INTEGRANTES:

Castillo Díaz Margarita

Cruz Fuentes Manuel

Barco Gonzáles Rina

Custodio Llontop Miguel

Aquino Odar Gustavo

CAPITULO 1

ALGORITMOS Y PROGRAMAS

CONTENIDO

La principal razón para que las personas aprendan lenguajes y técnicas de programación es utilizar la computadora como una herramienta para resolver problemas. La resolución de un problema exige al menos los siguientes pasos:

1. Definición o análisis del problema.
2. Diseño del algoritmo.
3. Transformación del algoritmo en un programa.
4. Ejecución y validación del programa.

Uno de los objetivos fundamentales dee este libro es el *aprendizaje* y *diseño* de algoritmos. Este capítulo introduce al lector en el concepto de algoritmo y de programa, así como las herramientas que permiten <<dialogar>> al usuario con la máquina: los lenguajes de programación.

Un algoritmo es un método para resolver un problema. Aunque la popularización del término ha llegado con el advenimiento de la era informática, algoritmo proviene de Mohammed al–Khowarizmí,

matemático persa que vivió durante el siglo IX y alcanzó gran reputación por el enunciado de las reglas paso a paso para sumar, restar, multiplicar y dividir números decimales; la traducción al latín del apellido en la palabra *algorismus* derivó posteriormente en algoritmo. Euclides, el gran matemático griego (del siglo IV antes de Cristo), que inventó un método para encontrar el máximo común divisor de dos números, se considera con Al–Khowarizmí el otro gran padre de la algoritmia (ciencia que trata de los algoritmos).

El resto del capítulo trata de los datos y las operaciones elementales necesarias para el diseño del algoritmo. El profesor Niklaus Wirth –inventor de Pascal, Modula-2 y Oberon– tituló uno de sus más famosos libros, Algoritmos + Estructuras de datos = Programas, significándonos que sólo se puede llegar a realizar un buen programa con el diseño de un algoritmo y una correcta estructura de datos. Esta ecuación será una de las hipótesis fundamentales consideradas en esta obra.

1.1. LOS SISTEMAS DE PROCESAMIENTO DE LA INFORMACIÓN.

Una definición antigua de computadora es: <<una máquina o aparato electrónico capaz de ejecutar operaciones repetitivas muy complejas a altas velocidades>>. Ahora bien, ésta definición no describe las modernas computadoras. Éstas son más que una máquina de ejecutar operaciones aritméticas. De hecho, los términos procesador de *datos y sistemas de procesamiento (tratamiento) de la información* se utilizan con frecuencia en lugar de computadora (*ordenador*, en la jerga informática usual en España).

En el uso diario, datos e información son esencialmente sinónimos. Sin embargo, los informáticos suelen hacer una diferencia: datos se refiere a la representación de un hecho, concepto o entidad real (los datos pueden tomar diferentes formas: por ejemplo, palabras escritas o habladas, números y dibujos); *información* implica datos procesados y organizados.

Un sistema en general se define como conjunto de componentes conectados e interactivos, que tienen un propósito y una unidad total. **Sistema de procesamiento de información** es un sistema que transforma datos brutos en *información organizada, significativa y útil*.

La Figura 1.1 muestra los tres componentes de un sistema de proceso de la información: *entrada, salida y procesador.* El *procesador*, que puede ser bastante complicado, se representa por una simple caja y puede aceptar datos llamados entrada, y esta entrada se transforma entonces para producir una información denominada *salida o resultados.*

Basados en este esquema, muchos dispositivos u organismos pueden ser considerados sistemas de procesamiento de la información. Un termostato que controla la temperatura de un edificio es un sistema de procesamiento de la información. La entrada es la temperatura media y la salida es una señal que controla la caldera del aire acondicionado. El corazón de un animal o un ser humano es un sistema complejo de procesamiento de la información.

El conjunto de instrucciones que especifican la secuencia de operaciones a realizar, en orden, para resolver un sistema específico o clase de problemas, se denomina algoritmo. En otras palabras, un **algoritmo** es una *fórmula* para la resolución de un problema.

Figura 1.1. Sistema de proceso de la información

Para realizar un proceso se le debe suministrar al procesador un algoritmo adecuado. Por ejemplo, al cocinero debe dársele una receta, al pianista la partitura y asi sucesivamente, considerando al cocinero y al pianista como procesadores.

Cuando el procesador es una computadora, el algoritmo ha de expresarse de una forma que recibe el nombre de programa. Un programa sen escribe en un lenguaje de programación y a la actividad de expresar un algoritmo en forma de programa se le denomina programación. Cada paso en algoritmo está expresado por medio de una instrucción en el programa. Por consiguiente, un programa consta de una secuencia de instrucciones, cada una de las cuales especifica las operaciones que debe realizar la computadora.

Existen dos conceptos importantes a considerar en los sistemas de procesamiento de la información: hardware y software. **Hardware** es el conjunto de componentes físicos de una computadora (Figura 1.2) –equipo físico– y **software** es el conjunto de programas que controlan el funcionamiento de una computadora –equipo lógico– .

EL hardware de una computadora se compone de:

1. La Unidad Central de Proceso, UCP (Central Processing Unit, CPU). La UCP es el conjunto de circuitos electrónicos capaces de ejecutar algunos cálculos sencillos como suma o multiplicación de números. La potencia de una computadora depende completamente de la velocidad y fiabilidad de la UCP.
2. Memoria central. La información procesada por la UCP se almacena normalmente en la memoria central hasta que se terminan los cálculos. Los programas de computadora se almacenan también en la memoria central.
3. Dispositivos de almacenamiento secundario (memoria auxiliar). Diferentes dispositivos, tales como discos y cintas magnéticas, se usan para almacenar grandes cantidades de información. Para ser procesados por la UCP, los datos se almacenan en dispositivos de almacenamiento auxiliar y luego tienen que llevarse a la memoria central.
4. Periférico o dispositivos de entrada/salida (E/S). Estos dispositivos permiten al usuario comunicarse con la computadora. Un sistema de computadoras puede tener diferentes dispositivos periféricos conectados a ella.

En la práctica, una instalación grande de computadora puede tener diferentes UCP, cada una con su propia memoria central compartida, una variedad de dispositivos de almacenamiento secundario y periféricos localizados en diferentes partes de un mismo edificio o diferentes edificios e incluso diferentes ciudades o países.

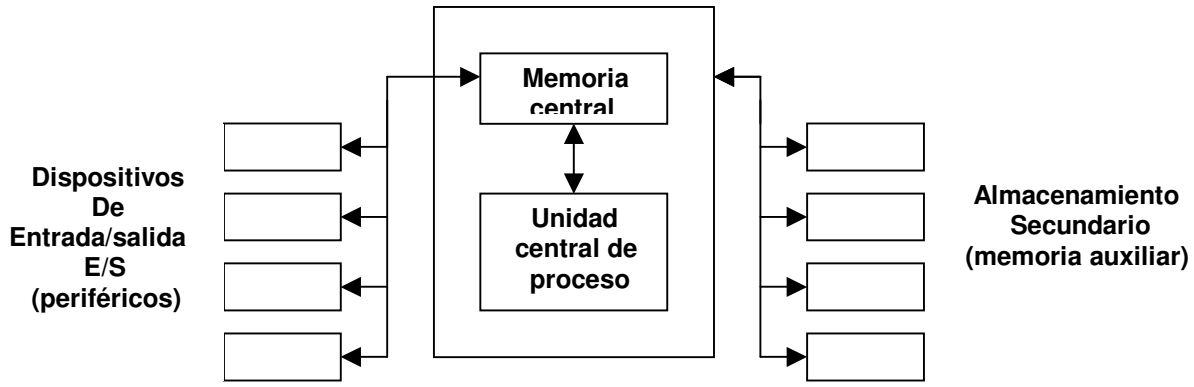


Figura 1.2. Diagrama esquemático de una computadora (hardware)

1.2. CONCEPTO DE ALGORITMO

El objetivo fundamental de esta texto es enseñar a resolver problemas mediante una computadora. Un programador de computadora es antes que nada una persona que resuelve problemas, por lo que para llegar a ser un programador eficaz se necesita aprender a resolver problemas de un modo riguroso y sistemático. A lo largo de todo este libro nos referimos a la metodología necesaria para resolver problemas mediante programas, al concepto se denomina **metodología de la programación**. El eje central de esta metodología es el concepto, ya tratado, de algoritmo.

La resolución de un problema exige el diseño de un algoritmo que resuelva el problema propuesto.

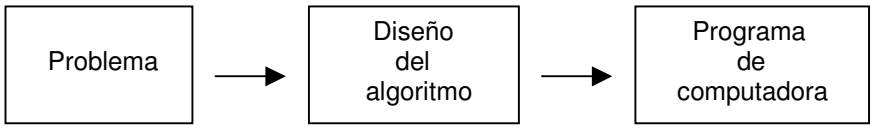


Figura 1.3. Resolución de un problema

Los pasos para la resolución de un problema son:

1. Diseño del algoritmo que describe la secuencia ordenada de pasos –sin ambigüedades– que conducen a la solución de un problema dado. (Análisis del programa y desarrollo del algoritmo.)
2. Expresar el algoritmo como un programa en un lenguaje de programación adecuado. (Fase de codificación.)
3. Ejecución y validación del programa por la computadora.

Para llegar a la realización de un programa es necesario el diseño previo de un algoritmo, de modo que sin algoritmo no puede existir un programa.

Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que los ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y ejecutarse en una computadora distinta; sin embargo, el algoritmo será siempre el mismo. Así, por ejemplo, en una analogía con la vida diaria, una receta de un plato de cocina se puede expresar en español, inglés o francés, pero cualquiera que sea el lenguaje, los pasos para la elaboración del plato se realizarán sin importar el idioma del cocinero.

En la ciencia de la computación y en la programación, los algoritmos son mas importantes que los lenguajes de programación o las computadoras. Un lenguaje de programación es tan solo un medio para expresar un algoritmo y una computadora es sólo un procesador para ejecutarlo. Tanto el lenguaje de programación como la computadora son los medios para obtener un fin: conseguir que el algoritmo se ejecute y se efectúe el proceso correspondiente.

Dada la importancia del algoritmo en la ciencia de la computación, un aspecto muy importante será el diseño de algoritmos. A la enseñanza y práctica de esta tarea se dedica gran parte de esta libro.

El diseño de la mayoría de los algoritmos requiere creatividad y conocimientos profundos de la técnica de la programación. En esencia, la solución de un problema se puede expresar mediante un algoritmo.

1.2.1. CARACTERÍSTICAS DE LOS ALGORITMOS

Las características fundamentales que debe cumplir todo algoritmo son:

- Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.
- Un algoritmo debe estar definido. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser finito. Si se sigue un algoritmo, se debe terminar en algún momento; osea, debe tener un número finito de pasos.

La definición de un algoritmo debe describir tres partes: Entrada, proceso y salida. En el algoritmo de receta de cocina citado anteriormente se tendrá:

Entrada: Ingredientes y utensilios empleados
Proceso: Elaboración de la receta en la cocina.
Salida: Terminación del plato (por ejemplo, cordero).

Ejemplo 1.1

Un cliente ejecuta un pedido a una fábrica. La fábrica examina en su banco de datos la ficha del cliente, si el cliente es solvente entonces la empresa acepta el pedido; en caso contrario, rechazará el pedido. Redactar el algoritmo correspondiente.

Los pasos del algoritmo son:

1. Inicio.
2. Leer el pedido.
3. Examinar la ficha del cliente
4. Si el cliente es solvente, aceptar pedido; en caso contrario, rechazar pedido
5. Fin

Ejemplo 1.2

Se desea diseñar un algoritmo para saber si un número es primo o no.

Un número es primo si solo puede dividirse por si mismo y pro la unidad (es decir no tiene más divisores que él mismo y la unidad). Por ejemplo, 9, 8, 6, 4, 12, 16, 20, etc, no son primos ya que son divisibles por números distintos en a ellos y a la unidad. Así, 9 es divisible por 3, 8 lo es por 2, etc.

El algoritmo de resolución del problema pasa por dividir sucesivamente el número por 2, 3, 4. . ., etc.

1. Inicio.
2. Poner X igual a 2 (X = 2, X, variable que representa a los divisores del número que se busca N).
3. Dividir N por X (N / X).
4. Si el resultado de N / X es entero, entonces N no es un número primo y bifurcar al punto 7; en caso contrario continuar el proceso.
5. Suma 1 a X (X • X + 1).
6. Si X es igual a N, entonces N es un número primo en caso contrario bifurcar al punto 3.
7. Fin.

---

1 Esta condición puede ser sustituida por X = n **div** 2 (donde div es el operador división entera). Véase tabla1.1 página 19

Por ejemplo, si N es 131, los pasos anteriores serian:

```
1. Inicio.
2. X = 2.
3 y 4.  131 / X. Como el resultado no es un entero, se continua el
        proceso.
5.   X • 2 + 1, luego X = 3 .
6.   Como X no es 131, se bifurca al punto 3.
3 y 4.   131 / X resultado no es un entero.
5.   X • 3 + 1, X = 4 .
6.   Como X no es 131, se bifurca al punto 3.
3 y 4.  131 / X..., etc .
7.    Fin.
```

Ejemplo 1.3

Realizar la suma de todos los números pares entre 2 y 1000.

El problema consiste en sumar 2 + 4 + 6 + 8 + ... + 1000.

Utilizaremos las palabras `SUMA Y NUMERO` (variables, serán denominadas mas tarde) para representar las sumas sucesivas (2 + 4), (2 + 4 + 6), (2 + 4 + 6 + 8), etc.

La solución se puede escribir con el siguiente algoritmo:

```
1. Inicio.
2. Establecer SUMA a 0 .
3. Establecer NUMERO a 2 .
4. Sumar NUMERO A SUMA .El resultado será el nuevo valor de la suma
   (SUMA) .
5. Incrementar NUMERO 2 unidades .
6. El NUMERO =< 1000 bifurca al paso 4 ; en caso contrario, escribir
   el ultimo valor de SUMA y terminar el proceso .
7. Fin .
```

1.3. LOS LENGUAJES DE PROGRAMACIÓN

Como se ha visto en el apartado anterior, para que un procesador realice un proceso se le debe suministrar en primer lugar un algoritmo adecuado. El procesador debe ser capaz de interpretar el algoritmo, lo que significa:

- comprender las instrucciones de cada paso,
- realizar las operaciones correspondientes.

Cuando el procesador es un computadora, el algoritmo se ha de expresar en un formato que se denomina programa. Un programa se escribe en un lenguaje de programación y las operaciones que conducen a expresar en un algoritmo en forma de programa se llaman programación. Así pues, los lenguajes utilizados para escribir programas de computadoras son los lenguajes de programación y programadores son los escritores y diseñadores de programas.

Los principales tipos de lenguajes utilizados en la actualidad son tres:

- lenguaje maquina,
- lenguaje de bajo nivel (ensamblador),
- lenguajes de alto nivel.

1.3.1. Instrucciones a la computadora

Los diferentes pasos (acciones) de un algoritmo se expresan en los programas como instrucciones, sugerencias o proposiciones (normalmente el término instrucción se suele referir a los lenguajes maquina y bajo nivel, reservando la sentencia o proposición para los lenguajes de alto nivel). Por consiguiente, un programa consta de una secuencia de instrucciones, cada una de las cuales especifica ciertas operaciones que debe ejecutar la computadora.

La elaboración de un programa requerirá conocer el juego o repertorio de instrucciones del lenguaje. Aunque en el Capítulo 3 se analizarán con mas detalle las instrucciones, adelantaremos los tipos fundamentales de instrucciones que una computadora es capaz de manipular y ejecutar. Las instrucciones básicas y comunes a casi todos los lenguajes de programación se pueden condensar en cuatro grupos:

- Instrucciones de entrada/salida. Instrucciones de transferencia de información y datos entre periféricos (teclado, impresora, unidad de disco, etc.) y la memoria central.
- Instrucciones aritmético-lógicas. Instrucciones que ejecutan operaciones aritméticas (suma, resta, multiplicación, división, potenciación), lógicas (operaciones and, or, not, etc).
- Instrucciones selectivas. Instrucciones que permiten la selección de tareas alternativas en función de los resultados de diferentes expresiones condicionales.
- Instrucciones repetitivas. Instrucciones que permiten la repetición de secuencias de instrucciones un número determinado o indeterminado de veces.

1.3.2. Lenguajes máquina

Los **lenguajes máquina** son aquellos que están escritos en lenguajes directamente inteligibles por la máquina (computadora), ya que sus instrucciones son cadenas binarias (cadenas o series de caracteres –dígitos- 0 y 1) que especifican una operación y las posiciones (dirección) de memoria implicadas en la operación se denominan instrucciones de máquina o código máquina. El código máquina es el conocido código binario.

Las instrucciones en lenguaje máquina dependen del hardware de la computadora y, por tanto, diferirán de una computadora a otra. El lenguaje máquina de una PC (computadora personal) será diferente de un sistema HP 9000 (HP, Hewlett Packard) o un sistema 6000 IBM.

Las ventajas de programar en lenguaje máquina son la posibilidad de cargar (transferir un

Memoria

Dirección	Contenido			
0100	0010	0000	0000	0100
0101	0100	0000	0000	0101
0102	0011	0000	0000	0110

Posiciones de memoria

Instrucciones binarias (código máquina)

Figura 1.4. instrucciones en lenguaje máquina

programa a la memoria) sin necesidad de traducción posterior lo que supone una velocidad de ejecución superior a cualquier otro lenguaje de programación.

Los *inconvenientes* –en la actualidad– superan a las ventajas, lo que hace prácticamente no recomendables los lenguajes máquina. Estos inconvenientes son:

- Dificultad y lentitud en la codificación.
- Poca fiabilidad
- Dificultad grande de verificar y poner a punto los programas.
- Los programas sólo son ejecutables en el mismo procesador (UCP, Unidad Central de Proceso).

Para evitar los lenguajes máquina, desde el punto de vista del usuario, se han creado otros lenguajes que permiten escribir programas con instrucciones similares al lenguaje humano (por desgracia, casi siempre inglés, aunque existen excepciones, como es el caso de las versiones españolas del lenguaje LOGO).

Estos lenguajes son los de alto nivel y bajo nivel.

1.3.3 Lenguajes de bajo nivel

Los lenguajes de bajo nivel son más fáciles de utilizar que los lenguajes máquina, pero, al igual que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel por excelencia es el ensamblador *(assembly language)*. Las instrucciones en lenguaje ensamblador son conocidas como nemotécnicos (*mnemonics*). Por ejemplo, nemotécnicos típicos de operaciones aritméticas son: en inglés, ADD, SUB, DIV, etc: en español, SUM, RES, DIV, etc.

