1
ALGORITMOS Y PROGRAMAS

La principal raz·n para que las personas aprendan lenguajes de programaci·n es utilizar la computadora como una herramienta en la resoluci·n de problemas. La resoluci·n de un problema exige al menos los siguientes pasos:

1. Definici·n o an§lisis del problema.
2. Dise¶o del algoritmo o m®todo para resolverlo.
3. Transformaci·n del algoritmo en un programa.
4. Ejecuci·n y validaci·n del programa.

Uno de los objetivos fundamentales de este libro es el aprendizaje y dise¶o de algoritmos. Este cap²tulo introduce al lector en el concepto de algoritmo y programa.

1.1. CONFIGURACIÓN DE UNA COMPUTADORA

Una computadora es un dispositivo electrónico utilizado para procesar información y obtener resultados. Los componentes físicos que constituyen la computadora, junto con los dispositivos que realizan las tareas de entrada y salida, se conocen con el término *hardware*. En su configuración física más básica una computadora está constituida por una Unidad Central de Proceso (**UCP**)*, una Memoria Principal y unas Unidades de Entrada y Salida. Las funciones desempeñadas por cada una de estas partes son las siguientes:

- Las unidades de Entrada/Salida se encargan de los intercambios de información con el exterior.
- La memoria principal almacena tanto las instrucciones que constituyen los programas como los datos y resultados obtenidos; para que un programa se ejecute, tanto él como sus datos, se deben situar en memoria central. Sin embargo, la información almacenada en la memoria principal se pierde cuando se desconecta la computadora de la red eléctrica y, además, dicha memoria es limitada en capacidad. Por estas razones, los programas y datos necesitan ser transferidos a *dispositivos de almacenamiento secundario,* que son dispositivos periféricos que actúan como medio de soporte permanente. La memoria caché es una memoria intermedia, de acceso aleatorio muy rápida entre la UCP y la memoria principal, que almacena los datos extraídos más frecuentemente de la memoria principal.

* **CPU**, Central Processing Unit.

- La UCP es la encargada de la ejecución de los programas almacenados en memoria. La UCP consta de Unidad de Control (UC), que se encarga de ejecutar las instrucciones, y la Unidad Aritmético Lógica (UAL), que efectúa las operaciones.

El *bus del sistema* es una ruta eléctrica de múltiples líneas —clasificables como líneas de direcciones, datos y control— que conecta la UCP, la memoria y los dispositivos de entrada/salida. Los *dispositivos de entrada/salida* son heterogéneos y de características muy variadas; por esta razón cada dispositivo o grupo de ellos cuenta *con controladores* específicos que admiten órdenes e instrucciones muy abstractas que les puede enviar la UCP y se encargan de llevarlas a cabo generando microórdenes para gobernar a los periféricos y liberando a la UCP de estas tareas; en síntesis, un *controlador* es un programa que enlaza un dispositivo de la computadora y el sistema operativo que controla la misma. Los dispositivos de entrada/salida pueden producir *interrupciones*, que son sucesos que se producen inesperadamente pero a los que generalmente se debe atender inmediatamente, por lo que la UCP abandona momentáneamente las tareas que estaba ejecutando para atender a la interrupción.

En cuanto a las operaciones que debe realizar el hardware, éstas son especificadas por una lista de instrucciones, llamadas *programas*, o *software*. El software se divide en dos grandes grupos: software del sistema y software de aplicaciones. *El software del sistema* es el conjunto de programas indispensables para que la máquina funcione, se denominan también programas del sistema, mientras que los programas que realizan tareas concretas, nóminas, contabilidad, análisis estadístico, etc. se denominan programas de aplicación o *software de aplicaciones*.

1.2. LENGUAJES DE PROGRAMACIÓN

Un programa se escribe en un lenguaje de programación y los lenguajes utilizados se pueden clasificar en:

- Lenguajes máquina.
- Lenguajes de bajo nivel (ensamblador).
- Lenguajes de alto nivel.

