Construcciones condicionales

Una de las construcciones importantes que pueden especificarse en un programa es el hecho de realizar diferentes tareas en función de ciertas condiciones. Esto es, ejecutar una parte del código u otra, condicionalmente. Para ello será necesario especificar dichas condiciones y disponer de un mecanismo para indicar qué acciones tomar dependiendo de cómo se evalúe una determinada condición en un momento dado de la ejecución del programa.

Antes de empezar, un recordatorio. El lenguaje C no dispone de valores booleanos o lógicos, que podrían usarse en la evaluación de condiciones. En su defecto, C ”simula” los valores falso y cierto, como el valor numérico cero, y cualquier valor no cero (incluyendo negativos), respectivamente.

Así pues, en este capítulo veremos las distintas maneras que C ofrece para controlar el flujo de ejecución de un programa de forma condicional, que son:

- La sentencia if.
- La sentencia if-else.
- La sentencia while.
- La sentencia do-while.
- La sentencia for.

* La sentencia *switch.*

Construcción *if*

La construcción ***if*** es similar a la existente en otros lenguajes de programación, aunque en C posee ciertas peculiaridades. El formato general de esta construcción para decidir si una determinada sentencia debe ejecutarse o no (alternativa simple) es el siguiente:

```
if (condición)
sentencia;
```

```
if (condición)
{
  sentencia 1;
      ……..
  sentencia N;
}
```

La construcción ***if*** puede escribirse también de forma más general para controlar la ejecución de un grupo de sentencias, de la siguiente manera:

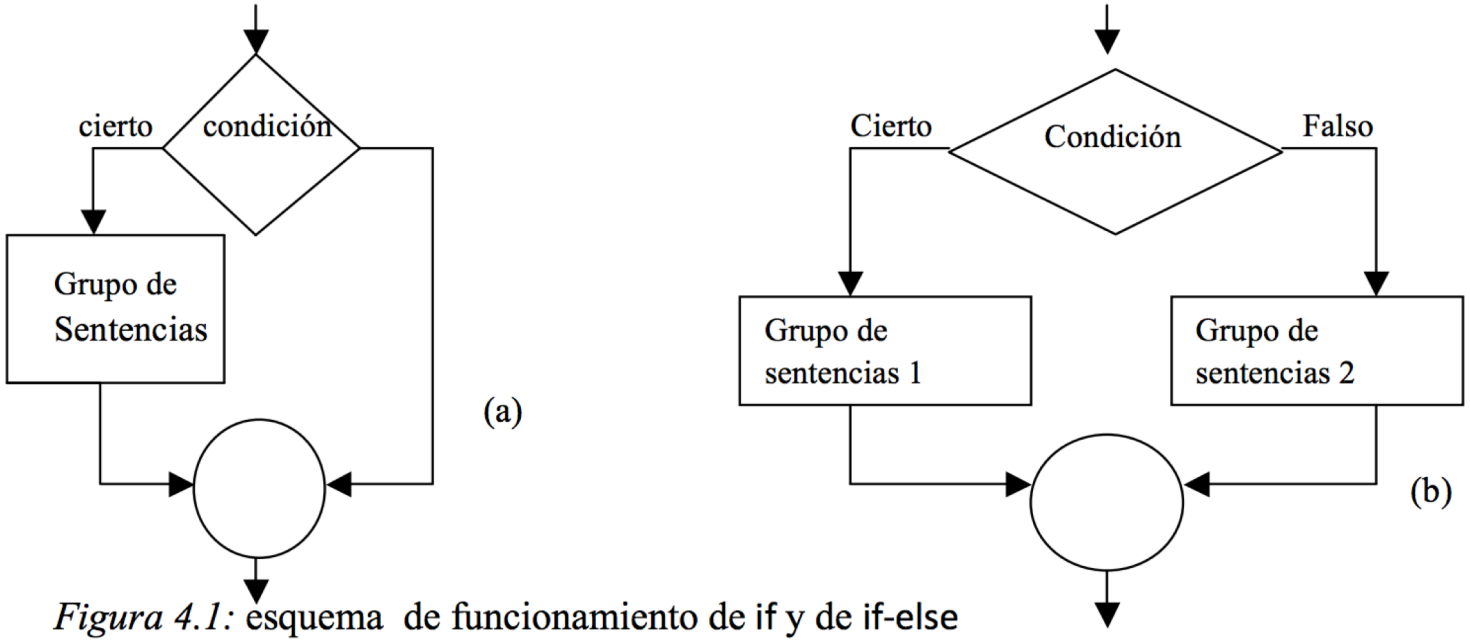

Figura 4.1: esquema de funcionamiento de if y de if-else

```
If (condición)
{
  Sentencia_1;
  Sentencia_2;
  .   .   .
  Sentencia_N;
}
```

