es significativamente diferente a la comunicación interpersonal) y al medio o canal (el simbolismo y la semántica que sirven de soporte a la construcción de programas).

Estas consideraciones nos permiten concluir con la siguiente definición: un *lenguaje de programación* es un lenguaje cuya gramática ha sido diseñada específicamente para que una *persona* pueda expresar formalmente el *proceso* que un *computador* deberá seguir para resolver un *problema.*

Los cuatro conceptos escritos en cursiva polarizan la atención a la hora de abordar una clasificación metodológica de los lenguajes de programación; pudiendo hablarse de lenguajes de programación orientados a la persona, al proceso, al computador y al problema, como se muestra en la figura 1.

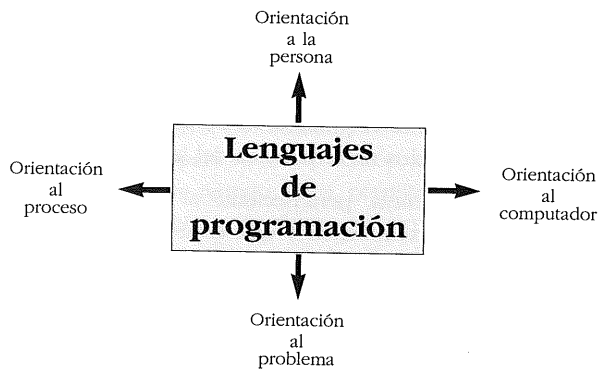


Figura 1. Orientaciones de los lenguajes de programación

Según qué polo sea el dominante se podrán establecer cuatro modelos de lenguajes que dan lugar a otras tantas metodologías de la programación, las cuales aparecen en la figura 2.

El modelo imperativo o procedimental

Este modelo asume la perspectiva del computador o, más exactamente, la perspectiva de la arquitectura SISD[4] o modelo Von Newmann. Teniendo en cuenta que bajo este modelo la computación se concibe como tratamiento (almacenamiento, manipulación y recuperación) de la informa-

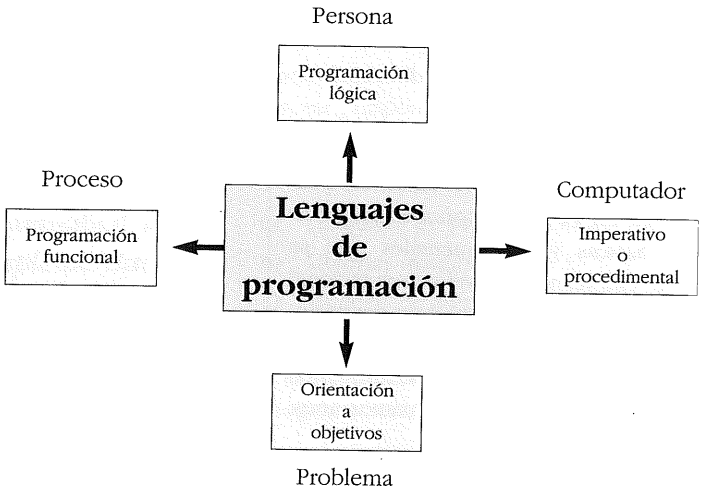


Figura 2. Metodología de programación

Sea G una gramática definida como $G = (T,N,P,I)$.

(a) Se entenderá por *cadena de símbolos* s cualquier secuencia, con o sin repeticiones, de símbolos del vocabulario de G

$$T \cup N$$

(b) Por *tamaño de una cadena*: $|s|$ se indicará el número de símbolos, incluidas repeticiones, de la cadena s.

(c) Se define una cadena especial: la *cadena vacía*: ϕ.

(d) Para cualquier conjunto (A) de símbolos se define el producto cartesiano $A \times A$, o A^2, como:

$$A \times A = \{s = ab \,/\, a \in A,\ b \in A\}$$

(e) A^i se define como la aplicación i-1 veces del producto cartesiano sobre A.