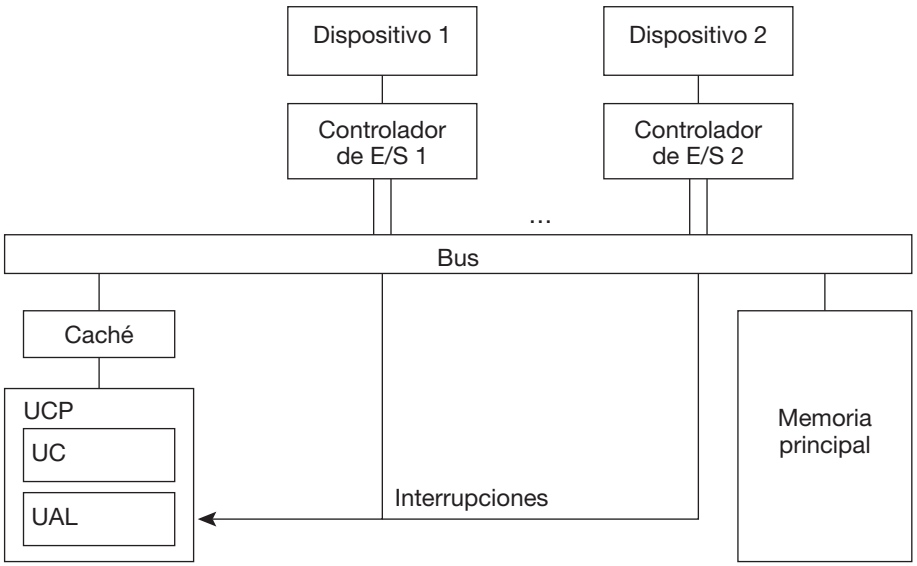

Figura 1.1. Organización física de una computadora.

Los *lenguajes máquina* proporcionan instrucciones específicas para un determinado tipo de *hardware* y son directamente inteligibles por la máquina.

El *lenguaje ensamblador* se caracteriza porque sus instrucciones son mucho más sencillas de recordar, aunque dependen del tipo de computadora y necesitan ser traducidas a lenguaje máquina por un programa al que también se denomina ensamblador.

Los *lenguajes de alto nivel* proporcionan sentencias muy fáciles de recordar, que no dependen del tipo de computadora y han de traducirse a lenguaje máquina por unos programas denominados compiladores o intérpretes. Los programas escritos en un lenguaje de alto nivel se llaman programas fuente y el programa traducido programa objeto o código objeto. El código objeto queda ligado al hardware y al sistema operativo. Casos especiales en cuanto a esto son *Java* y los lenguajes soportados por *.NET Framework*. En Java el archivo que se genera al compilar se llama *bytecode* y puede ejecutarse en cualquier plataforma donde esté instalado un determinado software denominado máquina virtual (*Java Virtual Machine*). La máquina virtual interpreta el *bytecode* para la plataforma, hardware y sistema operativo, en el que se está ejecutando el programa. Los lenguajes soportados por *.NET Framework*, como *C#*, se comportan de forma similar a Java y compilan por defecto a un *código intermedio*, denominado *Intermediate Language* (IL*Common Language Runtime*, CLR).

1.3. RESOLUCIÓN DE PROBLEMAS

Dos fases pueden ser identificadas en el proceso de creación de un programa:

- Fase de resolución del problema.
- Fase de implementación (realización) en un lenguaje de programación.

La fase de resolución del problema implica la perfecta comprensión del problema, el diseño de una solución conceptual y la especificación del método de resolución detallando las acciones a realizar mediante un algoritmo.

1.3.1. Fase de resolución del problema

Esta fase incluye, a su vez, el análisis del problema así como el diseño y posterior verificación del algoritmo.

