Capítulo

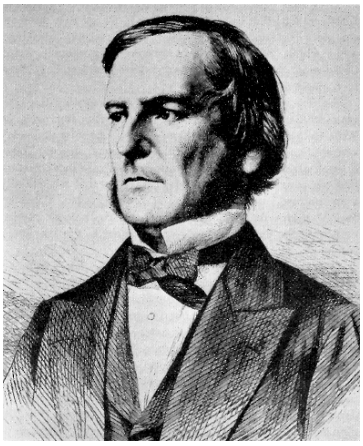
2

Sentencias de selección en C

En este capítulo y en el siguiente se describen las sentencias clave para la realización de cualquier programa, como son las sentencias de selección y de iteración, que permitirán ir avanzando en el grado de dificultad de los programas realizados, bien haciendo preguntas en el programa, bien repitiendo un conjunto de sentencias.

Operadores de relación y lógicos

Hasta ahora hemos visto que las sentencias se ejecutan secuencialmente según se han escrito, pero es muy habitual que una sentencia se ejecute o no, según una condición anterior. Para ello deberemos tener la capacidad de hacer preguntas en el programa y de obtener una respuesta del tipo sí o no. Raro es el algoritmo (instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad, en términos coloquiales la receta) que no incluye alguna selección.



Algo quizás nuevo para algunos alumnos es que en C cualquier expresión tiene un valor. Por ejemplo, una expresión de asignación tiene el valor de la variable de la parte izquierda. Esto unido a que en ANSI C no existen operadores booleanos[1], ya que como hemos dicho son enteros, siendo 0 falso, y cierta cualquier cosa distinta de 0 (normalmente 1), hace que las condiciones o las preguntas que se pueden hacer en el programa sean más generales (en C99 se incluyó el tipo de dato *bool* pero no forma parte del ANSI C). Como curiosidad prueba:

```
printf("El valor de false y true es %d %d \n"
,3>4,2<3);
```

Las preguntas se hacen en forma de expresiones enteras que tienen un valor y cuya respuesta es verdadera o falsa. Una expresión puede ser:

1. Una variable (incluida una asignación como exotismo).
2. Dos expresiones entre un operador relacional.
3. Dos expresiones entre un operador lógico.

[1] El nombre procede del padre de la lógica George Boole (ver retrato) que fue el primero en definir un sistema algebraico de lógica que define las funciones AND, OR y NOT.

Por lo tanto, para realizar preguntas debemos emplear operadores especiales que pueden ser de dos tipos: operadores relacionales y operadores lógicos.

Los operadores relacionales sirven para comparar y tienen una precedencia media entre los operadores aritméticos y la asignación (que es la de menor precedencia). Así, si definimos B como una variable entera podemos hacer: B = 4 > 5; lo cual es falso, por lo tanto, B valdrá 0.

Los operadores relacionales son los siguientes:

== != >= <= > <

No hay que confundir el operador relacional "==" con el operador de asignación "=". Uno sirve para comparar dos operandos, el otro cambia el valor de uno de ellos. El "!=" es el distinto (en otros lenguajes el <>) y los demás tienen el significado obvio: mayor o igual, menor o igual, mayor y menor. Hay que tener en cuenta que siempre se deberán comparar cosas iguales, es decir, enteros con enteros, flotantes con flotantes y caracteres con caracteres (es mala práctica y no tiene sentido matemático comparar con "==" flotantes).

El otro tipo de operadores son los lógicos (los definidos por Boole), hay tres:

AND (^) con && OR (v) con || NOT (~) con !

con ellos se forman las tablas de verdad, que nos dirán el resultado de utilizar uno de estos operadores. Las tablas son (T es TRUE y F FALSE):

OP 1	OP 2	AND
F	F	F
F	T	F
T	F	F
T	T	T

OR
F
T
T
T

OP	NOT
F	T
T	F

Los operadores "&&" y "||" son binarios (afectan a dos operandos) y el "!" es unario (un operando). El "!" es el que tiene la precedencia más alta (correspondería con el signo menos unario), después le sigue el "&&" (que está entre los operadores de relación y la asignación) y el "||" entre el "&&" y la asignación "=".

La tabla de precedencia (mayor arriba) es la siguiente[1]:

NOT	!
Relacionales	< <= >= >
Relacionales	== !=
AND	&&
OR	||
asignación	=

El orden de evaluación no está garantizado en expresiones aritméticas (en C clásico), pudiendo hacerse antes un término u otro, pero en las expresiones lógicas se garantiza que el orden es de izquierda a derecha (como las aritméticas del ANSI C):

```
(ch = getchar()) != EOF && ch != '\n'
```

primero se asignará valor a ch (por los paréntesis, si no, no funcionaría) y después se comparará con EOF y '\n'.