Una instrucción típica de suma sería:

```
ADD    M, N, P
```

Esta instrucción podría significar <<sumar el número contenido en la posición de memoria M al número almacenado en la posición de memoria N y situar el resultado en la posición de memoria P>>. Evidentemente, es mucho más sencillo recordar la instrucción anterior con un nemotécnico que su equivalente en código máquina:

```
0110          1001          1010          1011
```

Un programa escrito en lenguaje ensamblador no puede ser ejecutado directamente por la computadora –en esto se diferencia esencialmente del lenguaje máquina–, sino que requiere una fase de *traducción* al lenguaje máquina.

El programa original escrito en lenguaje ensamblador se denomina *programa fuente* y el programa traducido en lenguaje máquina se conoce como *programa objeto*, ya directamente inteligible por la computadora.

El traductor de programas fuente a objeto es un programa llamado *ensamblador (assembler)*, existente en casi todas las computadoras (Figura 1.5).

No se debe confundir –aunque en español adoptan el mismo nombre– el programa *ensamblador (assembler)*, encargado de efectuar la traducción del programa fuente escrito a lenguaje máquina, con el lenguaje ensamblador *(assembly language)*, lenguaje de programación con una estructura y gramáticas definidas.

Los lenguajes ensambladores presentan la *ventaja* frente a los lenguajes máquina de su mayor facilidad de codificación y, en general, su velocidad de cálculo.

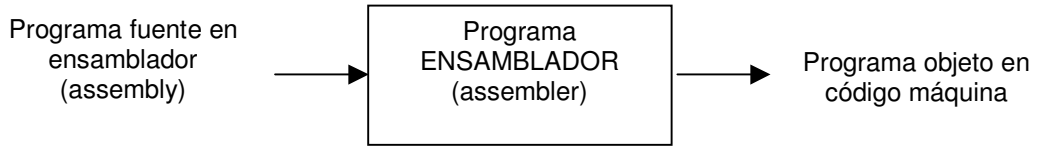


Figura 1.5. Programa ensamblador

Los *inconvenientes* más notables de los lenguajes ensambladores son:

- Dependencia total de la máquina, lo que impide la transportabilidad de los programas (posibilidad de ejecutar un programa en diferentes máquinas). El lenguaje ensamblador del PC es distinto del lenguaje ensamblador del Apple Macintosh.
- La formación de los programadores es más compleja que la correspondiente a los programadores de alto nivel, ya que exige no sólo las técnicas de programación, sino también el conocimiento del interior de la máquina.

Hoy día los lenguajes ensambladores tienen sus aplicaciones muy reducidas en la programación de aplicaciones y se centran en aplicaciones de tiempo real, control de proceso y de dispositivos electrónicos etc.

1.3.4. Lenguajes de alto nivel

Los lenguajes de alto nivel son los más utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensambladores. Otra razón es que un programa escrito en un lenguaje de alto nivel es independiente de la máquina; esto es, las instrucciones del programa de la computadora en particular. En consecuencia, los programadores escritos en lenguajes de alto nivel son portables o transportables, lo que significa la posibilidad de poder ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras; al contrario que los programas en lenguaje máquina o ensamblador, que sólo se pueden ejecutar en un determinado tipo de computadora.

Los lenguajes de alto nivel presentan las siguientes *ventajas*:

- El tiempo de formación de los programadores es relativamente corto comparador con otros lenguajes.
- La escritura de programas se basa en reglas sintácticas similares a los lenguajes humanos.
  Nombres de las instrucciones, tales como READ, WRIET, PRINT, IOPEN, etc.
- Las modificaciones y puestas a punto de los programas son más fáciles.
- Reducción del coste de los programas.
- Transportabilidad

Los inconvenientes se concretan en:

- Incremento del tiempo de puesta a punto, al necesitarse diferentes traducciones del programa fuente para conseguir el programa definitivo.
- No se aprovechan los recursos internos de la máquina, que se explotan mucho mejor en lenguajes máquina y ensambladores.
- Aumento de la ocupación de memoria.
- El tiempo de ejecución de los programas es mucho mayor.

Al igual que sucede con los lenguajes ensambladores, los programas fuente tienen que ser traducidos por programas traductores, llamados en este caso *compiladores o intérpretes*.
Los lenguajes de programación de alto nivel existentes hoy son muy numerosos aunque la práctica demuestra que su uso mayoritario se reduce a:

C C++ COBOLFORTRAN BASIC Pascal Visual BASIC.

están muy extendidos

Clipper Ada Modula-2 Prolog LISP Smalltalk

y comienzan a difundirse:

Visual Object Delphi Miranda Eiffel

aunque los dos primeros no dejan de ser entornos de programación orientados a objetos con soporte en los lenguajes Clipper y Turbo Pascal.

1.3.5. Traductores de lenguaje

Los *traductores de lenguaje* son programas que traducen a su vez los programas fuente escritos en lenguajes de alto nivel a código máquina.

Los traductores se dividen en:

- Compiladores.
- Intérpretes.

1.3.5.1. Intérpretes

Un *intérprete* es un traductor que toma un programa fuente, lo traduce y a continuación lo ejecuta.

Los programas intérpretes clásicos como BASIC, prácticamente ya no se utilizan, aunque las versiones Qbasic y QuickBASIC se comercializan todavía con el Sistema Operativo DOS que corre en las computadoras personales. Sin embargo, está muy extendida la versión interpretada del lenguaje Smalltalk, un lenguaje orientado o objetos puro.

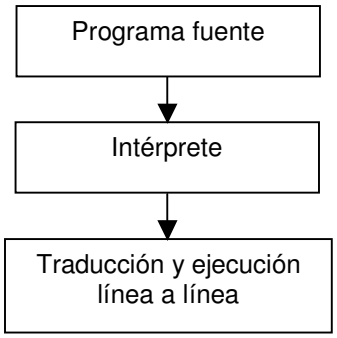


Figura 1.6. Intérprete.

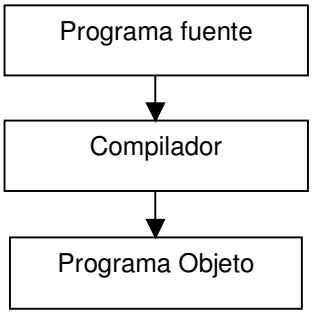


Figura 1.7. La compilación de programas.

1.3.5.2. Compiladores

Un *compilador* es un programa que traduce los programas fuente escritos en lenguajes de alto nivel –Pascal, FORTRAN, ...–a lenguaje máquina.

Los programas escritos en lenguajes de alto nivel se llaman *programas fuente* y el programa traducido *programa objeto o código objeto*. El compilador traduce –sentencia a sentencia– el programa fuente.

Los lenguajes compiladores típicos son: C, C++, PASCAL, FORTRAN, COBOL.

1.3.6. La compilación y sus fases.

La *compilación* es el proceso de traducción de programas fuente a programas objeto. El programa objeto obtenido de la compilación ha sido traducido normalmente a código máquina.

Para conseguir el programa máquina real se debe utilizar un programa llamado *montador o enlazador (linker).* El proceso de montaje conduce a un programa en lenguaje máquina directamente ejecutable (Figura 1.8).

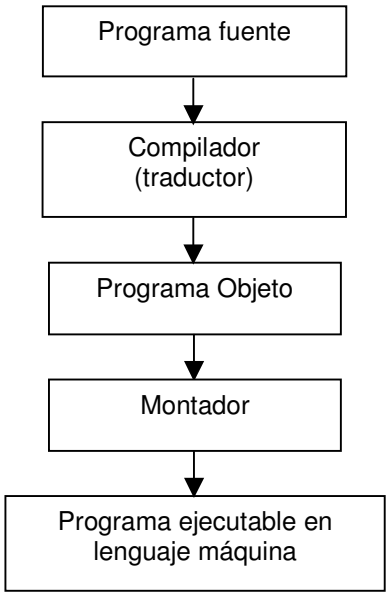


Figura 1.8. Fases de la compilación.

El proceso de ejecución de un programa Pascal, por ejemplo, tiene los siguientes pasos:

1. Escritura del programa *fuente* con un editor (programa que permite a una computadora actuar de modo similar a una máquina de escribir electrónica) y guardarlo en un dispositivo de almacenamiento (por ejemplo un disco).
2. Introducir el programa fuente en memoria.
3. *Compilar* el programa con el compilador Pascal.
4. *Verificar y corregir* errores de compilación (listado de errores).
5. Obtención del programa objeto.

6. El montador obtiene el programa ejecutable.
7. Se ejecuta el programa y, si no existen errores, se tendrá la salida del programa.

El proceso de ejecución sería el mostrado en las Figuras 1.9 y 1.10.

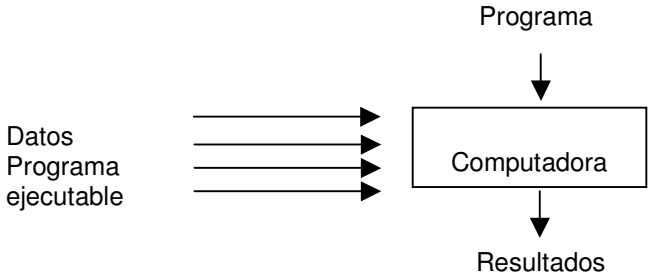


Figura 1.9. Ejecución de un programa.