1.3.1.1. Análisis del problema

El primer paso para encontrar la solución a un problema es el análisis del mismo. Se debe examinar cuidadosamente el problema a fin de obtener una idea clara sobre lo que se solicita y determinar los datos necesarios para conseguirlo.

1.3.1.2. Diseño del algoritmo

La palabra algoritmo deriva del nombre del famoso matemático y astrónomo árabe **Al-Khôwarizmi** (siglo IX) que escribió un conocido tratado sobre la manipulación de números y ecuaciones titulado *Kitab al-jabr w'almugabala*.

Un algoritmo puede ser definido como la secuencia ordenada de pasos, sin ambigüedades, que conducen a la solución de un problema dado y puede ser expresado en lenguaje natural, por ejemplo el castellano.

Todo algoritmo debe ser:

- **Preciso**. Indicando el orden de realización de cada uno de los pasos.
- **Definido.** Si se sigue el algoritmo varias veces proporcionándole los mismos datos, se deben obtener siempre los mismos resultados.

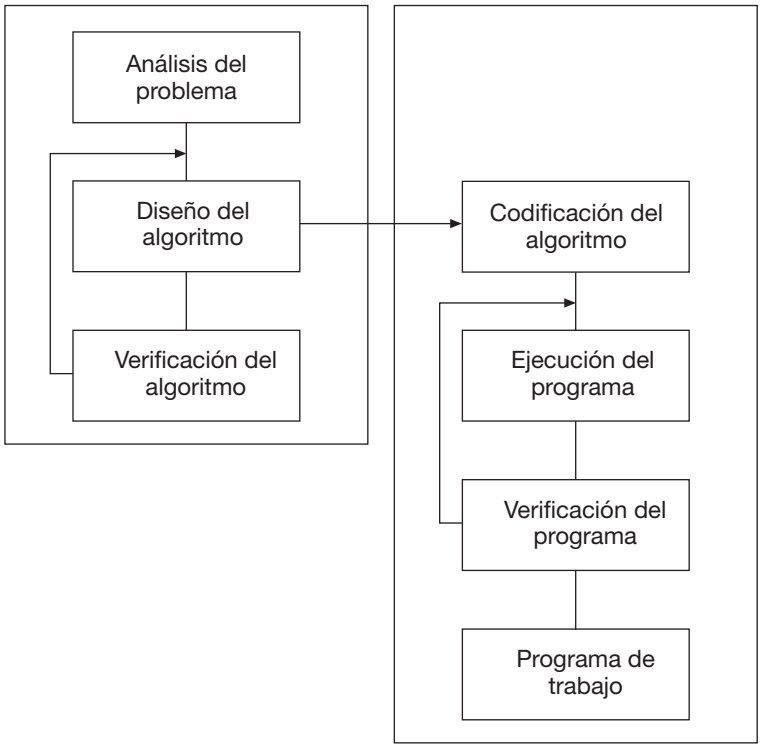

Figura 1.2. Resolución de problemas por computadora.

- **Finito**. Al seguir el algoritmo, éste debe terminar en algún momento, es decir tener un número finito de pasos.

Para diseñar un algoritmo se debe comenzar por identificar las tareas más importantes para resolver el problema y disponerlas en el orden en el que han de ser ejecutadas. Los pasos en esta primera descripción de actividades deberán ser refinados, añadiendo más detalles a los mismos e incluso, algunos de ellos, pueden requerir un refinamiento adicional antes de que podamos obtener un algoritmo claro, preciso y completo. Este método de diseño de los algoritmos en etapas, yendo de los conceptos generales a los de detalle a través de refinamientos sucesivos, se conoce como método descendente (top-down).

En un algoritmo se deben de considerar tres partes:

- **Entrada.** Información dada al algoritmo.
- **Proceso.** Operaciones o cálculos necesarios para encontrar la solución del problema.
- **Salida**. Respuestas dadas por el algoritmo o resultados finales de los cálculos.

