7

LENGUAJES DE PROGRAMACIÓN

7.1. Lenguajes de programación. Evolución.

El soporte lógico, o software, de una computadora es el conjunto de programas asociados a dicha computadora.

Hemos visto en el tema anterior el sistema operativo, como pieza fundamental en el funcionamiento de una computadora. En esta lección nos centraremos en los lenguajes de programación, como una forma de comunicarse con la computadora para indicar la tarea que se quiere realizar y la forma de llevarla a cabo.

Desde el comienzo de la historia de la informática, la programación de computadores se ha convertido en una disciplina por derecho propio. Los primeros sistemas de programación, que utilizaban conexiones eléctricas realizadas con cables sobre tableros móviles, fueron rápidamente sustituidos por otros que se apoyaban en métodos cada vez más sencillos y que, en consecuencia permitieron alcanzar niveles más altos de complejidad. Esta evolución afectó simultáneamente a los medios de introducción de programas en los computadores (cinta de papel, tarjetas perforadas, teletipo, máquina de escribir, terminal provisto de pantalla, etc.) y a la forma de programar (lenguaje máquina, lenguaje simbólico, lenguajes de alto nivel, sistemas de desarrollo de aplicaciones, entornos de generación de sistemas de bases de conocimiento, etc.)

Un programa completo consta siempre de dos partes: las instrucciones ejecutables (o programa en sentido estricto) y los datos sobre los que actúan estas instrucciones. Según la forma en que se organicen los datos y programas, se distinguen las formas de programar: programación lógica, programación orientada a objetos o programación procedimental; en esta última las instrucciones que componen los programas se ejecutan secuencialmente en un orden preestablecido, que solo depende de los valores de los datos a los que se aplica y que se puede deducir de estos, inspeccionando el programa.

7.2. Algoritmo: noción de programa.

Para que un computador pueda llevar a cabo una tarea se le debe de proporcionar un método para su ejecución descrito en una forma muy precisa, en términos de sus pasos diferentes. Un algoritmo es la descripción de los pasos de una tarea, usando un método particular. Definir un algoritmo es el primer paso en la preparación de una tarea para que la ejecute un computador.

Si se quiere utilizar un algoritmo en un computador se debe de expresar una tarea en términos de un número finito de pasos. Por ejemplo, un algoritmo para sumar un conjunto de números se podría expresar de la siguiente forma:

Poner total a cero
Mientras queden más números, repetir
    Sumar el número siguiente al total

Es interesante resaltar, que en este caso la tercera línea está controlada por la segunda.

7.3. El lenguaje ensamblador.

El objetivo fundamental de un ensamblador es traducir un programa escrito en lenguaje ensamblador al lenguaje máquina de un determinado computador, al tiempo que facilita la tarea de desarrollo de los programadores. El emsamblador es específico para cada procesador, en consecuencia computadores con distinto procesador tendrán lenguaje emsamblador distinto. Se denomina de igual forma al lenguaje que al traductor. Con el lenguaje se desarrollan los programas y con el traductor se convierten de lenguaje ensamblador (lenguaje de bajo nivel, próximo al lenguaje máquina que es lo que realmente "entiende" la máquina) al lenguaje máquina.

Veamos unos ejemplos comparativos escritos en lenguaje máquina y en ensamblador para una máquina supuesta, teniendo en cuenta que son parecidos en cuanto la utilización de nemónicos.

lenguaje máquina	ensamblador
1312 00AB	ADD A NUM

La interpretación del comando anterior, sería: Sumar el número contenido en la dirección 00AB, cuya dirección simbólica es NUM, al acumulador.

El siguiente ejemplo realizaría la comparación del número que se encuentra en el acumulador, con el que está en la dirección NM1.

lenguaje máquina	ensamblador
1A12 0002	CMP A NM1

7.4. Lenguajes de alto nivel.

El desarrollo de los lenguajes de alto nivel comenzó a mediados de los años cincuenta, unos diez años después del nacimiento de los computadores electrónicos digitales. Durante estos años se puso de manifiesto que la mayor limitación de los computadores estaba a nivel de software y no de hardware. Escribir programas que funcionen correctamente en lenguaje ensamblador y aún más en lenguaje máquina, es una tarea pesada, difícil y cara en cuanto al tiempo dedicado.

