Estructuras de control

Programación estructurada

Estructuras condicionales

- La sentencia `if`
- La cláusula `else`
- Encadenamiento y anidamiento
- El operador condicional `?:`
- La sentencia `switch`

Estructuras repetitivas/iterativas

- El bucle `while`
- El bucle `for`
- El bucle `do…while`
- Bucles anidados

Cuestiones de estilo

Vectores y matrices

- Algoritmos de ordenación
- Algoritmos de búsqueda

Las estructuras de control
controlan la ejecución de las instrucciones de un programa

Programación estructurada

IDEA CENTRAL:
Las estructuras de control de un programa
sólo deben tener un punto de entrada y un punto de salida.

La programación estructurada...

mejora la productividad de los programadores.

mejora la legibilidad del código resultante.

La ejecución de un programa estructurado progresa
disciplinadamente,
en vez de saltar de un sitio a otro de forma impredecible

Gracias a ello, los programas…

resultan más fáciles de probar

se pueden depurar más fácilmente

se pueden modificar con mayor comodidad

En programación estructurada sólo se emplean tres construcciones:

- **Secuencia**
  Conjunto de sentencias que se ejecutan en orden

  Ejemplos:
  Sentencias de asignación y llamadas a rutinas.

- **Selección**
  Elige qué sentencias se ejecutan en función de una condición.

  Ejemplos:
  Estructuras de control condicional `if-then-else` y `case/switch`

- **Iteración**
  Las estructuras de control repetitivas repiten conjuntos de instrucciones.

  Ejemplos:
  Bucles `while`, `do...while` y `for`.

Teorema de Böhm y Jacopini (1966):
Cualquier programa de ordenador
puede diseñarse e implementarse
utilizando únicamente las tres construcciones estructuradas
(secuencia, selección e iteración; esto es, sin sentencias `goto`).

Böhm, C. & Jacopini, G.: "Flow diagrams, Turing machines, and languages only with two formation rules". Communications of the ACM, 1966, Vol. 9, No. 5, pp. 366-371

Dijkstra, E.W.: "Goto statement considered harmful". Communications of the ACM, 1968, Vol. 11, No. 3, pp. 147-148

Estructuras de control condicionales

Por defecto,
las instrucciones de un programa se ejecutan secuencialmente:

> El orden secuencial de ejecución no altera el flujo de control del programa respecto al orden de escritura de las instrucciones.

Sin embargo, al describir la resolución de un problema, es normal que tengamos que tener en cuenta condiciones que influyen sobre la secuencia de pasos que hay que dar para resolver el problema:

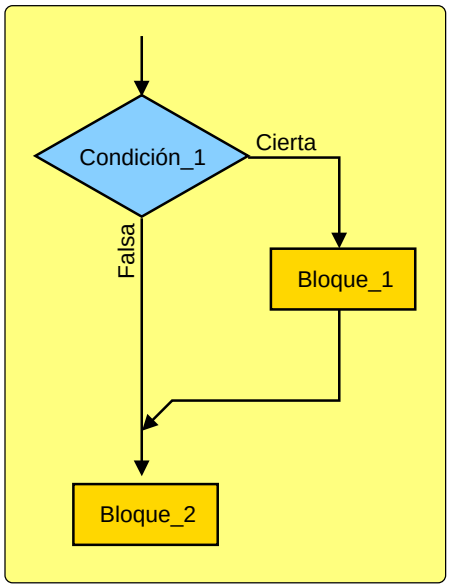
> Según se cumplan o no determinadas condiciones,
> la secuencia de pasos involucrada en la realización de una tarea será diferente

Las estructuras de control condicionales o selectivas nos permiten decidir qué ejecutar y qué no en un programa.

Ejemplo típico
Realizar una división sólo si el divisor es distinto de cero.

La estructura de control condicional `if`

La sentencia `if` nos permite elegir
si se ejecuta o no un bloque de instrucciones.



Sintaxis

```
if (condición)
   sentencia;


if (condición) {
   bloque
}
```

donde `bloque` representa un bloque de instrucciones.

Bloque de instrucciones:
Secuencia de instrucciones encerradas entre dos llaves {....}

Consideraciones acerca del uso de la sentencia `if`

- Olvidar los paréntesis al poner la condición del `if` es un error sintáctico (los paréntesis son necesarios)

- Confundir el operador de comparación == con el operador de asignación = puede producir errores inesperados

- Los operadores de comparación ==, `!=`, <= y >= han de escribirse sin espacios.