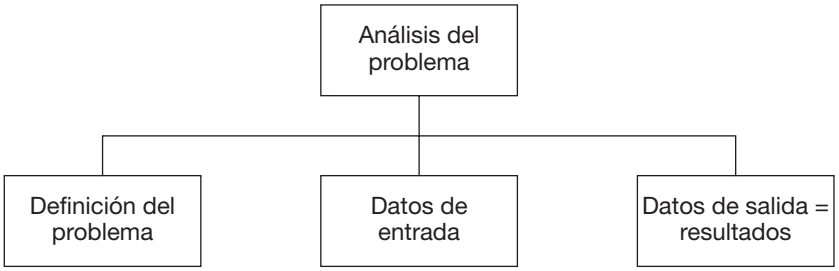

Figura 1.3. Análisis del problema.

Como ejemplo imagine que desea desarrollar un algoritmo que calcule la superficie de un rectángulo proporcionándole su base y altura. Lo primero que deberá hacer es plantearse y contestar a las siguientes preguntas:

Especificaciones de entrada

¿Qué datos son de entrada?
¿Cuántos datos se introducirán?
¿Cuántos son datos de entrada válidos?

Especificaciones de salida

¿Cuáles son los datos de salida?
¿Cuántos datos de salida se producirán?
¿Qué precisión tendrán los resultados?
¿Se debe imprimir una cabecera?

El algoritmo en el primer diseño se podrá representar con los siguientes pasos:

Paso 1. Entrada desde periférico de entrada, por ejemplo teclado, de base y altura.
Paso 2. Cálculo de la superficie, multiplicando la base por la altura.
Paso 3. Salida por pantalla de base, altura y superficie.

El lenguaje algorítmico debe ser independiente de cualquier lenguaje de programación particular, pero fácilmente traducible a cada uno de ellos. Alcanzar estos objetivos conducirá al empleo de métodos normalizados para la representación de algoritmos, tales como los diagramas de flujo, diagrama Nassi-Schneiderman o pseudocódigo, comentados más adelante.

1.3.1.3. Verificación de algoritmos

Una vez que se ha terminado de escribir un algoritmo es necesario comprobar que realiza las tareas para las que se ha diseñado y produce el resultado correcto y esperado.

El modo más normal de comprobar un algoritmo es mediante su ejecución manual, usando datos significativos que abarquen todo el posible rango de valores y anotando en una hoja de papel las modificaciones que se producen en las diferentes fases hasta la obtención de los resultados. Este proceso se conoce como prueba del algoritmo.

1.3.2. Fase de implementación

Una vez que el algoritmo está diseñado, representado mediante un método normalizado (diagrama de flujo, diagrama N-S o pseudocódigo), y verificado se debe pasar a la fase de codificación, traducción del algoritmo a un determinado lenguaje de programación, que deberá ser completada con la ejecución y comprobación del programa en la computadora.

1.4. EJERCICIOS RESUELTOS

Desarrolle los algoritmos que resuelvan los siguientes problemas:

1.1. *Ir al cine.*

Análisis del problema

DATOS DE SALIDA: Ver la película
DATOS DE ENTRADA: Nombre de la película, dirección de la sala, hora de proyección
DATOS AUXILIARES: Entrada, número de asiento

Para solucionar el problema, se debe seleccionar una película de la cartelera del periódico, ir a la sala y comprar la entrada para, finalmente, poder ver la película.

