Introducción al lenguaje C
3. Estructuras de control

Las estructuras de control permiten modificar el orden secuencial de las instrucciones de un programa.

3.1. Estructuras selectivas

3.1.1. Sentencia if-else

Se trata de la estructura selectiva simple y doble de los principios de programación estructurada. El formato de la instrucción es:

```
if(expresión)
   sentencia1
[else
   sentencia2]
```

La sentencia evalúa la *expresión*, si ésta es verdadera (es decir, si su valor es distinto de 0) ejecuta la *sentencia1*. Si no incluye la cláusula **else**, en el caso de que la *expresión* sea falsa no haría nada; si incluye la cláusula **else** y la expresión es false ejecuta la *sentencia2*.

Hay que tener en cuenta que, normalmente la expresión será de tipo lógico, pero puede ser cualquier valor numérico.

```
if(a+b)
   printf("a+b es distinto de 0");
else
   printf("a+b es igual a 0");
```

También hay que tener en cuenta que tanto *sentencia1* como *sentencia2* pueden ser una proposición simple o una compuesta. En el segundo caso las sentencias que forman el bloque deberían ir encerradas entre llaves y no llevaría punto y coma final.

```
if(a ¡= b){
   a = b;
   printf("a es distinto de b");
}else{
   b = a;
   printf("a es igual a b");
}
```

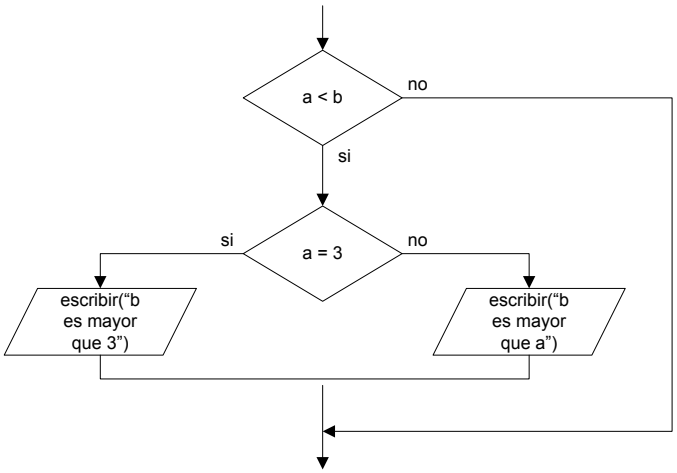
sentencia1 y *sentencia2* pueden ser a su vez estructuras de control (por ejemplo sentencias **if-else**) creando estructuras anidadas.

```
if(a<b)
   c=a;
```

```
else
   if(a>b)
      c=b;
   else
      c =0;
```

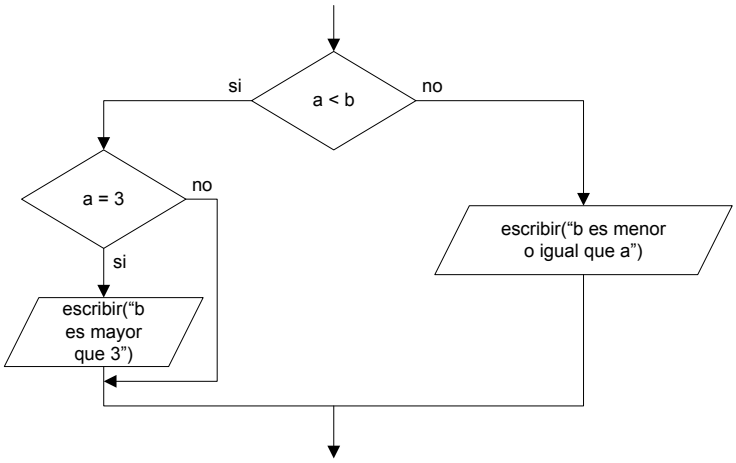
La asociación entre el **`else`** y su **`if`** se hace a partir de la estructura más interna. Esto es válido también si se omite alguna cláusula **`else`**.

```
if(a<b)
   if (a==3)
      printf("b es mayor que 3");
   else
      printf("sólo sabemos que a es menor que b");
```



