

Algoritmos: descripción y notación algorítmica

Introducción

El objetivo primordial de la asignatura es proporcionar al alumno los conocimientos que le permitan diseñar algoritmos propios para la resolución de problemas, parte de tales conocimientos versarán sobre Java para describir dichos algoritmos así como técnicas para desarrollar programas eficientes y comprensibles.

Aquellos alumnos que no tengan experiencia en el manejo de ordenadores no deben preocuparse, los contenidos de la asignatura se proporcionarán de forma gradual y dando por supuesto que los alumnos nunca han aprendido un lenguaje de programación. Aquellos alumnos que conozcan algún lenguaje tienen ciertas ventajas pero también inconvenientes al tener que luchar contra “vicios” adquiridos.

Aproximación intuitiva a los algoritmos

Como ya se ha dicho, un lenguaje de programación no es más que una forma de representar un algoritmo, asípues, ¿qué es un algoritmo? Antes de proporcionar una definición precisa del término daremos un pequeño rodeo supuesto que los algoritmos han acompañado a la humanidad desde hace mucho tiempo, con la salvedad de que la mayorvparte eran dados por supuesto o bien se les denominaba con términos diferentes. A continuación se muestran algunosvejemplos clásicos.

Algoritmo “obvio a posteriori”: Instrucciones para subir una escalera

[...] Las escaleras se suben de frente, pues hacia atrás o de costado resultan particularmente incómodas. La actitud natural consiste en mantenerse de pie, los brazos colgando sin esfuerzo, la cabeza erguida aunque no tanto que los ojos dejen de ver los peldaños inmediatamente superiores al que se pisa, y respirando lenta y regularmente. Para subir una escalera se comienza por levantar esa parte del cuerpo situada a la derecha abajo, envuelta casi siempre en cuero o gamuza, y que salvo excepciones cabe exactamente en el escalón. Puesta en el primer peldaño dicha parte, que para abreviar llamaremos pie, se recoge la parte equivalente de la izquierda (también llamada pie, pero que no ha de confundirse con el pie antes citado), y llevándola a la altura del pie, se le hace seguir hasta colocarla en el segundo peldaño, con lo cual en éste descansará el pie, y en el primero descansará el pie.

Los primeros peldaños son siempre los más difíciles, hasta adquirir la coordinación necesaria. La coincidencia de nombre entre el pie y el pie hace difícil la explicación.

Cuídese especialmente de no levantar al mismo tiempo el pie derecho y el pie izquierdo.

Llegando en esta forma al segundo peldaño, basta repetir alternadamente los movimientos hasta encontrarse con el final de la escalera. Se sale de ella fácilmente, con un ligero golpe de talón que la fija en su sitio, del que no se moverá hasta el momento del descenso

Algoritmo de “andar por casa”: Tortilla de patatas a la española (6 personas)

Ingredientes:

- 2 vasos (de los de agua) de aceite (1/2 litro)
- sal
- 8 huevos
- 1 kg de patatas

Se lavan las patatas una vez peladas, y se secan con un paño; se parten en dos a lo largo y después se cortan en láminas finitas. Se pone el aceite en la sartén a calentar y se fríen las patatas, moviéndolas de vez en cuando y echándoles un poco de sal.

Una vez fritas (más o menos doradas, según gusten), se separan y se ponen a escurrir en un colador grande. Se quita el aceite sobrante de la sartén.

Aparte se baten los huevos con tenedor y muy fuerte; se pone un poco de sal; en el mismo plato de los huevos se echan las patatas y se mueven con un tenedor.

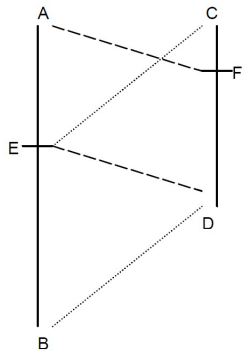
En una sartén grande (o en dos pequeñas) se ponen 3 cucharadas soperas de aceite para que sólo cubra el fondo. Cuando está caliente se vierte la mezcla de huevos y patatas.

Se mueve la sartén por el mango para que no se pegue la tortilla. Cuando se vea que está bien despegada y dorada (esto depende del gusto de cada cual), se pone una tapadera encima, se vuelca la sartén y se escurre suavemente la tortilla otra vez en la sartén. Se vuelve a mover por el mango y cuando esté cuajada (a gusto) se pasa a una fuente redonda y se sirve.

Algoritmo “infantil”: Multiplicación de números enteros

Para obtener el producto de dos números enteros utilizando lápiz y papel se debe escribir el primer factor (multiplicando) y, justo debajo y alineado a la derecha, el segundo factor (multiplicador). Se recorren todas las cifras del multiplicador de derecha a izquierda y se operan con cada una de las cifras el multiplicando, también de derecha a izquierda, escribiendo los resultados intermedios en líneas separadas; cada línea estará desplazada una posición a la izquierda respecto a la línea inmediatamente superior. Una vez se han obtenido todos los resultados intermedios se suman columna a columna obteniéndose el resultado final.

Algoritmo “clásico”: Algoritmo de Euclides



Sean AB y CD dos números cualesquiera no primos entre sí.

Es preciso hallar la medida común máxima (máximo común divisor) de AB y CD.

Si CD mide (divide) a AB, como se mide también a sí mismo, entonces CD es medida común de CD y AB. Y está claro que también es la máxima, pues ninguna mayor que CD medirá a CD.

Sin embargo, si CD no mide a AB, entonces, restándose sucesivamente el menor de los AB y CD del mayor, quedará un número que medirá al anterior. No quedará una unidad: porque en otro caso AB y CD serían primos entre sí [VII, 1], lo que contradice la hipótesis de partida.

Así pues, quedará un número que medirá al anterior. Ahora bien, CD, al medir a BE, deja EA menor que él mismo, y EA, al medir a DZ, deje ZC menor que él mismo, y medirá CZ a AE. Así pues, como CZ mide a AE, y AE mide a DZ, entonces CZ medirá también a DZ; pero se mide también a sí mismo; entonces medirá también al total CD. Como CD mide a BE entonces CZ mide a BE; y mide también a EA; por tanto medirá también al total BA; al medir también a CD; entonces CZ

mide a AB y CD. Por tanto, CZ es medida común a AB y CD.

También en esta ocasión es la máxima. Pues, si CZ no es la medida común máxima de AB y CD, un número que sea mayor que CZ medirá a los números AB, GD. Supongamos que existe y se denomina H.