Diseño del algoritmo

```
inicio
   //seleccionar la película
   tomar el periódico
   mientras no llegemos a la cartelera
      pasar la hoja
   mientras no se acabe la cartelera
      leer película
      si nos gusta, recordarla
   elegir una de las películas seleccionadas

   leer la dirección de la sala y la hora de proyección

   //comprar la entrada
   trasladarse a la sala
   si no hay entradas, ir a fin
   si hay cola
      ponerse el último
      mientras no lleguemos a la taquilla
         avanzar
         si no hay entradas, ir a fin
   comprar la entrada

   //ver la película
   leer el número de asiento de la entrada
   buscar el asiento
   sentarse
   ver la película
fin
```

1.2. *Comprar una entrada para ir a los toros.*

Análisis del problema

DATOS DE SALIDA: La entrada
DATOS DE ENTRADA: Tipo de entrada (sol, sombra, tendido, andanada, etc.)
DATOS AUXILIARES: Disponibilidad de la entrada

Hay que ir a la taquilla y elegir la entrada deseada. Si hay entradas se compra (en taquilla o a los reventas). Si no la hay, se puede seleccionar otro tipo de entrada o desistir, repitiendo esta acción hasta que se ha conseguido la entrada o el posible comprador ha desistido.

Diseño del algoritmo

```
inicio
   ir a la taquilla
   si no hay entradas en taquilla
      si nos interesa comprarla en la reventa
         ir a comprar la entrada
      si no ir a fin
   //comprar la entrada

   seleccionar sol o sombra
   seleccionar barrera, tendido, andanada o palco
   seleccionar número de asiento
   solicitar la entrada
   si la tienen disponible
      adquirir la entrada
      si no
         si queremos otro tipo de entrada
            ir a comprar la entrada
fin
```

1.3. *Poner la mesa para la comida.*

Análisis del problema

DATOS DE SALIDA:	La mesa puesta
DATOS DE ENTRADA:	La vajilla, los vasos, los cubiertos, la servilletas, el número de comensales
DATOS AUXILIARES:	Número de platos, vasos, cubiertos o servilletas que llevamos puestos

Para poner la mesa, después de poner el mantel, se toman las servilletas hasta que su número coincide con el de comensales y se colocan. La operación se repetirá con los vasos, platos y cubiertos.

Diseño del algoritmo

```
inicio
   poner el mantel
repetir
   tomar una servilleta
hasta que el número de servilletas es igual al de comensales

repetir
   tomar un vaso
hasta que el número de vasos es igual al de comensales

repetir
   tomar un juego de platos
hasta que el número de juegos es igual al de comensales
```

```
repetir
   tomar un juego de cubiertos
hasta que el número de juegos es igual al de comensales

fin
```

1.4. *Hacer una taza de té.*

Análisis del problema

DATOS DE SALIDA: Taza de té
DATOS DE ENTRADA: Bolsa de té, agua
DATOS AUXILIARES: Pitido de la tetera, aspecto de la infusión

Después de echar agua en la tetera, se pone al fuego y se espera a que el agua hierva (hasta que suena el pitido de la tetera). Introducimos el té y se deja un tiempo hasta que está hecho.

Diseño del algoritmo

```
inicio
   tomar la tetera
   llenarla de agua
   encender el fuego
   poner la tetera en el fuego
   mientras no hierva el agua
      esperar
   tomar la bolsa de té
   introducirla en la tetera
   mientras no esté hecho el té
      esperar
   echar el té en la taza
fin
```

1.5. *Fregar los platos de la comida.*

Análisis del problema

DATOS DE SALIDA: Platos limpios
DATOS DE ENTRADA: Platos sucios
DATOS AUXILIARES: Número de platos que quedan

Diseño del algoritmo

```
inicio
   abrir el grifo
   tomar el estropajo
   echarle jabón
   mientras queden platos
      lavar el plato
      aclararlo
      dejarlo en el escurridor
   mientras queden platos en el escurridor
      secar plato
fin
```

1.6. *Reparar un pinchazo de una bicicleta.*

Análisis del problema

DATOS DE SALIDA: La rueda reparada
DATOS DE ENTRADA: La rueda pinchada, los parches, el pegamento
DATOS AUXILIARES: Las burbujas que salen donde está el pinchazo

Después de desmontar la rueda y la cubierta e inflar la cámara, se introduce la cámara por secciones en un cubo de agua. Las burbujas de aire indicarán donde está el pinchazo. Una vez descubierto el pinchazo se aplica el pegamento y ponemos el parche. Finalmente se montan la cámara, la cubierta y la rueda.