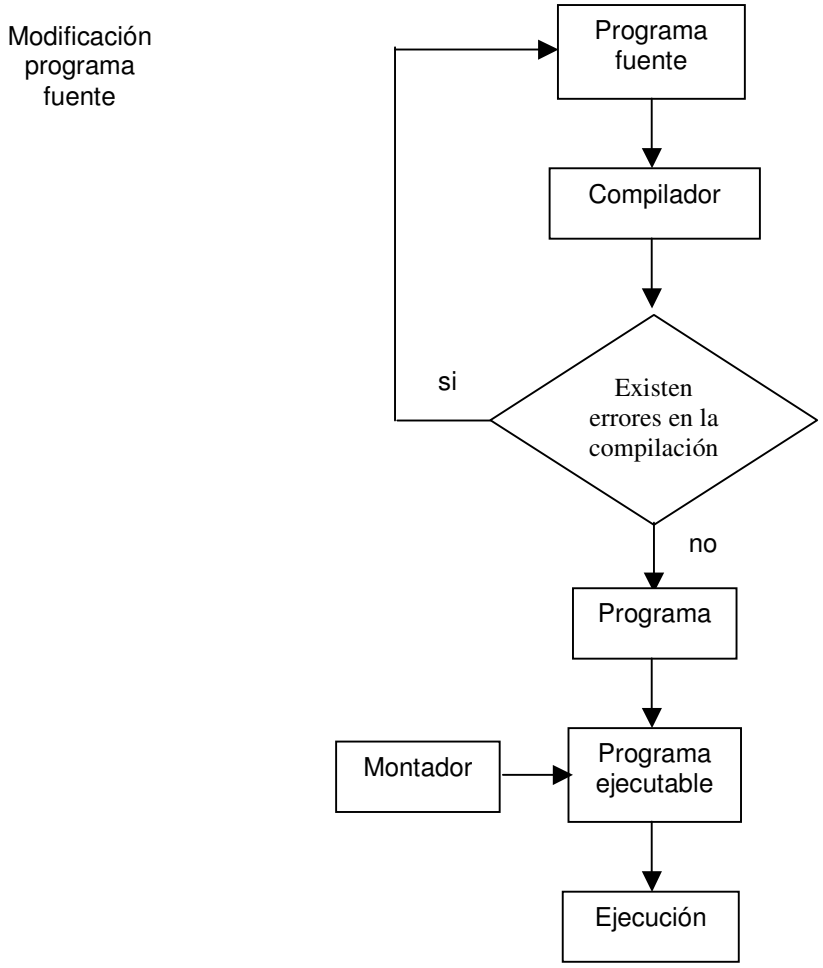


Figura 1.10. Fases de la ejecución de un programa.

1.4. DATOS, TIPOS DE DATOS Y OPERACIONES PRIMITIVAS

El primer objetivo de toda computadora es el manejo de la información o datos. Estos datos pueden ser las cifras de ventas de un supermercado o las calificaciones de una clase. Un *dato* es la expresión general que describe los objetivos con los cuales opera una computadora.

La mayoría de las computadoras pueden trabajar con varios tipos (modos) de datos. Los algoritmos y los programas correspondientes operan sobre datos.

La acción de las instrucciones ejecutables de las computadoras en cambios en los valores de las partidas de datos. Los datos de entrada se transforman por el programa, después de las etapas intermedias, en datos de salida.

En el proceso de solución de problema el diseño de la estructura de datos es tan importante como el diseño del algoritmo y del programa que se basa en el mismo.

Existen dos clases de tipos de datos: *simples* (sin estructura) y *compuestos* (estructurados). Los datos estructurados se estudian a partir del Capítulo 6 y son conjuntos de partidas de datos simples con relaciones definidas entre ellos.

Los distintos tipos de datos se representan en diferentes formas en la computadora. A nivel de máquina, un dato es un conjunto o secuencia de bits (dígitos 0 y 1). Los lenguajes de alto nivel permiten basarse en abstracciones e ignorar los detalles de la representación interna. Aparece el concepto de tipo de datos, así como su representación.

Los tipos de datos simples son los siguientes:

numéricos (integer, real),
lógicos (bolean),
carácter (char, string).

Existen algunos lenguajes de programación –FORTRAN esencialmente– que admite otros tipos de datos; complejos, que permiten tratar los números complejos, y otros lenguajes –Pascal– que también permiten declarar y definir sus propios tipos de datos: **enumerados** (enumerated) y **subrango** (subrange).

1.4.1 Datos numéricos

El tipo *numéricos* es el conjunto de los valores numéricos. Éstos pueden representarse en dos formas distintas.

- Tipo numérico *entero* (integer),
- Tipo numérico *real* (real).

Enteros: El tipo entero es un subconjunto finito de los número enteros. Los enteros son números complejos, no tienen componentes fraccionarios o decimales y pueden ser negativos o positivos.

Ejemplos de números enteros son:

```
  5      6
-15     -4
 20     17
1340    26
```

Los enteros se denominan en ocasiones números de punto o coma fija. Los números enteros máximos y mínimos de una computadora[2] suelen ser –32768 a +32767–. Los números enteros

2 En computadoras de 16 bits como IBM PC o compatibles.

fuera de este rango no se suelen representar como enteros, sino como reales, aunque existen excepciones (enteros largos: FORTRAN, Quick/Qbasic, C, C++, etc).

Reales: El tipo real consiste en un subconjunto de los números reales. Los números reales siempre tienen un punto decimal y pueden ser positivos o negativos. Un número real consta de un entero y una parte decimal.

Los siguientes ejemplos son números reales:

```
 0.08.1          3739.41
 3.7452          -52.321
-8.12              3.0
```

En aplicaciones científicas se requiere una representación especial para manejar números muy grandes, como la masa de la Tierra, o muy pequeños, como la masa de un electrón. Una computadora sólo puede representar un número fijo de dígitos. Este número puede variar de una máquina a otra, siendo ocho dígitos un número típico. Este límite provocará problemas para representar y almacenar números muy grandes o muy pequeños como son los ya citados o los siguientes:

4867213432 0.00000000387

Existe un tipo de representación denominado notación exponencial o científica y que se utiliza para números muy grandes o muy pequeños. Así,

367520100000000000000

se representa en notación científica descomponiéndolo en grupos de tres dígitos:

367520 100 000 000 000 000

y posteriormente en forma de potencias de 10:

3.675201×10^{20}

y de modo similar:

.0000000000302579

se representa como:

3.02579×10^{-22}

La representación en coma flotante es una generalización de notación científica. Obsérvese que las siguientes expresiones son equivalentes:

$3.675201 \times 10^{19} = .3675207 \times 10^{20} = .03675201 \times 10^{21} = \ldots$
$= 36.75201 \times 10^{18} = 367.5201 \times 10^{17} = \ldots$

en estas expresiones se considera la *mantisa* (parte decimal) al número *real* y al *exponente* (parte potencial) el de la potencia de diez.

36.75201 mantisa 18 exponente

1.4.2. Datos lógicos (booleanos)

El tipo *lógico* –también denominado booleano– es aquel dato que solo puede tomar uno de dos valores:

cierto o **verdadero** (true) y **falso** (false).

Este tipo de datos se utiliza para representar las alternativas (si/no) a determinadas condiciones.
Por ejemplo, cuando se pide si un valor entero es par, la respuesta será verdadera o falsa, según sea par o impar.

1.4.3. Datos tipo carácter y tipo cadena

El tipo *carácter* es el conjunto finito y ordenado de caracteres que la computadora reconoce. Un dato tipo carácter contiene un solo carácter.

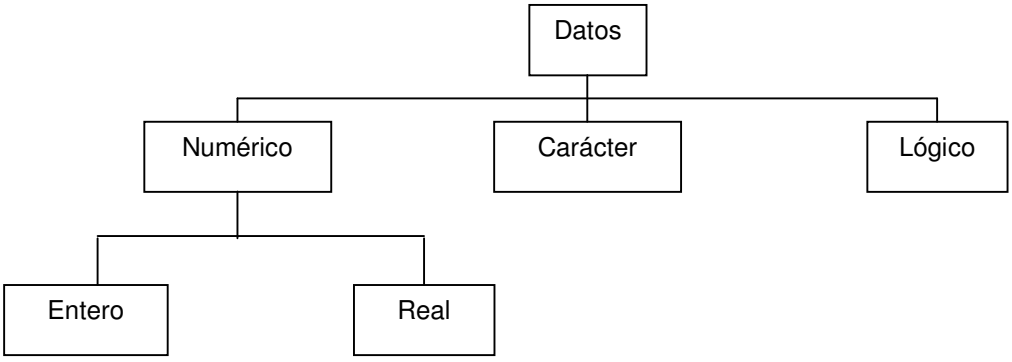
Los caracteres que reconocen las diferentes computadoras no son estándar; sin embargo, la mayoría reconoce los siguientes caracteres alfabéticos y numéricos:

- caracteres alfabéticos (A, B, C, . . . , Z) (a, b, c, . . . , z),
- caracteres numéricos (1, 2, . . . , 9, 0),
- caracteres especiales (+, −, *, /, ^, . , ; . <, >, $, . . .).

Una cadena (string) de caracteres es una sucesión de caracteres que se encuentran delimitados por una comilla (apóstrofo) o dobles comillas, según el tipo de lenguaje de programación. La *longitud* de una cadena de caracteres es el número de ellos comprendidos entre los separadores o limitadores. Algunos lenguajes tiene datos tipo *cadena*.

´Hola Mortimer´
´8 de Octubre de 1946´
´Sr. Mckenna´

RESUMEN: Los tipos de datos primitivos se clasifican en:



1.5.CONSTANTES VARIABLES

Los programas de computadora contiene ciertos valores que no deben cambiar durante la ejecución del programa. Tales valores se llaman *constantes*. De igual forma, existen otros valores que cambiarán durante la ejecución del programa; a estos valores se les llama *variables*.

Una ***constante*** es una partida de datos (objetos) que permanecen sin cambios durante todo el desarrollo del algoritmo o durante la ejecución del programa.