Existía una resistencia notable a la idea de los lenguajes de alto nivel, se suponía que serían ineficaces comparados con los programas escritos en código máquina. Considerando el pequeño tamaño de la memoria y la escasa potencia de los procesadores de los años cincuenta, este temor tenía cierta justificación, dado que los lenguajes de alto nivel necesitan el traductor que lo convierte en lenguaje máquina, que está ocupando memoria.

A pesar de los problemas mencionados, una vez que se dispuso de los lenguajes de alto nivel, su uso se extendió rápidamente. Se han escrito muchos lenguajes y se han diseñado traductores (generalmente compiladores más que intérpretes) para implementar estos lenguajes en muchos de los computadores actualmente disponibles. Citamos como más conocidos los lenguajes COBOL, BASIC, FORTRAN, PASCAL, C, ALGOL, ADA, LISP, PROLOG, C++, JAVA.

7.4.1. ¿Qué es un lenguaje de alto nivel?

Un lenguaje de alto nivel es un lenguaje orientado hacia la resolución de una determinada clase de problema, mientras que un lenguaje de bajo nivel esta orientado a una determinada máquina o clase de máquina.

El lenguaje de alto nivel es independiente de la arquitectura del computador que lo soporta. Esto presenta dos ventajas principales. En primer lugar, la persona que desarrolla los programas no tiene por que saber nada del computador donde se va a ejecutar ese programa. Y, en segundo lugar, los programas son portátiles, es decir, el mismo programa puede funcionar sobre otro tipo de computador, siempre que el lenguaje este soportado por esa máquina.

En la mayoría de los casos los programas de alto nivel son mas cortos que el correspondiente en bajo nivel, sin embargo la cercanía al lenguaje máquina de los de bajo nivel les hacen más rápidos. Una instrucción en alto nivel se desglosa, en general, en varias de bajo nivel.

7.4.2. Tipos de lenguajes.

Los lenguajes de alto nivel se pueden clasificar de forma general en lenguajes de propósito general y lenguajes de propósito especial. Los lenguajes de propósito general se emplean igualmente en negocios, que aplicaciones científicas o en resolución de problemas de ingeniería, como en tareas de desarrollo de software de sistemas. Entre los lenguajes de propósito general cabe citar al PASCAL, C y ADA.

Las categorías más comunes de lenguajes de propósito especial son los lenguajes comerciales, científicos y educativos. En el campo comercial citamos al COBOL, en el campo científico, FORTRAN, y en el campo educativo, aunque ya en desuso, BASIC.

Otra forma de clasificar los lenguajes es en lenguajes de procedimiento y lenguajes declarativos:

- Los lenguajes de procedimiento establecen como debe de ejecutarse una tarea, dividiéndola en áreas de procedimiento (procedures en inglés) que especifican como realizar cada una de las tareas. Todos los lenguajes de alto nivel desarrollados en las primeras épocas eran lenguajes de procedimiento.
- Los lenguajes declarativos describen estructuras de datos y las relaciones entre ellas, siendo significativo para ejecutar una determinada tarea, a la vez indican cual es el objetivo de esa tarea. El proceso por el que se ejecuta la tarea no se establece de forma explícita en el programa. Este proceso se determina por el sistema de traducción del lenguaje. Prolog es un ejemplo de lenguaje de programación declarativo.

7.4.3.- Conceptos de Compilador e Intérprete

El objetivo principal de los programas de traducción de lenguajes es el de convertir un programa, o parte de un programa, escrito en lenguaje de alto nivel, en su equivalente en lenguaje máquina, utilizando en algunas ocasiones un lenguaje intermedio. El programa inicial se denomina fuente, el programa final es el programa objeto.

Un segundo objetivo de los traductores de lenguajes (compiladores e intérpretes) es la depuración de errores en el programa fuente. El programa produce errores de diagnostico que ayudan al programador a corregir los errores. Evidentemente el traductor debe generar un código eficiente, no solo depurado de errores, sino también que consiga realizar la tarea de una forma óptima. Es decir, no debe tardar demasiado en traducir el programa fuente, ni ocupar excesiva memoria principal. Por último, cabe citar que los programas de traducción forman parte de un conjunto de herramientas de software, entre las cuales están el editor y las librerías que acompañan al proceso de compilación, aportando facilidades ya desarrolladas.

