

Lenguajes de programación

Introducción

Ya sabemos que un lenguaje de programación es una forma de representar un algoritmo de tal forma que es comprensible tanto para el humano como para el ordenador; con anterioridad se han mostrado algunos ejemplos de algoritmos escritos en el lenguaje FORTRAN (que es el que se utilizará durante el curso), sin embargo, no se han dado detalles sobre la forma en que un ordenador manipula un algoritmo en un lenguaje de programación ni las características de los distintos lenguajes existentes.

Antes de explicar todos esos detalles y presentar de una forma más clara las características fundamentales del lenguaje FORTRAN conviene entender, de manera muy básica, la forma en que funcionan los ordenadores electrónicos, cómo representan la información y cómo representan y ejecutan las instrucciones de un algoritmo.

¿Cómo funciona un ordenador electrónico?

En el capítulo anterior se describió de manera muy somera la historia de las máquinas algorítmicas. Vimos cómo durante los siglos XVI, XVII y XVIII hubo varios diseños (e implementaciones) de mecanismos que permitían realizar operaciones aritméticas; tales mecanismos constituyeron un primer paso hacia máquinas multipropósito, como los dispositivos de Babbage que aún eran puramente mecánicos, no electrónicos.

A fines del siglo XIX se desarrollaron máquinas parcialmente inspiradas por las ideas de Babbage pero que añadían elementos para introducir información (las famosas tarjetas perforadas) y durante el siglo XX se impuso la utilización de elementos electromecánicos (relés) y, finalmente, electrónicos (válvulas, transistores y chips) en la construcción de tales máquinas.

Durante la Segunda Guerra Mundial ambos bandos dedicaron importantes esfuerzos al desarrollo de máquinas electrónicas que permitieran llevar a cabo cálculos más veloces y decodificar mensajes cifrados; en 1944 John von Neumann, que participaba en el Proyecto Manhattan, percibiendo la importancia que tendrían las computadoras en el desarrollo de armas nucleares escribió un artículo en el que describía una arquitectura, inspirada por las ideas de Babbage, que perdura aún hoy.

Arquitectura Von Neumann

La arquitectura Von Neumann divide el ordenador en cuatro partes principales:

1. Unidad Aritmética (UA).
2. Unidad de Control (UC).
3. Memoria (M).
4. Dispositivos de entrada/salida (E/S).

La Unidad Aritmética es la encargada de realizar las operaciones aritméticas básicas y, quizás, funciones más complejas como raíces, logaritmos y funciones trigonométricas.

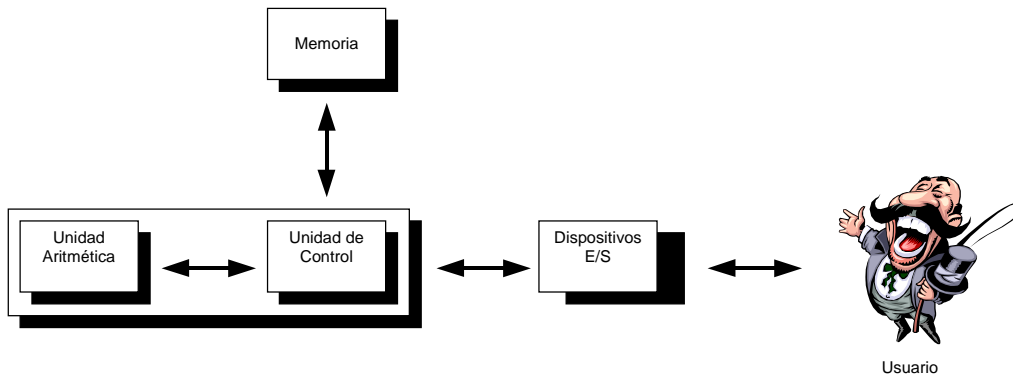
La Unidad de Control controla la ejecución de las operaciones de forma apropiada y dirige el funcionamiento del resto de unidades de tal forma que el trabajo conjunto de todas conduzca a la consecución de las tareas específicas programadas en el sistema. La Unidad de Control realiza las siguientes operaciones:

1. Recupera de memoria la siguiente instrucción a ejecutar.
2. Decodifica la instrucción y determina las acciones que debe llevar a cabo.
3. Envía órdenes a la memoria para recuperar o almacenar datos, a la UA para llevar a cabo operaciones y a los dispositivos de E/S para recibir o enviar datos al usuario.