El funcionamiento de la *construcción if* es muy simple. En primer lugar se evalúa la condición, que no es otra cosa que una expresión de tipo entero. A continuación, si la expresión se ha evaluado como cierta, se ejecuta la sentencia o grupo de sentencias. En caso contrario la ejecución del programa continua por la siguiente sentencia en orden secuencial (ver Fig. 4.1 (a)).

Construcción *if - else*

La construcción ***if - else*** es usada de la siguiente forma:

```
if (condición)
 sentencia 1;

else
 sentencia 2;
```

```
  if (condición)
{
   sentencia 1;
        ……..
   sentencia N;
}
   else
    sentencia N+1;
```

El funcionamiento de la *construcción* ***if - else*** es muy simple. En primer lugar se evalúa la condición, si se evalua como cierta la ***sentencia 1*** es **evaluada**. Si la expresión es evaluada como falsa se dirige a la condición ***else*** y se evalua la ***sentencia 2***.

Ejemplo

```
 1: // Demonstrates if and else statements and some of C's relational operators
 2:
 3:    #define CURRENTYEAR 2013
 4:    #include <stdio.h>
 5:
 6:    int birth_year, age;
 7:
 8:    int main(void)
 9:    {
10:        printf("Enter the year you were born: ");
11:        scanf("%d", &birth_year);
12:
13:        // Two tests to calculate whether the user was a leap year birth
14:
15:        if (birth_year % 4 == 0)
16:            printf("You were born in a leap year!\n");
17:        else
18:            printf("You were not born in a leap year!\n");
19:
20:        age = CURRENTYEAR - birth_year;
21:
22:        // Can check on voting age as well as drinking age
23:
24:        if (age >= 18)
25:            printf("You can vote this year!\n");
26:        if (age <= 21)
27:            printf("It is illegal for you to drink alcohol!\n");;
28:
29:        return(0);
30:
31:    }
```

Construcción *while*

La construcción ***while*** es similar a la existente en otros lenguajes de programación. Sin embargo, debido a que en C toda sentencia puede considerarse como una expresión, la construcción *while* de C ofrece cierta potencia añadida.

```
while (condición)

sentencia;
```

```
while (condición)

{
   sentencia 1;
        ……..
   sentencia N;
}
```

Construcción *while*

El funcionamiento de esta construcción es bastante simple. El cuerpo del bucle, es decir, la sentencia o grupo de sentencias dentro del bucle, se ejecuta mientras el valor de la expresión que actúa de condición sea cierto. En el momento en que la condición sea falsa, la ejecución del programa continua secuencialmente con la siguiente instrucción tras el bucle.