El primer paso para el desarrollo de un programa es escribirlo, es decir, editar el código fuente. Cabe esperar una **diferencia entre compiladores e intérpretes en cuanto al editor**, que además es inherente al diferente concepto que hay entre uno y otro, y es que, en el del intérprete tendríamos un editor interactivo, que tan pronto como introducimos una línea la ensambla en código máquina con lo que nos presentaría los errores que se produjeran (sintaxis, etc.). Puede decirse que un intérprete dota al computador sobre el que se ejecuta de un lenguaje máquina de alto nivel. Cada una de las sentencias del programa se analiza y luego se ejecuta. No se produce código objeto y solo se pueden interpretar programas completos. BASIC es un ejemplo de lenguaje interpretado, ya en desuso.

Frente a esto un compilador traduce él código fuente (que puede ser un programa o una parte del programa) generando un programa en lenguaje máquina que contiene el código objeto equivalente. El código objeto puede ser "linkado" (enlazado con otros módulos y librerías) para generar un programa ejecutable.

Hay que hacer notar la diferencia entre el BASIC interpretado y el compilado, que sería un buen ejemplo entre ambas técnicas de traducción. En el BASIC intérprete se interpreta cada una de las líneas del programa, con lo que se enlentece la ejecución del programa; con el compilado se consigue un programa objeto que, una vez linkado, es un programa ejecutable globalmente, (obviamente mucho más rápido). Todos los demás lenguajes citados tienen un traductor del tipo compilador.

7.4.4. Objetivos de los lenguajes de alto nivel.

Las características que definen un lenguaje de alto nivel son que están orientados a un determinado tipo de problema y que son independientes de la máquina.

El primer objetivo de un lenguaje de alto nivel es el de proporcionar un medio conveniente para expresar la solución a un determinado problema. Existen otras dos formas frecuentes para lograr este objetivo, las matemáticas y los lenguajes naturales, como el español. El problema de los lenguajes naturales es su riqueza y su complejidad, que los hacen imposible de utilizar para programar un computador. Los conceptos matemáticos están claramente en los lenguajes de programación, añadiendo palabras tomadas de los lenguajes naturales (especialmente inglés) que combinadas con ciertos símbolos matemáticos y de acuerdo a unas reglas, sirven para crear un programa que pueda controlar el computador. El resultado es un buen lenguaje de alto nivel con una estructura clara, que no difiere demasiado de la forma de pensar y de expresarnos que tenemos normalmente.

Un segundo objetivo es la simplicidad, que se logra con un reducido conjunto de operaciones básicas, unas cuantas reglas para combinar estas operaciones, y por encima de todo, la falta de excepciones a estas reglas.

Como tercer objetivo está la eficiencia, los programa elegidos en el lenguaje deben traducirse a lenguaje máquina de una forma rápida y el código máquina resultante debe ser eficiente.

7.4.5. Características de los lenguajes de alto nivel.

Las características que definen un lenguaje de alto nivel son su juego de caracteres, las palabras reservadas de que dispone, las facilidades para estructurar los programas (es decir, que se pueda escribir un programa empleando proceso secuencial [IF-THEN-ELSE] y bucles [WHILE] y no utilizando la sentencia GOTO) y los datos, las operaciones que realiza, las facilidades de E/S y su estructura de control.

7.4.6. Programación clásica (o de procedimientos).

Un programa procedimental (escrito en BASIC, FORTRAN, PASCAL, COBOL, C, etc.), se caracteriza porque las instrucciones que lo componen se ejecuta secuencialmente, en un orden preestablecido, que solo depende de los valores de los datos a los que se aplica y que se puede deducir de estos, inspeccionando el programa.

La instrucción fundamental de la programación procedimental, la que permite construir la jerarquía de programas en que se basa este tipo de programación, es la instrucción CALL. Una aplicación determinada suele estar formada por una jerarquía de programas, módulos o subrutinas, cada uno de los cuales es capaz de invocar a otros de la jerarquía. Uno de los programas, situado en lo más alto de la jerarquía, recibe el nombre de programa principal. Los demás se conocen con el nombre de subprogramas, subrutinas, funciones o procedimientos (dependiendo del lenguaje con el que se realice la programación). En cuanto a los datos, dependiendo del lenguaje de programación, pueden ser globales (accesibles para todos los

programas de la jerarquía) o locales (utilizables solo por el programa al que pertenecen y, a veces, por aquellos que se encuentran en niveles jerárquicos inferiores a este).

