Tema 5

Algoritmos y estructuras de programación

Versión: 2 de marzo de 2009

5.1 Fases de creación de un programa

El proceso de resolución de problemas en un ordenador conduce a la escritura de un **programa** y su ejecución. Las fases en el desarrollo de un programa pueden resumirse de la siguiente forma:

1. **Analizar el problema** consiste en conocer perfectamente en qué consiste y qué resultados se desean obtener.

2. **Planificación** de la resolución del problema, dividiéndolo, si es complicado, en una secuencia de etapas más simples. Esta fase se lleva a cabo EN UN PAPEL, estableciendo lo más claramente posible la finalidad de cada etapa, los datos que se necesitan de entrada, los datos que producirían en salida, los **algoritmos** (ver la Sección 5.2) que se utilizarán, etc.

3. **Edición** del código fuente, es decir, escritura del mismo utilizando un editor de textos simple (sin formato) y un lenguaje de programación. Los programas fuente serán almacenados en ficheros de texto, normalmente en el disco duro del ordenador.

4. **Compilación y ejecución** del programa al lenguaje máquina.

5. **Corrección de errores** del programa. Los errores se corregirán en el código fuente, repitiendo los pasos 3 y 4 tantas veces como sea necesario. Si se producen errores en la lógica del programa, es decir, si el programa "funciona" pero produce resultados incorrectos, hay que modificar el algoritmo volviendo al paso 2. Estos errores son los más difíciles de detectar.

6. **Documentación**. Una vez que el programa funcione correctamente, es conveniente **revisar** el código fuente para ordenarlos, eliminar cálculos innecesarios e incluir las líneas de comentario necesarias, que normalmente deben incluir unas **breves** explicaciones al principio del código sobre la finalidad del programa y sus argumentos de entrada y de salida.

5.2 Algoritmos

Un ordenador es capaz de realizar "sólo" determinadas acciones sencillas, tales como sumar, comparar o transferir datos, pero los problemas que normalmente interesa resolver son más complejos. Para resolver un problema real es necesario, en primer lugar, encontrar un método de resolución y,

posteriormente, determinar la sucesión de acciones sencillas (susceptibles de ser ejecutadas por un ordenador) en que se descompone dicho método.

No todos los métodos de solución de un problema pueden ser puestos es práctica en un ordenador. Para que un procedimiento pueda ser implantado en un ordenador debe ser:

- **Preciso:** estar compuesto de pasos **bien definidos** (no ambiguos) y **ordenados**.
- **Definido:** si se sigue dos veces, se obtiene el mismo resultado cada vez.
- **Finito:** tener un número finito de pasos.

Un procedimiento o método para resolver un problema que cumpla los requisitos anteriores se dice que es un **algoritmo**. Se puede dar por tanto la siguiente definición:

> Un **algoritmo** es un método para resolver un problema mediante una secuencia de pasos bien definidos, ordenados y finitos.

Para que se pueda ejecutar el algoritmo es preciso, además, que se disponga de las "herramientas" adecuadas para llevar a cabo cada uno de los pasos. Si no es así, estos deberán, a su vez, ser descompuestos en una secuencia (algoritmo) de pasos más simples que sí se puedan llevar a cabo.

> Un **programa de ordenador** es una sucesión de órdenes que describen un algoritmo, escritas de forma que puedan ser entendidas por el ordenador.

En un algoritmo (y por tanto en un programa) se distinguen las siguientes acciones:

- **Entrada:** es la información de partida que necesita el algoritmo para arrancar.
- **Proceso:** es el conjunto de todas las operaciones a realizar.
- **Salida:** son los resultados obtenidos.

Un ejemplo elemental es el Algoritmo 5.1.

Algoritmo 5.1 Preparar una taza de té.

```
Entrada: tetera, taza, bolsa de té
Salida: taza de té

Inicio
   Tomar la tetera
   Llenarla de agua
   Encender el fuego
   Poner la tetera en el fuego
   Esperar a que hierva el agua
   Tomar la bolsa de té
   Introducirla en la tetera
   Esperar 1 minuto
   Echar el té en la taza
Fin
```

5.3 Representación de algoritmos

Las dos herramientas más utilizadas comúnmente para describir algoritmos son:

Diagramas de flujo: son representaciones gráficas de secuencias de pasos a realizar. Cada operación se representa mediante un símbolo normalizado el Instituto Norteamericano de Normalización (ANSI - American National Standars Institute). Las líneas de flujo indican el orden de ejecución. Algunos de los símbolos principales se muestran en la Figura 5.1, como son: `Inicio/Fin` del algoritmo, `Lectura/Escritura` de datos que el programa necesita o genera (por ejemplo,lectura de datos que se teclean o escritura de datos en un fichero); `Proceso` conjunto de instrucciones secuenciales; `Decisión` es una bifurcación en el flujo del algoritmo en base a que se verifique o no cierta condición (ver la Sección 5.5).