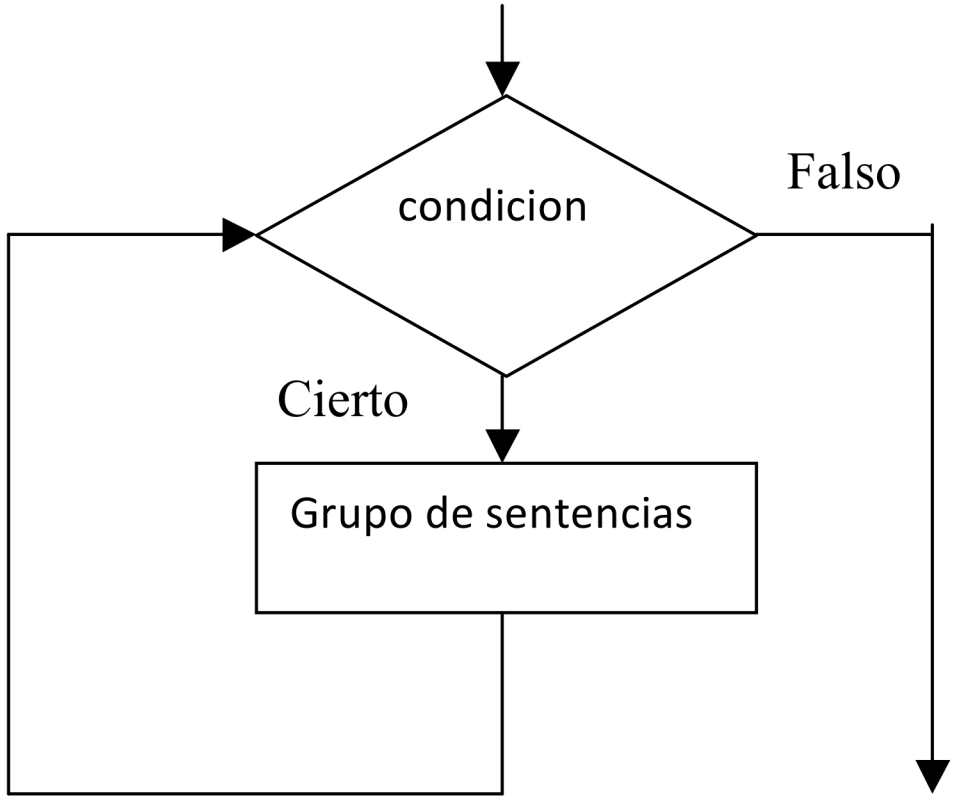

Figura: Esquema de funcionamiento de ***while***

El siguiente ejemplo calcula la media de una secuencia de números enteros leídos por teclado acabada en -1:

```
#include<stdio.h>
void main()
{
 int num,cont,suma;
 cont=0;
 suma=0;
 printf("\n Teclea una secuencia de numeros, -1 para salir:");
 scanf("%d",&num);
 while(num!=-1)
 {
      cont++;
```

```
        suma=suma+num;
        scanf("%d",&num);
        }
        if(cont!=0)
            printf("\n La media es %d",sum/cont);

        else
            printf("\n La secuencia esta vacia");

        return;
}
```

Construcción *do - while*

La construcción ***do - while*** ejecuta primero el cuerpo del bucle y después evalúa la condición. Por lo cual, el cuerpo del bucle se ejecuta como mínimo una vez.

```
do
{
                    sentencia o grupo de sentencias;
}while(condición);
```

El siguiente ejemplo cuenta el número de veces que aparece el número en una secuencia de números enteros acabada en -1:

```
#include <stdio.h>
void main()
{   int num, cont;
    cont = 0;

   do
   {
        scanf( "%d", &num );
        if (num == 3)
            cont++;
   } while (num != -1);
   printf( "El 3 ha aparecido %d veces\n", cont );
}
```

Construcción *for*

La construcción ***for*** iterativa no presenta un formato fijo estricto, sino que admite numerosas variantes, lo que la dota de gran potencia y flexibilidad.

```
for (sentencia inicial; condición; incremento/decremento)
    sentencia;
```

```
for (sentencia inicial; condición; incremento/decremento)
{
    sentencia 1;
    ……..
    sentencia N;
}
```