Dentro de la programación procedimental, nos encontramos con la programación estructurada o programación sin GOTO, que presenta mejoras como son la facilidad de mantenimiento de los programas, gracias a la modularidad que se realiza en el diseño y posterior desarrollo de los programas. Cabe considerar a los lenguajes estructurados como PASCAL o C como una mejora aconsejable frente a los no estructurados como BASIC, FORTRAN o COBOL (FORTRAN 77 se considera un lenguaje estructurado)

Ejemplo: Escribir un programa que lea 10 números enteros, uno cada vez, los presente, y escriba su suma.

a) Realización en Pascal (lenguaje estructurado):

```
program enter10 (input,output);
var
   suma,dato,conta: integer;
begin
   suma:= 0;
   conta:= 0;
   while conta < 10 do
      begin
        readln (dato);
        writeln (dato);
        suma:= suma+dato;
        conta:= conta+1;
      end;
   writeln (' suma de 1o enteros = ',suma);
end.
```

b) Realización en Basic (lenguaje no estructurado):

```
10 suma=0
20 dato=0
30 conta=0
40 Input " Introduce el número"; dato
50 Print dato
60 suma=suma+dato
70 conta=conta+1
80 If conta < 10 then goto 40
90 Print " suma de enteros"; suma
100 end
```

7.4.7. Programación lógica.

Hacia los años setenta comenzaron a aparecer lenguajes basados en la utilización de nuevos métodos de programación, frente a los procedimentales, que se denominaron programación lógica. Un programa lógico, escrito en PROLOG (por ejemplo), recibe el control de un motor de

inferencia, que no es otra cosa que un programa procedimental clásico, que decide en cada momento el orden en que ha de recibir el control cada una de las instrucciones del programa.

Una forma típica de programación lógica utiliza reglas de producción para escribir las instrucciones. Una regla de producción es una estructura como la que sigue:

conclusión ← premisa & premisa & .. .& premisa

que se interpreta así: *se puede deducir la conclusión indicada si su cumplen todas las premisas de la regla*. Por ejemplo:

```
hijo(A,B) ← varón(A) & padre(B,A)
```

El conjunto de las reglas de producción, que vienen a corresponder a las instrucciones ejecutables de la programación procedimental, está desorganizado. En principio no tiene por qué existir ningún tipo de jerarquía entre las reglas, y una regla no debe de invocar a otra más que indirectamente, a través del motor de inferencia. Esto puede decidir que una regla sea aplicable:

- En función del comienzo de la propia regla (de su conclusión o de sus premisas)
- En función de los datos existentes
- En función del proceso de inferencia empleado, que puede ser dirigido hacia delante o hacia atrás y que depende del motor de inferencia que incorpore el sistema de programación lógica que se utilice.

El hecho de utilizar una regla para deducir un dato nuevo a partir de otros preexistentes es lo que se llama activar o desencadenar la regla. Veamos un ejemplo en PROLOG.

Problema: Partición de una lista

Dada una lista L y un elemento E, subdividir L en dos listas L1 y L2, tal que los elementos de L1 sean menores que E y los elementos de L2 mayores o iguales que E.

```
1.- partición(E,[P|L],[P|L1],L2) ← E =< P,|,partición (E,L,L1,L2)
2.- partición (E,[P|L],L1,[P|L2]) ← partición (E,L,L1,L2)
3.- partición (E,[],[],[])
```

7.4.8. Programación orientada a objetos.

En la década de los ochenta surge un nuevo conjunto de técnicas que recibe el nombre de programación orientada a objetos. A partir de ahora son los datos los que constituyen le jerarquía básica, un dato se puede ligar a otro a través de una relación de parentesco, o de cualquier otro tipo. Esto da lugar a la aparición de una red (árbol o grafo) semejante a la que suele formarse en las aplicaciones procedimentales con los programas.