Los diagramas de flujo suelen ser usados sólo para representar algoritmos pequeños, ya que abarcan mucho espacio.

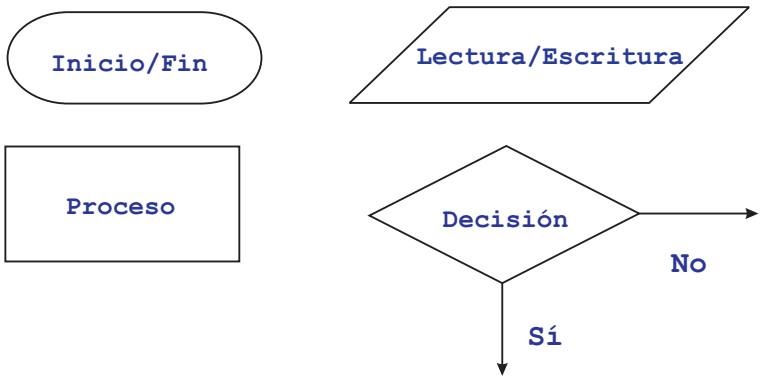

Figura 5.1: Símbolos en diagramas de flujo.

Pseudocódigos: describen un algoritmo de forma similar a un lenguaje de programación pero sin su rigidez, de forma más parecida al lenguaje natural. Presentan la ventaja de ser más compactos que los diagramas de flujo, más fáciles de escribir para las instrucciones complejas y más fáciles de transferir a un lenguaje de programación. El pseudocódigo no está regido por ningún estándar.

En estos apuntes usaremos las palabras LEER/IMPRIMIR para representar las acciones de lectura de datos (el programa recibe datos desde algún sitio) y salida de datos (el programa escribe información en algún medio)

El Algoritmo 5.2 y la Figura 5.2 muestran respectivamente el pseudocódigo y el diagrama de flujo del algoritmo para calcular la altura de una persona en pulgadas y pies a partir de la altura en centímetros introducida por el teclado.

Algoritmo 5.2 Calcular una altura en pulgadas (1 pulgada=2.54 cm) y pies (1 pie=12 pulgadas), a partir de la altura en centímetros, que se introduce por el teclado.

```
Inicio
   1- IMPRIMIR 'Introduce la altura en centimetros: '
   2- LEER: altura
   3- CALCULAR pulgadas=altura/2.54
   4- CALCULAR pies=pulgadas/12
   5- IMPRIMIR 'La altura en pulgadas es: ', pulgadas
   6- IMPRIMIR 'La altura en pies es : ', pies
Fin
```

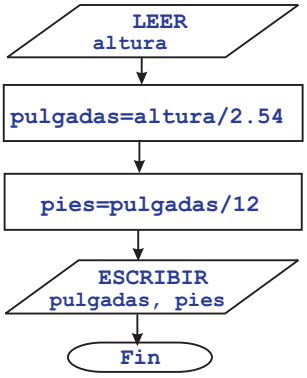

Figura 5.2: Diagrama de flujo para determinar la altura de una persona en pulgadas y pies a partir de la altura en centímetros introducida por el teclado.

5.4 Estructura secuencial

Es aquella en la que una acción (instrucción) sigue a la otra en el orden en el que están escritas. Su representación y el diagrama de flujo se muestra en la Figura 5.3. Los Algoritmos 5.1 y 5.2 son ejemplos de algoritmos secuenciales.

```
...
Instrucción 1
Instrucción 2
Instrucción 3
...
```

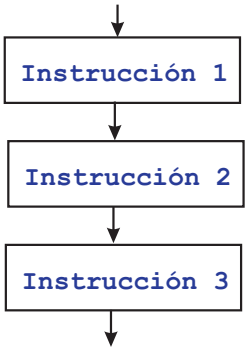

Figura 5.3: Estructura secuencial de instrucciones.