Si se desea otro comportamiento habría que utilizar las llaves.

```
if(a<b){
   if (a==3)
      printf("b es mayor que 3");
}else
   printf("b es menor o igual que a");
```



En el caso de utilizar varias estructuras selectivas anidadas anidadas su escritura puede quedar más clara si se hace de esta forma:

```
if(expresión1)
   sentencia1
else if(expresión2)
   sentencia2
else if(expresión3)
   sentencia3
else if(expresiónn)
   sentenciaN
else
  sentenciaN+1
```

```
if(precio >= 1000)
   descuento = 0.05;
else if(precio < 2000)
   descuento = 0.08;
else if(precio < 5000)
   descuento = 0.1;
else
   descuento = 0.12;
```

Obsérvese que la estructura no cambia respecto a la sentencia **if-else** anidada que aparece más arriba.

3.1.2. Sentencia switch

Proporciona la estructura selectiva múltiple. Evalúa una expresión de tipo entero y comprueba su valor con cada una de las cláusulas **case**; si coincide ejecuta la sentencia y sigue la comprobación, en caso contrario pasa a la siguiente cláusula. **default** incluirá las sentencias que se ejecutan si fallan todas las comprobaciones anteriores.

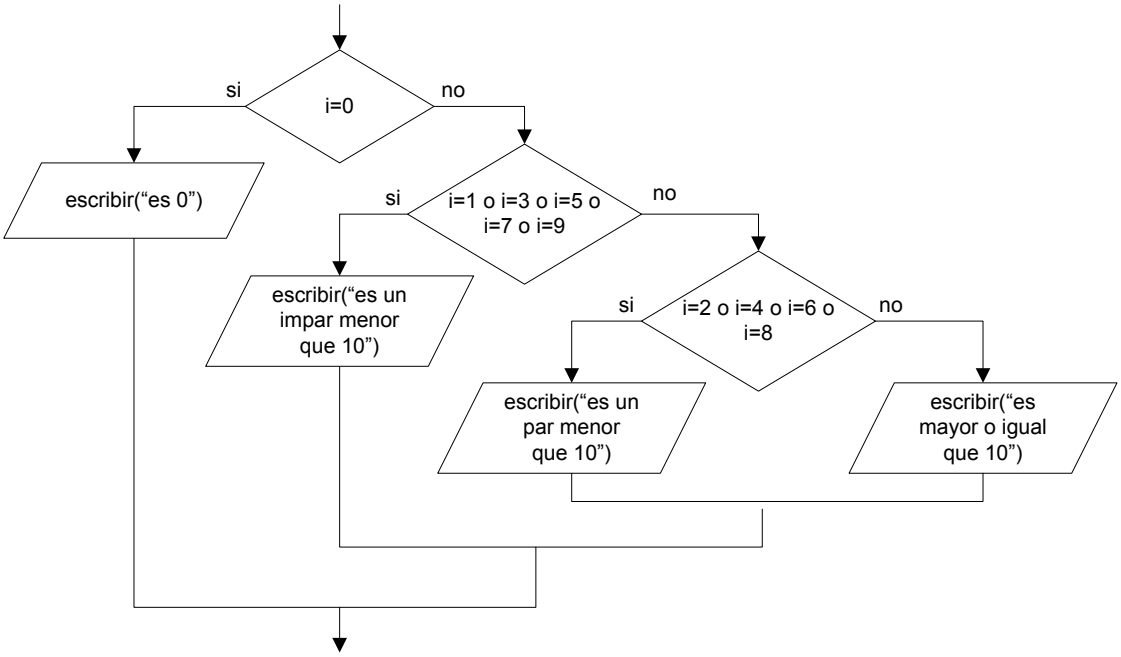
```
switch(expresión){
   case valor1: sentencia1;
   case valor2: sentencia2;
   case valorn: sentencian;
default:
   sentenciaDefault
}
```

A diferencia de lo que ocurre en otros lenguajes, la instrucción **switch** no hace cortocircuito, es decir, una vez que ha encontrado una coincidencia en el valor de un **case**, continúa la comprobación de los otros valores. Para evitar esto se puede incluir una sentencia **break** después de cada caso.

```
switch(expresión){
   case valor1: sentencia1;        //No hace cortocircuito
   case valor2: sentencia2; break; //Hace cortocircuito
   case valorn: sentencian;
default:
   sentenciaDefault
}
```