También en la programación orientada a objetos existen programas, pero éstos están desorganizados y se convierten en meros apéndices de los datos, como en la programación procedimental son los datos meros apéndices de los programas. Es posible disponer de programas globales, accesibles por todos los objetos (datos) que forman la jerarquía, y de programas locales, que solo son accesibles para ciertos objetos y para alguno de sus descendientes.

Por tanto, la programación orientada a objetos viene a ser una forma de programar opuesta a la programación procedimental, en el sentido de que los datos y los programas desempeñan papeles diametralmente opuestos. En la programación clásica, los programas están organizados, mientras los datos no lo están. En la programación orientada a objetos, son los datos los que forman una organización, pero no ocurre lo mismo con los programas. A su vez ambas formas de programar se oponen a la programación lógica, donde ni los datos, ni los programas tienen por que estar organizados.

Veamos un ejemplo sencillo en *Smalltalk* (lenguaje puro de programación orientado a objetos, ya que no permite la instrucción CALL).

Problema: Calcular el perímetro de un polígono

Supongamos que la clase **Poligono_regular** tiene dos atributos llamados, respectivamente, **lado** (su valor es la longitud del lado del polígono) y **nlados** (su valor es el número de lados del polígono)

```
perímetro aPoligono
^((aPoligono nlados)*(aPoligono lado))
```

La primera línea del listado indica que dicho método actúa sobre casos particulares descendientes de la clase **Polígono** (indicada por el término **aPoligono**).

7.5. Ingeniería del software.

Actualmente se están produciéndose cambios de gran alcance en la forma en que se desarrolla el software de los computadores. Entre las causas para estos cambios están:

- El coste de los desarrollos software; la complejidad y tamaño de los productos software; la creciente dependencia de muchas organizaciones de sus sistemas informáticos y la insatisfacción respecto al grado de adaptación del software desarrollado para sus equipos informáticos, y el avance a los computadores de la quinta generación, que tendrán unas características hardware y software muy diferentes a los actuales.

- La ingeniería del software es la profesión que probablemente reemplazará a la de programador y a la de analista de sistemas en un futuro no muy lejano. Las personas dedicadas a esta profesión necesitarán dominar las técnicas de programación, poseer unos amplios conocimientos de matemáticas y lógica formal, entender en detalle el funcionamiento de los sistemas informáticos, y poseer conocimientos de economía y gestión.

7.5.1. Objetivos de la ingeniería del software.

La ingeniería de software se ocupa del ciclo completo de vida de un producto software: diseño, desarrollo, comprobaciones, uso y mantenimiento.

La ingeniería del software es una tarea de equipo. Cuando comienza un proyecto de desarrollo se constituye una serie de equipos con una estructura paralela a la estructura del programa en sí, se establece un calendario para el proyecto y se asignan costes a cada una de las partes y etapas del proyecto. Cada equipo tiene un responsable, cuya tarea es la de comprobar que el software desarrollado por ese equipo sea correcto, esté estructurado con propiedad, disponga de las interfaces apropiados con el software desarrollado por otros equipos y se desarrolle de acuerdo a las previsiones de tiempo y coste. Esta es una tarea muy difícil, que requiere de unos amplios conocimientos tanto técnicos como de gestión.

7.6. Cómo se deben estructurar los programas.

Actualmente está claro que la única forma de alcanzar los requisitos de corrección, prestaciones y fiabilidad del software es utilizando un diseño muy cuidadoso en la estructura de un programa. Un programa bien estructurado debe satisfacer (independientemente del lenguaje en que esté escrito) las siguientes condiciones:

- El programa debe disponer de una estructura general clara en términos de módulos, cada uno para una tarea específica. Los módulos se implementan en forma de funciones, procedimientos o segmentos, dependiendo del lenguaje de programación empleado.

- Debe de existir una interface claramente definida entre los distintos módulos. Esto es particularmente importante cuando los módulos son el trabajo de distintos grupos o personas.

- Cada módulo debe ser una combinación sencilla de construcciones elementales de un lenguaje de programación. Los módulos deben ser fáciles de leer por una persona que no sea el programador de ese módulo.

- Tiene que existir una fuerte correspondencia entre la estructura de módulos y los datos correspondientes.