5.5 Estructuras de control: condicionales y bucles

Son parte fundamental de cualquier lenguaje. Sin ellas, las instrucciones de un programa sólo podrían ejecutarse en el orden en que están escritas (orden secuencial). Las estructuras de control permiten modificar este orden. Hay dos categorías de estructuras de control:

Condicionales o bifurcaciones: permiten que se ejecuten conjuntos distintos de instrucciones, en función de que se verifique o no determinada condición.

Bucles o repeticiones: permiten que se ejecute repetidamente un conjunto de instrucciones, bien un número pre-determinado de veces, o bien hasta que se verifique una determinada condición.

En términos de un lenguaje de programación, que se verifique o no una condición se traduce en que una (adecuada) **expresión lógica** tome el valor `VERDADERO` (`TRUE`) o tome el valor `FALSO` (`FALSE`). En los casos más sencillos y habituales la condición suele ser una comparación entre dos datos, como por ejemplo: si $a < b$ hacer una cosa y en caso contrario hacer otra distinta.

A continuación se describen las distintas estructuras de control. Para cada una de ellas se describe el diagrama de flujo y la sintaxis de la sentencia correspondiente en lenguaje MATLAB. Obsérvese que todas ellas tienen una **única entrada** y una **única salida**.

5.5.1 Estructura condicional simple: `IF`

Este es el tipo más sencillo de estructura condicional. Sirve para implementar acciones condicionales del tipo siguiente:

- Si se verifica una determinada `condición`, ejecutar una serie de instrucciones y luego seguir adelante.
- Si la `condición` NO se cumple, NO se ejecutan dichas instrucciones y se sigue adelante.

```
...
if condición
   instrucciones
end
...
```

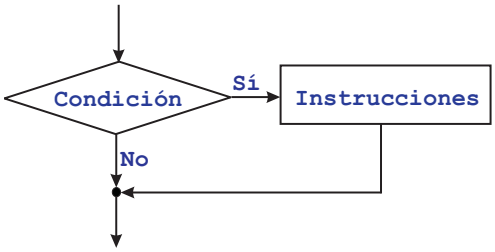

Figura 5.4: Estructura condicional simple: `IF`.

Obsérvese que, en ambos casos (que se verifique o no la condición), los "caminos" bifurcados se unen posteriormente en un punto, es decir, el flujo del programa recupera su carácter secuencial, y se continúa ejecutando por la instrucción siguiente a la estructura `IF`.

Como ejemplo de utilización de este tipo de condicional, se considera el cálculo del valor en un punto x de una función definida por partes, como por ejemplo:

$$f(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ x^2 & \text{si } x > 0 \end{cases}$$

El Algoritmo 5.3 muestra el pseudocódigo correspondiente.

Algoritmo 5.3 Cálculo del valor de la función $f(x) = 0$ si $x \leq 0$, $f(x) = x^2$ si $x > 0$.

```
Inicio
   1- LEER x
   2- HACER f=0
   3- Si x>0
        HACER f=x^2
      Fin Si
   4- IMPRIMIR 'El valor de la funcion es: ', f
Fin
```

5.5.2 Estructura condicional doble: IF - ELSE

Este tipo de estructura permite implementar condicionales en los que hay dos acciones alternativas:

- Si se verifica una determinada condición, ejecutar un serie de instrucciones (`bloque 1`).
- Si no, esto es, si la `condición` NO se verifica, ejecutar otra serie de instrucciones (`bloque 2`).

En otras palabras, en este tipo de estructuras hay una alternativa: se hace una cosa o se hace la otra. En ambos casos, se sigue por la instrucción siguiente a la estrucutra `IF - ELSE`.

```
...
if condición
   bloque-1
else
   bloque-2
end
...
```

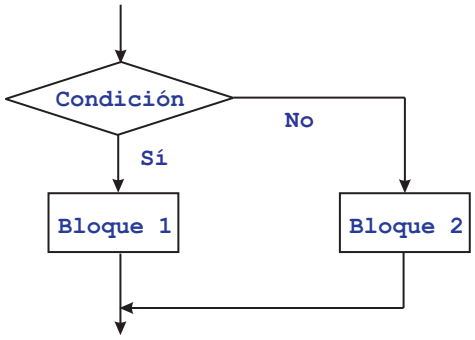

Figura 5.5: Estructura condicional doble: `IF - ELSE`.

Como ejemplo de utilización de este tipo de estructuras se plantea el problema de calcular las raíces de una ecuación de segundo grado

$$ax^2 + bx + c = 0$$

distinguiendo dos casos: que las raíces sean reales o que sean complejas (no se contempla, de momento, dintinguir entre una o dos raíces reales). Ver a continuación el diagrama de flujo (Figura 5.6) y el pseudocódigo correspondiente (Algoritmo 5.4).