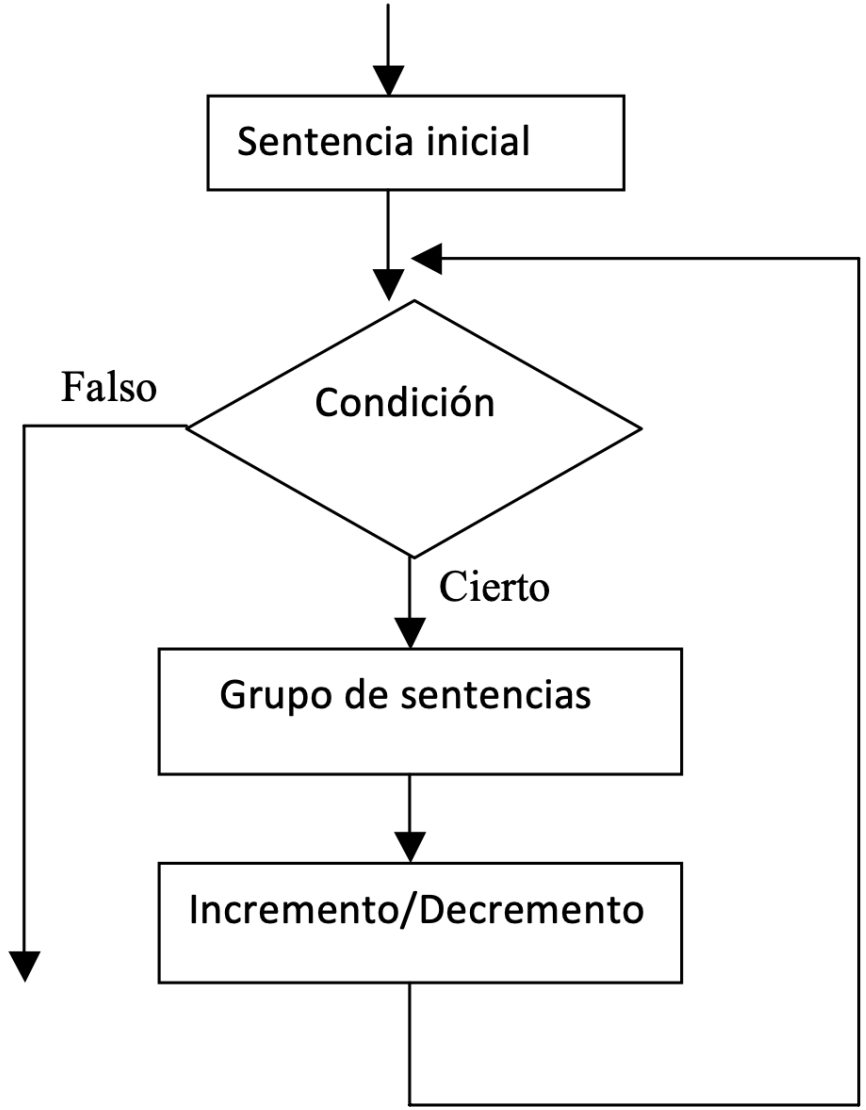

Figura: Esquema de funcionamiento de ***for***

Ejemplo:

El programa del siguiente ejemplo utiliza la construcción ***for*** para calcular la sumatoria

$$\sum_{i=1}^{i=10} i^3 = ?$$

```
#include<stdio.h>
    int main()
    {
     int i, cubo, suma;
     suma=0;
     for (i=1;i<=10;i++)
          {
               cubo=i*i*i;
               suma+=cubo;
          }
     printf(“\n La suma de los cubos es: %d”,suma);
     return 0;
    }
```

Construcción *switch*

La construcción ***switch*** permite especificar múltiples sentencias al estilo if-else-if, pero de manera más compacta, legible y elegante. Su forma general es la siguiente:

```
switch (expresión)
    {
        case constante1:
        sentencia o grupo de sentencias 1;
        break;
        case constante2:
        sentencia o grupo de sentencias 2;
        break;
        ……..
        case constanteN :
        sentencia o grupo de sentencias N;
        break;

        default:
        sentencia o grupo de sentencias por defecto;
        break;
    }
```

Construcción *while*

donde la expresión debe ser de tipo entero o carácter, al igual que todas las constantes asociadas a cada etiqueta case. Es importante resaltar que no pueden usarse variables o expresiones en los distintos case, sino sólo constantes.

El funcionamiento de la ***construcción switch*** es como sigue. En primer lugar se evalúa la expresión. Seguidamente su valor es comparado secuencialmente con el de las diferentes constantes en los case. Si el valor de la expresión coincide con alguna de ellas, se ejecuta el grupo de sentencias correspondiente y **switch** concluye gracias a la sentencia break. En caso contrario, y si existe el caso default (que es opcional), se ejecutaría el grupo de sentencias por defecto.