Otra característica de la sentencia es que se pueden utilizar varios casos para una misma sentencia. Si se utilizan varias etiquetas **`case`** para una misma sentencia, la sentencia se ejecutará si se cumplen esas condiciones.

```
switch(i){
   case 0:
      printf("Es 0\n");
      break;
   case 1: case 3: case 5: case 7: case 9:
      printf("Es un impar menor que 10\n");
      break;
   case 2: case 4: case 6: case 8:
      printf("Es un par menor que 10\n");
      break;
   default:
      printf("Es mayor que 10\n");
}
```



3.2. Estructuras repetitivas

3.2.1. Sentencia while

Ejecuta las sentencias del bucle mientras una expresión es verdadera. Tiene la condición de salida al comienzo de la estructura por lo que las sentencias se ejecutarían de 0 a n veces.

```
while(expresión)
   sentencia;
```

Si *expresión* es distinta de 0 (es decir, si es verdadera) ejecuta la *sentencia*; cuando es 0 finaliza la estructura y el flujo del programa continuaría a la instrucción siguiente. Por ejemplo, el fragmento de código siguiente escribiría los números entre 0 y 5.

```
int j=0;
while(j<=5){
   printf("%i\n",j);
   j++;
}
```

3.2.2. Sentencia for

La sentencia **for** también proporciona un bucle que se ejecutará de 0 a n veces pero que incorpora un mecanismos para la inicialización de variables del bucle, controlar la salida o modificar la variable del bucle.

```
for(expresión1;expresión2;expresión3){
   sentencia;
```

La *expresión1* se ejecutará antes de entrar en el bucle la primera vez, la *expresión3* se ejecuta al final de cada iteración del bucle y la *expresión2* se encargará, al comienzo de cada iteración de comprobar si deben continuar las repeticiones: si la *expresión2* es verdadera se entra en el bucle y en caso contrario se sale. Cualquiera de las tres puede ser una expresión de cualquier tipo e incluso se puede omitir cualquiera de las tres. Sin embargo, lo normal es utilizar la *expresión1* para inicializar la variable del bucle, la *expresión2* para comprobar la salida del bucle a partir de la variable y la *expresión3* para modificar el valor de la variable del bucle. El siguiente ejemplo, también escribiría los números entre 0 y 5.

```
for(j=0;j<=5;j++)
   printf("%i\n",j);
```

El funcionamiento sería exactamente el mismo que el del fragmento de código anterior. Normalmente se elegirá un bucle **for** cuando se sepa cuantas veces se debe ejecutar el bucle antes de entrar en el (bucles controlados por contador), mientras que la sentencia **while** se utilizará cuando se desconozca ese dato (bucle controlado por centinela).

3.2.3. Sentencia do-while

Los bucles generados tanto por la sentencia **for**, como por la sentencia **while** comprueban si el bucle debe ejecutarse al comienzo del mismo, pudiendo darse el caso de que el bucle se repita 0 veces. La sentencia **do..while** haría la comprobación al final.

```
do
   sentencia
while(expresión);
```

Primero ejecuta la *sentencia* y al final del bucle hace la comprobación. Si la *expresión* es distinta de 0 (es verdadera) repite la sentencia; en caso contrario se sale del bucle.

3.3. Sentencias de salto

3.3.1. Sentencia break

La sentencia **`break`** permite la salida de un bucle sin pasar por la comprobación del mismo (es decir, permite crear bucles en los que la comprobación de salida no esté sólo al comienzo o al final). Una sentencia **`break`** dentro de un bucle **`while`**, **`for`** o **`do..while`** (al igual que ocurría en la sentencia **`switch`**) realiza un salto incondicional al final de la sentencia.