La Memoria almacena tanto datos numéricos como instrucciones (también codificadas de forma numérica); la Memoria está dividida en celdas, cada una con una dirección única que permite el acceso a su contenido (datos o instrucciones).

Por último, los dispositivos de entrada/salida facilitan la interacción de los usuarios con la máquina.

Un aspecto interesante de esta descripción es que en ningún momento se hace referencia a cuestiones de índole tecnológica; una máquina electrónica puede implementar una arquitectura Von Neumann de la misma forma que un dispositivo mecánico o una persona (Unidad de Control) con lápiz y papel (memoria) y una calculadora (Unidad Aritmética).



Componentes y relación entre los mismos en la arquitectura Von Neumann

El código binario

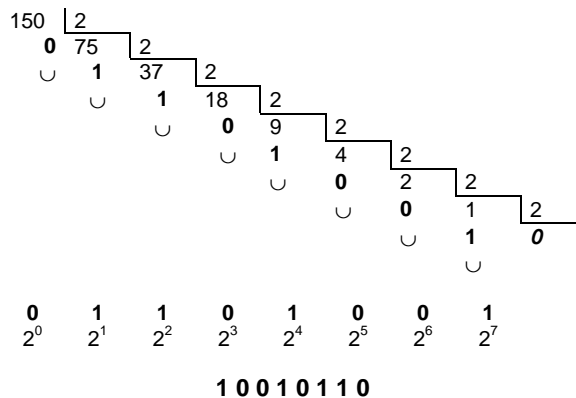
Los ordenadores utilizan internamente el código binario (base 2) puesto que es el sistema más fácilmente implementable con la actual tecnología electrónica; así, la mínima información que puede manipular un ordenador electrónico es un dígito binario, también llamado bit: 0 y 1. Los bits pueden agruparse formando unidades superiores:

- $2^3=8$ bits forman un byte, octeto o carácter (más adelante veremos el por qué de este nombre).
- $2^{10} = 1024$ bytes = 2^{13} bits forman un kilobyte o K. Un kilobyte permite almacenar, aproximadamente, 10 mensajes SMS (mensajes de teléfonos móviles).
- $2^{10} = 1024$ kilobytes = 2^{20} bytes = 2^{23} bits forman un megabyte o mega. Un megabyte es aproximadamente el tamaño de un disquete (un disquete dispone de 1.44 MB de espacio).
- $2^{10} = 1024$ megabytes = 2^{30} kilobytes = 2^{30} bytes = 2^{33} bits forman un gigabyte o giga. Un gigabyte puede almacenar una biblioteca de 1000 volúmenes o 150 minutos de audio. El código genético de un ser humano ocupa apenas 3 gigabytes.
- $2^{10} = 1024$ gigabytes = 2^{40} megabytes = 2^{40} kilobytes = 2^{40} bytes = 2^{43} bits forman un terabyte o tera. Un terabyte puede almacenar más de 100 días de audio.

El sistema binario permite representar cualquier número natural (con tal de disponer de suficientes dígitos); por ejemplo el número binario 10010110 se corresponde al número 150 decimal:

$$10010110 = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 128 + 0 + 0 + 16 + 0 + 4 + 2 + 0 = 150$$

Para convertir un número decimal a binario basta con realizar una serie de divisiones en cascada como se muestra en el ejemplo que sigue:



Representación de datos en código binario

A la vista de lo anterior podría parecer que un ordenador sólo puede representar números naturales; sin embargo, lo cierto es que cualquier tipo de dato puede ser codificado de una forma u otra. Cuando hablamos de la notación algorítmica mencionamos los siguientes: *caracter*, *logico*, *entero* y *real*.

Representación de caracteres

Para representar caracteres en un sistema binario debe desarrollarse un código que todos los usuarios tienen que conocer, dicho código establece lo siguiente:

- Los caracteres de los que dispondrá el código.
- El número asociado a cada carácter (dicho número se podrá representar, obviamente, en binario).

De forma tradicional los códigos de caracteres codifican cada elemento utilizando un byte (de ahí el sobrenombre de carácter); al disponer de 8 bits por carácter es posible representar 256 caracteres. Los caracteres a representar se agrupan en cinco categorías:

- Caracteres alfabéticos: letras mayúsculas y minúsculas del alfabeto latino.
- Caracteres numéricos: los diez dígitos decimales.
- Caracteres especiales: signos de puntuación, comparación, etc.
- Caracteres de control: fin de línea, pitido, fin de página, etc.
- Caracteres expandidos: símbolos no existentes en el alfabeto latino, como vocales acentuadas, la letra ñ, etc.