- => y =< no son operadores válidos en C.

- El fragmento de código afectado por la condición del `if` debe sangrarse para que visualmente se interprete correctamente el ámbito de la sentencia `if`:

```
if (condición) {
   // Aquí se incluye el código
   // que ha de ejecutarse sólo
   // si se cumple la condición del if
   // (sangrado para que se vea dónde
   //  empieza y dónde acaba el if)
}
```

Error común:

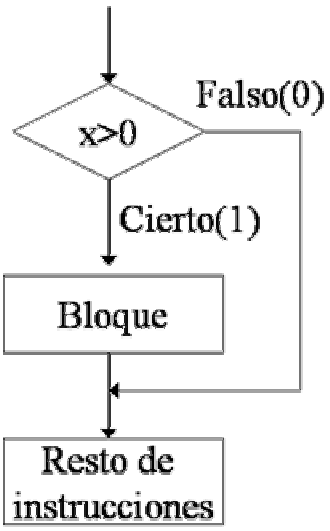
```
if (condición);
   sentencia;
```

es interpretado como

```
if (condición)
   ;   // Sentencia vacía

sentencia;
```

¡¡¡La sentencia siempre se ejecutaría!!!

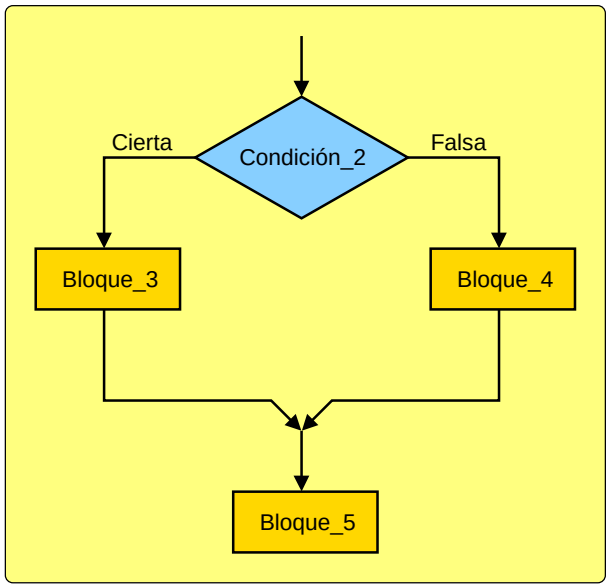
Ejemplo
Decir si un número es positivo




```
#include <stdio.h>

int main (int argc, char *argv[])
{
  int x;

  printf("Déme un numero");
  scanf("%d",&x);

  if (x>0) {
     printf("El numero %d es positivo",x);
  }

  return 0;
}
```

La cláusula `else`

Una sentencia `if`, cuando incluye la cláusula `else`, permite ejecutar un bloque de código si se cumple la condición y otro bloque de código diferente si la condición no se cumple.



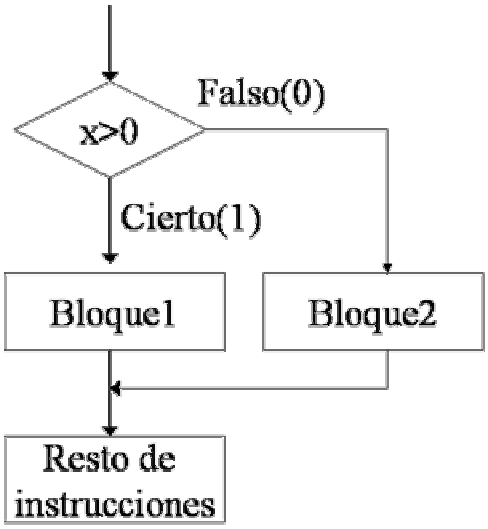
Sintaxis

```
if (condición)
   sentencia1;
else
   sentencia2;
```

```
if (condición) {
   bloque1
} else {
   bloque2
}
```

Los bloques de código especificados
representan dos alternativas complementarias y excluyentes

Ejemplo
Decir si un número es positivo o negativo




```
#include <stdio.h>

void main ()
{
  int x;

  printf("Déme un numero: ");
  scanf("%d",&x);

  if (x>=0) {
     printf("El numero %d es positivo", x);
  } else {
     printf("El numero %d es negativo", x);
  }
}
```

La sentencia `if` anterior es equivalente a

```
  if (x>=0) {
     printf("El numero %d es positivo", x);
  }
  if (x<0) {  // Condición complementaria a x>=0
     printf("El numero %d es negativo", x);
  }
```

si bien nos ahorramos comprobar una condición al usar `else`.