Constantes reales válidas :	***Constantes Reales no válidas :***
`1.234`	`1,752.63` *(comas no permitidas)*
`- 0.1436`	`82` *(normalmente contienen un punto decimal, aunque existen*
`+ 54437324`	*lenguajes que lo admiten sin punto)*

Constantes reales en notación científica :

`3.374562E2` *equivale a* `3.374562 x 10`

Una constante *tipo carácter* o *constante de caracteres* consiste en un carácter válido encerrado dentro de apóstrofos: por ejemplo :

```
'B'      '+'   '4'   ';'
```

Si se desea incluir el apóstrofo en la cadena, entonces debe aparecer como un par de apóstrofos, encerrados dentro de simples comillas.

Una secuencia de caracteres se denomina normalmente una *cadena* y una *constante tipo cadena* es una cadena encerrada entre apóstrofos. Por consiguiente,

```
'Juan Dominguez'
```

y

```
'Pepe Luis García'
```

son constantes de cadena válidas. Nuevamente, si un apóstrofo es uno de los caracteres es una constante de cadena, debe aparecer como un par de apóstrofos:

```
'Juan "s'
```

Constantes lógicas (boolean)

Sólo existen dos constantes *lógicas* o *boolean*:

```
verdad   falso
```

La mayoría de los lenguajes de programación permiten diferentes tipos de constantes: *enteras, reales, caracteres* y *boolean* o *lógicas*, y representan datos de esos tipos.

Variables

Una variable es un objeto o partida de datos cuyo valor puede cambiar durante el desarrollo del algoritmo o ejecución del programa.

Dependiendo del lenguaje, hay diferentes tipos de variables, tales como *entera, reales, carácter, lógicas* y *de cadena*.

Una variable que es de un cierto tipo puede tomar únicamente valores de ese tipo. Una variable de carácter, por ejemplo, puede tomar como valor sólo caracteres, mientras que una variable entera puede tomar sólo valores enteros.

Si se intenta asignar un valor de un tipo a una variable de otro tipo se producirá un *error de tipo*.

Una variable se identifica por los siguientes atributos: *nombre* que lo asigna y *tipo* que describe el uso de la variable.

Los nombres de las variables, a veces conocidos como *identificadores*, suelen constar de varios caracteres alfanuméricos, de los cuales el primero normalmente es una letra. No se deben utilizar - aunque lo permita el lenguaje, caso de FORTRAN - como nombres de identificadores palabras reservadas del lenguaje de programación.

Nombres válidos de variables son:

```
A510
NOMBRES
NOTAS
NOMBRE_APELLIDOS[3]
```

Los nombres de las variables elegidas para el algoritmo o el programa deben ser significativos y tener relación con el objeto que representan, como pueden ser los casos siguientes:

```
NOMBRE    para representar nombres de personas
PRECIOS   para representar los precios de diferentes artículos.
NOTAS     para representar las notas de una clase
```

Existen lenguajes - Pascal - en los que es posible darles nombre a determinadas constantes típicas utilizadas en cálculos matemáticos, financieros, etc. Por ejemplo, las constantes π = 3.141592... y e = 2.7182818 (base de los logaritmos naturales) se les pueden dar los nombres `Pi` y `E`.

```
PI  =     3.141592
E   =     2.718282
```

1.6. EXPRESIONES

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales. Las mismas ideas son utilizadas en notación matemática tradicional; por ejemplo,

$$A+ (b+3) + \sqrt{c} \qquad +b + (b-5) + \sqrt{c}$$

Aquí los paréntesis indican el orden de cálculo y $\sqrt{}$ representa la función raíz cuadrada.

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas.

---

[3] Algunos lenguajes de programación admiten como válido el carácter subrayado en los identificadores.

Una expresión consta de *operandos* y *operadores*. Según sea el tipo de objetos que manipulan, las expresiones se clasifican en:

- *aritméticas,*
- *lógicas,*
- *carácter.*

El resultado de la expresión aritmética es de tipo numérico; el resultado de la expresión relacional y de una expresión lógica es de tipo lógico; el resultado de una expresión carácter es de tipo carácter.

1.6.1.Expresiones aritméticas

Las *expresiones aritméticas* son análogas a las fórmulas matemáticas. Las variables y constantes son numéricas (real o entera) y las operaciones son las aritméticas.

+	Suma
-	Resta
*	Multiplicación
/	División
-,**,^	Exponenciación
`Div`	División entera
`Mod`	Módulo (resto)

Los símbolos +,-,*,**,^(- o **) y las palabras clave `div` y `mod` se conocen como *operadores aritméticos*. En la expresión

```
5 + 3
```

los valores 5 y 3se denominan *operandos*. El valor de la expresión 5 +3 se conoce como *resultado* de la expresión.

Los operadores se utilizan de igual forma que en matemáticas. Por consiguiente, `A x B` se escribe en algoritmo como `A * B`, y `1/4 x C` como `C/4`. Al igual que en matemáticas el signo menos juega un doble papel, como resta en `A - B` y como cambio de signo en `- A`.

No todos los operadores aritméticos existen en todos los lenguajes de programación; por ejemplo, en FORTRAN no existe `div` ni `mod`.

El operador exponenciación es diferente según sea el tipo de lenguaje de programación elegido (^,- en BASIC, ** en FORTRAN).

Los cálculos que implican tipos de datos relaes y enteros suelen dar normalmente resultados del mismo tipo si los operandos lo son también. Por ejemplo, el producto de operandos reales produce un real (véase Tabla 1.1).

Tabla 1.1 Operadores aritméticos

Operador	Significado	Tipos de operandos	Tipo de resultado
-,^,**	Suma	Entero o real	Entero o real
+	Resta	Entero o real	Entero o real
-	Multiplicación	Entero o real	Entero o real

*	División	Entero o real	Entero o real
/	Exponenciación	Real	Real
Div	División entera	Entero	Entero
mod	Módulo (resto)	Entero	Entero

Ejemplos:

```
5 x 7      se representa por     5 *  7
6/4        se representa por     6/4
4
3          se representa por     3^7
```

Operadores DIV y MOD

El símbolo / se utiliza para la división real y el operador `div` - en algunos lenguajes, por ejemplo BASIC, se suele utilizar el símbolo \ - representa la divisón entera.

```
A  div  B
```

Sólo se puede utilizar si `A` y `B` son expresiones enteras y obtiene la parte entera de `A/B`. Por consiguiente,

```
19 div 6
```

toma el valor 3.

Otro ejemplo puede ser la división 15/6:

```
15  | 6
3     2   cociente
|
resto
```

En forma de operadoresresultará la operación anterior:

```
15 div   6 = 2       15  mod  6 = 3
```

Otros ejemplos son:

```
19 div  3 equivale a 6
19 mod 6  equivale a 1
```

Ejemplo 1.4

*Los siguientes ejemplos muestran resultados de expresiones aritméticas:*3.

Expresión	Resultado	Expresión	Resultado
10.5/3.0	3.5	10 **div** 3	3
1/4	0.25	18 **div** 2	9
2.0/4.0	0.5	30 **div** 30	1
6/1	6.0	6 **div** 8	0
30/30	1.0	10 **mod** 3	1
6/8	0.75	10 **mod** 2	0

1.6.1.1. Reglas de prioridad

Las expresiones que tienen dos o más operandos requieren unas reglas matemáticas que permitan determinar el orden de las operaciones, se denominan reglas de *prioridad* o *precedencia* y son:

1. Las operaciones que están encerradas entre paréntesis se evalúan primero. Si existen diferentes paréntesis animados (interiores unos a otros), las expresiones más internas se evalúan primero.
2. Las operaciones aritméticas dentrode una expresión suelen seguirel siguiente orden de prioridad:

   - operador exponencial (, ↑ o bien **).
   - Operadores *, /, \,
   - Operadores **div** y **mod**.
   - Operadores + ,-.

En caso de coincidir varios operadores de igual prioridad en una expresión o subexpresión encerrada entre paréntesis, el orden de prioridad en este caso es de izquierda a derecha.

Ejemplo 1.5.
¿Cuál es el resultado de las siguientes expresiones?

```
a) 3 + 6 * 14
a) 3 + 6 * 14
     3  +  84
        87
```

```
b) 8 + 7 * 3 + 4 * 6
b) 8 + 7 * 3 + 4 * 6
    8 +   21      24
       29      +   24
             53
```

Ejemplo 1.6
Obtener los resultados de las expresiones:

```
-4 * 7 + 2 ^ 3 / 4 - 5
1. -4  * 7 + 8 / 4 - 5
2. -28 + 8 / 4 - 5
3. -28 + 2 - 5
4. -26 - 5
   -31
```

Ejemplo 1.7
Convertir en expresiones aritméticas algorítmicas las siguientes expresiones algebraicas:

$5 \cdot (x + y)$

$a^2 + b^2$

$$\frac{x + y}{u + \frac{w}{u}}$$

$$\frac{x}{y} . (z + w)$$

Los resultados serán :

```
5 * (x + y)
a ^ 2 + b ^ 2
(x + y) / ( u + w / a)
x / y * ( z+ w)
```

Ejemplo 1.8
Los paréntesis tienen prioridad sobre el resto de las operaciones:

A * (B + 3)	*la constante 3 se suma primero al valor de B, después este resultado se multiplica por el valor de A.*
(A * B) + 3	*A y B se multiplican primero y a continuación se suma 3.*
A + (B + C) + D	*Esta expresión equivale a A + B + C + D.*
(A + B / C) +D	*Equivale a A - B / C+ D.*
A * B / C * D	*Equivale a ((A * B) / C) * D y no a (A * B) / (C * D).*

Ejemplo 1.9
Evaluar la expresión `12 + 3 * 7 + 5 *4.`

En este ejemplo existen dos operadores de igual prioridad, * (multiplicación); por ello los pasos sucesivos son:

```
12 + 3 * 7 + 5 * 4
     ------
       21

12 + 21 + 5 * 4
          -----
            20

12 + 21 + 20 = 53
```

1.6.2. Expresiones lógicas (booleanas)

Un segundo tipo de expresiones es la *expresión lógica o booleana*, cuyo valor es siempre verdadero o falso. Recuerde que existen dos constantes lógicas, *verdad (true)* y *falso (false)* y que las variables lógicas pueden tomar sólo estos dos valores. En esencia, una expresión lógica es una expresión que sólo puede tomar estos dos valores, *verdad* y *falso*. Se denominan también *expresiones booleanas* en honor del matemático británico George Boole, que desarrollo el Álgebra lógica de Boole.
Las expresiones lógicas se forman combinando constantes lógicas, variables lógicas y otras expresiones lógicas, utilizando los *operadores lógicos* **`not`**, **`and`** y **`or`**, y los *operadores relacionales* (de relación o comparación) =,<,>,<=,>=,<>.

