# CAPITULO 5: INSTRUCCIONES DE CONTROL

## 1. INTRODUCIÓN

En este capitulo, denominaremos **bloque de sentencias** al conjunto de sentencias individuales encerradas con un par de llaves. Este conjunto se comportará sintácticamente como una **sentencia simple** y la llave de cierre del bloque no debe ir seguida de un punto y coma.

| Sentencias simples: | Bloque de sentencias: |
|---|---|
| `scanf("%d", n);`<br>`printf("n = %d,  \n", n);` | `{`<br>`c = getchar();`<br>`printf("valor hexadecimal de c: %x",c);`<br>`}` |

**Las instrucciones de control** se utilizan para conseguir ciertas acciones especiales en los programas, tales como las selección condicional, bucles y bifurcaciones. El flujo de programa se controla, en general, mediante estas instrucciones que a su vez requieren tanto las instrucciones simples como los bloques de ellas.

## 2. INSTRUCCIÓN if-eslse

**Sintaxis: if(expresión) sentencia;**
**ó if(expresion) bloque_de_sentencia;**

La sentencia sólo se ejecutará si el resultado de **expresión** es distinto de cero (**verdadero**) , en caso contrario el programa salteará dicha sentencia , realizando la siguiente en su flujo.
Veamos unos ejemplos de las distintas formas que puede adoptar la **expresión** dentro de un **if** :

| | | |
|---|---|---|
| `if(a>b)` | `if((a>b)!=0)` | 1. Las dos expresiones son idénticas. |
| `if(a)` | `if(a!=0)` | 2. Las dos expresiones son idénticas. |
| `if(!a)` | `if(a==0)` | 3. Obsérvese que (!a) dará un valor verdadero sólo cuando **a** sea falso. |
| `if(a==b)` | `if(a=b)` | 4. En la primera se hace una comparación entre a y b; en la segunda se b a a. Ambas son correctas pero distintas. |
| `if(a)` | `if a` | 5. Muy simplificada, poca legibilidad. |

**Proposición else.**

El uso del **else** es optativo , y su aplicación resulta en la ejecución de una , ó una serie de sentencias en el caso de que el resultado de la expresión del **if** sea **falso.**

Su aplicación puede verse en el ejemplo siguiente :

```
If(expresión)
{
Sentencia 1 ;
Sentencia 2 ;
}
sentencia 3 ;
sentencia 4 ;
sentencia 5 ;
```

```
if(expresión)
{
sentencia 1 ;
sentencia 2 ;
}
else
{
sentencia 3 ;
sentencia 4 ;
}
sentencia 5 ;
```

```
1.   En primer ejemplo, no se usa el else y
por lo tanto las sentencias 3 , 4 y 5
se ejecutarán siempre.
2.   En el segundo, las sentencias 1 y 2 se
ejecutan solo si la expresión es cierta y no
se ejecutarán la 3 y la 4 para saltar
directamente a la 5.
   En caso de que la expresión resulte falsa
   se realizarán la 3 y la 4
   en lugar de la 1 y la 2 y luego la 5.
```

En caso de decisiones múltiples, es común el uso de anidamientos **else-if** de la forma indicada abajo:

```
if(exp1)
  sentencia 1;
else if(exp2)
  sentencia 2;
else if(exp3)
  sentencia 3;
else
  sentencia 5;
```

```
if(exp1)
  sentencia 1;
else if(exp2)
      sentencia 2;
     else if(exp3)
          sentencia 3;
          else
          sentencia 5;
```

```
Se suele escribir según la
modalidad de la izquierda.

A la derecha se han expresado
las  asociaciones entre los
distintos else-if para mayor
legibilidad.
```