[1] Hay una tabla completa de precedencia en el apéndice D.

Además, se hará en cortocircuito (si es 0 en AND y 1 en OR se para la evaluación):

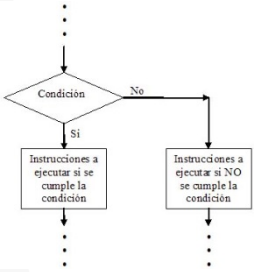
```
numero != 0 && 12 / numero == 2
```

por la precedencia no necesitamos poner paréntesis a los lados del && ya que los relacionales tienen mayor precedencia, además se garantiza que primero se va a evaluar numero != 0 y al ser en cortocircuito, si "numero" es igual a 0 no se producirá la división por 0 ya que la expresión completa será 0 (0 && cualquier cosa es 0).

Sentencias condicionales

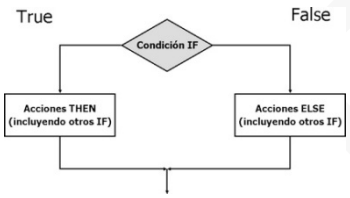
En cualquier programa normal es necesario tomar decisiones sobre si se ejecuta una cosa u otra. En C, la estructura que permite este control se llama *if*. La sentencia *if*, nos permitirá ejecutar una instrucción dependiendo del valor de una expresión, la estructura es:

```
if ( expresión )
        ejecuta sentencia 1;
```



Si la expresión es cierta (distinto de 0), ejecutará la sentencia 1, si no, no la ejecutará y pasará a la siguiente sentencia. Si queremos que ejecute más de una sentencia las tendremos que poner en un cuerpo/bloque (entre llaves).

Otra estructura muy común de selección es el *if* con *else*. Es similar a la anterior, pero en este caso, si la expresión se evalúa a 0, en vez de realizarse la siguiente instrucción después del *if*, se realizará la instrucción a continuación del *else*. La estructura sería:




```
if ( expresión )
        ejecuta sentencia 1;
else
        ejecuta sentencia 2;
```

También aquí, si se quiere realizar un grupo de instrucciones, tanto en el *if* como en el *else* se utilizarán los marcadores de bloque: las llaves.

En C se pone un punto y coma después de cada sentencia (actúa como terminador no como separador), por lo tanto, la sentencia 1 debe terminar en punto y coma (a diferencias de otros lenguajes como Pascal).

Veamos el ejemplo de la resolución de una ecuación de segundo grado (se añade la librería matemática *math.h* para obtener la raíz cuadrada, una referencia de esta librería se encuentra en el capítulo sexto):

```
#include <stdio.h>
#include <math.h>        /* hay que compilar con   -lm  */
int main()
{
  int coef1, coef2, coef3;           /* declaraciones */
  float discrimi, sol1, sol2;

  printf("Introduce los coeficientes de la ecuacion \n");
  printf("Coeficiente A : ");
  scanf("%d", &coef1);
  printf("Coeficiente B : ");
  scanf("%d", &coef2);
  printf("Coeficiente C : ");
  scanf("%d", &coef3);
```

```
  discrimi = coef2 * coef2 - 4 * coef1 * coef3;
  if (discrimi >= 0.0)
  {
         sol1 = (-coef2 + sqrt(discrimi)) / 2 / coef1;
         sol2 = (-coef2 - sqrt(discrimi)) / 2 / coef1;
         printf("La solución primera es : %f", sol1);
         printf("La solución segunda es : %f", sol2);
  } // fin de if
  else
  {
         sol1 = - coef2 / 2 / coef1;
         sol2 = sqrt(-discrimi) / 2 / coef1;
         printf("La parte real es : %f", sol1);
         printf("La parte imaginaria es : %f", sol2);
  } // fin de else
   return(0);
} // fin de main
```

Vemos como, dependiendo del valor del discriminante, tomamos la decisión de resolver la ecuación de forma real o de forma compleja, además, después de cerrar un bloque, no hace falta poner un punto y coma.

No hay ninguna restricción en cuanto a las instrucciones que se pueden introducir dentro de una sentencia *if*, de hecho se pueden introducir sentencias *if* dentro de sentencias *if* (anidamiento, en el ANSI C se permiten al menos 15 niveles), el factor limitativo será la legibilidad del programa que se produzca. Cuantos más *if* anidados se utilicen, más enmarañado y difícil de entender se hará el programa.

Cuando se anidan muchos *if* con *else*, tenemos que saber con seguridad a que *if* corresponde el *else*. El sangrado es algo que escribimos nosotros y que al compilador le resulta indiferente, por lo tanto la regla que se sigue es que el *else* siempre pertenece al *if* que tenga más cercano, empezando por el más interno. Otra manera de estar seguros de la correspondencia es poniendo estructuras de bloque { }, aunque sólo haya una sentencia interna, en los lugares adecuados.