El código de caracteres más comúnmente utilizado es el ASCII (*American Standard Code for Information Interchange*); los ordenadores PC, por ejemplo, emplean ASCII para la representación de caracteres.

Existen idiomas que emplean caracteres no latinos así como lenguas ideográficas que requieren muchísimos más símbolos; por esa razón ha surgido la codificación UNICODE que emplea 16 bits, UNICODE permite representar más de 65.000 símbolos con lo cual se pueden satisfacer las necesidades de la mayor parte de idiomas incluyendo los asiáticos.

Representación de valores lógicos

Como sabemos, los valores lógicos son dos: “verdadero” y “falso”; para codificarlos bastaría, en teoría, con emplear un solo bit asignando de manera arbitraria el 1 y el 0 a cada uno de esos valores.

Sin embargo, por cuestiones de eficiencia, los ordenadores no manipulan bits individuales sino bytes por lo que la representación de los valores lógicos emplea siempre un byte. Dependiendo de la máquina, dicho byte puede tener todos los bits a 1 o a 0, o emplear un bit determinado para la representación del valor lógico.

Representación de números enteros

Los números enteros pueden tener o no tener signo; en caso de que se trate de enteros sin signo (naturales) la representación es muy sencilla pues basta con codificar el número directamente en base 2. Sin embargo, no se puede utilizar un número arbitrario de bits para representar un valor, sino que es necesario especificar la “longitud” de los mismos.

Generalmente los ordenadores utilizan enteros sin signo de 2 bytes (16 bits) o enteros cortos y enteros sin signo de 4 bytes (32 bits) o enteros largos. Con 16 bits es posible representar 65.536 valores con lo cual el rango de los enteros cortos sin signo es el [0 , 65.535]; en el caso de los enteros largos sin signo es [0 , 4.294.967.296].

La cuestión fundamental es que un ordenador no puede representar cualquier valor sino un valor perteneciente al rango de valores representables.

Por lo que respecta a los enteros con signo se utiliza un bit (generalmente el situado a la izquierda de la palabra) para representar el signo, la asignación de 1 o 0 a la presencia o ausencia del signo es arbitraria. El rango de enteros cortos con signo es [-32.768 , 32.767], y en los enteros largos con signo [-2.147.483.648 , 2.147.483.647].

Representación de números reales

Para codificar los números reales se utiliza el formato exponencial, es decir de la forma:

$$\text{mantisa} \times 2^{\text{exponente}}$$

Para codificar un número en este formato se utilizan una parte de los bits para la representación de la mantisa y otra parte para la representación del exponente. Los reales más habituales utilizan 4 bytes (precisión simple), 8 bytes (doble precisión) y 10 bytes (precisión extendida).

Como en el caso de los enteros habrá un rango de valores pero, además, habrá una limitación en la precisión debido a que el exponente también se deberá limitar a un rango específico; por esa razón existe un error de representación puesto que habrá infinitos números que no se pueden representar de forma exacta en un ordenador:

- Números con infinitas cifras decimales (como los irracionales).
- Números con más cifras decimales que las representables.
- Números con demasiadas cifras significativas (números excesivamente grandes o excesivamente pequeños).

Por ejemplo, supongamos que queremos representar un número real empleando 4 bytes (32 bits), asignaremos 1 bit para el signo de la mantisa, 23 bits para la mantisa, 1 bit para el signo del exponente y 7 bits para el exponente.

De esta forma la mantisa se moverá en el rango [-8.388.608 , 8.388.607] y el exponente en el rango [-128, 127]; de esta forma tendríamos que:

- $8.388.607 \times 2^{127} = 1,43 \text{ E}+45$ sería el máximo número real positivo.
- $-8.388.608 \times 2^{127} = -1,43 \text{ E}+45$ sería el máximo número real negativo.
- $8.388.607 \times 2^{-128} = 2,47 \text{ E}-32$ sería el mínimo número real positivo.
- $-8.388.608 \times 2^{-128} = -2,47 \text{ E}-32$ sería el mínimo número real negativo.

A la vista de los resultados está claro que, aunque la precisión y el rango sea limitado, es posible representar números reales en formato binario con una precisión más que aceptable para cualquier aplicación; después de todo, en caso de necesitarse más precisión tan sólo sería necesario utilizar más bytes para cada valor real.