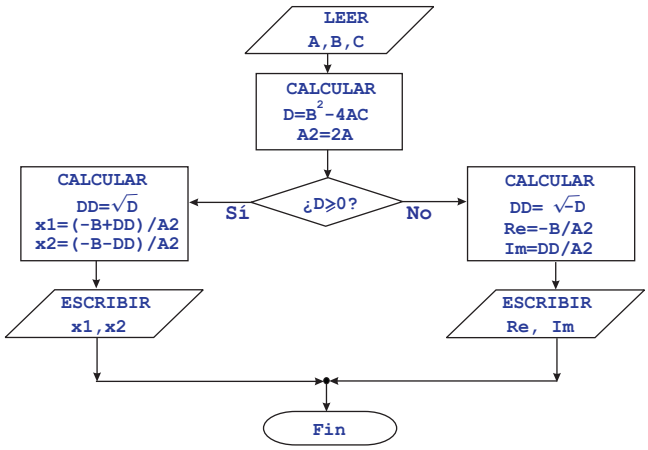

Figura 5.6: Diagrama de flujo para determinar las raíces reales o complejas de la ecuación de segundo grado $Ax^2 + Bx + C = 0$.

Algoritmo 5.4 Cálculo de las raíces de la ecuación de segundo grado $Ax^2 + Bx + C = 0$, distinguiendo los casos de raíces reales y complejas.

```
Inicio
   1- LEER A,B y C
   2- CALCULAR D=B²-4*A*C
   3- CALCULAR AA=2*A
   4- Si D≥0
       CALCULAR DD=√D
                x1=(-B+DD)/AA
                x2=(-B-DD)/AA
       IMPRIMIR 'La ecuación tiene raíces reales:', x1, x2
      Si no
       CALCULAR DD=√-D
                Re=-B/AA
                Im=DD/A2
       IMPRIMIR 'La ecuación tiene raíces complejas conjugadas:'
       IMPRIMIR 'Parte real:', Re
       IMPRIMIR 'Parte imaginaria:', Im
      Fin Si
Fin
```

5.5.3 Estructura condicional múltiple: IF - ELSEIF - ELSE

En su forma más general, la estructura IF - ELSEIF - ELSE permite implementar condicionales más complicados, en los que se "encadenan" condiciones en la forma siguiente:

- Si se verifica la `condición 1`, ejecutar las instrucciones del `bloque 1`.
- Si no se verifica la `condición 1`, pero SÍ se verifica la `condición 2` , ejecutar las instrucciones del `bloque 2`.
- Si no, esto es, si no se ha verificado ninguna de las condiciones anteriores, ejecutar las instrucciones del `bloque 3`.

En cualquiera de los casos, el flujo del programa continúa por la instrucción siguiente a la estructura `IF - ELSEIF - ELSE`.

La sintaxis en lenguaje MATLAB y el diagrama de flujo, se muestran en la Figura 5.7.

```
...
if condición-1
   bloque-1
elseif condición-2
   bloque-2
else
   bloque-3
end
...
```

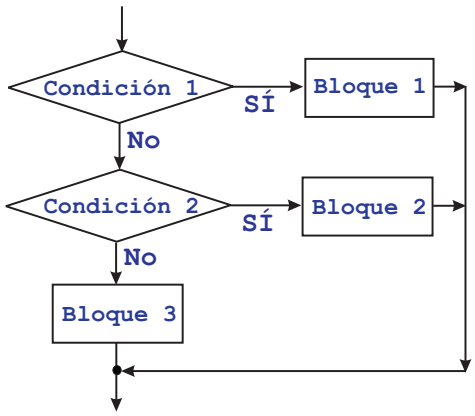

Figura 5.7: Estructura condicional múltiple: `IF- ELSEIF - ELSE`.

El Algoritmo 5.5 contiene un ejemplo de utilización de este tipo de estructura.