Veamos un ejemplo de esto con el siguiente trozo de código:

```
if (numero > 6 )
   if ( numero < 12 )
          printf("Caliente !\n");
else
   printf("Lo siento has perdido!\n");
```

¿Cuándo pintará "Lo siento has perdido"?, cuando numero sea <= que 6 o cuando sea >= que 12, es decir, ¿a qué *if* pertenece el *else*? La respuesta es al segundo, da igual el sangrado que se utilice. Si quisiéramos que fuera al primero habría que hacer:

```
if (numero > 6 )
{
   if ( numero < 12 )
          printf("Caliente !\n");
}
else
   printf("Lo siento has perdido!\n");
```

Algunas veces podemos evitar anidar los *if*, por ejemplo, si tenemos las sentencias:

```
if ( condicion1 )
       if ( condicion2 )
              if ( condicion3 )
```

podemos convertirla en una sola sentencia *if*:

```
if (condicion1 && condicion2 && condicion3)
```

Existe otra forma más de selección que es la estructura *if else if*. Ocurre cuando en la parte *else* se utilizan sentencias *if*.

La estructura sería:

```
if (  expresión  )
       ejecuta sentencia 1;
else if (expresión)
       ejecuta sentencia 2;
else
       ejecuta sentencia 3;
```

Un ejemplo de esta sentencia sería:

```
#include <stdio.h>
int main()        /* recibo de la luz */
{
   float kilo;
   float recibo;
   printf("Introduce el gasto en kilowatios: ");
   scanf("%f", &kilo);
   if (kilo < 500.0)
          recibo = 5.0 * kilo;
   else if (kilo < 1000.0)
          recibo = 4.0 * kilo;
   else
          recibo = 3.0 * kilo;
   printf("La cuenta por %.2f kw. es %.2f € \n", kilo, recibo);

        return(0);
}
```

Existe en C un operador, el operador condicional, que realiza sentencias *if* de manera abreviada y que sirve cuando se tienen que utilizar dos alternativas para calcular una expresión.

Es ternario, en la primera se hace una pregunta, en la segunda se ejecuta la sentencia si fuera la respuesta cierta y en la tercera lo contrario. Estas tres partes van separadas por " ? : ".

Así, para asignar el valor absoluto de un número se puede hacer:

```
x = (y < 0) ? -y : y;    <=>   if (y < 0)  x = -y;  else  x = y;
```

Selección simple y múltiple

Cuando tenemos que elegir entre muchas alternativas, es poco eficiente y claro utilizar estructuras *if else if else...*, para ello el C tiene una sentencia *switch* que permite elegir entre distintas alternativas. La forma general de esta sentencia es:

```
switch ( expresión )
{      case primero :       sentencias;
                            break;        /* opcional */
       case segundo :       sentencias;
                            break;        /* opcional */
                      ...
       case último  :       sentencias;
                            break;        /* opcional */
       default :     sentencias;
                            break;        /* opcional */
}
```

La expresión que selecciona las sentencias que se ejecutarán debe ser de tipo escalar discreto (un *char*, un entero o un enumerado), ya que hay que escoger con seguridad una. Los valores que puede tomar esta expresión son los casos que aparecen en el esquema anterior (en el estándar ANSI C al menos 257 casos), por supuesto tienen que coincidir en tipo con la expresión selectora. La sentencia *default*, que es opcional, se ejecutará siempre que ningún valor de los casos coincida con el valor de la expresión de selección.

En cuanto a esta sentencia hay que considerar los siguientes puntos:

1. Dentro de un caso pueden aparecer distintos valores del tipo de la expresión selectora, tendrán que estar especificados como *case* primero : *case* segundo : ...
2. Los casos no tienen por qué estar en orden, pero no se pueden repetir.
3. Dentro de un caso se pueden ejecutar varias sentencias, incluso otras sentencias *switch* (al menos 15 en ANSI C).
4. La sentencia *break* es opcional y sirve para terminar de ejecutar un caso. Si no se pone, se ejecutarán, para una selección dada, todas sus sentencias y las sentencias de los casos inferiores hasta que se encuentre un *break*.
5. La opción *default* también es opcional, si no se pone y no coincide, se ejecutará la sentencia posterior al *switch*.

Para verlo vamos a poner un ejemplo sencillo:

```
#include <stdio.h>
int main()
{
   char ch;

   printf("Dame una letra y te dare una marca de coches:");
   if ((ch = getchar()) >= 'a' && ch <= 'z')
          switch (ch)
          {
            case 's':   printf("seat\n");
                        break;
            case 'p':   printf("peugot\n");
            case 'c':   printf("tambien citroen\n");
                        break;
            default :   printf("renault\n");
                        break;  // para que ?
          }
   return(0);
}
```

De tal manera que si pulsamos 'p' se ejecutarán dos sentencias *printf* hasta llegar al *break* de la opción 'c', y si no es la 'p', 's', o 'c' se ejecutará el *default* al que no es necesario poner un *break*.