En resumen, aunque un ordenador electrónico sólo puede manipular números en base 2 es posible representar cualquier tipo de dato:

- Los caracteres se representan mediante la definición de un conjunto de símbolos para cada uno de los cuales se asocia un número natural (representable en binario).
- Los valores lógicos se representan de forma inmediata asociando a los valores “verdadero” y “falso” un valor binario arbitrario.
- Los enteros sin signo se representan de forma directa y para los enteros con signo se emplea un bit para indicar el signo del entero.
- Los reales se dividen en mantisa y exponente representando ambas partes como enteros con signo.

Esta forma de representar la información supone la existencia de límites: conjuntos de caracteres limitados, rangos de enteros limitados, rangos de reales limitados y con precisión limitada; sin embargo, los rangos y límites establecidos para cada ordenador son siempre adecuados de cara a las tareas para las que vaya a ser destinado.

Lenguajes de programación

El lenguaje máquina

Hasta el momento se ha visto la forma en que es posible representar de forma adecuada datos en un ordenador; dichos datos se almacenarán en la memoria y se trabajará con ellos en la Unidad Aritmética. Sabemos que la Unidad de Control también recupera instrucciones de la memoria y realiza acciones más sencillas según indique cada instrucción.

Aunque no entraremos en detalles es necesario que el alumno sepa que la Unidad de Control “entiende” un conjunto de instrucciones o léxico limitado; dichas instrucciones irán codificadas en binario e indicarán, además de la acción a realizar (por ejemplo, “sumar”) los datos con los que se va a trabajar (por ejemplo, direcciones de memoria donde hay datos almacenados).

Este conjunto de instrucciones codificadas en binario comprensibles por la unidad de control se conoce con el nombre de “lenguaje máquina”; se trata del lenguaje de programación más básico que existe y es el único que entiende un ordenador.

Resulta extraordinariamente tedioso programar directamente en código máquina, como ejemplo se muestran dos instrucciones en un código máquina (el lenguaje máquina depende del ordenador, no hay uno único):

```
0000 0000 0010 0000 0000 0000 0010 0000
1110 0010 0010 0001 0000 0000 0010 0000
```

La primera instrucción permite sumar dos números enteros y almacenar el resultado en una tercera posición mientras que la segunda permite restar dos números reales y almacenar el resultado en una tercera posición. Está claro, ¿verdad!?

En los primeros tiempos del desarrollo de los ordenadores era necesario programarlos directamente de esta forma, sin embargo, eran máquinas extraordinariamente limitadas, con muy pocas instrucciones por lo que aún era posible; en la actualidad esto es completamente irrealizable por lo que es necesario utilizar lenguajes más fácilmente comprensibles para los humanos que deben ser traducidos a código máquina para su ejecución.

El lenguaje ensamblador

Los lenguajes ensambladores son en esencia una versión simbólica de los lenguajes máquina; por cada instrucción de la máquina o cada elemento capaz de almacenar datos se crea un símbolo que puede utilizar el programador; así, las instrucciones anteriores se escribirían como:

```
add.i c a b
sub.f c a b
```

Aunque aún es bastante críptico resulta más sencillo programar en un lenguaje de este tipo que en código máquina. Los ensambladores fueron desarrollados de forma muy temprana y recibieron este nombre porque las instrucciones básicas del lenguaje ensamblador eran en realidad pequeños programas escritos directamente en código máquina; así cuando un programador debía escribir un nuevo programa con ese lenguaje en realidad estaba “ensamblando” código máquina reutilizable.

Una característica tanto de los lenguajes ensambladores como del código máquina es que son totalmente dependientes del ordenador: un programa en código máquina (o en ensamblador) sólo funciona en un tipo de ordenador y no en otro; la ventaja es que al estar perfectamente adaptados a una máquina concreta son programas muy rápidos que permiten que el programador aproveche todas y cada una de las características del ordenador.

Lenguajes de alto nivel

Aunque los lenguajes ensambladores supusieron una mejora respecto a la programación directamente en código máquina seguían siendo engorrosos, excesivamente alejados de la forma de pensar humana y específicos de cada tipo de ordenador por lo que era muy difícil, por no decir imposible, transportar un algoritmo de un ordenador a otro.