**El operador condicional (? :)**

**`Sintaxis:  (condicion) ? (expresion 1) : (expresion 2);`**

En las sentencias de selección en las que hay solo un **else** se pueden reescribir mediante el operador condicional (**? :**) , donde las expresiones 1 y 2 deben ser simples y no pueden formar bloques de instrucciones. Primero se evalúa la **condicion**, si es verdadera, se ejecuta la **expresion1**, y si es falsa, se ejecuta **expresion2.**

```
if(x<1)
 puts("No toques nada...");
else
 puts("Continúe,joven...");
```

```
El código se puede reescribir como:

(x<1)? puts("No toques nada..."):
       puts("Continúe,joven...");
```

## 3. INSTRUCCIÓN switch

```
Sintaxis: switch (expresion)
          {
          case constante: sentencia;
                          break;
          case constante: sentencia;
                          break;
          ............
          default:        sentencia;
          }
```

La proposición **switch** se puede considerar como la proposición **if,** múltiple puesto que permite seleccionar tomar decisiones para más de dos posibilidades. Su uso nos permite evitar largos y confusos anidamientos de **else-if .**

El **switch** empieza con la evaluación de **expresión** cuyo resultado debe ser entero (**int , char) ,** y luego se compara sucesivamente con todas las etiquetas**.** Cuando se iguala a una de ellas, se ejecuta la sentencia correspondientes. Si no aparece la palabra **break,** continúa la comparación con el resto de las opciones, pero si aparece un **break,** pues se termina la ejecución del **switch**. Al final del **switch**, aparece una opción optativa llamada **default**, que implica: si no se ha cumplido ningún **case**, ejecute lo que sigue. En este caso no es necesario un **break**. Una característica poco obvia del **switch**, es que si se eliminan los **break** al resultar **cierta** una sentencia de comparación, se ejecutarán las sentencias de ese **case** particular pero también la de todos los **case** por debajo. La razón para este poco comportamiento del **switch** es que se permite que varias comparaciones compartan las mismas sentencias de programa.

```
switch(ch = getchar())
{
 case 'a': printf("Azul");
           break;
 case 'b': printf("Blanco");
           break;
 case 'r': printf("Rojo");
           break;
 default : printf("Error");
}
```

Selección de colores.

Cada case termina con un break, con lo cual se selecciona el color y el control sale fuera del switch.

```
..............
case 'a':
case 'A': printf("Azul");
          break;
case 'b':
case 'B': printf("Blanco");
          break;
case 'r':
case 'R': printf("Rojo");
          break;
.........
Los case 'a' y 'A', tiene la
misma sentencia.
```

```
...........
case 'a': printf("Azul");
case 'b': printf("Blanco");
case 'r': printf("Rojo");
............


Si en la comparación se elige el case
'a', se ejecutarán todas las
sentencias que hay a continuación
hasta el siguiente break o la opción
default.
```

## 4. INSTRUCCIÓN while

**Sintaxis: while(expresion)**
**instruccion;**

Esta instrucción significa: mientras **expresion** dé un resultado **cierto** ejecútese la **instruccion** y, por lo general, contiene algún elemento que altere el valor de **expresion** proporcionando así la condición de salida del bucle.

```
int digito = 0;
while(digito <= 9){
   printf("%d \n", digito);
   ++digito;
}
Programa que visualiza los
Dígitos del 0 al 9.
```

```
int digito = 0;
while(digito <= 9)
   printf("%d \n", digito++);


Una variante más concisa que la primera.
```

## 5. INSTRUCCIÓN do-while.

**Sintaxis: do instruccion**
**while(expresion);**

Significa: ejecute **instruccion**, luego repita la ejecución mientras **expresion** dé un resultado **cierto.** La diferencia fundamental entre esta iteración y la anterior es que el **do-while** se ejecuta siempre al menos una vez , sea cual sea el resultado de **expresion.** En el bucle **while**, la comprobación se realiza al comienzo de cada iteración, mientras que en el bucle **do-while** , la comprobación se hace al final del bucle.

```
void leer_comando()


do{
  mensajes();
  }while(condicion_salida);
```

```
Programa que lee comandos del terminal.

  El chequeo se hace al final puesto que
el cuerpo, como mínimo, se ejecuta una
vez.
```