Si H mide a CD y CD mide a BE, entonces H mide también a BE; pero también mide al total BA; entonces medirá también al resto AE. Pero AE mide a DZ; por tanto, H medirá a DZ y mide también al total DC; luego medirá también al resto CZ, esto es: el mayor al menor, lo cual es imposible; así pues, no medirá a los números AB y CD un número que sea mayor que CZ. Por consiguiente, CZ es la medida común máxima de AB y CD. A partir de esto queda claro que, si un número mide a dos números, medirá también a su medida común máxima.

Si se estudia con atención cada uno de los algoritmos anteriores descubriremos una serie de características interesantes que definen la naturaleza de lo que es un algoritmo:

- **Un algoritmo resuelve un problema específico:** subir una escalera, obtener una tortilla de patatas, hacer una multiplicación o determinar el máximo común divisor de dos números.
- **Un algoritmo es llevado a cabo por una entidad que trabaja en un entorno dado:** una persona cuyo universo inmediato se reduce a su propio cuerpo y una escalera; un cocinero con una sartén, huevos, patatas, aceite, sal y cebolla; o un niño con lápiz y papel.
- **Un algoritmo consta de una serie de pasos que deben llevarse a cabo siguiendo una secuencia marcada:** algunos de los pasos en uno de los algoritmos anteriores serían: dar la vuelta a la tortilla, batir los huevos, pelar las patatas o cascar los huevos; dichos pasos deben aplicarse en un orden prefijado y no de cualquier manera.
- **Un algoritmo se aplica de forma mecánica:** un algoritmo no precisa decisiones subjetivas ni creatividad en su aplicación, cualquiera con una receta adecuada para obtener tortilla de patatas logrará una tortilla de patatas. Sin embargo, sí es necesario un acto creativo para desarrollar un nuevo algoritmo.
- **Un algoritmo termina en un tiempo finito:** todos los algoritmos deben finalizar, pueden tardar más o menos tiempo en lograr un resultado pero dicho tiempo debe ser finito.

Definición de algoritmo. Etimología del término algoritmo

Definición1: Un algoritmo es una secuencia de pasos que es llevado a cabo de forma mecánica y sistemática por un actor que se desenvuelve en un entorno dado para resolver un problema determinado en un tiempo finito.

Definición2: Un algoritmo es una combinación de instrucciones combinadas de forma adecuada para resolver un determinado problema en una cantidad finita de tiempo. Cada instrucción es una indicación sencilla y no ambigua.

El término “ALGORITMO” proviene de Mahommed ibn Musa al-Khowârizmî (Mahommed, hijo de Musa, natural de Kharizm), matemático persa del siglo IX; las matemáticas le deben la introducción del sistema de numeración actual y del álgebra. En su libro *De numero indiorum* (Sobre los números hindúes) proporciona las reglas para realizar las operaciones aritméticas (con los nuevos números, por supuesto), dichas reglas se denominaron “reglas de al-Khowârizmî” y, por deformación “algoritmos”, haciéndose extensivo el término a cualquier conjunto de reglas para resolver un problema determinado.

Máquinas para aplicar algoritmos

El hombre siempre ha deseado tener herramientas que le ayudaran a efectuar cálculos precisos y rápidos; aunque los actuales ordenadores electrónicos tienen una historia relativamente corta (apenas medio siglo) son muchos los precedentes mecánicos y electromecánicos que han aparecido a lo largo de la historia.

Algunos de los hitos más interesantes serían los siguientes:

- 3000 A.C: Ábaco de arena (Oriente medio).
- 1274: Ramón Llull ideó dispositivos mecánicos para realizar demostraciones lógicas.
- 1500: Leonardo DaVinci diseñó máquinas de calcular mecánicas.
- 1624: Wilhelm Schickard desarrolló una calculadora con cuatro operaciones básicas.
- 1642: Blaise Pascal creó la “Máquina Aritmética” que permitía realizar sumas.
- 1671: Gottfried Leibniz construyó una calculadora mecánica que permitía sumar, restar, multiplicar, dividir y calcular raíces cuadradas.
- 1830: Charles Babbage diseñó y desarrolló la primera computadora de uso general, la “Máquina Analítica”; dicha máquina funcionaría impulsada mediante vapor y, aunque nunca llegó a ser construida. Ada Lovelace (hija del poeta Lord Byron y colaboradora de Babbage) se convirtió en la primera “programadora” del mundo al escribir programas para dicha máquina; el primero de ellos permitiría calcular la serie de Bernoulli.

A finales del siglo XIX se comenzaron a aplicar nuevas técnicas como las tarjetas perforadas (inicialmente utilizadas para controlar telares) y la electricidad en el desarrollo de máquinas precursoras de los ordenadores; durante la segunda guerra mundial se dio un impulso definitivo hacia el desarrollo de los mismos y se estableció una arquitectura que aún hoy perdura; la revolución iniciada por el transistor y, posteriormente, por el chip condujeron a la actual explosión informática.

Elementos participantes en la realización de un algoritmo

Al mencionar las características fundamentales de los algoritmos decíamos que un algoritmo es llevado a cabo por una entidad que trabaja en un entorno dado. En los ejemplos anteriores la entidad era un ser humano y el entorno era el propio entorno físico que le rodeaba; sin embargo, en esta asignatura los algoritmos que nos interesan no van a ser aplicados por un individuo (no sería práctico ni eficiente) sino por un instrumento mecánico. Así, de cara a una descripción “aséptica” de los algoritmos se emplearán los siguientes términos para referirnos a los elementos que participan en un algoritmo:

- **Procesador:** un procesador es un ente que es capaz de entender los pasos (acciones) que componen el algoritmo y llevarlos a cabo (ejecutarlos).
- **Entorno:** es el conjunto de materiales necesarios para la ejecución del algoritmo.
- **Acción:** es un suceso, llevado a cabo por el procesador, que modifica el entorno.

Al aplicar estos conceptos al algoritmo de la tortilla de patatas podemos ver que el procesador es el cocinero, el entorno son los ingredientes (huevos, patatas, cebolla) y las acciones cada uno de los actos que se aplican sobre los ingredientes (cascar, batir, pelar, trocear, salar, freír, etc).

Si, en cambio, se aplican a un algoritmo que se vaya a ejecutar en un ordenador tenemos que el procesador es el ordenador, el entorno son los datos con los que va a trabajar y las acciones serían todas aquellas operaciones que el ordenador puede realizar sobre los datos (sumar, restar, comparar, etc).