Encadenamiento

Las sentencias `if` se suelen encadenar:

if … else if …

```
#include <stdio.h>

void main ()
{
  float nota;

  printf("Déme una nota: ");
  scanf("%f",&nota);

  if (nota>=9) {
     printf("Sobresaliente");
  } else if (nota>=7) {
     printf("Notable");
  } else if (nota>=5) {
     printf("Aprobado");
  } else {
     printf("Suspenso");
  }
}
```

El `if` encadenado anterior equivale a:

```
  if (nota>=9) {
     printf("Sobresaliente");
  }
  if ((nota>=7) && (nota<9)) {
     printf("Notable");
  }
  if ((nota>=5) && (nota<7)) {
     printf("Aprobado");
  }
  if (nota<5) {
    printf("Suspenso");
  }
```

Anidamiento

Las sentencias `if` también se pueden anidar unas dentro de otras.

Ejemplo

Resolución de una ecuación de primer grado ax+b = 0

```
#include <stdio.h>

void main()
{
  float a,b;

  printf("Coeficientes de la ecuación ax+b=0:");
  scanf ("%f %f",&a,&b);

  if (a!=0) {
     printf("La solución es %f", -b/a);
  } else {
     if (b!=0) {
        printf("La ecuación no tiene solución.");
     } else {
        printf("Solución indeterminada.");
     }
  }
}
```

El `if` anidado anterior equivale a …

```
  if (a!=0) {
     printf("La solución es %f", -b/a);
  }

  if ((a==0) && (b!=0)) {
     printf("La ecuación no tiene solución.");
  }

  if ((a==0) && (b==0)) {
     printf("Solución indeterminada.");
  }
```

En este caso, se realizarían 5 comparaciones en vez de 2.

El operador condicional **`?:`**

C proporciona una forma de abreviar una sentencia `if`

> El operador condicional `?:`
> permite incluir una condición dentro de una expresión.

Sintaxis

```
condición? expresión1: expresión2
```

"equivale" a

```
if (condición)
   expresión1
else
   expresión2
```

Sólo se evalúa una de las sentencias,
por lo que deberemos ser cuidadosos con los efectos colaterales.

Ejemplos

```
max = (x>y)? x : y;

min = (x<y)? x : y;

med = (x<y)? ((y<z)? y: ((z<x)? x: z)):
             ((x<z)? x: ((z<y)? y: z));
```

Selección múltiple con la sentencia **`switch`**

Permite seleccionar entre varias alternativas posibles

Sintaxis

```
switch (expresión) {

  case expr_cte1:
       sentencia1;

  case expr_cte2:
       sentencia2;
  ...
  case expr_cteN:
       sentenciaN;

  default:
       sentencia;
}
```

- Se selecciona a partir de la evaluación de una única expresión.
- La expresión del `switch` ha de ser de tipo entero.
- Los valores de cada caso del `switch` han de ser constantes.
- La etiqueta `default` marca el bloque de código que se ejecuta por defecto (cuando al evaluar la expresión se obtiene un valor no especificado por los casos del `switch`).
- En C, se ejecutan todas las sentencias incluidas a partir del caso correspondiente, salvo que explícitamente usemos `break`:

```
switch (expresión) {
  case expr_cte1:
       sentencia1;
       break;
  case expr_cte2:
  case expr_cte3:
       sentenciaN;
}
```

```
switch (expresión) {
  case expr_cte1:
       sentencia1;
       break;
  case expr_cte2:
       sentencia2;
       break;
  default:
       sentencia;
}
```

Ejemplo

```
#include <stdio.h>

void main()
{
  int nota;

  printf(“Calificación: ”);
  scanf(“%d”, &nota);

  switch (nota) {

    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
         printf(“Suspenso”);
         break;

    case 5:
    case 6:
         printf(“Aprobado”);
         break;

    case 7:
    case 8:
         printf(“Notable”);
         break;

    case 9:
         printf(“Sobresaliente”);
         break;

    case 10:
         printf(“Matrícula”);
         break;

    default:
         printf(“Error”);
  }
}
```

Si trabajamos con datos de tipo real,
tendremos que usar sentencias `if` encadenadas.