Algoritmo 5.5 Determinación del signo de un número: positivo, negativo o nulo.

```
Inicio
   1- LEER X
   2- Si X>0
        IMPRIMIR 'El número tiene signo positivo'
      Si no, si X<0
        IMPRIMIR 'El numero tiene signo negativo'
      Si no
        IMPRIMIR 'El numero es nulo'
Fin
```

En la estructura `IF - ELSEIF - ELSE` se puede multiplicar la cláusula `ELSEIF`, obteniéndose así una "cascada" de condiciones, como se muestra en el organigrama, cuyo funcionamiento es claro.
En este tipo de estructura condicional, la cláusula `ELSE` junto con su bloque de instrucciones pueden no estar presente.
Las distintas estructuras condicionales descritas pueden ser **anidadas**, es decir, puede incluirse una estructura `IF` (de cualquier tipo), como parte de las instrucciones que forman el bloque de uno de los casos de otro `IF`. Como es lógico, no puede haber solapamiento. Cada estructura `IF` debe tener su propio fin (`end`).

```
if condición-1
   instrucciones-1
elseif condición-2
   instrucciones-2
elseif condición-3
   instrucciones-3
   ...
else
   instrucciones-k
end
```

Como ejemplo de utilización de este tipo de estructura condicional y de estructuras anidadas, véase el Algoritmo 5.6.

Algoritmo 5.6 Dados dos números reales, **a** y **b**, y el símbolo, **S** (carácter), de un operador aritmético (+, -, *, /), imprimir el resultado de la operación **a S b**

```
Inicio
   LEER a
   LEER b
   LEER S
   Si S='+'
       IMPRIMIR 'El resultado es =', a+b
   Si no, si S='-'
       IMPRIMIR 'El resultado es =', a-b
   Si no, si S='*'
       IMPRIMIR 'El resultado es =', a*b
   Si no, si b=0
       Si a=0
         IMPRIMIR 'El resultado es =', NaN (indeterminación)
       Si no
         IMPRIMIR 'El resultado es =', Inf (infinito)
       Fin Si
   Si no
       IMPRIMIR 'El resultado es =', a/b
   Fin Si
Fin
```

5.5.4 Estructura de repetición indexada: FOR

Este tipo de estructura permite implementar la repetición de un cierto conjunto de instrucciones un número pre-determinado de veces.

Para ello se utiliza una variable de control del bucle, llamada también `índice`, que va recorriendo un conjunto pre-fijado de valores en un orden determinado. Para cada valor del `índice` en dicho conjunto, se ejecuta una vez el mismo conjunto de instrucciones.

En la Figura 5.8 se han representado la forma de escribir esta estructura en MATLAB y el organigrama correspondiente: el bloque de instrucciones se ejecuta una vez para cada valor del `indice`, que va tomando sucesivamente el valor de cada componente del vector `V`, de longitud `N`.

```
...
for indice=V
   instrucciones
end
...
```

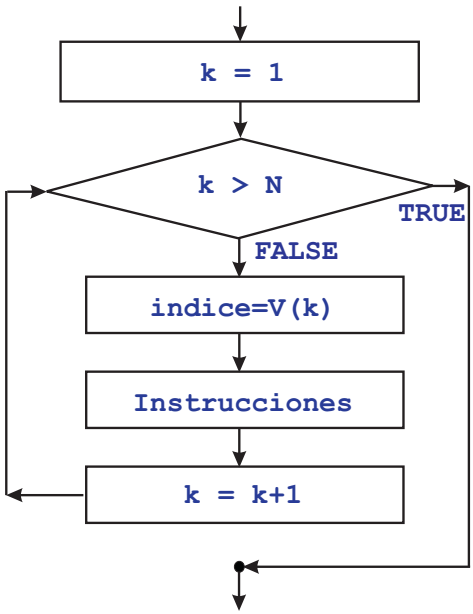

Figura 5.8: Repetición indexada: sintaxis MATLAB y diagrama de flujo. El `indice` del bucle recorre los valores de un vector `V` de longitud `N`.

Como ejemplo de utilización de la estructura `FOR`, véanse los Algoritmos 5.8 y 5.9 para calcular la suma de los `n` primeros números impares.

Nota 5.7

a) El valor de la variable de control `indice` puede ser utilizado o no dentro del conjunto de instrucciones que forman parte del cuerpo del `FOR`, pero no debe ser modificado.

b) El conjunto de valores que debe recorrer el `indice` puede ser **vacío** (`N`=0). En ese caso, el bloque de instrucciones no se ejecuta ninguna vez.

c) Las estructuras `FOR` e `IF` pueden "anidarse", es decir, incluir una dentro de la otra, con la restricción (de sentido común) de que la interior tiene que estar completamente contenida en uno de los bloques de instrucciones de la otra. Véase, como ejemplo, el Algoritmo 5.10.