Formas de describir un algoritmo

Dado que los algoritmos permiten resolver problemas de forma mecánica, está claro que resulta muy interesante compartir dicho algoritmo de tal manera que otras personas puedan conocerlo y aplicarlo; así surge el problema de describir los algoritmos de forma tal que todas las características que los definen se mantengan invariables.

Lenguaje natural

La primera y más sencilla forma de describir un algoritmo es empleando el lenguaje natural; por ejemplo, el algoritmo para encontrar las raíces de una ecuación de segundo grado podría describirse así:

1. Definir los coeficientes de la ecuación de segundo grado: a , b y c .
2. Determinar el valor del discriminante: $b^2 - 4ac$.
3. Si el discriminante es cero sólo hay una solución: $-b/(2a)$.
4. Si el discriminante es positivo pero no cero hay dos soluciones: $(-b \pm \sqrt{\text{discr}})/(2a)$.
5. Si el discriminante es negativo no hay soluciones reales.

La ventaja fundamental es la facilidad de comprensión, cualquier persona (hispanoparlante, por supuesto) que lea dicho algoritmo podría entenderlo y aplicarlo; sin embargo, son varios los problemas que plantea describir un algoritmo de esta forma:

- **El lenguaje natural no es universal**, este algoritmo sería completamente inútil para los no hispanoparlantes.



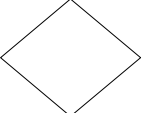

- **El lenguaje natural es ambiguo** y, por tanto, susceptible de errores.
- **El lenguaje natural es demasiado amplio**, lo que para una persona puede ser una instrucción sencilla puede no serlo para otra y desde luego no lo será para un ordenador.

Por todo ello, se han buscado nuevas formas de describir los algoritmos que, cuando menos, sean más universales, estén mejor delimitadas y no sean ambiguas; dos técnicas que logran esto son los organigramas y las notaciones en pseudocódigo.

Organigramas

Los organigramas o diagramas de flujo permiten describir los algoritmos de forma gráfica; para ello utilizan una serie de bloques que indican distintas circunstancias y flechas que muestran bajo qué condiciones se pasa de un bloque a otro.

Algunos de los símbolos son los siguientes:

 <p>Terminal Punto de comienzo o final de un programa.</p>	 <p>Entrada/Salida Información introducida para su proceso o generada como resultado.</p>	 <p>Decisión Operación que determina varios caminos alternativos a seguir.</p>	 <p>Proceso Cualquier proceso distinto de la E/S o las decisiones.</p>
--	---	---	--

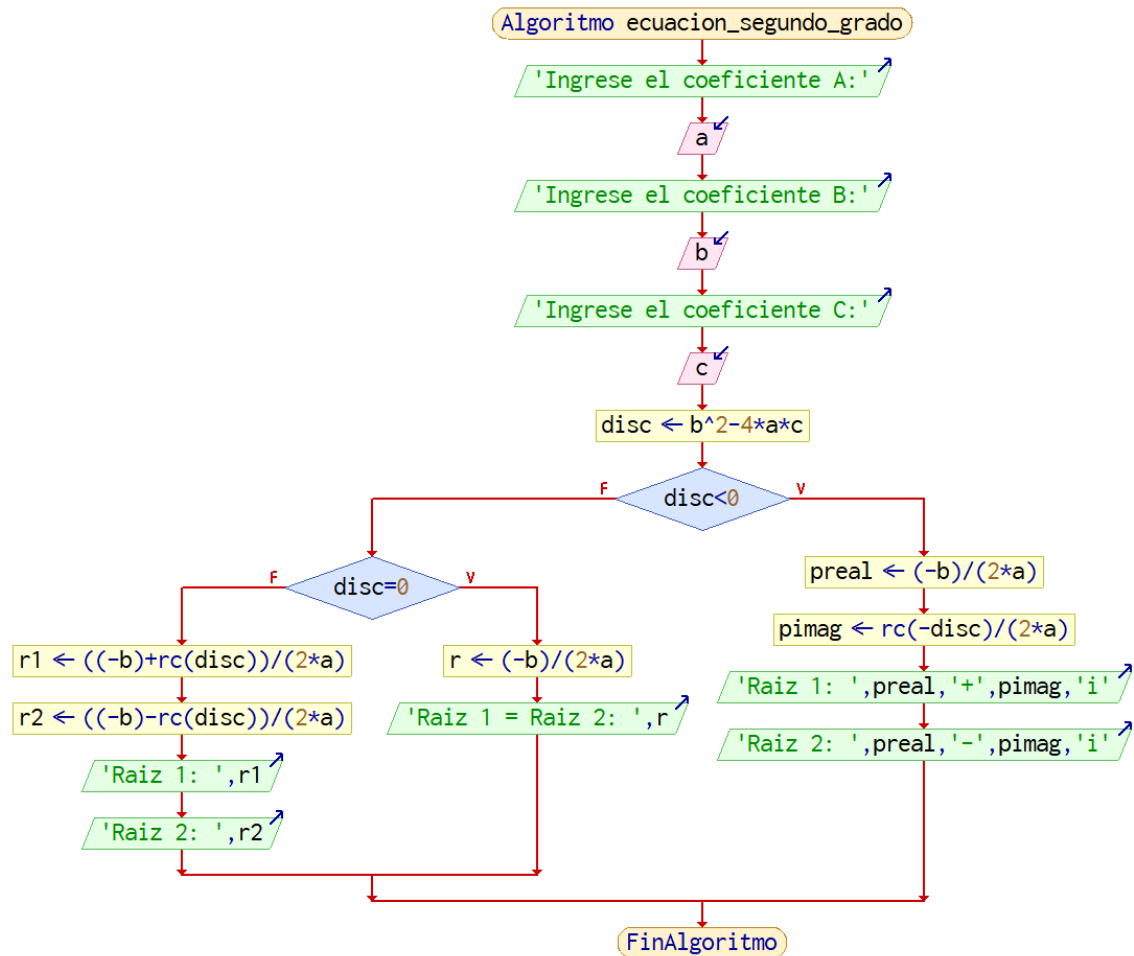
Los organigramas presentan varias ventajas frente al lenguaje natural:

- Los símbolos son universales.
- Son menos propensos a la ambigüedad.
- Por estar basados en un número pequeño de bloques y reglas para su empleo permiten delimitar mejor los algoritmos.
- Se aproximan más a la forma en que trabaja el ordenador.