(f) $A^* = \Sigma_{\geq 0} A^i \qquad A^+ = \Sigma_{\geq 0} A^i = AA^* = A^* - \{\varepsilon\}$

A partir de estos conceptos se pueden obtener las especificaciones formales de los cuatro tipos de gramáticas, según la forma de sus producciones:

Tipo 0: Gramáticas generales

$$\forall (\alpha, \beta) \in P,\ \alpha \in (T \cup N)^+,\ \beta \in (T \cup N)^+$$

Tipo 1: Gramáticas sensibles al contexto

$$\forall (\alpha, \beta) \in P,\ \alpha \in (T \cup N)^+,\ \beta \in (T \cup N)^+,\ |\alpha| \leq |\beta|$$

Tipo 2: Gramáticas de contexto libre

$$\forall (\alpha, \beta) \in P,\ \alpha \in N,\ \beta \in (T \cup N)^*$$

$$|\alpha| \leq |\beta|$$

Tipo 3: Gramáticas regulares

$$\forall (\alpha, \beta) \in P,\ \alpha \in N,$$

$$\begin{cases} \beta = Ab, \text{ donde } A \in N,\ b \in T \\ \text{ó} \\ \beta = b, \text{ donde } b \in T \end{cases}$$

3 Una caracterización, aunque somera, de los tipos de gramáticas de la clasificación de Chomsky viene expresada en el recuadro adjunto.

4 Las siglas SISD corresponden a uno de los grupos (Single Instruction Single Data) de la clasificación de Flynn en 1966. Los computadores pertenecientes a este grupo se caracterizarían por tener CPUs capaces de procesar simultáneamente sólo una instrucción (Single Instruction) que opera sólo sobre un dato (Single Data).

ción contenida en la memoria del computador, los lenguajes que se basan en él reflejan de una forma bastante directa la ejecución secuencial de comandos y la utilización de datos (variables) cuyo contenido se modifica durante el proceso.

Debido a la cercanía con la arquitectura del procesador, su traducción a código máquina resulta relativamente sencilla por lo que fueron los primeros en aparecer, y los únicos hasta la década pasada. Entre los lenguajes que se pueden adscribir a este modelo destacan Fortran, Pascal, Ada y C.

La orientación lógica

La programación u orientación lógica de los lenguajes de programación se centra en la perspectiva de la persona, encarando el problema desde un punto de vista lógico. Es decir, aborda la especificación de un problema no desde el enfoque basado en *cómo hacer* (secuencias de pasos necesarios para obtener la solución) sino en *qué hacer* (hechos y reglas relevantes que pueden ser utilizados para deducir la solución).

La consecuencia principal de este modelo es la separación, tanto en el ámbito conceptual como en el de implementación, entre lo que es la *base de conocimientos* específica para el problema particular (y que en sí constituye el programa) y el *motor inferencial* genérico que incorpora los mecanismos de razonamiento que se aplican a la base de conocimiento para obtener la solución del problema (y que de alguna forma constituye el lenguaje).

Desde un punto de vista lógico, un programa escrito según el modelo que nos ocupa estaría formado por un conjunto de axiomas (hechos y reglas que se asumen como verdaderos) y una sentencia objetivo o meta a demostrar. A partir de estos datos, las reglas de inferencia deberán determinar si los axiomas permiten deducir el objetivo propuesto. La forma como esto se hace no es competencia del programador sino que forma parte del lenguaje mismo. Es decir, los programas escritos según el modelo lógico indican una meta y los hechos que son relevantes para su demostración, pero no expresan el método de derivación, sino que éste podrá ser el resultado de la ejecución del programa, en el caso en que la meta sea derivable de los hechos.

Desde un punto de vista lógico, un programa escrito [...] estaría formado por un conjunto de axiomas [...] y una sentencia objetivo o meta a demostrar. A partir de estos datos, las reglas de inferencia deberán determinar si los axiomas permiten deducir el objetivo propuesto.

Los ejemplos clásicos de orientación lógica lo constituyen Prolog y los lenguajes de interrogación para bases de datos relacionales.

El modelo funcional

Este modelo adopta el punto de vista centrado en el proceso de solución del problema. La idea que le sirve de base es la noción matemática de función, concebida como un tipo especial de correspondencia entre un conjunto dominio y un conjunto imagen o destino. La traducción gramatical de este concepto para servir como modelo de programación supone la equiparación entre el conjunto dominio con todas las posibles entradas y el conjunto destino con las salidas posibles. La definición de la función muestra cómo un elemento del conjunto imagen es obtenido a partir de un elemento del conjunto dominio.

A nivel formal se suele utilizar en el modelo funcional la denominada *expresión lambda* desarrollada por Church en 1941.