Estructuras de control repetitivas/iterativas

A menudo es necesario ejecutar
una instrucción o un bloque de instrucciones más de una vez.

Ejemplo

Implementar un programa que calcule
la suma de N números leídos desde teclado.

Podríamos escribir un programa en el que apareciese repetido el código que deseamos que se ejecute varias veces, pero…

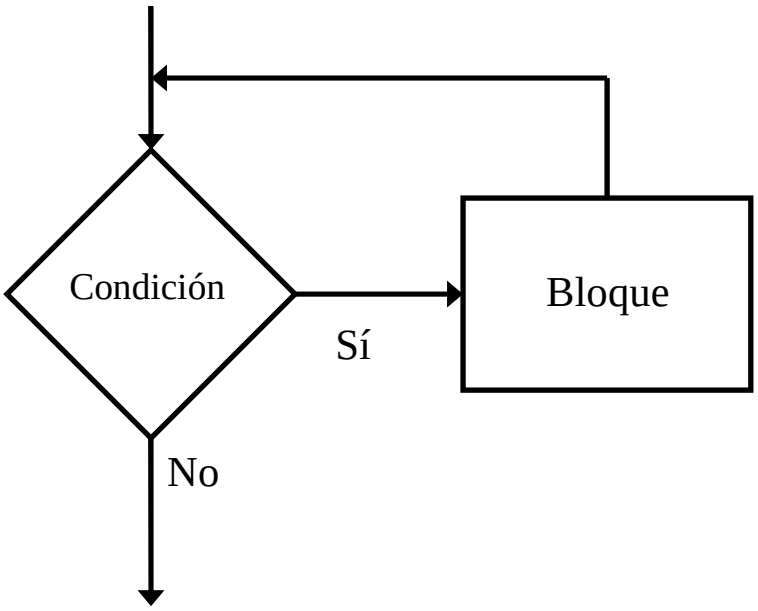
- Nuestro programa podría ser demasiado largo.
- Gran parte del código del programa estaría duplicado, lo que dificultaría su mantenimiento en caso de que tuviésemos que hacer cualquier cambio, por trivial que fuese éste.
- Una vez escrito el programa para un número determinado de repeticiones (p.ej. sumar matrices 3x3), el mismo programa no podríamos reutilizarlo si necesitásemos realizar un número distinto de operaciones (p.ej. matrices 4x4).

Las **estructuras de control repetitivas o iterativas**, también conocidas como "**bucles**", nos permiten resolver de forma elegante este tipo de problemas. Algunas podemos usarlas cuando conocemos el número de veces que deben repetirse las operaciones. Otras nos permiten repetir un conjunto de operaciones mientras se cumpla una condición.

Iteración: Cada repetición de las instrucciones de un bucle.

El bucle **while**

```
while (condición)
      sentencia;

while (condición) {
      bloque
}
```



MUY IMPORTANTE

En el cuerpo del bucle debe existir algo que haga variar el valor asociado a la condición que gobierna la ejecución del bucle.

Ejemplo
Tabla de multiplicar de un número

```
#include <stdio.h>

void main ()
{
  int n, i;

  printf (“Introduzca un número: ”);
  scanf (“%d”, &n);

  i=0; // Inicialización del contador

  while (i<=10) {
    printf (“%d x %d = %d\n“, n, i, n*i);
    i++;
  }
}
```

Ejemplo
Divisores de un número

```
#include <stdio.h>

void main ()
{
  int n;
  int divisor;

  printf (“Introduzca un número: ”);
  scanf (“%d”, &n);

  printf (“Los divisores del número son:\n”);

  divisor = n;

  while (divisor>0) {

    if ((n%divisor) == 0)
       printf (“%d\n”, divisor);

    divisor--;
  }
}
```

Ejemplo
Suma de N números

```
#include <stdio.h>

void main()
{
  int    total;  // Número de datos
  double suma;   // Suma de los datos
  double n;      // Número leído desde el teclado
  int    i;      // Contador

  printf (“Número total de datos: ”);
  scanf (“%d”,&total);

  suma = 0; // Inicialización de la suma
  i = 0;    // Inicialización del contador

  while (i<total) {

    printf (“Dato %d: ”, i);
    scanf (“%lf”,&n);

    suma = suma + n;
    i++;
  }

  printf (“Suma total = %lf\n”, suma);
}
```

Tal como está implementado el programa, fácilmente podríamos calcular la media de los datos (`suma/total`).

