

TEMA 5. CONTROL DE FLUJO DEL PROGRAMA

5.1 Sentencias

Una sentencia es una expresión seguida de un punto y coma.

Sentencia \equiv Instrucción \equiv Expresión

↑ Operadores + Operandos

Sintaxis: Sentencia ; ← El ; es obligatorio

Las sentencias se ejecutan una tras otra, en EL MISMO ORDEN en que han sido escritas → Para cambiar ese orden están las SENTENCIAS DE CONTROL DE FLUJO → En función de una condición deciden qué sentencia, o bloque de sentencias, es la siguiente en ejecutarse. Pueden ser clasificadas en dos categorías:

- Estructuras de selección
- Estructuras de repetición

5.2 Sentencias compuestas (bloques)

Una sentencia compuesta o bloque es un grupo de sentencias agrupadas entre llaves { y }. Son sintácticamente equivalentes a una sentencia simple. Presentan la siguiente sintaxis:

```
{ /* comienzo del bloque */  
    <directivas>      /* (#include, #define, etc.) */  
    <declaraciones>   /* declaraciones de variables locales */  
    <sentencias>      /* conjunto opcional de sentencias */  
} /* final del bloque */
```

5.3 Estructuras de selección (sentencias condicionales)

Permiten ejecutar unas sentencias u otras en función de una condición:

- if else, switch

5.3.1 if else

Se utiliza para expresar decisiones. Sintaxis general:

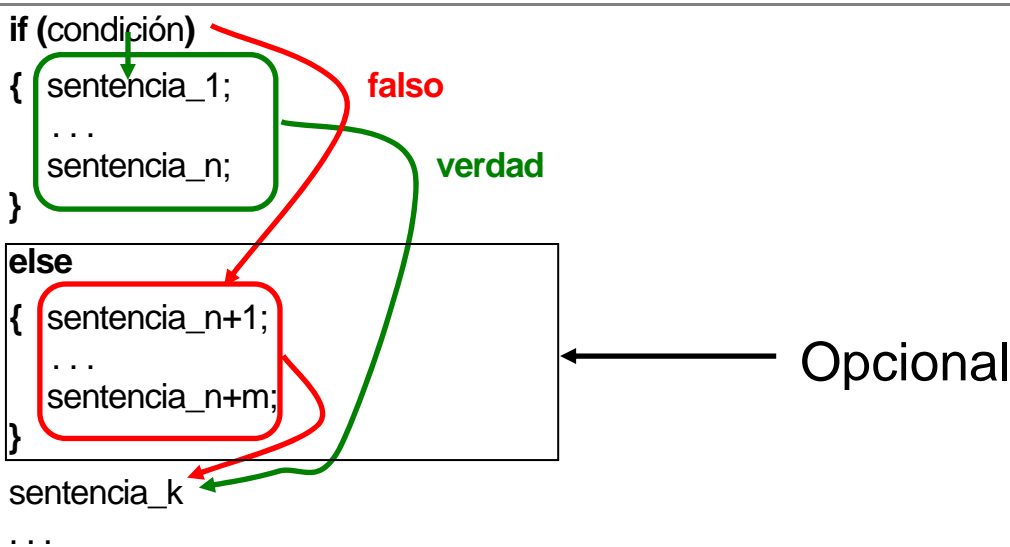
```
if (condición)
    sentencia1; /* se ejecuta si condición es verdadera */
else
    sentencia2; /* se ejecuta si condición es falsa */
```

← Opcional

La forma general del **if** con bloques de sentencia es:

```
if (condición)
{
    sentencia_1;
    ...
    sentencia_n;
}
else
{
    sentencia_n+1;
    ...
    sentencia_n+m;
}
sentencia_k
...
```