El lenguaje funcional más clásico es Lisp, el cual ha servido de modelo para otros lenguajes y entornos.

La programación orientada a objetos

Por último, y de aparición más reciente, encontramos la programación orientada a objetos, cuya filosofía de diseño se preocupa fundamentalmente por la perspectiva del problema real.

Desde un punto de vista estructural, este modelo está formado por cuatro componentes básicos:

— *Clases.* Una clase define un modelo de objetos. Puede concebirse como un tipo de datos abstracto (TDA) que incluye una estructura interna y un conjunto de operaciones. Con mayor detalle, una clase está formada por una serie de métodos y descripciones que asumirán automáticamente todos los objetos de la clase sin necesidad de una redefinición para cada caso.

— *Métodos.* Se trata de las descripciones de las operaciones que un objeto perteneciente a una clase realizará cuando reciba un mensaje. Es decir, la recepción de un mensaje por parte de un objeto desencadenará la ejecución de un método (que estará en función del tipo del mensaje, el tipo de objeto y su estado actual).

— *Mensajes.* Son los ítems de información transferidos desde un objeto hasta otro con la finalidad de obtener un resultado determinado, estando predefinidos en el objeto receptor las relaciones entre mensajes y métodos; además el objeto emisor no tiene por qué conocer el mecanismo de resolución del problema, sino que una vez enviado el mensaje, las operaciones necesarias que se deban ejecutar para obtener el resultado requerido estarán determinadas y controladas exclusivamente por el objeto receptor.

— *Objetos.* Los objetos son las instancias particulares de las clases y, por lo tanto están constituidas por un conjunto encapsulado de operaciones y una descripción actual del estado del objeto, que se encarga de mantener permanentemente los efectos de las operaciones ejecutadas sobre él. A partir de la recepción de un mensaje asociado con una operación, el objeto activa el método correspondiente, pudiendo, entre otras cosas, enviar nuevos mensajes a otros objetos.

Las principales características o propiedades del modelo son:

- *Herencia*: es posible definir una jerarquía de clases, y por extensión, de objetos, de forma que sea automática la asunción de propiedades, métodos, etc., por parte de las clases inferiores en la escala; permitiéndose, no obstante, la especificación de excepciones.
- *Encapsulado*: esta propiedad permite ver a los objetos como estructuras autónomas, donde el modelo descriptivo del estado y los pares mensajes-métodos vienen predefinidos por la clase a la que pertenece el objeto y el valor actual del objeto está determinado por los métodos activados en respuesta a los mensajes de una ejecución particular.
- *Polimorfismo*: lo que significa que el mismo mensaje puede ser enviado a diferentes objetos, y que el método adecuado se establecerá según la clase a la que pertenezca cada uno de los objetos receptores. E incluso, se podrán enviar mensajes sin conocer la clase de objetos receptora.

El primer lenguaje que implementó la metodología de la programación orientada a objetos fue Simula en 1966, posteriormente revisado en Smalltalk. C++ se considera un lenguaje híbrido, donde a partir de un lenguaje imperativo se han añadido algunas de las características del modelo orientado a objetos.

Una clasificación histórica de los lenguajes de programación

Son muchas las clasificaciones de los lenguajes de programación en etapas a lo largo de los últimos 40 o 50 años. De entre ellas se ha elegido la siguiente por ser representativa de la evolución que se ha logrado a este nivel. Básicamente se estructura la historia de los lenguajes de programación en cinco generaciones.

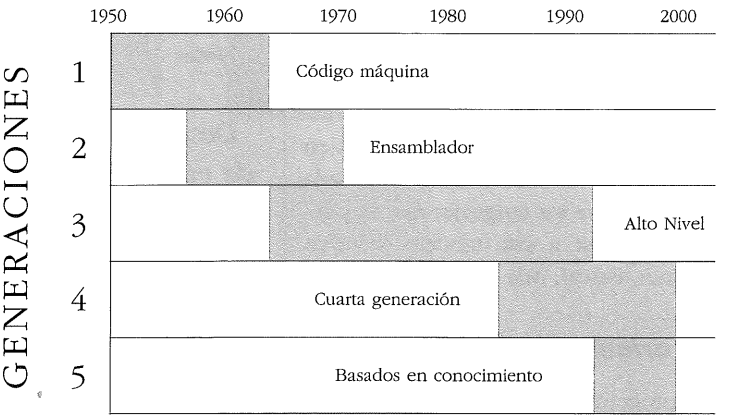


Figura 3. Las generaciones de los lenguajes de programación

Primera generación: Lenguajes máquina

La primera generación (1950-1960)[5] está basada en la utilización de instrucciones en código máquina, es decir, la programación se realiza utilizando directamente los códigos binarios de los procesadores. Los dos problemas de esta metodología eran la dificultad del código y la dependencia de la arquitectura.