Diseño del algoritmo

```
inicio
   desmontar la rueda
   desmontar la cubierta
   sacar la cámara
   inflar la cámara
   meter una sección de la cámara en un cubo de agua
   mientras no salgan burbujas
      meter una sección de la cámara en un cubo de agua
   marcar el pinchazo
   echar pegamento
   mientras no esté seco
      esperar
   poner el parche
   mientras no esté fijo
      apretar
   montar la cámara
   montar la cubierta
   montar la rueda
fin
```

1.7. *Pagar una multa de tráfico.*

Análisis del problema

DATOS DE SALIDA: El recibo de haber pagado la multa
DATOS DE ENTRADA: Datos de la multa
DATOS AUXILIARES: Impreso de ingreso en el banco, dinero

Debe elegir cómo desea pagar la multa, si en metálico en la Delegación de Tráfico o por medio de una transferencia bancaria. Si elige el primer caso, tiene que ir a la Delegación Provincial de Tráfico, desplazarse a la ventanilla de pago de multas, pagarla en efectivo y recoger el resguardo.

Si desea pagarla por transferencia bancaria, hay que ir al banco, rellenar el impreso apropiado, dirigirse a la ventanilla, entregar el impreso y recoger el resguardo.

Diseño del algoritmo

```
inicio
   si pagamos en efectivo
      ir a la Delegación de Tráfico
      ir a la ventanilla de pago de multas
```

```
      si hay cola
         ponerse el último
         mientras no lleguemos a la ventanilla
            esperar
      entregar la multa
      entregar el dinero
      recoger el recibo
   si no
   //pagamos por trasferencia bancaria
      ir al banco
      rellenar el impreso
      si hay cola
         ponerse el último
         mientras no lleguemos a la ventanilla
            esperar
      entregar el impreso
      recoger el resguardo
fin
```

1.8. *Hacer una llamada telefónica. Considerar los casos:* a) *llamada manual con operador;* b) *llamada automática;* c) *llamada a cobro revertido.*

Análisis del problema

Para decidir el tipo de llamada que se efectuará, primero se debe considerar si se dispone de efectivo o no para realizar la llamada a cobro revertido. Si hay efectivo se debe ver si el lugar a donde vamos a llamar está conectado a la red automática o no.

Para una llamada con operadora hay que llamar a la centralita y solicitar la llamada, esperando hasta que se establezca la comunicación. Para una llamada automática se leen los prefijos del país y provincia si fuera necesario, y se realiza la llamada, esperando hasta que cojan el teléfono. Para llamar a cobro revertido se debe llamar a centralita, solicitar la llamada y esperar a que el abonado del teléfono al que se llama dé su autorización, con lo que se establecerá la comunicación.

Como datos de entrada tendríamos las variables que nos van a condicionar el tipo de llamada, el número de teléfono y, en caso de llamada automática, los prefijos si los hubiera. Como dato auxiliar se podría considerar en los casos a y c el contacto con la centralita.

Diseño del algoritmo

```
inicio
   si tenemos dinero
      si podemos hacer una llamada automática
         leer el prefijo de país y localidad
         marcar el número
      si no
         //llamada manual
         llamar a la centralita
         solicitar la comunicación
      mientras no contesten
         esperar
      establecer comunicación
   si no
      //realizar una llamada a cobro revertido
      llamar a la centralita
```

```
      solicitar la llamada
      esperar hasta tener la autorización
      establecer comunicación
fin
```

1.9. *Cambiar el cristal roto de la ventana.*

Análisis del problema

DATOS DE SALIDA: La ventana con el cristal nuevo
DATOS DE ENTRADA: El cristal nuevo
DATOS AUXILIARES: El número de clavos de cada una de las molduras