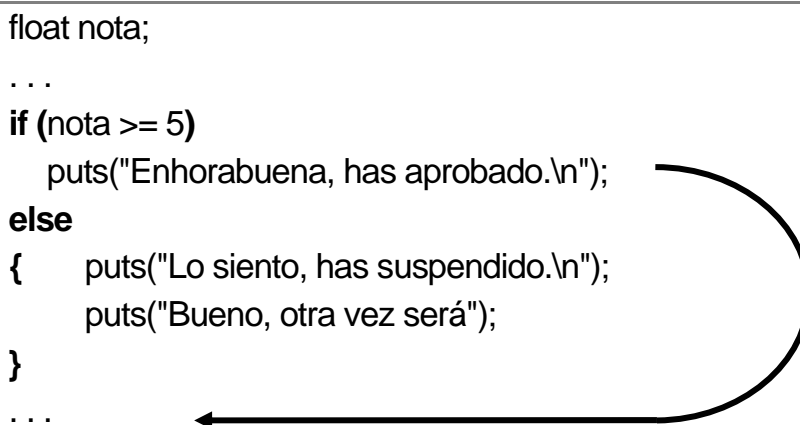
← Opcional



condición == true → se ejecuta sentencias 1, 2, ... n y k

condición == false → se ejecuta sentencias n+1, n+2,...n+m y k

```
float nota;
...
if (nota >= 5)
    puts("Enhorabuena, has aprobado.\n");
else
{
    puts("Lo siento, has suspendido.\n");
    puts("Bueno, otra vez será");
}
...
```



Como en la parte **if** tenemos una sentencia simple las '{' '}' no son necesarias, pero en la parte **else** si hacen falta ya que tenemos un bloque formado por dos sentencias.

5.3.1.1 if anidados

Una sentencia **if** puede tener anidada otra sentencia **if** y así sucesivamente. Un **if** anidado es una sentencia **if** insertada dentro de otro **if** o **else**. Debido a que la parte **else** es optativa, existe una ambigüedad cuando un **else** se omite de una secuencia **if** anidada.

```
float nota;  
...  
if (nota < 5)  
    if (practica > 6)  
        nota = nota + 1;  
    else  
        nota = nota - 0.5;  
printf("La nota es %.2f \n", nota);  
...
```

```
float nota;  
...  
if (nota < 5)  
{ if (practica > 6)  
    nota = nota + 1;  
    else  
        nota = nota - 0.5;  
}  
printf("La nota es %.2f \n", nota);  
...
```

¿A que **if** está asociado el **else**?

El **else** se asocia con el **if** anterior más cercano que no tenga **else**.

Ambos ejemplos son, por tanto, totalmente equivalentes.

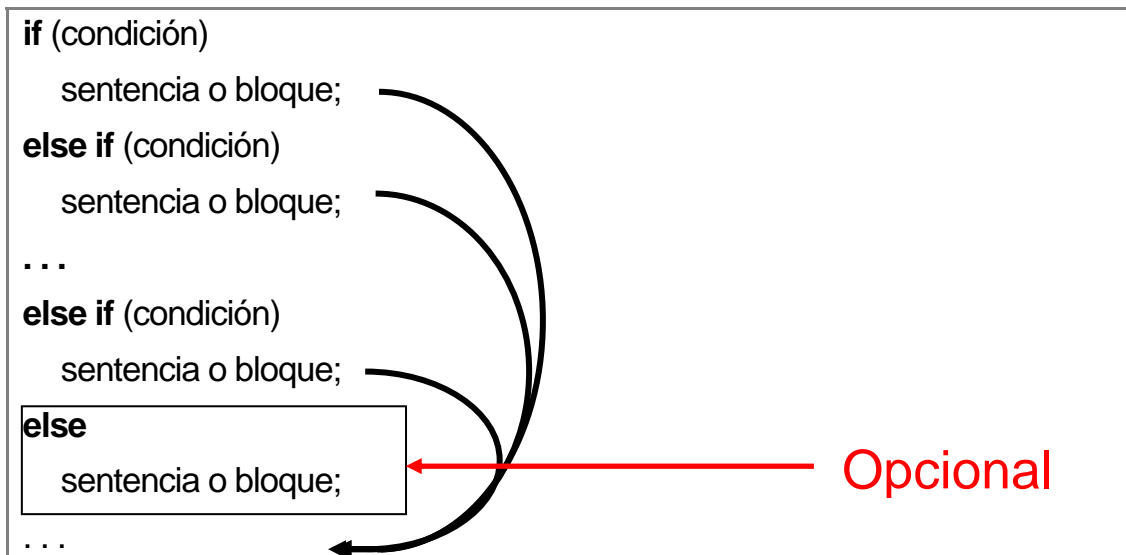
En caso de duda, o para realizar una acción diferente (asociar el **else** con el otro **if**) podemos utilizar llaves para alterar la asociación:

Dependiendo del lugar donde pongamos las llaves, el significado del mismo código fuente varía considerablemente:

```
float nota;  
...  
if (nota < 5)  
{ if (practica > 6)  
    nota = nota + 1;  
}  
else  
    nota = nota - 0.5;  
printf("La nota sacada es %f \n", nota);  
...
```

5.3.1.2 else-if

Es la forma más general de escribir una decisión múltiple:

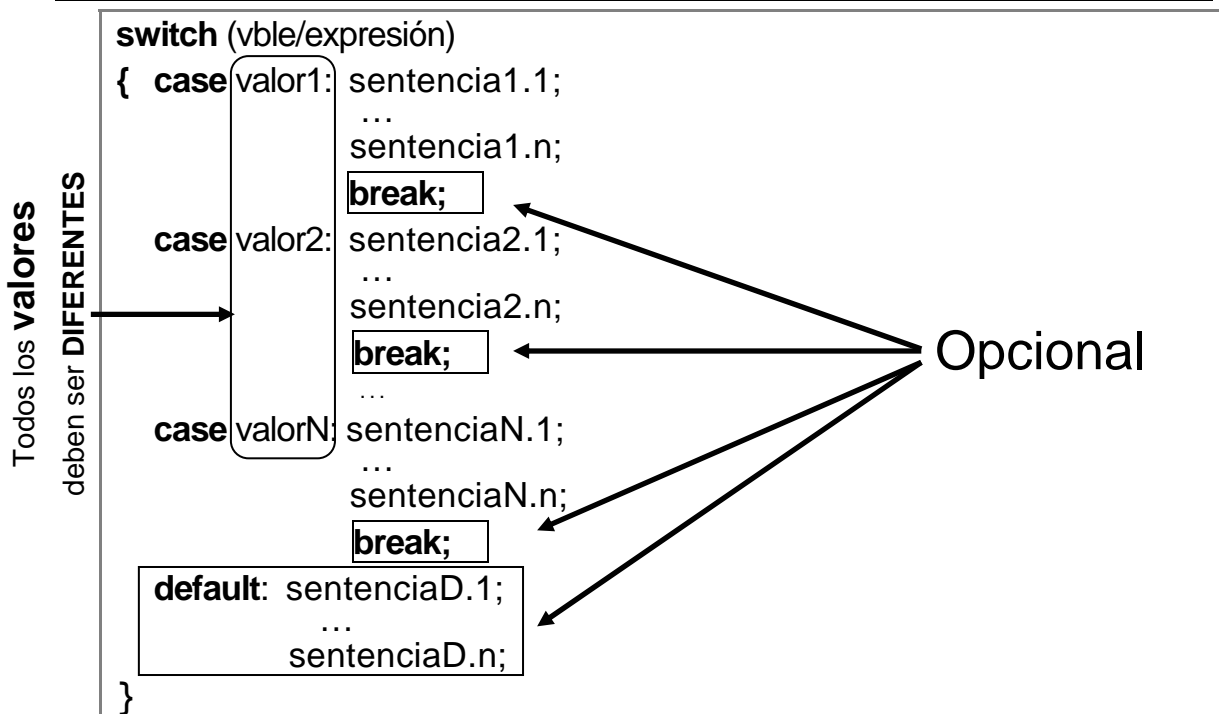


El último **else** maneja el caso "si no se cumple ninguno de los anteriores". Este último **else** no es obligatorio.