1.6.2.1. Operadores de relación

Los operadores relacionales o de relación permiten realizar comparaciones de valores de tipo numérico o carácter. Los operadores de relación sirven para expresar las condiciones en los algoritmos.

Los operadores de relación se recogen en la Tabla 1.2.

El formato general para las comparaciones es:

expresión1	**OPERADOR DE RELACIÓN**	expresión2

Tabla 1.2. Operadores de relación

Operador	Significado
<	menor que
>	mayor que
=	igual que
<=	menor o igual que
>=	mayor o igual que
<>	distinto de

Y el resultado de la operación será verdadero o falso. Así, por ejemplo. Si A = 4 y B = 3. Entonces:

A > B es verdad

Mientras que:

(A – 2) > (B – 4) es falso

Los operadores de relación se pueden aplicar a cualquiera de los cuatro tipos de datos estándar: enteros, real, lógico, caracter.

La aplicación a valores numéricos es evidente. Los ejemplos siguientes sin significativos.

N1	N2	EXPRESIÓN LÓGICA	RESULTADO
3	6	3 < 6	Verdadero
0	1	0 > 1	Falso
4	2	4 = 2	Falso
8	5	8 < = 5	Falso
9	9	9 > = 9	Verdadero
5	5	5 < > 5	Falso

Para realizar comparaciones de datos tipo caracter, se requiere una secuencia de ordenación de los caracteres, similar al orden creciente o decreciente. Esta ordenación suele ser alfabética, tanto, mayúsculas como minúsculas y numérica, considerándolas de modo independiente. Pero si se consideran caracteres mixtos, se debe recurrir a un código normalizado como es el ASCII. Aunque todas las computadoras siguen el código normalizado en su juego completo de caracteres, si son prácticamente estándar los códigos de los caracteres alfanuméricos más usuales.

Estos códigos normalizados son:

- Los caracteres especiales #, %, $, (,), +, -, /, ... exigen la consulta del código de ordenación.
- Los valores de loscaracteres que representan a los dígitos están en su orden natural. Esto es, ‘0’ < ‘1’, ‘1’ < ‘2’ ..... ‘8’ < ‘9’
- Las letras mayúsculas A a Z siguen el orden alfabético (‘A’ < ‘B’, ‘C’ < ‘F’, etc.)
- Si existen letras minúsculas, éstas siguen el mismo criterio alfabético (‘a’ < ‘b’, ‘c’<’h’, etc.)

En general, los cuatro grupos anteriores están situados en el código ASCII en orden creciente. Así, ‘F’< ’A’ y ‘B’< ’C’. Sin embargo, para tener completa la seguridad será preciso consultar el código de caracteres de su computadora (normalmente, el ASCII (American Estándar Code for Información Interchange) o bien el EBCDIC (Extended Binary – Coded Decimal Interchange Code), utilizado en computadoras en computadoras IBM diferentes a los modelos PC y PC/2).

Cuando se utilizan los operadores de relación, con valores lógicos, la constante False (Falso) es menor que la constante True (Verdad).

False < True	Falso < Verdad
True > False	Verdad > Falso

Cuando se utilizan los operadores relacionales = y < > para comparar cantidades numéricas, es importante recordar que la mayoría de los valores reales no pueden ser almacenados exactamente. En consecuencia, las expresiones lógicas formales con comparación de cantidades reales con (=), a veces se evalúan como falsas, incluso aunque esas cantidades sean algebraicamente iguales. Así.

(1.0 / 3.0) * 3.0 = 1.0

Teóricamente es verdadera y, sin embargo, al realizar el cálculo en una computadora se puede obtener .999999... y, en consecuencia, el resultado es falso; esto es debido a la precisión limitada de la aritmética real en las computadoras. Por consiguiente, a veces deberá excluir las comparaciones con datos de tipo real.

OPERADORES LÓGICOS

Los operadores lógicos o boléanos básicos son not (no), and (y) y or (o). La tabla siguiente recoge el funcionamiento de dichos operadores:

OPERADORES LÓGICOS

OPERADOR LÓGICO	EXPRESIÓN LÓGICO	SIGNIFICADO
NO (NOT)	NO p (NOT p)	NEGACIÓN DE p
Y (AND)	p Y q (p AND q)	CONJUNCIÓN DE p Y q
O (OR)	p O q (p OR q)	DISYUNCIÓN DE p Y q

Las definiciones de las operaciones no, y, o se resumen en unas tablas conocidas como **Tablas de Verdad.**

a	**no a**		
Verdad	Falso		No (6 > 11) es verdad
Falso	Verdad		Ya que (6 > 11) es falso

a	**b**	**a y b**	
Verdad	Verdad	Verdad	a y b es verdad sólo
Verdad	Falso	Falso	si a y b son verdad
Falso	Verdad	Falso	
Falso	Falso	Falso	

a	**b**	**a o b**	a o b son verdad
Verdad	Verdad	Verdad	cuando a, b o ambas
Verdad	Falso	Verdad	son verdad
Falso	Verdad	Verdad	
Falso	Falso	Falso	

En las expresiones lógicas se pueden mezclar operadores de relación y lógicos. Así por ejemplo:

(1 < 6) y (6 < 10)	Es verdad
(6 > 10) o (A < B)	Es verdad, ya que A < B

PRIORIDAD DE LOS OPERADORES EN LAS EXPRESIONES LÓGICAS

Los operadores aritméticos seguían un orden específico de prioridad cuando existían más de un operador en las expresiones. De modo similar, los operadores lógicos y relaciones, tienen un orden de prioridad.

EXPRESIONES LÓGICAS

EXPRESIÓN LÓGICA	RESULTADO	OBSERVACIONES
(1 > 0) Y (3 = 3)	VERDAD	
NO PRUEBA	VERDAD	• PRUEBA es un valor lógico Falso
(0 < 5) O (0 > 5)	VERDAD	
(5 < = 7) Y (2 > 4)	FALSO	
NO (5 < > 5)	VERDAD	
(NÚMERO = 1) O (7 > = 4)	VERDAD	• NÚMERO es una variable entera de valor 5

PRIORIDAD DE OPERADORES (LENGUAJE PASCAL)

EXPRESIÓN LÓGICA	PRIORIDAD
NO (NOT)	MÁS ALTA (PRIMERA OPERACIÓN EJECUTADA)
/,*, DIV, MOD, Y (AND)	
-, +, O (OR)	
<, >, =, < =, > =, < >	MAS BAJA (ÚLTIMA OPERACIÓN EJECUTADA)

PRIORIDAD DE OPERADORES (LENGUAJE BASIC)

EXPRESIÓN LÓGICA	PRIORIDAD
/	MÁS BAJA
*	
DIV ( )	
MOD	
-, +	
<, >, =, < >, < =, > =	
NO (NOT)	
Y (AND)	
O (OR)	MÁS ALTA

Al igual que en las expresiones aritméticas, los paréntesis se pueden utilizar y tendrán prioridad sobre cualquier operación.

Ejemplo:

NO 4 > 5	Produce un error, ya que el operador no se aplica a 4
NO (4 > 14)	Produce un valor verdadero
(1.0 < X) Y (X < Z – 7.0)	Si X vale 7 y Z vale 4, se obtiene un valor verdadero.

FUNCIONES INTERNAS

Las operaciones internas que se requieren en los programas exigen numerosas ocasiones, además de las operaciones de las operaciones aritméticas básicas, ya tratadas, un número determinado de operaciones especiales que se denominan Funciones Internas, incorporadas o estándar. Por ejemplo, la función ***1n*** se puede utilizar para determinar el logaritmo neperiano de un número y la función ***raíz2 (sqrt)*** calcular la raíz cuadrada de un número positivo. Existen otras funciones que se utilizan para determinar las funciones geométricas.