## 6. INSTRUCCIÓN for

**Sintaxis: for(expresion1; expresion2 ; expresion3)instruccion;**

donde **expresion1** es una asignación de una ó más variables que equivale a una inicialización

de las mismas, (llamada algunas veces índice del bucle) , **expresion2** representa una condición que debe ser satisfecha para que continúe la ejecución de las iteraciones y **expresion3** es otra asignación, que comúnmente varía alguna de las variables contenida en **expresión2.** Cuando se ejecuta un **for**, en primer lugar se evalúa **expresion2** y se comprueba antes de cada iteración del bucle, y al final de cada pasada se evalúa **expresion3,** por lo tanto un bucle **for** es equivalente a un bucle **while;**

```
expresion1 ;
while(expresión2) {
  instruccion;
  expresion3 ;
 }
```

| | |
|---|---|
| `for(digito=0; digito<=9; ++digito)`<br>`  printf("%d \n", digito);` | Programa que visualiza los dígitos del 0 al 9.<br>Ver ejemplo (while). |

Dentro del bucle **for,** todas las expresiones pueden omitirse conservando los puntos y comas. Si se omite **expresion2** , se asumirá que ésta tiene el valor **cierto**, con lo cual habrá que finalizar el bucle mediante algún otro mecanismo, tal como **break.** Otra característica es la de poder tener varios contadores en el índice separados por una coma.

| | |
|---|---|
| `for( ; ; )  printf("Esto no termina");` | 1. bucle infinito. |
| `for( ;cañas <=20; )`<br>`  { cañas += 1;`<br>`  borachera *=2;`<br>`  }` | 2. Sin índice ni su actualización, pero se actualiza dentro del cuerpo del bucle. |
| `for(cont=0, total=0; cont<10; ++cont)`<br>`{total += cont;`<br>`printf("cont=%d,total=%d\n",cont,total);`<br>`}` | 3. Dos contadores en el índice. |

## 7. LAS SENTENCIAS break, continue y goto.

### La sentencia break.

La sentencia **break**, ya descripta con la proposición **switch,** sirve también para terminar bucles producidos por **while, do-while** y **for** antes de cumplirse la condición normal de terminación. Esta sentencia es ideal para terminar con bucle si se detecta un error o alguna otra condición irregular. Veamos su uso para terminar un bucle **while** indefinido.

```
char c ;
printf("Este es bucle indefinido");
 while(1){
   printf( " Apriete una tecla:");
    if( (c = getch()) == 'S')
     break ;
   printf("\n No fue la correcta");
 }
 printf("\n Tecla correcta.");
```

Obsérvese que la expresión while(1) siempre es cierta con lo que el se ejecutará hasta que el operador oprima la tecla S, permitiendo la ejecución de la instrucción break dando por finalizado el bucle while.

## La sentencia continue

La sentencia **continue** es distinta de **break.** En vez de terminar un bucle, termina con la realización de la iteración en curso saltando el resto de la pasada. Es decir todas las instrucciones que se encuentran después de **continue** para ir a la siguiente iteración. Igual que **break,** se puede usar en los bucle **while, do-while** o **for.**

```
char c ;
for(;;){
   c = getch(); /*lee sin eco */
   if(c=='a' || c=='e' || c=='i'||
      c=='o' || c=='u')
     continue;
   putchar(c);
   putchar(c);
 }
```

En este programa, se leen los caracteres que se pulsen. Si se pulsa una vocal minúscula, se realiza un salto al comienzo del bucle, en caso contrario el carácter se escribe dos veces en pantalla.

## La sentencia goto

El hecho de que C tenga la sentencia **goto**, **NO LO OBLIGA A USARLA. E**n programación estructurada siempre hay remedios para evitarlo. El uso del **goto** implica un salto incondicional de un lugar a otro. Esta práctica hace que los programas sean muy difíciles de corregir ó de mantener.