Sin embargo:

- El hecho de emplear símbolos supone que una persona que desconozca los símbolos puede tener dificultades para comprender el algoritmo o no entenderlo en absoluto.
- Aunque los símbolos son universales el texto que se coloca en su interior sigue siendo lenguaje natural.
- La representación gráfica puede resultar bastante tediosa y en el caso de algoritmos complejos extremadamente confusa.
- Un ordenador no es capaz de utilizar una representación visual como descripción de un algoritmo.

Actualmente, los organigramas no son muy utilizados aunque para mostrar el funcionamiento de algoritmos sencillos siguen siendo resultando prácticos.



Pseudocódigo

El pseudocódigo pretende aunar en un solo tipo de representación las ventajas del lenguaje natural y de los organigramas sin ninguno de sus problemas; por tanto, el pseudocódigo:

- Es fácilmente comprensible para una persona que lo vea por vez primera.
- Está bien delimitado.
- Elimina las ambigüedades del lenguaje natural.
- Se representa de una forma compacta.

De esta forma, el pseudocódigo se suele ver como un subconjunto de un lenguaje natural que proporciona un número limitado de operaciones para la construcción de algoritmos; la única finalidad del pseudocódigo (como de los organigramas) es la comunicación entre seres humanos. A continuación se muestra un ejemplo de algoritmo descrito mediante un pseudocódigo:

// calcula las raices de una ecuacion de segundo grado

```

Proceso ecuacion_segundo_grado
  // cargar datos
  Escribir "Ingrese el coeficiente A:"

```

```

Leer a
Escribir "Ingrese el coeficiente B:"
Leer b
Escribir "Ingrese el coeficiente C:"
Leer c

// determinar si son reales o imaginarias
disc <- b^2-4*a*c
Si disc<0 Entonces
    // si son imaginarias
    preal<- (-b)/(2*a)
    pimag<- rc(-disc)/(2*a)
    Escribir "Raiz 1: ",preal,"+",pimag,"i"
    Escribir "Raiz 2: ",preal,"-",pimag,"i"
Sino
    Si disc=0 Entonces // ver si son iguales o distintas
        r <- (-b)/(2*a)
        Escribir "Raiz 1 = Raiz 2: ",r
    Sino
        r1 <- ((-b)+rc(disc))/(2*a)
        r2 <- ((-b)-rc(disc))/(2*a)
        Escribir "Raiz 1: ",r1
        Escribir "Raiz 2: ",r2
    FinSi
FinSi

FinProceso

```

Veamos algunos ejemplos usando Pseint en <http://pseint.sourceforge.net/index.php?page=ejemplos.php>. Hay que mirar las bases del pseudocódigo para realizar nuevos ejercicios <http://pseint.sourceforge.net/index.php?page=pseudocodigo.php>

Diagramas de Cajas o de Chapin o Nassi-Shneiderman

Los diagramas Nassi-Schneiderman son similares a un diagrama de flujo pero omite las flechas que representan el flujo de control.

En lugar de las flechas se ocupan rectángulos colocados en forma sucesiva, dentro de cada rectángulo se pone la instrucción en forma de pseudocódigo u otro conjunto de rectángulos o símbolos para representar una condición o ciclo.



Alternativa simple

Alternativa doble

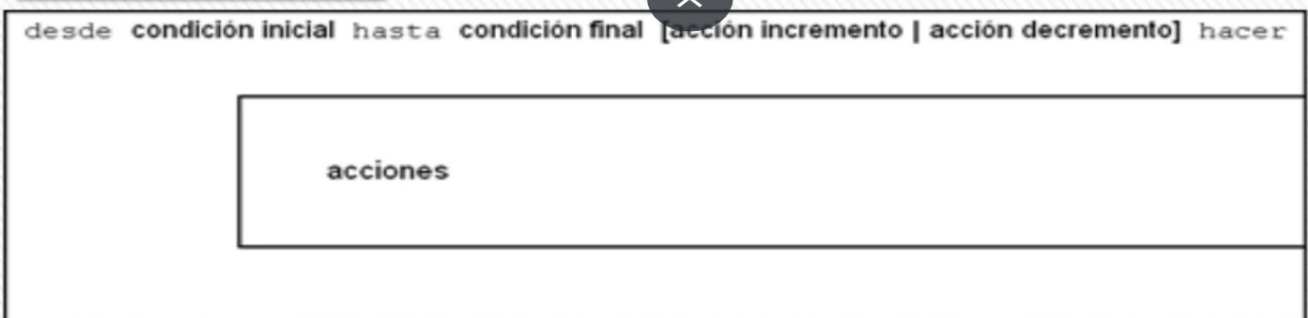
Alternativa múltiple

condición				
caso 1	caso 2	caso 3	caso 4	
acciones 1	acciones 2	acciones 3	acciones 4	

Ciclo Repetir



×



		condición	
SI		NO	
acciones		acciones	
		condición	
		SI	NO
		acciones	
		condición	
SI		NO	acciones

Diagramas Warnier

Este diagrama permite la descripción de los procedimientos y los datos de la organización. Se desarrollaron inicialmente en Francia por Jean-Dominique Warnier y Kenneth Orr.

Estos diagramas permiten diseñar un programa identificando primero su salida y entonces trabajar hacia atrás para determinar las combinaciones de pasos y de entradas para producirlas.

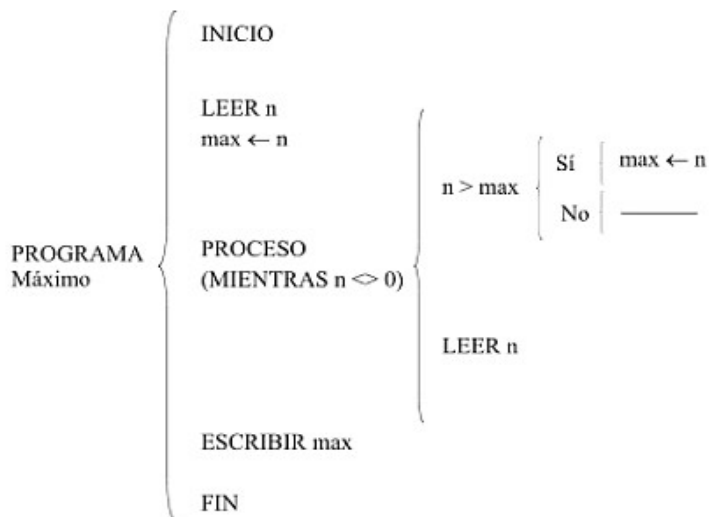
La ventaja de estos diagramas consiste en su apariencia simple y sencilla de entender. En ellos se agrupan los procesos y datos de un nivel a otro.