En C hay otras sentencias de control, por un lado está la propia *break*, que no sólo se puede utilizar en el *switch*, sino en las estructuras de iteración (bucles) que veremos en el capítulo siguiente, y sirve para interrumpir el control de una instrucción y pasar a la siguiente.

Por otro lado está *continue*, que se utiliza en los bucles de la misma forma que *break* pero sólo para una iteración.

También está la instrucción maldita *goto*, que va en contra de la programación estructurada, y que produce un salto incondicional allá donde indique la etiqueta asociada. No recomendable.

Y por último esta la sentencia *exit* y que produce la terminación del programa ejecutado. El argumento que se le da indica situaciones de error. Si es cero es una salida sin error, si es distinto de cero con error (la codificación es libre del programador).

Ejemplos

```c
/* Ejemplo 7.c
Comprende las operaciones que hay.
Piensa como aparecerán en pantalla lo indicado en los printf.
Observa como las condiciones (expresiones booleanas) se evalúan siempre de izquierda a derecha.
Observa como las condiciones (expresiones booleanas) se evalúan siempre en cortocircuito.      */


#include <stdio.h>                              /* fichero de cabecera para el preprocesador */
int main()
{
        int a=2,b=0;
        char ch='a';

        printf("El valor de false y true es %d y %d \n",3>4,2<3);

        if (a)
            printf("a es cierto \n");
        else
            printf("a es falso \n");

        printf("a & b es %d , a | b es %d, \n", a&&b, a||b);

        if ((ch = getchar()) != EOF && ch != '\n')
            printf("ch es %c \n ", ch);

        printf("Numero  %d  \n", 12 / b);       // dara excepcion
        if (b != 0 && 12 / b == 2)              // esto no dara excepcion
            printf("Numero  %d  \n", b);

        return(0);
}
```

```c
/* Ejemplo 8.c
Comprende las operaciones que hay.
Se trata de resolver una ecuación de segundo grado.
Se usa la librería m.       */


#include <stdio.h>
#include <math.h>        /* hay que compilar con   -lm  */
int main()
{
    int coef1, coef2, coef3;           /* declaraciones */
    float discrimi, sol1, sol2;

    printf("Introduce los coeficientes de la ecuacion \n");
    printf("Coeficiente A : ");
    scanf("%d", &coef1);
    printf("Coeficiente B : ");
    scanf("%d", &coef2);
    printf("Coeficiente C : ");
    scanf("%d", &coef3);

    discrimi = coef2 * coef2 - 4 * coef1 * coef3;
    if (discrimi >= 0.0)
    {
        sol1 = (-coef2 + sqrt(discrimi)) / 2 / coef1;
        sol2 = (-coef2 - sqrt(discrimi)) / 2 / coef1;
        printf("La soluciÃ³n primera es : %f   ", sol1);
        printf("La soluciÃ³n segunda es : %f\n", sol2);
    }
    else
    {
        sol1 = - coef2 / 2 / coef1;
        sol2 = sqrt(-discrimi) / 2 / coef1;
        printf("La parte real es : %f   ", sol1);
        printf("La parte imaginaria es : %f\n", sol2);
    }

        return(0);
}
```

```c
/* Ejemplo 9.c
Comprende las operaciones que hay.
Practica con las sentencias de selección
     if  - else if -  else.
Explica que hace el operador condicional.      */


#include <stdio.h>
int main()        /* recibo de la luz */
{
    float kilo;
    float recibo;
    printf("Introduce el gasto en kilowatios: ");
    scanf("%f", &kilo);

        kilo = (kilo < 0.0) ? -kilo:kilo;

    if (kilo < 500.0)
        recibo = 5.0 * kilo;
    else if (kilo < 1000.0)
        recibo = 4.0 * kilo;
    else
        recibo = 3.0 * kilo;
    printf("La cuenta por %.2f kw. es %.2f â,¬ \n", kilo, recibo);

        return(0);
}

```

```c
/* Ejemplo 10.c
Comprende las operaciones que hay.
Practica con las sentencias de selección múltiple borrando algún break que no sea necesario.      */


#include <stdio.h>
int main()
{
       char ch;

   printf("Dame una letra y te dare una marca de coches:");

   if ((ch = getchar()) >= 'a' && ch <= 'z')
       switch (ch)
       {
         case 's': printf("seat\n");
               break;
         case 'p': printf("peugot\n");
         case 'c': printf("tambien citroen\n");
               break;
         default : printf("renault\n");
               break;  // para que ?
       }

       return(0);
}

```