Para solucionar estos inconvenientes se desarrollaron los lenguajes de alto nivel, este tipo de lenguajes proporcionan características más avanzadas que los lenguajes ensambladores como estructuras de control, estructuras de datos, etc.

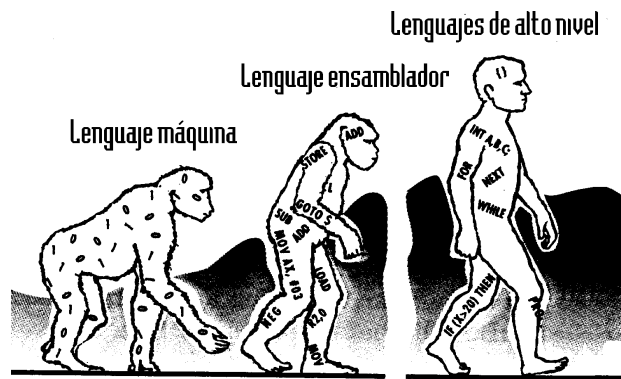
Los lenguajes de alto nivel son independientes de la máquina y, por tanto, portables; es decir, un algoritmo escrito en un lenguaje de programación de alto nivel puede utilizarse en ordenadores diferentes. Esto es posible porque los lenguajes de alto nivel son traducidos a lenguaje máquina por un tipo de programa especial denominado "compilador", un compilador toma como entrada un algoritmo escrito en un lenguaje de alto nivel y lo convierte a instrucciones inteligibles por el ordenador; los compiladores deben estar adaptados a cada tipo de ordenador pues deben generar código máquina específico para el mismo.

Las anteriores instrucciones podrían representarse en un lenguaje de alto nivel (como FORTRAN) de la forma siguiente:

$$c = a + b$$

$$c = a - b$$

FORTRAN, Pascal, C, C++ o Java son ejemplos de lenguajes de alto nivel; todos ellos comparten muchas similitudes entre sí por lo que una vez aprendido un lenguaje de programación de alto nivel es bastante sencillo aprender otros nuevos.



Metáfora sobre los lenguajes de programación

El lenguaje FORTRAN

Historia del lenguaje

FORTRAN es un acrónimo de *FOR*mula *TRAN*slator (Traductor de Fórmulas). Se trata del más antiguo lenguaje de alto nivel; fue desarrollado por John Backus para IBM a finales de los años 50. FORTRAN fue diseñado para su uso por matemáticos, ingenieros y científicos en general y sigue siendo de gran importancia en estos campos.

A lo largo de estos 40 años han surgido varias versiones de FORTRAN, con cada versión se ha tratado de actualizar el lenguaje a las tendencias más modernas en lenguajes de programación manteniendo la esencia original del lenguaje; la evolución de FORTRAN se resume a continuación:

- FORTRAN (I): El primer compilador de FORTRAN (y con éste el lenguaje) fue desarrollado por un equipo de IBM liderado por John Backus entre 1954 y 1957; esta primera versión tuvo un enorme éxito pues se trataba del primer lenguaje de programación de alto nivel en la historia. El compilador fue distribuido con todos los modelos de ordenador 704 de IBM logrando una difusión extraordinaria.
- FORTRAN II (1958), FORTRAN III (1958) y FORTRAN IV (1961) fueron mejorando ciertos aspectos del lenguaje y de los compiladores.
- FORTRAN 66: En 1966 un comité de expertos desarrolló un estándar del lenguaje FORTRAN, se trató de un paso muy importante puesto que el hecho de disponer de una especificación estandarizada supone que múltiples fabricantes pueden proporcionar compiladores que funcionarán de manera análoga en cualquier ordenador; FORTRAN 66 supuso un gran impulso en la popularidad del lenguaje.
- FORTRAN 77: En esta versión se añadieron características que hicieron de FORTRAN un lenguaje mucho más estructurado.
- FORTRAN 90 y 95: Se añaden nuevas estructuras de control y tipos abstractos de datos.

De todas las versiones de FORTRAN las más comúnmente utilizadas son FORTRAN 77 y FORTRAN 90; en este curso se utilizará FORTRAN 90 aunque no se usarán todas las características que proporciona el lenguaje.

Introducción de la notación FORTRAN

Como en el caso de la notación algorítmica, el lenguaje FORTRAN se presentará de forma incremental; sin embargo, es necesario proporcionar una breve introducción de la notación del lenguaje, esta introducción tocará los mismos puntos que se vieron en el caso de la notación algorítmica.