Para hacer estos diagramas el diseñador trabaja de atrás hacia adelante, comienza con las salidas del sistema y en un papel escribe de derecha a izquierda. Primero se ponen salidas o resultados de los procesos.

En el siguiente nivel se escribe una inclusión con una llave los pasos necesarios para producir la salida. Las llaves agrupan los procesos requeridos para producir el resultado del siguiente nivel.

Elementos:

Elemento	Descripción
{ Conjunto	Delimita un bloque de información jerarquizada que pueden ser datos o acciones, de derecha a izquierda denota los niveles de abstracción, de arriba abajo, muestra la secuencia y las relaciones lógicas entre las funciones.
(0,1) Condicionalidad	La información entre los paréntesis (variable o cantidad) indica el número de veces que ocurrirá el conjunto. Si se coloca una letra C indica que un ciclo se termina cuando la condición se cumpla.
+ Secuencia de acciones mutuamente excluyentes	Indica que una acción o grupo de acciones son mutuamente excluyentes dadas las condiciones que se establezcan.



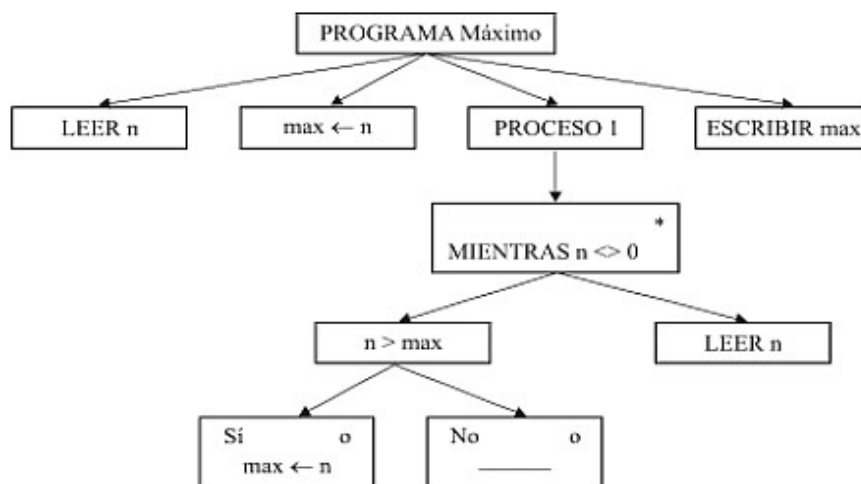
Método Jackson:

Se trata de un método de representación de programa en forma de árbol denominado diagrama arborescente de Jackson, un diagrama de Jackson consta de:

- Definición detallada de los datos de entrada y salida incluyendo los archivos lógicos utilizados.
- Representación del proceso o algoritmo.

La lectura del árbol se realiza en preorden raiz, subárbol izquierdo, subárbol derecho.

El arbol consta de nodos dibujados como rectángulos que indican la operación a realizar.



Método Bertini

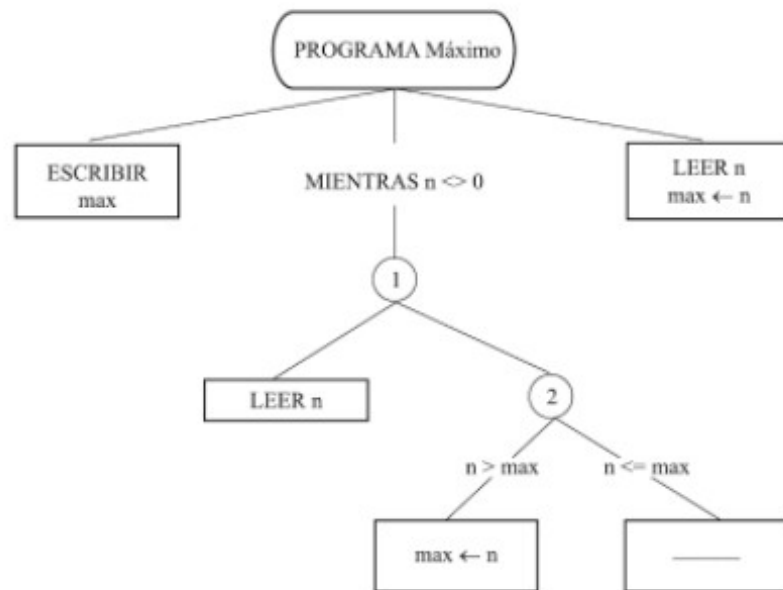
Al igual que Jackson, la representación de programas es en forma de árbol denominado diagrama arborescente de Bertini.

Un diagrama de Bertini consta de:

- Definición detallada de los datos de entrada y salida.
- Representación del proceso o algoritmo.

En este caso el árbol se recorre en orden inverso Raiz, subárbol derecho y subárbol izquierdo.

Los nodos se representan como círculos de los que cuelgan las estructuras.

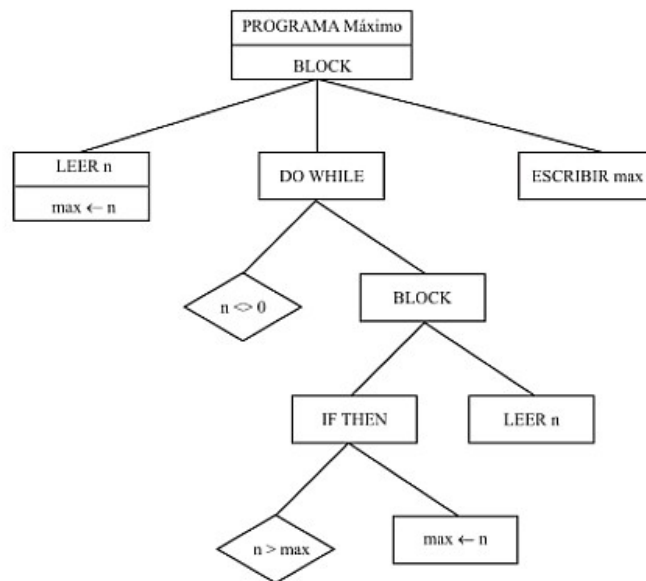


Método Tabourier

Se representa en forma de árbol con recorrido en preorden y utilizando rectángulos y rombos (condicionales y bucles).

El programa comienza con un rectángulo dividido horizontalmente con el nombre del programa en la parte superior y en la inferior la palabra BLOCK

- Definición detallada de los datos de entrada y salida incluyendo los archivos lógicos utilizados
- Representación del proceso o algoritmo.