EJERCICIO
Ampliar el programa anterior para que, además de la suma y de la media, nos calcule también el máximo, el mínimo, la varianza y la desviación típica de los datos.

Pista: La varianza se puede calcular a partir de la suma de los cuadrados de los datos.

En los ejemplos anteriores se conoce de antemano el número de iteraciones que han de realizarse (cuántas veces se debe ejecutar el bucle):

En este caso, la expresión del `while` se convierte en una simple comprobación del valor de una variable contador (una variable que se incrementa o decrementa en cada iteración y nos permite contabilizar la iteración en la que nos encontramos).

En otras ocasiones, puede que no conozcamos de antemano cuántas iteraciones se han de realizar.

Ejemplo
Sumar una serie de números
hasta que el usuario introduzca un cero

```
#include <stdio.h>

void main()
{
  double suma;   // Suma de los datos
  double n;      // Número leído desde el teclado

  suma = 0; // Inicialización de la suma

  printf ("Introduzca un número (0 para terminar): ");
  scanf ("%lf", &n);

  while (n!=0) {
    suma = suma + n;
    printf ("Introduzca un número (0 para terminar): ");
    scanf ("%lf", &n);
  }

  printf ("Suma total = %lf\n", suma);
}
```

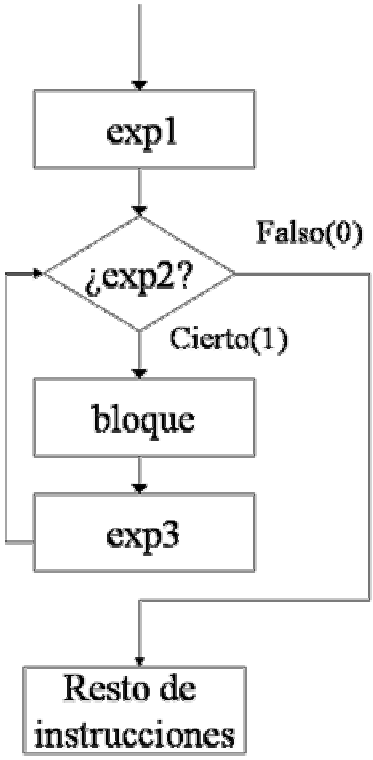
El valor introducido determina
si se termina o no la ejecución del bucle.

El bucle `for`

Se suele emplear en sustitución del bucle `while`
cuando se conoce el número de iteraciones que hay que realizar.

Sintaxis

```
for (exp1; exp2; exp3) {
    bloque;
}
```



Equivalencia entre `for` y `while`

```
for (expr1; expr2; expr3) {
    bloque;
}
```

equivale a

```
expr1;
while (expr2) {
      bloque;
      expr3;
}
```

Ejemplo
Cálculo del factorial de un número

Bucle **for**

```
#include <stdio.h>

void main()
{
  long i, n, factorial;

  printf ("Introduzca un número: ");
  scanf ("%ld", &n);

  factorial = 1;

  for (i=1; i<=n; i++) {
      factorial *= i;
  }

  printf ("factorial(%ld) = %ld", n, factorial);
}
```

Bucle **while**

```
#include <stdio.h>

void main()
{
  long i, n, factorial;

  printf ("Introduzca un número: ");
  scanf ("%ld", &n);

  factorial = 1;

  i = 1;
  while (i<=n) {
    factorial *= i;
    i++;
  }

  printf ("factorial(%ld) = %ld", n, factorial);
}
```

```
for (expr1; expr2; expr3) {
    bloque;
}
```