El siguiente fragmento de código escribe los número entre 1 y m, pero pararía si alguno de esos números el múltiplo de n:

```
for(int j=1;j<=m;j++){
   if(j % n == 0)
      break;
   printf("%i\n",j);
}
```

Hay que tener en cuenta que este comportamiento también se puede realizar utilizando bucles **`while`** con condiciones múltiples:

```
int j=1;
while(j<=m && j%n !==0){
   printf("%i\n",j);
   j++;
}
```

Utilizando la sentencia **`break`** y con bucle infinitos, sería posible crear estructuras iterativas en las que la condición de salida no tuviera que estar necesariamente al comienzo o al final.

```
while(1){
   sentencia1
   if(expresión)
      break;
   sentencia2
}

for(;;){
   sentencia1
   if(expresión)
      break;
   sentencia2
}

do{
   sentencia1
   if(expresión)
      break;
   sentencia2
}while(1);
```

3.3.2. Sentencia continue

La sentencia **`continue`** permite omitir algunas de las acciones del bucle, ya que realiza un salto a la siguiente iteración; es decir, en el **`while`** y el **`do..while`** se pasa a la comprobación, mientras que en el **`for`** se ejecuta la *`expresión3`*, la que se utiliza normalmente para el incremento.

Por ejemplo, el siguiente fragmento de código procesa de forma distinta los números pares y los impares:

```
for(j=1;j<=m;j++){
   if(j % 2 == 0){
      printf("%i es par\n",j);
      continue;
   }
   printf("%i es impar\n",j);
}
```

3.3.3. Sentencia return

La sentencia **`return`** también se puede utilizar, como **`break`**, para terminar un bucle. La diferencia es que esta sentencia hace que finalice la función dónde se encuentra tomando el control la función llamadora (o el sistema operativo si es que se encuentra en la función `main`).

Además, como se verá más adelante, la sentencia **`return`** puede ir acompañada de una expresión que será el valor que devolverá la función.

3.3.4. Sentencia Goto

La sentencia **`goto`** transfiere el control del programa a las sentencias situadas después de una etiqueta:

`goto` *`etiqueta;`*

Una etiqueta es un identificador seguido de dos puntos que sirve para referenciar una sentencia o grupo de sentencias. La etiqueta siempre debe estar asociada a una sentencia, aunque esté vacía. Por ejemplo,

```
{
   miEtiqueta:
}
```

generaría un error, mientras que

```
{
   miEtiqueta: ;
}
```

sería sintácticamente correcto.

El siguiente ejemplo escribiría los números entre 1 y 5:

```
i = 0;
inicio:;
   i++;
   printf("%i\n",i);
if(i<5) goto inicio;
```

Hay que tener en cuenta que la sentencia **goto** permite saltar a cualquier punto del programa, incluso dentro de un bucle saltándose las sentencias de inicialización. Esta característica la hace muy poco recomendable, y se puede evitar en prácticamente todas las ocasiones, sobre todo teniendo en cuenta que se puede sustituir por una sentencia estructurada.

Uno de los pocos casos en los que se justificaría su utilización sería para forzar la salida de bucles anidados. En estas condiciones, la utilización de un **break** sólo permitiría salir de uno de los bucles; sin embargo con un **goto** sería posible salir de todos los niveles de anidamiento:

```
...
for(int i = 0;i<=m;i++){
   ...
   for(int j=0; j<=n;j++){
      ...
      if(condicionError) goto error;
      ...
   }
}
error:
...
```

Si se produce la `condicionError`, el flujo del programa se derivaría directamente a la etiqueta `error`, saliendo de los dos bucles.De cualquier forma, esto también se podría solucionar utilizando bucles **while** o **do..while** con las condiciones apropiadas:

```
do{
   ...
   do{
      ...
   }while(i<=m || ¡condicionError);
}while(i<=n || ¡condicionError);
...
```

3.4. Ejercicios

1. Codificar un programa que permita determinar si un número entero mayor o igual que 0 es par, impar o 0.
2. Codificar un programa que lea una nota y devuelva su calificación. La calificación será suspenso (menor que 5), aprobado (entre 5 y 6.5), notable (mayor que 6.5 y menor o igual que 8.5) o sobresaliente (mayor que 8.5).
3. Codificar un programa que permita calcular el sueldo neto semanal de un trabajador según los siguientes criterios:

- El sueldo bruto se calculará según las horas semanales trabadas. Las primeras 40 horas semanales se pagarán a la tarifa normal (35 €). Cada hora extra trabajada se pagará a 1,5 veces la tarifa normal.