Diseño del algoritmo

```
inicio
   repetir cuatro veces
      quitar un clavo
      mientras el número de clavos quitados no sea igual al total de clavos
         quitar un clavo
      sacar la moldura
   sacar el cristal roto
   poner el cristal nuevo
   repetir cuatro veces
      poner la moldura
      poner un clavo
      mientras el número de clavos puestos no sea igual al total de clavos
      poner un clavo
   poner el cristal nuevo
fin
```

1.10. *Diseñar un algoritmo que imprima y sume la serie de números 3,6,9,12,...,99.*

Análisis del problema

Se trata de idear un método con el que obtengamos dicha serie, que no es más que incrementar una variable de tres en tres. Para ello se hará un bucle que se acabe cuando el número sea mayor que 99 (o cuando se realice 33 veces). Dentro de ese bucle se incrementa la variable, se imprime y se acumula su valor en otra variable llamada suma, que será el dato de salida.

No tendremos por tanto ninguna variable de entrada, y sí dos de salida, la que nos va sacando los números de tres en tres (`núm`) y suma.

Diseño del algoritmo

```
inicio
   asignar a suma un 0
   asignar a núm un 3
   mientras núm <= 99
      escribir núm
      incrementar suma en núm
      incrementar núm en 3
   escribir suma
fin
```

1.11. *Diseñar un algoritmo para calcular la velocidad (en metros/segundo) de los corredores de una carrera de 1500 metros. La entrada serán parejas de números (minutos, segundos) que darán el tiempo de cada corredor. Por cada corredor se imprimirá el tiempo en minutos y segundos, así como la velocidad media. El bucle se ejecutará hasta que demos una entrada de 0,0 que será la marca de fin de entrada de datos.*

Análisis del problema

DATOS DE SALIDA: `v` (velocidad media)
DATOS DE ENTRADA: `mm, ss` (minutos y segundos)
DATOS AUXILIARES: `distancia` (distancia recorrida, que en el ejemplo es de 1500 metros) y `tiempo` (los minutos y los segundos que ha tardado en recorrerla)

Se debe efectuar un bucle que se ejecute hasta que mm sea 0 y ss sea 0. Dentro del bucle se calcula el tiempo en segundos con la fórmula tiempo = ss + mm * 60. La velocidad se hallará con la fórmula `velocidad = = distancia/tiempo`.

Diseño del algoritmo

```
inicio
   asignar a distancia 1500
   leer desde teclado mm,ss
   mientras mm y ss sea alguno de ellos distinto de 0
      asignar a tiempo el resultado de sumar a ss el producto de mm por 60
      asignar a v el cociente entre distancia y tiempo
      escribir mm, ss, v
      leer mm, ss
fin
```

1.12. *Escribir un algoritmo que calcule la superficie de un triángulo en función de la base y la altura.*

Análisis del problema

DATOS DE SALIDA: `s` (superficie)
DATOS DE ENTRADA: `b` (base), `a` (altura)

Para calcular la superficie se aplica la fórmula `S = base * altura/2`.

Diseño del algoritmo

```
inicio
   leer b, a
   asignar a s el cociente de dividir entre dos el producto de b por a
   escribir s
fin
```

1.13. *Escribir un algoritmo que calcule la compra de un cliente en un supermercado.*

Análisis del problema

DATOS DE SALIDA: `Total`
DATOS DE ENTRADA: `Precio, unidades`
DATOS INTERMEDIOS: `Importe`

Por cada producto de la compra, se leerá el precio del producto y se calculará el importe mediante la expresión `precio * unidades`. Una vez calculado el importe se acumulará al total que, previamente, se habrá inicializado a 0. El proceso continuará mientras el cliente todavía tenga productos.

Diseño del algoritmo

```
inicio
   asignar 0 a total
   mientras haya productos
      leer precio y unidades
      asignar al importe la expresión precio * unidades
      acumular el importe al total
   escribir total
fin
```