En un bucle `for`

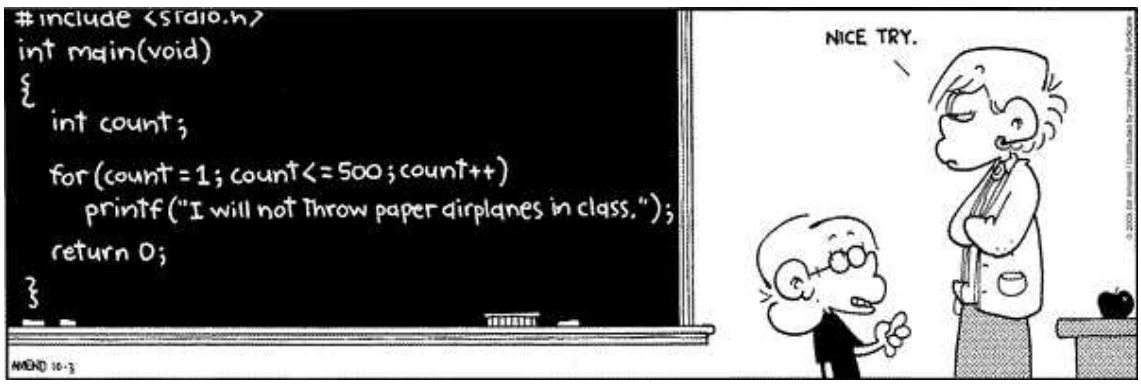
- La primera expresión, `expr1`, suele contener inicializaciones de variables separadas por comas. En especial, siempre aparecerá la inicialización de la variable que hace de contador.

- Las instrucciones que se encuentran en esta parte del `for` sólo se ejecutarán una vez antes de la primera ejecución del cuerpo del bucle (`bloque`).

- La segunda expresión, `expr2`, es la que contiene una expresión booleana (la que aparecería en la condición del bucle `while` equivalente para controlae la ejecución del cuerpo del bucle).

- La tercera expresión, `expr3`, contiene las instrucciones, separadas por comas, que se deben ejecutar al finalizar cada iteración del bucle (p.ej. el incremento/decremento de la variable contador).

- El bloque de instrucciones `bloque` es el ámbito del bucle (el bloque de instrucciones que se ejecuta en cada iteración).



Cabecera del bucle	Número de iteraciones
`for (i=0; i<N; i++)`	N
`for (i=0; i<=N; i++)`	N+1
`for (i=k; i<N; i++)`	N-k
`for (i=N; i>0; i--)`	N
`for (i=N; i>=0; i--)`	N+1
`for (i=N; i>k; i--)`	N-k
`for (i=0; i<N; i+=k)`	N/k
`for (i=j; i<N; i+=k)`	(N-j)/k
`for (i=1; i<N; i*=2)`	$\log_2 N$
`for (i=0; i<N; i*=2)`	∞
`for (i=N; i>0; i/=2)`	$\log_2 N + 1$

suponiendo que N y k sean enteros positivos

Bucles infinitos

Un bucle infinito es un bucle que se repite “infinitas” veces:

```
for (;;)        /*bucle infinito*/

while (1)       /*bucle infinito*/
```

Si nunca deja de cumplirse la condición del bucle, nuestro programa se quedará indefinidamente ejecutando el cuerpo del bucle, sin llegar a salir de él.

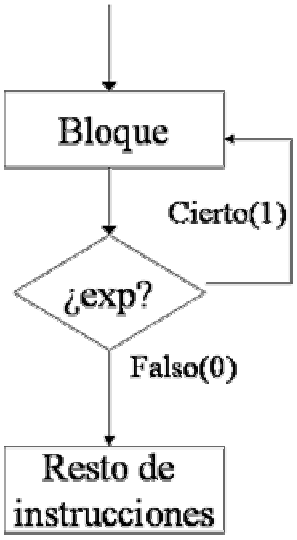
El bucle `do while`

Tipo de bucle, similar al `while`, que realiza la comprobación de la condición después de ejecutar el cuerpo del bucle.

Sintaxis

```
do
  sentencia;
while (condición);
```

```
do {
  bloque
} while (condición);
```



- El bloque de instrucciones se ejecuta al menos una vez.
- Especialmente indicado para validar datos de entrada (comprobar que los valores obtenidos están dentro del rango de valores esperados).

Ejemplo
Cálculo del factorial
comprobando el valor del dato de entrada

```
#include <stdio.h>

void main()
{
  long i, n, factorial;

  do {

    printf ("Introduzca un número (entre 1 y 12): ");
    scanf ("%ld", &n);

  } while ((n<1) || (n>12));

  factorial = 1;

  for (i=1; i<=n; i++) {
      factorial *= i;
  }

  printf ("factorial(%ld) = %ld", n, factorial);
}
```

IMPORTANTE

En todos nuestros programas debemos asegurarnos
de que se obtienen datos de entrada válidos
antes de realizar cualquier tipo de operación con ellos.