Segunda generación: Lenguajes ensambladores

La segunda generación (1955-1970) está caracterizada por la presencia de lenguajes ensambladores simbólicos. Básicamente pretendió corregir uno de los problemas de la generación previa: la dificultad del código binario, al ofrecer un código alfabético mnemotécnico para cada instrucción máquina, haciendo más sencilla la comprensión y el desarrollo de programas. No obstante, persistía la segunda dificultad, a saber, la dependencia de la arquitectura y los consiguientes problemas de portabilidad.

5 Como es obvio, todos los procesos históricos no son fácilmente demarcables. Esto se ha visto reflejado en la breve historia de la informática, y así las fechas que se indican para cada generación deben ser comprendidas como etapas con límites difusos, pues las ideas asociadas con cada metodología y cada lenguaje no aparecen ni se eliminan de una forma puntual.

Tercera generación: Lenguajes de alto nivel

La tercera generación de lenguajes agrupa los denominados lenguajes de alto nivel y se inserta dentro de una corriente general de renovación en el ámbito de la informática ejemplificada por la tendencia a la estandarización de los sistemas operativos, intentando asegurar de esta forma la portabilidad de las aplicaciones.

Quizás la característica principal de esta generación sea la presencia de un entorno de desarrollo cercano al programador, que realiza grandes abstracciones con respecto a las instrucciones nativas de la máquina.

Los primeros lenguajes de alto nivel en aparecer fueron Fortran y Cobol, a los que siguieron Pascal, C, PL/1, etc., que implementan ya metodologías estructuradas de programación.

Cuarta generación

Desde principios de los ochenta se viene hablando de lo que sería una cuarta generación de lenguajes concebidos como herramientas para el desarrollo rápido de aplicaciones. Enlazado con la clasificación metodológica descrita en la sección anterior, esta generación representa la imposición de la metodología de la orientación lógica sobre la programación imperativa característica de la tercera generación; es decir, el paso desde el *cómo hacer* al *qué hacer*.

La posibilidad de una quinta generación de lenguajes ha venido vinculada a la realizabilidad de las técnicas de inteligencia artificial, y realmente suponen un nuevo paso en la incorporación de conocimiento a los sistemas computacionales.

Dentro de los entornos de cuarta generación se incluyen:

- Sistemas de gestión de bases de datos.
- Lenguajes de interrogación (tipo SQL).
- Generadores de informes.
- Gráficos comerciales interactivos.
- Paquetes de software integrado. Etc.

Quinta generación: Sistemas basados en el conocimiento

La posibilidad de una quinta generación de lenguajes ha venido vinculada a la realizabilidad de las técnicas de inteligencia artificial, y realmente suponen un nuevo paso en la incorporación de conocimiento a los sistemas computacionales.

Entre los sistemas basados en conocimiento que caracterizan esta quinta generación se encuentran:

- Los sistemas expertos.
- Los motores de inferencia.
- El procesamiento del lenguaje natural. Etc.

Las especificaciones formales de estos entornos introducen fuertes requisitos en cuanto a potencia computacional para lograr implementaciones eficientes. En concreto, la investigación actual ha fijado su interés en el procesamiento masivamente paralelo.

Breve historia de los lenguajes de programación

En esta sección se pretende ofrecer una breve relación de las fechas más importantes desde el punto de vista de la historia de los lenguajes de programación:

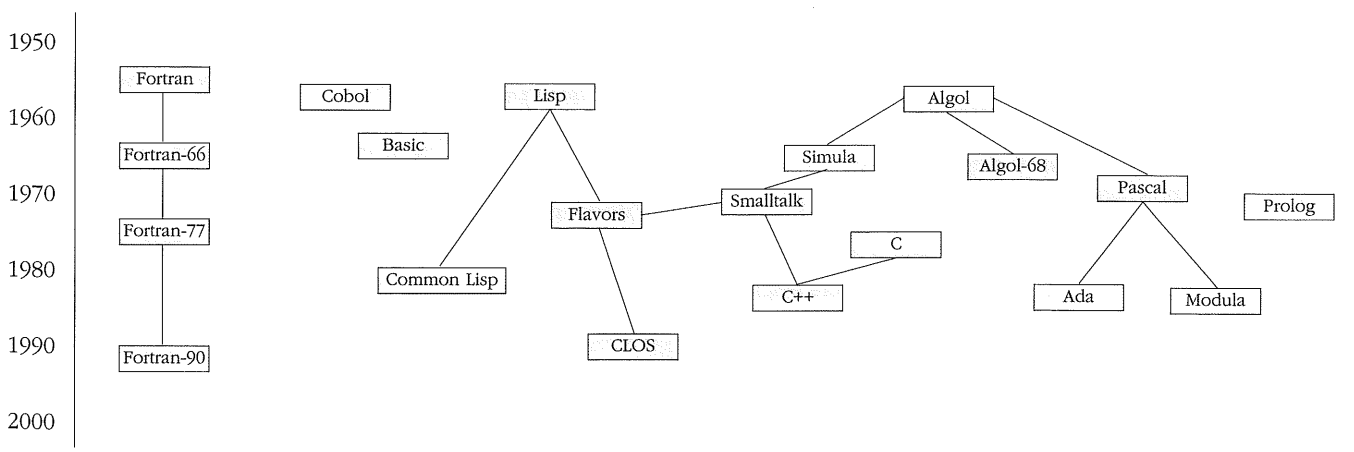


Figura 4. Principales relaciones entre los lenguajes de programación

1956 Fortran: diseñado e implementado por John Backus, IBM.

1960 COBOL.

ALGOL 60.

LISP, desarrollado por John McCarthy en el Artificial Intellegence Group del M.I.T

1962 APL, especificado por Kenneth Iverson.

1965 BASIC, desarrollado por Thomas Kurtz y John Kemeny es el Dartmouth College.

1966 Fortran 66, ANSI X3.9-1966.

1967 Simula 67, desarrollado por Ole-Johan Dahl y Kristan Nygaard.

1969 ALGOL 68.

Pascal.

1972 Prolog, desarrollado en la Universidad de Marsella por Alan Colmerauer.

1973 Smalltalk, desarrollado por Alan Kay en el centro de Investigación de Xerox en Palo Alto.

1978 C, desarrollado en los laboratorios Bell, desde 1972 por Dennis Ritchie.

Fortran 77, ANSI X3.9-1978.

1981 Common LISP.

1982 Modula-2, desarrollado por Niklaus Wirth.

1983 Ada ANSI/MILSTD 1815A.

1986 C++, Bjarne Stroustrup añadió en los laboratorios Bell de ATT características de orientación a objetos al lenguaje imperativo C.

1991 FORTRAN 90, ISO/IEC- 1953.

Los lenguajes de programación más importantes desde el punto de vista del software matemático

Prácticamente todos los lenguajes de programación incorporan elementos que los hacen susceptibles para ser utilizados en la implementación computacional de algoritmos matemáticos. No obstante, entre los lenguajes existen diferencias que afectan fundamentalmente a la cercanía notacional con respecto a las matemáticas, a la eficiencia con que ejecutan las implementaciones o a las herramientas de desarrollo que acompañan al lenguaje, desde librerías de funciones hasta editores sensibles al lenguaje. Esto ha hecho que no todos los lenguajes se hayan usado con la misma frecuencia en este ámbito, como lo demuestra la comparación, del que quizás es el caso más obvio, entre Fortran y Cobol.

Prácticamente todos los lenguajes de programación incorporan elementos que los hacen susceptibles para ser utilizados en la implementación computacional de algoritmos matemáticos.

Entre los lenguajes que se encontrarían en un primer escalafón en importancia con respecto al software matemático se podrían incluir Fortran, Lisp y C. A un segundo nivel se puede destacar también el uso extendido de Pascal, Ada, Modula, etc. Por motivos obvios de extensión centraremos nuestro estudio en los tres primeros.