Lenguajes de programación

Un lenguaje de programación comparte las mismas características que el pseudocódigo pero, además de ser comprensible para un ser humano, resulta comprensible para un ordenador; esto es, un algoritmo descrito mediante un lenguaje de programación puede ser ejecutado por una máquina. A continuación se muestra el algoritmo anterior descrito en el lenguaje de programación FORTRAN:

```

program segundoGrado
print *, 'Deme los coeficientes y resolveré una ecuación de 2º grado'
print *, '¿Cuánto vale A?'
read *, a
print *, '¿Cuánto vale B?'
read *, b
print *, '¿Cuánto vale C?'
read *, c
discr = b*b-4*a*c
if (discr==0) then
s = -b/(2*a)
print *, 'Sólo hay una solución: ', s
else
if (discr>0) then

```

```
s1 = (-b+sqrt(discr))/(2*a)
s2 = (-b-sqrt(discr))/(2*a)
print *, 'Las soluciones son: ',s1,s2
else
print *, 'No hay soluciones reales'
end if
end if
end
```

Como se puede ver, existen muchas similitudes entre el pseudocódigo y un lenguaje de programación como FORTRAN, de hecho la mayor parte de lenguajes de programación modernos resultan bastante inteligibles con unos conocimientos básicos de inglés; sin embargo, aunque ahora sea difícil de apreciar, el pseudocódigo aún es un lenguaje demasiado ambiguo y poco preciso para un ordenador.

Desarrollo e implementación de un algoritmo

A la hora de resolver un problema mediante la utilización de un ordenador no se codifica directamente un programa en un lenguaje dado; aún cuando los lenguajes de programación son inteligibles para los humanos están bastante alejados de nuestra forma de pensar y si el problema es complejo resulta muy difícil, por no decir imposible, escribir un programa en un solo paso.

Por esa razón, existen toda una serie de fases por las que es necesario pasar desde que se plantea un problema hasta que se obtiene un programa de ordenador que lo resuelve; la finalidad fundamental de esta asignatura no es convertir a los alumnos en expertos en el tema pero es necesario que conozcan unos aspectos metodológicos mínimos de cara a desarrollar su trabajo con ordenadores de una forma eficaz (y más agradable).

En una primera fase, análisis, se estudia el problema a resolver: qué datos precisan ser introducidos para obtener la solución, en qué consistirá dicha solución, qué errores puede presentar, etc. En caso de disponerse de más información relativa al problema, por ejemplo cualquier tipo de fórmula o expresión matemática también se incluiría como información de análisis. Un aspecto importante es que durante el análisis **no** se busca una forma de resolver el problema, tan sólo se trata de comprender la naturaleza del mismo; una forma práctica de enfocar el análisis es dividiéndolo, a su vez, en fases de entrada, proceso, salida y condiciones erróneas.

En una segunda fase, diseño, se busca una forma de resolver el problema, es decir, un algoritmo; para ello puede emplearse la técnica de “diseño descendente”. El diseño descendente se basa en el principio de “divide y vencerás”; este método consiste en resolver el problema mediante una aproximación con distintos niveles de abstracción, nunca se debe atacar desde el principio con un enfoque excesivamente detallado (muy próximo al ordenador).

En primer lugar se plantea el problema empleando términos del mismo problema (nivel de abstracción 1).

En segundo lugar, se descompone en varios subproblemas expresados también en términos del problema y tratando de hacerlos lo más independientes entre sí que sea posible.

Este paso se repite para cada subproblema tantas veces como sea necesario hasta llegar a una descripción del problema que emplee instrucciones sencillas que puedan ser transformadas de forma sencilla a código en un lenguaje de programación.

En la fase de diseño pueden emplearse gráficos en forma de árbol para representar los subproblemas y las relaciones entre ellos y pseudocódigo para la descripción “formal” de la forma de resolver cada subproblema.

A continuación se mostrará un ejemplo de análisis y diseño para un problema dado.

Enunciado:

Proporcionar un algoritmo que determine si un año indicado por el usuario es bisiesto.

Análisis:

Entrada: El usuario debería introducir un año, un año es un número entero positivo.

Proceso: Un año es bisiesto si es múltiplo de 4 pero no de 100, la excepción son los años múltiplos de 400.

Salida: Hay dos posibles salidas: “El año es bisiesto” y “El año no es bisiesto”.

Condiciones de error: Si el dato introducido no es válido (número negativo o cero) debería indicarse: “Dato no válido.”

Diseño

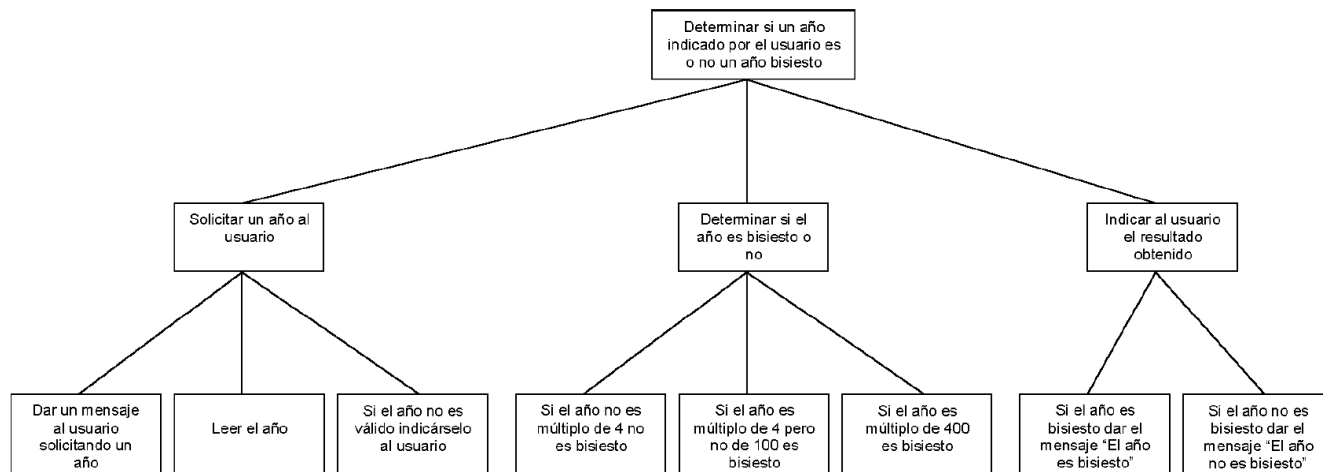
- | | |
|---------|--|
| Nivel 1 | Determinar si un año indicado por el usuario es o no un año bisiesto. |
| Nivel 2 | 2.1. Solicitar un año al usuario.
2.2. Determinar si el año es bisiesto o no.
2.3. Indicar al usuario el resultado obtenido. |
| Nivel 3 | 2.1.1. Dar un mensaje al usuario solicitando un año.
2.1.2. Leer el año.
2.1.3. Si el año no es válido indicárselo al usuario.