- **Variables:** Los nombres de variables en FORTRAN tienen las mismas características que en nuestra notación algorítmica: se forman con caracteres alfanuméricos (esto es, letras y números) incluyendo el carácter de subrayado y excluyendo los blancos y caracteres no latinos (como la 'ñ' y las vocales acentuadas).

Como ya sabemos, no pueden empezar por número, sólo por letra. Además, FORTRAN es insensible a mayúsculas.

En resumen,

- Ejemplos de identificadores válidos: `v`, `aceleracion`, `K`, `v1`, `b_n`, `Pot`, ...
- Ejemplos de identificadores no válidos: `1n` (empieza por número), `año` (incluye un carácter no válido, la 'ñ'), `aceleración` (incluye un carácter no válido, la 'ó'), `p v` (incluye un espacio en blanco), ...
- Ejemplos de identificadores equivalentes en FORTRAN: `v` y `V`; `anno`, `Anno`, `ANNO` y `AnnO`; `Pot`, `pot` y `poT`, ...
- **Tipos de datos:** En la notación algorítmica existían cuatro tipos de datos: `entero`, `real`, `logico` y `character`. En FORTRAN existen tipos equivalentes denominados: `integer`, `real`, `logical` y `character`. A continuación se mostrarán las similitudes y diferencias entre los tipos FORTRAN y los tipos de nuestra notación.
- Los tipos `entero` e `integer` son totalmente equivalentes; la única diferencia importante radica en la forma de realizar las operaciones división entera (`div`) y módulo/resto (`mod`). En el caso de la notación algorítmica se aplicarían como sigue:

```
division ← dividendo div divisor
modulo ← dividendo mod divisor
```

Mientras que en FORTRAN se haría de esta forma:

```
division = dividendo / divisor
modulo = mod (dividendo, divisor)
```

Como se puede ver, FORTRAN no dispone de un operador específico para la división entera, emplea el operador `/`; este hecho es fuente de múltiples confusiones puesto que el operador realiza divisiones enteras si el divisor es de tipo `integer` y reales si el divisor es de tipo `real`; así pues, **¡atención!**

Por lo que respecta al módulo no se trata de un operador sino de una función que recibe dos argumentos; más adelante se estudiarán con detenimiento las funciones.

- Los tipos `real` (de la notación) y `real` (de FORTRAN) son también equivalentes; por lo que respecta a las operaciones cambian de notación: la raíz cuadrada se corresponde con la función `sqrt` (*square root*), la potenciación se corresponde con el operador `**`, el logaritmo con la función `log`, y las funciones trigonométricas más habituales son `sin` (seno), `cos` (coseno), `tan` (tangente), `asin` (arco seno), `acos` (arco coseno) y `atan` (arco tangente).
- Los tipos `logico` y `logical` son equivalentes, con las diferencias habituales en notación; los valores admisibles son `.true.` y `.false.` (atención al punto inicial y final) y las operaciones admitidas son: `.and.` (y-lógico), `.or.` (o-lógico) y `.not.` (no-lógico). Al igual que en la notación algorítmica, los operadores de comparación de FORTRAN dan como resultado valores de tipo `logical`.
- Los tipos `character` y `character` son muy similares aunque presentan un par de diferencias importantes:
 - El operador de concatenación no es `+` sino `//`.
 - En FORTRAN hay que indicar la longitud de una variable de tipo `character`; es decir el número de caracteres máximo que admite. Por ejemplo, supongamos una variable de tipo `character` denominada `uno` y otra variable `character*7` denominada `siete`, si se intenta introducir el literal `'dificil'` en ambas variables veremos que `uno` almacena únicamente `'d'` mientras que `siete` almacena la cadena completa.

Todos los tipos admiten, además, las operaciones de comparación: los operadores de comparación disponibles son mayor (`>`), menor (`<`), mayor o igual (`>=`), menor o igual (`<=`), igual (`==`) y distinto (`!=`). Como ya se dijo, el resultado de una comparación es un valor de tipo `logical`. Obsérvese que la

igualdad es == y no =, eso es así porque el operador de asignación en FORTRAN es = y no ← lo cual ha sido una decisión bastante lamentable.