FUNCIONES INTERNAS

FUNCIÓN	DESCRIPCIÓN	TIPO DE ARGUMENTO	RESULTADO
Abs (x)	Valor Absoluto de x	Entero o real	Igual que argumento
Arctan (x)	Arco Tangente de x	Entero o real	Real
Cos (x)	Coseno de x	Entero o real	Real
Exp (x)	Exponencial de x	Entero o real	Real
1n (x)	Logaritmo Neperiano de x	Entero o real	Real
Log10 (x)	Logaritmo Decimal de x	Entero o real	Real
Redondeo (x) (round (x))	Redondeo de x	Real	Entero
Sen (x) (sin (x))	Seno de x	Entero o real	Real
Cuadrado (x) (sqr (x))	Cuadrado de x	Entero o real	Igual que argumento
Raiz2 (x) (sqrt (x))	Raíz cuadrada de x	Entero o real	Real
Trunc (x)	Truncamiento de x	real	Entero

Las funciones aceptan argumentos reales o enteros y sus resultados dependen de la tarea que realice la función:

EXPRESIÓN	RESULTADO
Raiz2 (25)	5
Redondeo (6.6.)	7
Redondeo (3.1.)	3
Redondeo (-3.2)	-3
Trunc (5.6)	5
Trunc (3.1)	3
Trunc (-3.8)	-3
Cuadrado (4)	16
Abs (9)	9
Abs (-12)	12

Ejemplo
Utilizar las funciones internas para obtener la solución de la ecuación cuadrática:
$ax^2 + bx + c = 0$; las raíces de la ecuación son:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

O lo que es igual:

$$x1 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad x2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Las expresiones se escriben como:

x1 = (-b raiz2 (cuadrado (b) – 4 * a * c)) / (2 * a)
x2 = (-b raiz2 (cuadrado (b) – 4 * a * c)) / (2 * a)

LA OPERACIÓN DE ASIGNACIÓN

La operación de asignación es el modo de darle valores a una variable. La operación de asignación se representa con el símbolo u operador ⇓ . La operación de asignación se conoce como instrucción o **sentencia de asignación** cuando se refiere a un lenguaje de programación.
El formato general de una operación es:
Nombre de la variable ⇓ expresión

La flecha (operador de asignación) se sustituye en otros lenguajes por = (BASIC, FORTRAN) o : = (PASCAL) C/C++. Sin embargo, es preferible el uso de la flecha en la redacción del algoritmo para evitar ambigüedades, dejando el uso del símbolo = exclusivamente para el operador de igualdad.

La operación de asignación:

A ⇓ 5 Significa que a la variable A se le ha asignado el valor 5

La acción de asignar es destructiva, ya que el valor que tuviera la variable antes de la asignación se pierde y se reemplaza por el nuevo valor. Así en la secuencia de operaciones:

A ⇓ 25
A ⇓ 134
A ⇓ 5

Cuando estas se ejecutan, el valor último que toma A será 5 (los valores 25 y 134 han desaparecido)

La computadora ejecuta la sentencia de asignación en dos pasos. En el primero de ellos se calcula el valor de la expresión al lado derecho del operador, obteniéndose un valor de un tipo específico. En el segundo paso, este valor se almacena en la variable cuyo nombre aparece a la izquierda del operador de asignación, sustituyendo al valor que tenía anteriormente.

X ⇓ Y – 2

El valor de la expresión Y – 2, se asigna a la variable X

Es posible utilizar el mismo nombre de la variable en ambos lados del operador de asignación. Por ello, acciones como:

N ⇓ N – 1

Las acciones de asignación se clasifican según sea el tipo de expresiones en aritmética, lógicas y de caracteres.

ASIGNACIÓN ARITMÉTICA

Las expresiones en las operaciones de asignación son aritméticas:

AMN ⇓ 3 + 14 + 8 Se evalúa la expresión 3 + 14 + 8 y se asigna a la variable AMN, es decir, 25 será el valor que toma AMN

TER1 ⇓ 14.5 + 8
TER2 ⇓ 0.75 * 3.4
COCIENTE ⇓ TER1 / TER2

Se evalúan las expresiones 14.5 + 8 y 0.75 * 3.4 y en la tercera acción se dividen los resultados de cada expresión y se asigna a la variable COCIENTE, es decir, las tres operaciones equivalen a COCIENTE ⇓ (14.5 + 8) / (0.75 * 3.4).

Otro ejemplo donde se pueden comprender las modificaciones de los valores almacenados en una variable es el siguiente:

A ⇓ 0 La variable A toma el valor 0
N ⇓ 0 La variable N toma el valor 0
A ⇓ N + 1 La variable A toma el valor 0 + 1, es decir, 1

El ejemplo anterior se puede modificar para considerar la misma variable en ambos lados del operador de asignación:

N ⇓ 2
N ⇓ N –1

En la primera acción N toma el valor 2 y en la segunda acción se evalúa la expresión N+1, que tomará el valor 2 + 1 = 3 y se asignará nuevamente a N, que tomará el valor 3.

ASIGNACIÓN LÓGICA

La expresión que se evalúa en la operación de asignación es lógica. Supóngase que M, N y P son variables de tipo lógico.

M ⇓ 8 < 5
N ⇓ M o (7 < = 12)
P ⇓ 7 > 6

Tras evaluar las operaciones anteriores, las variables M, N y P tomarán los valores falso, verdad.

ASIGNACIÓN DE CADENAS DE CARACTERES

La expresión que se evalúa es de tipo cadena:

X ⇓ 12 de octubre de 1492

La acción de asignación anterior asigna la cadena de caracteres - 12 de octubre de 1492 – a la variable tipo cadena X

CONVERSIÓN DE TIPO

En las asignaciones no se pueden asignar valores a una variable de un tipo diferente del suyo. Se presentará un error si se trata de asignar valores de tipo caracter a una variable numérica o un valor numérico a una variable de tipo caracter.

Ejemplo 1.14

¿Cuáles son los valores de A, B y C después de la ejecución de las siguientes operaciones?

A • 3
B • 4
C • A + 2 * B
C • C + B
B • C – A
A • B * C

En las dos primeras acciones A y B toman los valores de 3 y 4.

C • A + 2 * B LA EXPRESIÓN A + 2 * B TOMARA EL VALOR 3 + 2 * 4 = 3 + 8 = 11
C • 11

La siguiente acción

C • C + B

Producirá un valor de 11 + 4 = 15

C • 15

En la acción B • C – A se obtiene para B el valor 15 – 3 = 12 y por último:

A • B * C

A tomará el valor B * C , es decir, 12 * 15 = 180 ; por consiguiente; el último valor que toma A sera 180.

Ejemplo 1. 15

¿Cuál es el valor de X después de las siguientes expresiones?

```
X • 2
X • cuadrado (X + X)
X • raiz2 (X + raiz2(X) + 5)

Los resultados de cada expresión son:

X • 2                      X toma el valor 2
X • cuadrado (2 + 2)       X toma el valor4 al cuadrado; es decir, 16
X • raiz2 (16 + raiz2(16) + 5)
```

en esta expresión se evalúa primero **raiz2 (16)** , que produce 4 y por último, **raiz2(16+4+ 5)** proporciona **raiz2 (25)** , es decir, 5.

Los resultados de las expresiones sucesivas anteriores son:

X • 2
X • 16
X • 5

1.9. ENTRADA Y SALIDA DE INFORMACIÓN

Los cálculos que realizan las computadoras requieren para ser útiles la entrada de los datos necesarios para ejecutar las operaciones que posteriormente se convertirán en resultados, es decir, salida.

Las operaciones de entrada permiten leer determinados valores y asignarlos a determinadas variables. Esta entrada se conoce como operación de **lectura** (read). Los datos de entrada se introducen al procesador mediante dispositivos de entrada (teclado, tarjetas perforadas, unidades de disco, etc.). La salida puede aparecer en un dispositivo de salida (pantalla, impresora, etc.). La operación de salida se denomina **escritura** (write).

En la escritura de algoritmos las acciones de lectura y escritura se representan por los formatos siguientes:

leer (lista de variable de entrada)
escribir (lista de expresiones de salida)

Así por ejemplo

leer (A, B, C)

representa la lectura de tres valores de entrada que se asignan a las variables A, B y C.

escribir (hola Vargas)

visualiza en la pantalla –o escribe en el dispositivo de salida– el mensaje ‘ hola Vargas ‘

NOTA 1: si se utilizaran las palabras reservadas en ingles, como suele ocurrir en los lenguajes de programación se deberá sustituir

leer **escribir**

Por

read **write** o bien **print**

NOTA 2: Si no se especifica el tipo de dispositivo del cual se leen o escriben datos, los dispositivos de E/S por defecto son del teclado y la pantalla.

ATIVIDADES DE PROGRAMACIÓN RESUELTAS

1.1 Diseñar un algoritmo para cambiar una rueda de un coche.

Algoritmo pinchazo

1. Inicio.
2. si gato del coche esta averiado llamar a la estación y bifurcar al punto 11; en caso contrario, continuar con el proceso.
3. Levantar el coche con el gato.
4. Aflojar y sacar los tornillos de las ruedas.
5. Si NO todos los tornillos están flojos y quitados, bifurcar al paso 4; en caso contrario, continuar el proceso.
6. Quitar la rueda.
7. Poner rueda de repuesto.
8. Poner los tornillos y apretarlos.
9. Si NO están puestos todos los tornillos volver al paso 8, en caso contrario, continuar el proceso.
10. Bajar el gato.
11. Fin.