Algoritmo 5.8 Dado un entero, `n`, calcular la suma de los `n` primeros números impares.

```
Inicio
   LEER n
   HACER suma=0
   Para i= 1, 3, 5, ..., 2*n-1
     HACER suma=suma+i
   Fin Para
   IMPRIMIR 'La suma vale : ', suma
Fin
```

Algoritmo 5.9 Dado un entero, `n`, calcular

$$\sum_{k=0}^{n} \left(\frac{1}{2}\right)^k = 1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^n}$$

```
Inicio
   LEER n
   HACER suma=1
   HACER ter=1
   Para k= 1, 2, ..., n
     HACER ter=ter/2
     HACER suma=suma+ter
   Fin Para
   IMPRIMIR 'La suma vale : ', suma
Fin
```

Algoritmo 5.10 Dado un número natural, `n`, imprimir la lista de sus divisores, en orden decreciente.

```
Inicio
   LEER n
   IMPRIMIR ' Lista de divisores del numero: ', n
   Para i=ParteEntera(n/2) hasta 2 (incremento -1)
      Si resto(n/i)=0
         IMPRIMIR i
      Fin Si
   Fin Para
   IMPRIMIR 1
Fin
```

5.5.5 Estructura repetitiva condicional: WHILE

Permite implementar la repetición de un mismo conjunto de instrucciones mientras que se verifique una determinada condición: el número de veces que se repetirá el ciclo no está definido *a priori*.

El diagrama de flujo descriptivo de esta estructura se muestra en la Figura 5.9.

```
...
while expresión-lógica
   instrucciones
end
...
```

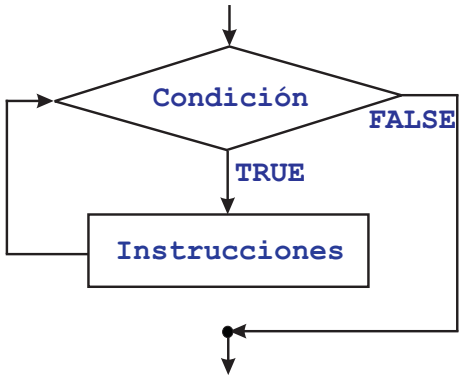

Figura 5.9: Estructura repetitiva WHILE: sintaxis MATLAB y diagrama de flujo.

Su funcionamiento es evidente, a la vista del diagrama:

1. Al comienzo de cada iteración se evalúa la expresión-lógica.
2. Si el resultado es VERDADERO, se ejecuta el conjunto de instrucciones y se vuelve a iterar, es decir, se repite el paso 1.
3. Si el resultado es FALSO, se detiene la ejecución del ciclo WHILE y el programa se sigue ejecutando por la instrucción siguiente al END.

El Algoritmo 5.11 es un ejemplo de utilización de esta estructura.

Algoritmo 5.11 Imprimir de forma ascendente los 100 primeros números naturales.

```
Inicio
   i=1
   Mientras que i ≤ 100
      IMPRIMIR i
      HACER i=i+1
   Fin Mientras
Fin
```

5.5.6 Ruptura de ciclos de repetición: BREAK y CONTINUE

En ocasiones es necesario interrumpir la ejecución de un ciclo de repetición **en algún punto interno del bloque de instrucciones que se repiten**. Lógicamente, ello dependerá de que se verifique o no alguna condición.

La interrupción puede hacerse de dos formas:

1. Abandonando el ciclo de repetición definitivamente.

2. Abandonando la iteración en curso, pero comenzando la siguiente.

Las instrucciones para poner esto en práctica tienen nombres diversos en los distintos lenguajes de programación. En MATLAB, la primera opción se implementa con la instrucción `BREAK` y la segunda con la instrucción `CONTINUE`. Ambas pueden utilizarse tanto para romper un ciclo `FOR` como un ciclo `WHILE`. Cuando se utiliza la orden `BREAK` dentro de un ciclo `FOR`, el `índice` del bucle conserva, fuera del mismo, el último valor que tomó.

Véanse los correspondientes diagramas de flujo en las Figuras 5.10 y 5.11.

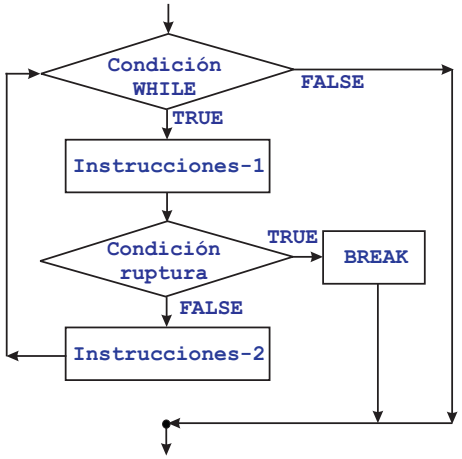

Figura 5.10: Diagrama de flujo de la ruptura de ciclo `BREAK`.