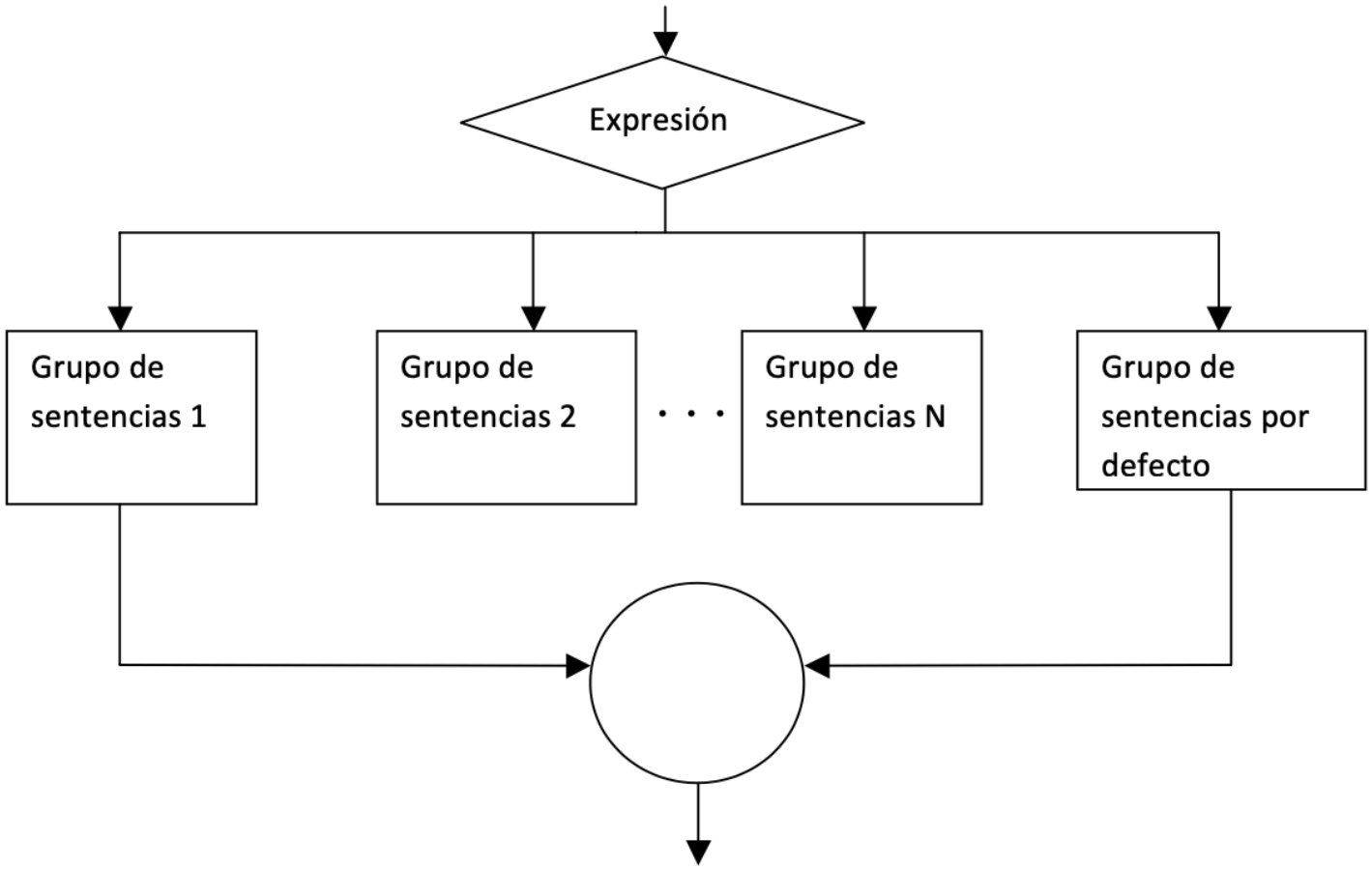

Figura: Esquema de funcionamiento de ***switch***

```
#include!<stdio.h>

 void main()
{
int num;

scanf("%d",&num);

  switch(num)

  {
        case 1:

        printf("Uno \n");
        break;
        case 2:

        printf(”Dos \n");
        break;

        case 3:

        printf(”Tres \n");
        break;

        case 4:

        printf(”Cuatro \n");
        break;

default:

printf("El digito esta fuera de rango.\n");

break;
  }
}
```