FORTRAN

Se trata del lenguaje de alto nivel más antiguo, ideado inicialmente por John Backus en IBM, y concebido específicamente para el desarrollo de aplicaciones propias de la ciencia y la ingeniería. Es interesante notar que ante las alternativas que suponía el código máquina y el código ensamblador, el grupo dirigido por Backus se preocupará por obtener una sintaxis bastante cercana a la notación matemática usualmente empleada, de ahí el origen del término FORTRAN: **FOR**mula **TRAN**slations. A través de su dilatada historia, que se ha reflejado en la aparición de distintas normalizaciones: Fortran 66, Fortran 77 y el reciente Fortran 90, Fortran se ha convertido quizás en el lenguaje más usado para computación científica, existiendo realmente una cantidad ingente de material ya implementado en este lenguaje.

Desde un punto de vista más técnico, Fortran es un claro representante de la orientación procedimental, a la que contribuyó de una forma especial incorporando, entre otras, las siguientes nociones:

- Las variables y las sentencias de asignación.
- Los tipos de datos.
- La modularidad, a través del uso de subprogramas.
- Formateo de las entradas y las salidas.

Entre las características incorporadas por el nuevo estándar, Fortran 90, merecen destacarse las siguientes:

- Almacenamiento dinámico.
- Un formato fuente que abandona las características de las tarjetas para adecuarse más al uso de terminales.

- Tratamiento de arrays.
- Punteros.
- El usuario puede definir tipos de datos compuestos a partir de otras estructuras de datos, y puede también definir operaciones sobre las nuevas estructuras.
- Procedimientos recursivos.
- Argumentos opcionales en la llamada a procedimientos. Etc.

C

Ideado inicialmente por Dennis Ritchie a principios de los setenta en los laboratorios Bell, es un lenguaje que, aún a pesar de estar dentro de la metodología procedimental a la que también pertenece Fortran, representa no obstante, lo que podría denominarse una segunda corriente caracterizada por la idea de sistemas abiertos, portabilidad y gran eficiencia. Las ventajas que suponen estas características se han visto corroboradas por la creciente popularidad de C y UNIX.

Entre las características más destacables del lenguaje merecen citarse:

- El énfasis que pone para conseguir una gran generalidad y economía de las expresiones.
- El incluir un potente conjunto de estructuras de control y operaciones de manipulación.
- El permitir al programador un control, a bajo nivel, del sistema, llegando incluso hasta manipulaciones a nivel de bit, etc.
- Los operadores y la aritmética de punteros le permiten al usuario acceder a las facilidades más básicas, y también más eficientes, de la máquina, convirtiéndose en un lenguaje muy adecuado para el diseño de programas del sistema o programas en los que los recursos (tiempo y memoria fundamentalmente) pasan a ser un criterio clave.
- En general, la filosofía de C consiste en darle al programador todo el control posible sobre la ejecución del programa.

El caso de LISP es históricamente similar al de Fortran.

Estas características han hecho que C se convierta en un importante foco de interés para la implementación de software matemático, donde los problemas no son triviales y hay que aumentar tanto como sea posible la eficiencia de los algoritmos, aprovechando las características de procesamiento a bajo nivel, sin perder de vista en ningún caso la necesidad de portabilidad que representa la filosofía de los sistemas abiertos.

LISP

El caso de LISP es históricamente similar al de Fortran. Se trata también de uno de los primeros lenguajes de alto nivel, diseñado en 1960 por John McCarthy, y cuya historia se ha dilatado hasta la actualidad, habiendo influido en el desarrollo de otros lenguajes y entornos, aunque Lisp como tal nunca haya sido estandarizado.

La metodología de diseño y la orientación básica de Lisp hacen que difiera bastante de Fortran o C. Lisp no es un lenguaje con pretensiones numéricas ni posee una filosofía orientada a la ejecución procedimental que lo hagan cercano o cómodo para la traducción de algoritmos matemáticos concebidos secuencialmente. Por contra, Lisp representa el interés por la computación simbólica.

La estructura básica de Lisp, y de la que ha tomado su nombre, es la lista (LISP = **LIS**t **P**rocessing); mediante la utilización de éstas, LISP implementa la programación funcional a través de la invocación de funciones y su composición para la resolución de problemas complejos.

Esta equiparación entre estructuras de datos y unidades de proceso (listas y funciones) le permite al programador de Lisp utilizar la misma estructura para almacenar los datos y los programas, lo que ofrece nuevas posibilidades en el ámbito de la metaprogramación (al concebir los programas como datos que pueden ser procesados por otros programas).