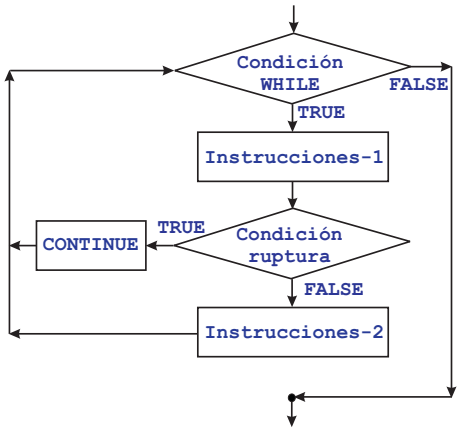

Figura 5.11: Diagrama de flujo de la ruptura de ciclo `CONTINUE`.

5.5.7 Estructura de elección entre varios casos: `SWITCH`

Este tipo de estructura permite decidir entre varios caminos posibles, en función del valor que tome una determinada instrucción.

El diagrama de flujo correspondiente a una de estas estructuras (con cuatro casos) se presenta en la Figura 5.12.

```
switch expresión
  case valor-1
    instrucciones caso 1
  case valor-2
    instrucciones caso 2
  ...
  case {valores...}
    instrucciones caso k
  otherwise
    instrucciones
end
```

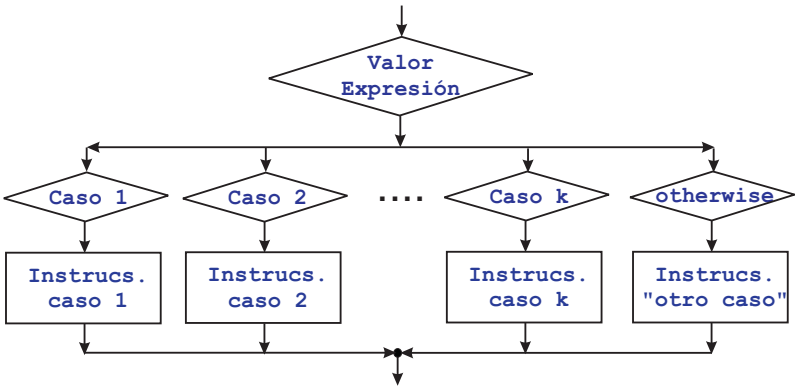

Figura 5.12: Estructura de elección de caso `SWITCH`: sintaxis MATLAB y diagrama de flujo.

En cada uno de los casos, el `valor` correspondiente puede ser o bien un sólo valor, o bien un conjunto de valores, en cuyo caso se indican entre llaves. La cláusula `OTHERWISE` y su correspondiente conjunto de instrucciones puede no estar presente.

El funcionamiento es el siguiente:

1. Al comienzo se evalúa la `expresión`.
2. Si `expresión` toma el valor (ó valores) especificados junto a la primera cláusula `CASE`, se ejecuta el conjunto de instrucciones de este caso y después se abandona la estructura `SWITCH`, continuando por la instrucción siguiente al `END`.
3. Se repite el procedimiento anterior, de forma ordenada, para cada una de las cláusulas `CASE` que siguen.
4. Si la cláusula `OTHERWISE` está presente y la `expresión` no ha tomado ninguno de los valores anteriormente especificados, se ejecuta el conjunto de instrucciones correspondiente.

Obsérvese que se ejecuta, **como máximo** el conjunto de instrucciones de uno de los casos, es decir, una vez que se ha verificado un caso y se ha ejecutado su conjunto de instrucciones, no se testea el resto de casos, ya que se abandona la estructura. Obviamente, si la cláusula `OTHERWISE` no está presente, puede ocurrir que no se dé ninguno de los casos.

Como ejemplo de utilización, se presenta el mismo proceso del Algoritmo 5.6, pero utilizando la sentencia `SWITCH`.