2.2.1. Si el año no es múltiplo de 4 no es bisiesto.
2.2.2. Si el año es múltiplo de 4 pero no de 100 es bisiesto.
2.2.3. Si el año es múltiplo de 400 es bisiesto.

2.3.1. Si el año es bisiesto dar el mensaje “El año es bisiesto”. |

2.3.2. Si el año no es bisiesto dar el mensaje “El año no es bisiesto”.

Este diagrama muestra mejor la descomposición descendente del problema planteado; cada “caja” es un problema o subproblema, las cajas “hijas” son subproblemas que componen un problema de un nivel de abstracción superior. Es importante señalar que el número de subproblemas que aparecen en un nivel varía (aunque se suele mantener en torno a 7) y que no es necesario profundizar por igual en todos los casos.



Si el problema es muy sencillo puede realizarse un organigrama que describa el algoritmo que lo resuelve, en caso contrario es preferible utilizar un pseudocódigo; es necesario decir que el pseudocódigo también se desarrolla por niveles, de hecho, por cada caja/problema anterior debería obtenerse un pseudocódigo que lo resuelva.

Problema	Pseudocódigo del algoritmo que lo resuelve
Determinar si un año indicado por el usuario es o no un año bisiesto.	Solicitar un año al usuario Determinar si el año es bisiesto o no Indicar al usuario el resultado obtenido
Solicitar un año al usuario	<pre> escribir "Por favor, deme un año" leer AÑO si AÑO ≤ 0 entonces escribir "El año no es válido" si no determinar si el año es bisiesto o no indicar al usuario el resultado obtenido fin_si </pre>
Determinar si el año es bisiesto o no	<pre> si el año es múltiplo de 4 entonces si el año es múltiplo de 400 entonces BISIESTO ← si si no si el año es múltiplo de 100 entonces BISIESTO ← no si no BISIESTO ← no </pre>

	<pre> BISIESTO ← si fin_si fin_si si no BISIESTO ← no fin_si</pre>
Indicar al usuario el resultado obtenido	<pre>si BISIESTO = si entonces escribir "El año es bisiesto" si no escribir "El año no es bisiesto" fin_si</pre>

Una vez hecho esto es posible obtener el pseudocódigo completo y detallado que resuelve el problema:

```
escribir 'Por favor, deme un año'
leer AÑO
si AÑO ≤ 0 entonces
    escribir 'El año no es válido'
si no
    si el año es múltiplo de 4 entonces
        si el año es múltiplo de 400 entonces
            BISIESTO ← si
        si no
            si el año es múltiplo de 100 entonces
                BISIESTO ← no
            si no
                BISIESTO ← si
        fin_si
    fin_si
si no
    BISIESTO ← no
fin_si

si BISIESTO = si entonces
    escribir 'El año es bisiesto'
si no
    escribir 'El año no es bisiesto'
fin_si
fin_si
```

Una vez se ha obtenido el pseudocódigo traducirlo a un lenguaje de programación es extremadamente sencillo.

Conceptos fundamentales

A lo largo de la asignatura se utilizarán en muchas ocasiones algunos términos que conviene definir claramente; por el momento, los más importantes serán los conceptos de: acción, indicador o variable, información, estado, datos y resultados, léxico, tipo de datos y operador.

- **Acción:** Como ya se dijo anteriormente, una acción es un suceso, llevado a cabo por un procesador. Una acción siempre modifica su entorno.

- **Indicador o variable:** El entorno de una acción está compuesto de indicadores que pueden tomar distintos valores (por esa razón también se denominan variables puesto que los valores pueden cambiar, es decir, variar); cada indicador o variable tiene asignado un nombre para poder referirse a él.
- **Información:** El hecho de que un indicador/variable tenga asociado un valor es irrelevante si dicha asociación no es interpretada; por ejemplo si sabemos que el indicador/variable “v” tiene asociado el valor “70.5” no nos aporta nada, es la interpretación de “v” como “velocidad en km/h” de dicha pareja la que nos proporciona una información. Así pues, la información es la asociación entre una variable, un valor y la interpretación que se hace.
- **Estado:** Puesto que el entorno está formado por indicadores/variables que pueden tomar diferentes valores está claro que el entorno no es inmutable sino dinámico; el conjunto de valores de los indicadores/variables del entorno en un instante se denomina estado. De esta forma, si una acción modifica su entorno (los valores de uno o más indicadores) se puede afirmar que las acciones hacen pasar el entorno de un estado a otro.
- **Datos/resultados:** Una acción emplea un tiempo finito en su ejecución; si la acción se inicia en t_0 y termina en t_1 , el estado del sistema en el instante t_0 (los valores de los indicadores) definen los datos de la acción mientras que el estado del sistema en el instante t_1 (los valores de los indicadores tras la ejecución de la acción) definen los resultados de la acción.
- **Léxico:** Como sabemos, un procesador comprende y ejecuta instrucciones que componen un algoritmo; el conjunto finito de acciones que el procesador puede comprender y ejecutar constituye el léxico del procesador.
- **Tipo de datos:** En matemáticas se clasifican los datos en función de ciertas características importantes; así tenemos naturales, enteros, reales, complejos, etc. Esta clasificación se hace mediante conjuntos; dichos conjuntos pueden estar formados por elementos individuales, **R**, o agregaciones de valores, **R_n**. Esta forma de clasificación resulta muy útil y, por tanto, a partir ahora todos los indicadores (variables) tendrán asignado un tipo.
- **Operador:** A lo largo de muchos de los ejemplos anteriores se ha dado por supuesto que el procesador conocía la forma de realizar asignaciones, comparar valores, etc. Dichas acciones se conocen con el nombre de operadores y las acciones que realizan son, por tanto, operaciones; los alumnos conocen la mayor parte de los operadores presentes en el léxico de la notación algorítmica que se va a estudiar; se trata de los operadores aritméticos, de comparación, etc.

Introducción de la notación algorítmica a emplear

A lo largo del curso se empleará una notación uniforme para el pseudocódigo; aunque dicha notación se irá explicando de forma paulatina en este apartado se explicarán algunos aspectos básicos que, además, servirán para ilustrar algunos de los conceptos anteriores.