Ejemplo
Cálculo de la raíz cuadrada de un número

```
#include <stdio.h>
#include <math.h>

void main()
{
  // Declaración de variables

  double n;     // Número real
  double r;     // Raíz cuadrada del número
  double prev;  // Aproximación previa de la raíz

  // Entrada de datos

  do {
     printf("Introduzca un número positivo: ");
     scanf("%lf", &n);
  } while (n<0);

  // Cálculo de la raíz cuadrada

  r = n/2;

  do {
     prev = r;
     r = (r+n/r)/2;
  } while (fabs(r-prev) > 1e-6);

  // Resultado

  printf("La raíz cuadrada de %lf es %lf", n, r);
}
```

NOTA:
La función `abs` nos da el valor absoluto de un númeo entero.
Para trabajar con reales hemos de usar `fabs`.

Bucles anidados

Los bucles también se pueden anidar:

```
for (i=0; i<N;i++) {
    for (j=0; j<N; j++) {
        printf("(%d,%d) ",i,j);
    }
}
```

genera como resultado:

```
(0,0) (0,1) (0,2) … (0,N)
(1,0) (1,1) (1,2) … (1,N)
                …
(N,0) (N,1) (N,2) … (N,N)
```

Ejemplo

```
#include <stdio.h>

void main ()
{
  int n,i,k;
  n = 0;                          // Paso 1
  for (i=1; i<=2; i++) {          // Paso 2
      for (k=5; k>=1; k-=2) {     // Paso 3
          n = n + i + k;          // Paso 4
      }                           // Paso 5
  }                               // Paso 6
  printf("N vale %d", n);         // Paso 7
}
```

Paso	1	2	3	4	5	3	4	5	3	4	5	3	6	2	3	4	5	3	4	5	3	4	5	3	6	2	7
N	0			6			10			12						19			24			27					
i	?	1											2												3		
k	?		5		3			1			−1				5		3			1			−1				

Cuestiones de estilo

Escribimos código para que lo puedan leer otras personas,
no sólo para que lo traduzca el compilador.

Identificadores

- Los identificadores deben ser **descriptivos**
  - ß `p, i, s...`
  - ü `precio, izquierda, suma...`

- En ocasiones, se permite el uso de nombres cortos para variables locales cuyo significado es evidente (p.ej. bucles controlados por contador)
  - ü `for (elemento=0; elemento<N; elemento++ )…`
  - ü `for (i=0; i<N; i++) …`

Constantes

- Se considera una mala costumbre incluir literales de tipo numérico (**"números mágicos"**) en medio del código. Se prefiere la definición de constantes simbólicas (con `#define` o, mejor aún, con `enum`).
  - ß `for (i=0; i<79; i++) …`
  - ü `for (i=0; i<columnas-1; i++) …`

Expresiones

- *Expresiones booleanas:*
  Es aconsejable escribirlas como se dirían en voz alta.

  ß
```
if ( !(bloque<actual) ) …
```

  ü
```
if ( bloque >= actual ) …
```

- *Expresiones complejas:*
  Es aconsejable dividirlas para mejorar su legibilidad

  ß
```
x += ( xp = ( 2*k<(n-m) ? c+k: d-k ));
```

  ü
```
if ( 2*k < n-m )
   xp = c+k;
else
   xp = d-k;

x += xp;
```

  ü
```
max = (a > b) ? a : b;
```

Comentarios

- Comentarios descriptivos: Los comentarios deben comunicar algo. Jamás se utilizarán para “parafrasear” el código y repetir lo que es obvio.

  ß
```
i++;     /* Incrementa el contador */
```

  ü
```
/* Recorrido secuencial de los datos*/

for (i=0; i<N; i++) ...
```

- Sangrías: Conviene utilizar espacios en blanco o separadores para delimitar el ámbito de las estructuras de control de nuestros programas.

- Líneas en blanco: Para delimitar claramente los distintos bloques de código en nuestros programas dejaremos líneas en blanco entre ellos.

- Salvo en la cabecera de los bucles `for`, sólo incluiremos una sentencia por línea de código.

- Sean cuales sean las convenciones utilizadas al escribir código (p.ej. uso de sangrías y llaves), hay que ser consistente en su utilización.

```
while (…) {
   …
}

for (…;…;…) {
   …
}

if (…) {
    …
}
```

```
while (…)
{
    …
}

for (…;…;…)
{
    …
}

if (…)
{
    …
}
```

El código bien escrito es más fácil de leer, entender y mantener (además, seguramente tiene menos errores)