- **Literales:** De forma análoga a la notación algorítmica es posible definir literales de cualquier tipo en FORTRAN; también es posible definir constantes aunque en FORTRAN se denominan *parámetros* (un nombre desafortunado). Para definir una constante o parámetro en FORTRAN emplearemos la notación siguiente:

```
tipo, parameter :: constante = literal
```

A continuación se muestran un par de ejemplos de definición de constantes/parámetros en FORTRAN:

```
real, parameter :: pi = 3.1415926535897932384626433832795
real, parameter :: NA = 6.0221367e23
```

- **Expresiones:** En FORTRAN también se dispone de expresiones que son análogas a las expresiones en la notación algorítmica; la diferencia fundamental radica en la notación del lenguaje; a continuación se muestran algunas expresiones típicas:

```
2 * pi * r
v * t
(a>5) .and. (a<10)
'Sub' // 'cadena'
```

- **Asignación:** el operador de asignación es = (como ya se ha dicho una decisión poco afortunada) y se emplea como el operador ← de la notación algorítmica.
- **Entrada/salida:** En FORTRAN también existen operaciones de entrada/salida; las acciones se denominan `read` (leer) y `print` (escribir). Aunque pueden utilizarse de una forma muy sencilla admiten una serie de opciones más avanzadas que sus homólogas por lo que la descripción en profundidad de las mismas se hará en una lección posterior.

Por lo que respecta a la estructura de un algoritmo descrito en FORTRAN también se dejará para una lección posterior puesto que, aunque similar a nuestra notación algorítmica, presenta algunas peculiaridades.

Resumen

1. Aunque los ordenadores actuales tienen una historia de aproximadamente 50 años hay toda una serie de precedentes durante los siglos XVI, XVII y XVIII.
2. Todos los ordenadores electrónicos se basan en la arquitectura Von Neumann que data de 1944 y divide el ordenador en cuatro partes:
 - i. Unidad Aritmética (UA): realiza operaciones.
 - ii. Unidad de Control (UC): procesa instrucciones y controla la operación conjunta de todas las partes.
 - iii. Memoria (M): almacena datos e instrucciones.
 - iv. Dispositivos de entrada/salida (E/S): permiten la comunicación entre el ordenador y el usuario.
3. Los ordenadores utilizan internamente el código binario (base 2); la mínima unidad de información en este código es el bit, las agrupaciones de bits reciben los siguientes nombres: byte (8 bits), kilobyte (1024 bytes), megabyte (1024 kilobytes), gigabyte (1024 megabytes) y terabyte (1024 gigabytes).
4. Es posible representar en binario cualquier tipo de dato:
 - Enteros sin signo: directamente.
 - Enteros con signo: utilizando un bit para el signo y el resto directamente.
 - Reales: codificando mantisa y exponente como enteros con signo en binario.
 - Lógicos: utilizando uno o más bits para representar los valores “verdadero” y “falso” con ambos valores binarios (0 y 1).
 - Caracteres: seleccionando un conjunto de caracteres y asociando a cada carácter un natural cuyo valor se codifica; el código de caracteres más común es ASCII.
5. Un ordenador representa las instrucciones que “comprende” también en forma binaria, ese lenguaje se denomina código máquina; el código máquina es el lenguaje más básico y resulta muy arduo programar en él. El código máquina no es portable pues es específico de cada tipo de ordenador.
6. El ensamblador es un lenguaje más avanzado que el código máquina, cada instrucción ensamblador es una representación simbólica de una instrucción máquina; aunque son aún muy crípticos resulta más sencillo programar en ensamblador que en código máquina. El ensamblador también es específico de cada ordenador y, por tanto, tampoco es portable.
7. Los lenguajes de alto nivel están más cercanos a la forma de resolver problemas los humanos, son independientes del ordenador y, por tanto, portables. Los lenguajes de alto nivel son traducidos a código máquina específico para cada máquina por programas especiales llamados compiladores. Ejemplos de lenguajes de programación del alto nivel son FORTRAN, Pascal, C, C++ o Java.

8. FORTRAN es un acrónimo de *FORmula TRANslator*. Es el lenguaje de alto nivel más antiguo y data de finales de los 50 cuando fue diseñado para su uso por matemáticos y científicos. Las versiones más comunes del lenguaje en la actualidad son FORTRAN 77 y FORTRAN 90.
9. FORTRAN, al igual que la notación algorítmica aunque con ciertas diferencias, soporta:
 - Variables.
 - Tipos de datos: `character`, `integer`, `real` y `logical`.
 - Literales y constantes.
 - Expresiones.
 - Operación de asignación: operador `=`.
 - Operaciones de entrada/salida: `read` y `print`.