1.2. Encontrar el valor de la variable VALOR después de la ejecución de las siguientes expresiones:

```
(a)  VALOR ← 4.0 * 5
(b)  X ← 3.0
     Y ← 2.0
     VALOR ← X ^ Y - Y
(c)  VALOR ← 5
     X ← 3
     VALOR ← VALOR * X
```

```
(a)  VALOR = 20.0
(b)  X = 3.0
     Y = 2.0
     VALOR = 3 ^ 2 - 2 = 9-2 = 7    VALOR = 7
(c)  VALOR = 5
     X = 3
     VALOR = VALOR * X = 5 * 3 = 15 VALOR =15
```

1.3. Deducir el resultado que se produce con las siguientes instrucciones:

```
var entero: X, Y
X ← 1
Y ← 5
Escribir (X, Y)
```

```
X e Y toman los valores 1 y 5
```

La instrucción de salida (`escribir`) presentará en el dispositivo de salida 1 y 5, con los formatos específicos del lenguaje de programación; por ejemplo,

```
1     5
```

1.4. Deducir el valor de las expresiones siguientes:

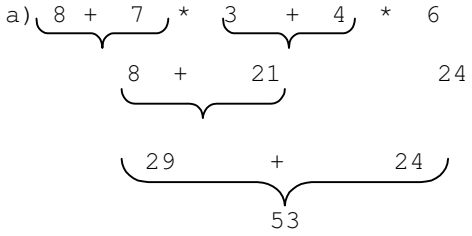
```
X ← A + B + C
X ← A + B * C
X ← A + B / C
```

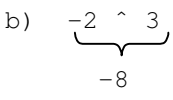
X • A + B \ C
X • A + B **mod** C
X • (A + B) / C
X • A + (B / C)
Siendo A = 5 B = 25 C = 10

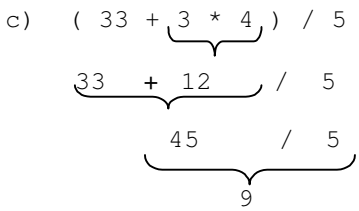
Expresión	X
A + B + C = 5 + 25 + 10	40
A + B * C = 5 + 25 * 10	255
A + B / C = 5 + 25 / 10	7.5
A + B \ C = 5 \ 25 + 10 = 5 + 2	7
A + B **mod** C = 5 + 25 **mod** 10 = 5 + 5	10
(A + B) / C = (5 + 25) / 10 = 30 / 10	3
A + (B / C)= 5 + (25 + 10) = 5 + 2.5	7.5

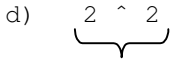
1.5 Calcular el valor de las siguientes expresiones:

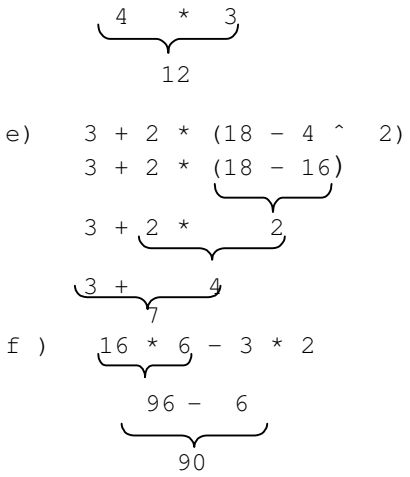
a) 8 + 7 * 3 + 4 * 6
b) -2 ^ 3
c) (33 + 3 * 4) / 5
d) 2 ^ 2 * 3
e) 3 + 2 * (18 - 4 ^ 2)
f) 16 * 6 – 3 * 2











1.6 Cómo se intercambian los valores de dos variables, A y B.

El procedimiento para conseguir intercambiar los valores de dos variables entre sí debe recurrir a una variable AUX y a las instrucciones siguientes asignadas

```
AUX • A
  A • B
  B • AUX
```

Veámoslo con el ejemplo del ejercicio, donde

A • 10

Instrucción	A	B	AUX	Observaciones
A • 10	10	–	–	
B • 5	10	5	–	
AUX • A	10	5	10	la variable AUX toma el valor de A
A • B	5	5	10	A toma el valor de B, 5
B • AUX	5	10	10	B toma el valor inicial de A, 10

Ahora A = 5 y B = 10

1.7 Deducir el valor que toma la variable tras la ejecución de las instrucciones:

```
A • 4
B • A
B • A + 3
```

Mediante una tabla se da un método eficaz para obtener los sucesivos valores:

		A	B
(1)	A • A	A	–
(2)	B • A	4	4
(3)	B • A + 3	4	7

Después de la instrucción (1) A contiene el valor 4.

La variable B no ha tomado todavía ningún valor y se presenta esa situación con un guión.
La instrucción (2) asigna el valor actual de A (4) a la variable B.
La instrucción (3) efectúa el cálculo de de la expresión a + 3 , lo que produce un resultado de (4+ 3) y este valor se asigna a la variable B, cuyo último valor (4) se destruye.
Por consiguiente, los valores finales que tiene la variable A y B son:

A = 4 B = 7

1.7 ¿Qué se obtiene de las variables A y B, después de la ejecución de las siguientes instrucciones?

A • 5
B • A + 6
A • A + 1
B • A – 5

Siguiendo la dirección del ejercicio anterior :

	Instrucción			Observaciones
(1)	A • 5	5	-	B no toma ningún valor
(2)	B • A + 6	5	11	Se evaluá A + 6 (5 + 6) y se asigna a B
(3)	A • A + 1	6	11	Se evalúa A + 1 (5 + 1) y se asigna a A borrándose el valor que tenía (5) y tomando el nuevo valor (6)
(4)	B • A – 5	6	1	Se evalúa A – 5 (6 - 1) y se asigna a B

Los valores últimos de A y B son : A = 6, B = 1

1.8 ¿Qué se obtiene en las variables A, B y C después de ejecutar las instrucciones siguentes?

A • 3
B • 20
C • A + B
B • A + B
A • B – C

Instrucción		A	B	C	Observación
(1)	A • 3	3	-	-	B y C no toman ningún valor
(2)	B • 20	3	20	-	C sigue sin valor
(3)	C • A + B	3	20	23	Se evalúa A + B (20 + 3) y se asigna a C
(4)	B • A + B	3	23	23	S evalúa A + B (20 + 3) y se asigna a B; destruye el valor antiguo (20)
(5)	A • B – C	0	23	23	Se evalúa B – C (23 – 23) y se asigna a A

Los valores finales de las variables son:

A = 0 B = 23 C = 23

1.9 ¿Qué se obtiene en A y B tras la ejecución de :

A • 10
B • 5
A • B
B • A

Instrucción		A	B	Observación
(1)	A • 10	10	-	B no toma valor
(2)	B • 5	10	5	B recibe el valor inicial 5
(3)	A • B	5	5	A toma el valor de B (5)
(4)	B • A	5	5	B toma el valor actual de A (5)

Los valores finales de A y B son 5
En este caso se podría decir que la instrucción (4) B • A es *redundante* respecto a las anteriores, ya que su ejecución no afecta al valor de las variables.

1.10 Determinar el mayor de tres números enteros
Los pasos a seguir son:

1. Comparar el primer y el segundo entero, deduciendo cual es el mayor.
2. Comparar el mayor anterior con el tercero y deducir cuál es el mayor. Este será el resultado.

Los pasos anteriores se pueden descomponer en otros pasos más simples en lo que se denomina *refinamiento del algoritmo.*

1. Obtener el primer número (entrada), denominarlo NUM1
2. Obtener el segundo número (entrada), denominarlo NUM2
3. Comparar NUM1 con NUM2 y seleccionar el mayor ; si los dos enteros son iguales, seleccionar NUM1. Llamar a este número MAYOR.
4. Obtener el tercer número (entrada) y denominarlo NUM3
5. Comparar MAYOR con NUM3 y seleccionar el mayor; si los dos enteros son iguales, seleccionar el MAYOR. Denominar a este número MAYOR
6. Presentar el valor de MAYOR (salida)
7. Fin

EJERCICIOS

1.1. Diseñar los algoritmos que resuelvan los siguientes problemas :

a) Ir al cine
b) Comprar una entrada para los toros
c) Colocar la mesa para comer
d) Cocer un huevo
e) Hacer una taza de té
f) Fregar los platos del almuerzo
g) Buscar el número de teléfono de un alumno
h) Reparar el pinchazo de una bicicleta
i) Pagar una multa de trafico
j) Cambiar un neumático pinchado(se dispone de herramientas y gata)
k) Hacer palomitas de maíz en una olla puesta al fuego con aceite, sal y maíz
l) Cambiar el cristal rito de una ventana
m) Hacer una llamada telefónica. Considerar los casos: a) manual, con operadora; b) automático; c) cobro revertido.
n) Quitar una bombilla quemada de un techo.
o) Encontrar la media de una lista indeterminada de números positivos teminada con un número negativo.

1.2. ¿ Cuál de las siguientes identificadores no son válidos ?

a) Xrayo b) X_Rayo
c) R2D2 d) X
e) 45 f) N14
g) ZZZZ h) 3u

1.3. ¿ Cuál de las siguientes constantes son válidas ?

a) 234 b) – 8.975
c) 12E – 5 d) 0
e) 32,767 f) ½
g) 3.6E+7 h) – 7E12
I) 3.5 x 10 j) 0,456
k) 0.000001 l) 224E1

1.4 Evaluar las siguientes expresiones para A = 2 y B = 5:

3 * A – 4 * B / A * 2

1.5 Evaluar la expresión

4 / 2 * 3 / 6 - 6 / 2 / 1 / 5 * 2 / 4 * 2

1.6 Encontrar el valor de cada una de las siguintes expresiones o decir si no es una expresión válida:

a) 9 – 5 – 3
b) 2 div 3 + 3/5
c) 9 div 2/5
d) 7 mod 5 mod 3
e) 7 mod (5 mod 3)
f) (7 mod 5) mod 3
g) (7 mod 5 mod 3)
h) ((12 + 3) div 2) / (8 – (5 + 1))
i) 12 / 2 * 3
j) raíz 2 (cuadrado (4))
k) cuadrado (raíz 2 (4))
l) trunc (81.5) + redondeo (81.5)