- **Variables:** Como se dijo con anterioridad el entorno en que trabajará el procesador viene definido por una serie de variables (también llamados indicadores) que toman valores

cambiantes; también se dijo que las variables tendrán nombres para poder referirse a las mismas.

Los nombres de las variables son identificadores y tradicionalmente los identificadores se han formado con caracteres alfanuméricos (esto es, letras y números) incluyendo el carácter de subrayado y excluyendo los blancos y caracteres no anglosajones (como la 'ñ' y las vocales acentuadas).

Otra característica de los identificadores es que no pueden empezar por número, sólo por letra (o subrayado). En algunas notaciones (y lenguajes de programación), las mayúsculas se consideran como caracteres diferentes (key sensitive) de las minúsculas mientras que en otros no se hace así; nuestra notación (como Java) será sensible 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), ...

El nombre de las variables debe ser representativo del valor que continen, así no serán válidas variables con el valor *a* o *b*, pero sí *dni* o *nombre_cliente*.

El nombre de las variables lo precederemos que un indicador del tipo de datos que contenga:

- Enteros: `int` por ejemplo `intCoeficiente`
- Reales: `rln` por ejemplo `rlnNota`
- Caracter: `chr` por ejemplo `chrLetra`
- Cadena: `str` por ejemplo `strNombre`
- Booleanos: `bln` por ejemplo `blnBandera`
- Arrays: antes del tipo indicamos que es un array `arrRlnNotas1DAM`
- **Tipos de datos:** Ya se ha mencionado el hecho de que las variables deben pertenecer a un tipo de datos determinado; en nuestra notación algorítmica tenemos los siguientes tipos: `entero`, `real`, `logico` y `caracter`. Los tipos se describen en función de las operaciones que admiten.
 - El tipo `entero` se corresponden con un subconjunto finito (un rango) de los enteros, variando el tamaño de dicho rango en función del computador; las operaciones soportadas son las operaciones aritméticas básicas: suma (+), resta (-), producto (*), división entera (`div`) y módulo/resto (`mod`). El resultado de operar dos valores de tipo `entero` resulta otro valor de tipo `entero` con un valor exacto (siempre y cuando no resulte un valor fuera del rango).
 - El tipo `real` se corresponde con un subconjunto de números reales; una diferencia fundamental entre un valor `real` y un valor `entero` es que el segundo siempre es exacto mientras que el primero es un valor aproximado, susceptible de errores de redondeo. El tipo

real admite las mismas operaciones que el tipo entero, exceptuando la división entera y el módulo (la división real emplea el operador `/`); además de estas, admite las típicas funciones reales: raíz cuadrada, exponenciación, potenciación, logaritmo y funciones trigonométricas. Al operar valores de tipo `real` se obtienen resultados de tipo `real`. Para facilitar las cosas es posible operar valores de tipo `real` con valores de tipo `entero` y aplicar operaciones del tipo `real` al tipo `entero`; en todos estos casos el resultado es de tipo `real`.

- El tipo `logico` o `booleano` admite tan sólo dos valores: `verdadero` y `falso`; las operaciones admitidas son las de la lógica booleana: y-lógico (`and` o `^`), o-lógico (`or` o `^`) y no-lógico (`not` o `¬`). Hay que señalar que hay operadores que, aún no operando sobre valores de tipo `logico`, dan como resultado valores de dicho tipo; nos referimos a los operadores de comparación.

- El tipo `character` comprende el conjunto de caracteres imprimibles; dicho conjunto tiene una serie de características:

- Contiene las 26 letras del alfabeto latino, los 10 dígitos de la numeración arábica y caracteres como símbolos de puntuación.
- Los subconjuntos de letras y números están ordenados; es decir si un carácter es mayor que la 'A' y menor que la 'Z' es una letra mayúscula, si es mayor que 'a' y menor que 'z' es una letra minúscula y si es mayor que '0' y menor que '9' es un número.
- Existe un carácter blanco o espacio.

El tipo `character` tiene una única operación propia, la concatenación de cadenas de caracteres que emplea el operador `(+)`.

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 `logico`.

- **Literales:** En todo momento es posible definir literales, expresiones de valor constante, de cualquier tipo; así es posible tener literales de tipo `entero` (por ejemplo, `45`, `0` o `-17`), de tipo `real` (por ejemplo, `3.141592`, `10.5e20` o `-5.24e-5`), de tipo `character` (por ejemplo, `'E'`, `'e'`) y de tipo `cadena` (por ejemplo, `"Esto es una cadena"`).

En muchas ocasiones es necesario utilizar un valor literal de forma reiterada, por ejemplo `3.141592`, en esos casos es preferible referirse a dicho valor mediante un identificador, por ejemplo `PI`, que utilizar el valor en sí; para ello se utilizan las denominadas **constantes**, identificadores a los que se les asigna un valor que no puede ser modificado en ningún punto del algoritmo. Se representan con letra mayúscula para todo el nombre.

- **Expresiones:** una expresión es una combinación de variables, constantes y literales de tipos compatibles entre sí, estos elementos están combinados mediante operadores válidos; la expresión más simple está formada por un único elemento (variable, literal o constante) sin ningún operador. Es posible tener expresiones con cualquier tipo de datos. A continuación se muestran algunas expresiones típicas:

```
2 * PI * r
intVelocidad * intTiempo
(intCont>5) and (intCont<10)
'Hola' + ' mundo'
```

- **Asignación:** el operador de asignación, \leftarrow , se emplea según la notación siguiente:

```
izquierda  $\leftarrow$  derecha
```

El elemento situado a la izquierda del operador es una variable a la que se asignará el valor de la expresión de la derecha; para poder realizar la asignación de forma adecuada ambos elementos deben ser del mismo tipo.

- **Entrada/salida:** La mayor parte de los algoritmos necesitan interactuar con el usuario, es posible que éste deba indicar una serie de valores (por ejemplo, los coeficientes de una ecuación de segundo grado) o que el algoritmo deba proporcionar unos resultados al mismo usuario (por ejemplo, las posibles soluciones de dicha ecuación). Las acciones de entrada salida son dos: `leer` y `escribir`.

El formato de la primera es muy sencillo: `leer variable`; el resultado de la acción es la lectura de un valor por teclado que será asignado a la variable `variable`.

El formato de la segunda es similar: `escribir expresión`; donde `expresión` puede ser de cualquier tipo. El resultado de esta acción es la visualización (en una pantalla) de la expresión.