Aunque Lisp, por las características mencionadas, no es un lenguaje adecuado para el procesamiento numérico, sus capacidades para el procesamiento simbólico lo han convertido en el lenguaje más frecuente en campos como la demostración automática de teoremas, el álgebra o la geometría computacional, la derivación e integración simbólicas, etc.

Resumen

El objetivo del artículo ha sido el análisis de los lenguajes de programación en su relación con el software matemático. Para ello se llevó a cabo una discusión inicial acerca de lo que se entenderá como software matemático, para pasar a continuación a un estudio de los lenguajes de pro-

gramación. Este estudio se ha estructurado a través de una clasificación de las cuatro metodologías básicas de programación, que son la procedimental o imperativa, la orientación lógica, la programación funcional y la orientación a objetos. Asimismo se ha presentado una estructuración histórica de las principales generaciones de los lenguajes y los hitos más significativos y representativos de la historia de éstos.

Por último se han analizado los tres lenguajes de programación cuya utilización es más común desde el ámbito del software matemático.

Bibliografía

BATE, J. J y D. B. VADINA (1987): *Fourth Generation Languages*, London, BSP Professional Books.

DERSHEM, H. L. y M. J. JIPPING (1990): *Programming Languajes: Structures and Models*, Wadsworth Publishing Company, Belmont, California.

KERNIGHAM, B. W y D. M. RITCHIE (1978, 1ª ed): *The C programming language*, Prentice Hall Software Series. (Existe traducción al expañol).

METCALF, M. y J. REID (1993): *Fortran 90 Explained*, Oxford University Press, Oxford.

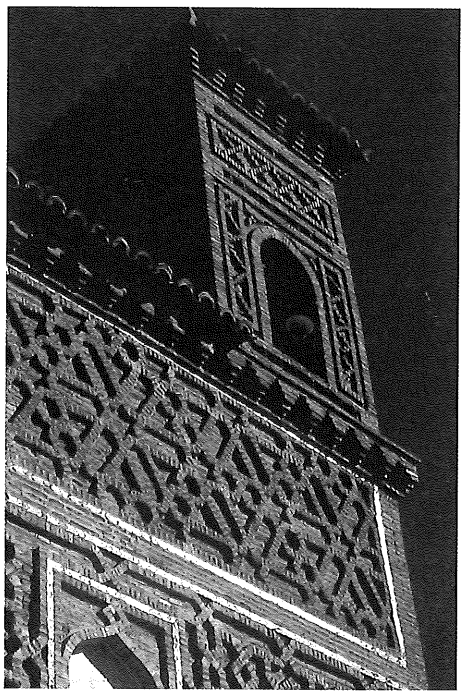
MUELLER, R. A. y R. L. PAGE (1988): *Symbolic Computing with Lisp and Prolog*, John Wiley & Sons, Inc.

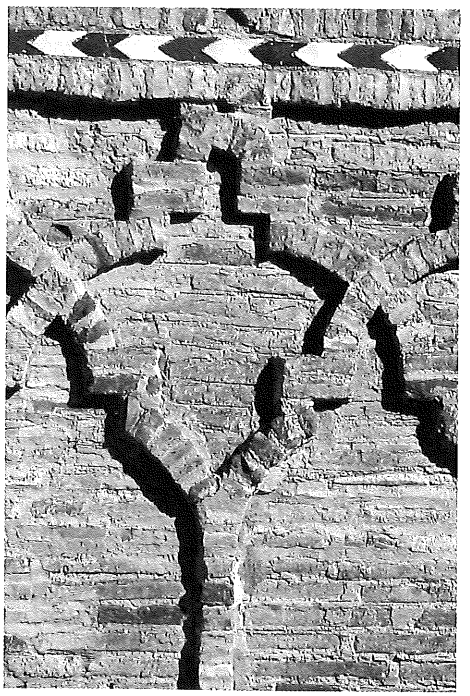
STEELE, G. L. (1990): *Common Lisp*, Digital Press.

STROUSTRUP, B. (1986): *The C++ programming language*, Addison Wesley Publishing Company.

WINBLAND, A. L; S. D. EDWARS y D. R. KING (1993): *Software orientado a objetos*, Addison-Wesley-Diaz de Santos.

José F. Quesada
C.I.C.A.
Sevilla





Geometría mudéjar
Iglesia de Tobed (Zaragoza).
Siglo XIV
(Fotos: F. Villarroya)