- El sueldo neto se calculará restando al sueldo bruto las retenciones. Los primeros 1000 euros no llevarán retención. El tramo del sueldo bruto entre 1000 y 1500 euros llevará una retención del 12%. Todo el tramo de sueldo que sobrepase los 1500 euros llevará un 18% de retención.

4. Codificar un programa que permita almacenar 3 números en tres variables y los muestre por pantalla ordenados de menor a mayor.
5. Codificar un programa que permita resolver la ecuación de segundo grado $ax^2+bx+c=0$. Se deberá comprobar:
   - Si a = 0, no es una ecuación de segundo grado.
   - Si el discriminante es 0, sólo tiene una solución.
   - Si el discriminante es negativo la solución es imaginaria.
   - En caso contrario calcular los valores de x_1 y x_2.
6. Codificar un algoritmo que permita introducir el número de un mes y devuelva su nombre.

7. Codificar un programa que permita introducir el número de un mes y devuelva su nombre.
8. Codificar un programa que permita sumar n números.
9. Codificar un programa que permita sumar una cantidad indeterminada de números enteros positivos.
10. Codificar un programa que calcule el factorial de un número entero mayor o igual que 0 introducido por teclado.
11. Codificar un programa que permita hallar la suma de n números, sumando por separado los números pares y los impares.
12. Codificar un programa que permita obtener la nota media de un número indeterminado de notas (antes de comenzar el bucle se desconoce el número de notas a procesar, por lo que habrá que utilizar un bucle controlado por centinela).

    Al final del programa será necesario mostrar la nota media y la calificación (menor que 5, suspenso, mayor que 5 y menor o igual que 6,5 aprobado, mayor de 6,5 y menor o igual que 8,5 notable y mayor que 8,5 sobresaliente).

13. Codificar un algoritmo que muestre por pantalla el máximo común divisor de dos números enteros positivos introducidos por teclado.
14. Codificar un programa que permita sacar por la pantalla el valor mayor de una serie de números positivos introducidos por teclado.
15. Codificar un programa que saque por pantalla los números primos menores que n. n es un número entero positivo introducido por teclado.
16. Codificar un programa que permita sacar todos los números abc que cumplen la siguiente condición abc=a3+b3+c3. Por ejemplo, el número 370 cumple la condición, ya que 33+73+03 = 370

17. Se tiene una cuba con 100 litros de cola y otra con 100 litros de ginebra. Además se tiene un recipiente de un litro. Con el recipiente se desea ir vaciando litro a litro la cuba con cola y llenarla con un litro de ginebra. Codificar un programa que cuente el número de veces que se repite la operación para que en la mezcla en la cuba de cola tenga más ginebra que refresco.
18. Codificar un programa que permita escribir un número al revés.
19. Dada una letra introducida por teclado y secuencia de caracteres también introducidos por teclado, codificar un programa que indique el número de veces que la letra aparece en la secuencia.

    Por ejemplo: si la letra es la "s", y la secuencia de caracteres introducida por teclado es "r, t, s, u, s, v, x, s, t", el algoritmo debería sacar un 3, ya que aparecen tres caracteres "s" en la secuencia.

20. Se desea realizar una estadística sobre el peso de los alumnos de un colegio. Codificar un programa que indique el número de alumnos en cada uno de los siguientes intervalos de peso:
    - Alumnos de menos de 40 kg.
    - Alumnos entre 40 y 60 kg.
    - Alumnos de más de 60 kg.
21. Codificar un programa que sea capaz de hallar la temperatura media mensual de un observatorio durante un mes del año 2009.
    - Se deberá introducir el número del mes por teclado para saber el número de días del mismo.
    - La temperatura media del mes será la media de las temperaturas medias diarias.
    - Por cada día se introducirá la temperatura máxima y mínima del día y la temperatura media de ese día será el valor medio entre ambas temperaturas.
22. Dada la serie a1=0, a2=1, an=3*an-1+2*an-2, codificar un programa que permita obtener el valor y el rango del primer término <= que 1000.