Algoritmo 5.12 Dados dos números reales, **a** y **b**, y el símbolo, **S** (carácter), de un operador aritmético (+, -, *, /), imprimir el resultado de la operación **a S b**

```
LEER a , b , S
Elegir caso S
  Caso '+'
     IMPRIMIR 'El resultado es =', a+b
  Caso '-'
     IMPRIMIR 'El resultado es =', a-b
  Caso '*'
     IMPRIMIR 'El resultado es =', a*b
  Caso '*'
     IMPRIMIR 'El resultado es =', a*b
  Caso '/'
     Si a≠0
        IMPRIMIR 'El resultado es =', a/b
     Si no, si b≠0
        IMPRIMIR 'El resultado es =', Inf (infinito)
     Si no
        IMPRIMIR 'El resultado es =', NaN (indeterminación)
     Fin Si
  En otro caso
     IMPRIMIR 'El operador no se reconoce'
  Fin Elegir caso
```

5.6 Funciones externas

Con lo expuesto hasta aquí se pueden escribir programas sencillos y no demasiado largos. Pero varias razones justifican la necesidad de disponer de otro tipo de recursos de programación. Por una parte, puede haber en la resolución de un problema, partes que se repitan. Por otra parte, es conveniente partir la resolución de un problema "largo" en una serie de etapas más cortas que se concatenan para resolver el problema global. Los programas demasiado largos son difíciles de revisar y de corregir.

En cualquier lenguaje de programación, existen las `funciones`. Son trozos de código que se escriben separadamente y que realizan cálculos o tareas específicas. Todos los lenguajes de programación tienen funciones incorporadas o intrínsecas, es decir, funciones que realizan cálculos o tareas de uso habitual que han sido ya programados y están disponibles para el usuario. Pero, además, todos los lenguajes tienen la posibilidad de que el usuario defina sus propias funciones, que reciben el nombre de funciones externas.

La forma de definir las funciones depende del lenguaje de programación. En Matlab, las funciones deben tener la estructura siguiente:

```
function [argumentos de salida] =nombre(argumentos de entrada)
...
instrucciones
...
```

Aquí, `function` es una palabra reservada: no se cambia y debe estar presente. `nombre` es el nombre que se quiere dar a la función, para identificarla. Los `argumentos de entrada` y los `argumentos de salida` son nombre de variable que hacen el papel de las variables mudas en la definición de una función matemática: símbolos para describir la función. En programación reciben, también, el nombre de variables mudas (de entrada o de salida).

Los argumentos de entrada y de salida, cuando hay varios, se separan por comas. Puede no haber argumentos de entrada, en cuyo caso, no es necesario poner los paréntesis. Puede, asimismo, no haber argumentos de salida, en cuyo caso no es necesario poner los corchetes ni el signo igual.

A continuación de la instrución de declaración (`function`), se escribe el código para llevar a cabo los cálculos o tareas correspondientes.

Para que una función MATLAB (llamada también M-función) esté disponible para su uso desde la ventana de comandos, desde un *script* o desde otra función, debe ser guardada en un fichero de nombre igual al de la función y extensión `.m`, y debe estar en el directorio actual de trabajo.

La ejecución de una M-función se termina cuando se terminan sus instrucciones. Cuando ello sucede el control de ejecución vuelve al punto desde el que se invocó a la función: la linea de comandos de MATLAB, un *script* o bien otra función.

A veces es necesario terminar la ejecución de una function **antes** de que se terminen sus instrucciones: ello se hace mediante la sentencia `RETURN`. Esta instrucción, en cualquier parte de una función o de un *script* provoca el inmediato abandono del mismo.

A modo de ejemplo de uso de una función, véase el siguiente, que muestra el código de una M-función que calcula el valor de una función matemática.

EJEMPLO:
M-función para calcular el valor en un punto x de la función

$$f(x) = e^{x\,\mathrm{sen}(x/3)} \cos \frac{x+\pi}{2}$$

```
function [y]=mifun(x)
v=exp(x.*sin(x/3));
w=cos((x+pi)/2);
y=v.*w;
```

Una vez escrita y salvada, como se ha dicho antes, en un fichero de nombre `mifun.m` en el directorio de trabajo, esta función estará disponible para su uso en forma similar a cualquier otra función intrínseca.

Las variables definidas dentro de una función son **variables locales**, en el sentido de que son inaccesibles desde "fuera" de esta función. Se puede decir que pertenecen al espacio de trabajo propio de la función y no son vistas desde otros espacios de trabajo. Por ejemplo, si se utiliza esta función desde la línea de comandos de Matlab, y se tiene, en el espacio de trabajo de Matlab, una variable de nombre, por ejemplo, `w`, esta variable y la `w` de la función **son distintas**: se refieren a una posiciones diferentes en memoria.