Ejemplo:

```
float nota;
...
if (nota == 10)
{ puts("Matricula de Honor \n");    puts("Enhorabuena \n");
}
else if (nota >= 9)
    puts("Sobresaliente \n");
else if (nota >= 7)
    puts("Notable \n");
else if (nota >= 5)
    puts("Aprobado \n");
else { puts("Suspenso \n");
       puts("Lo siento mucho \n");
    }
```

5.3.2 switch

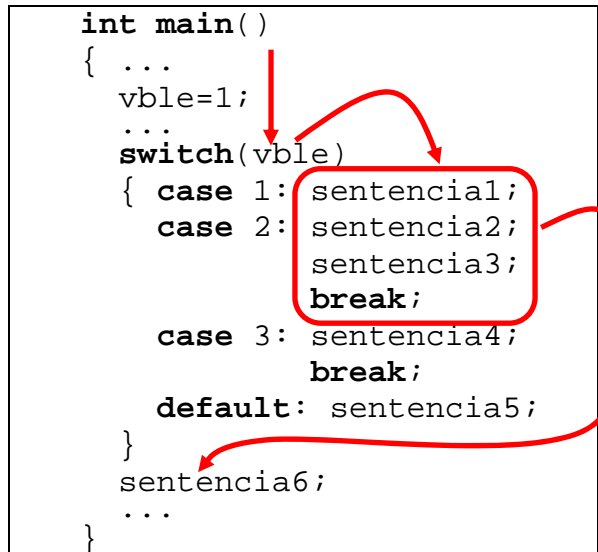
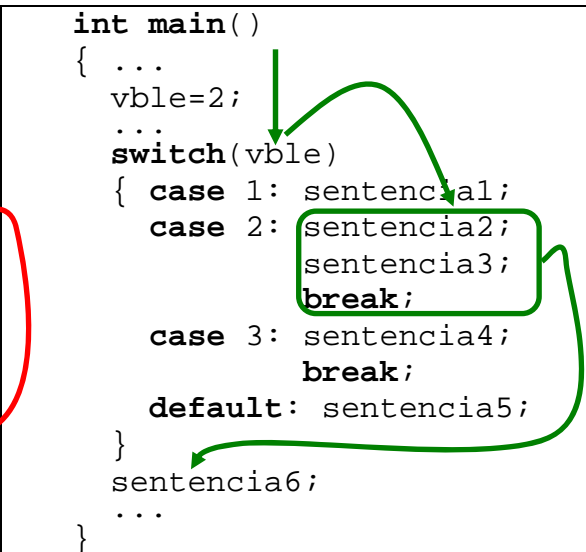
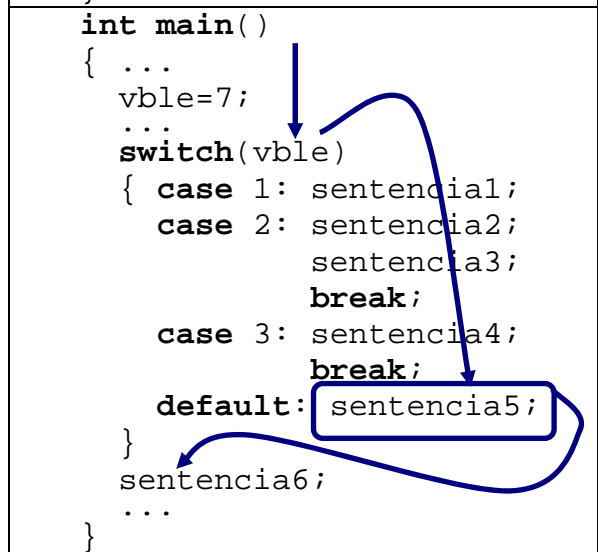
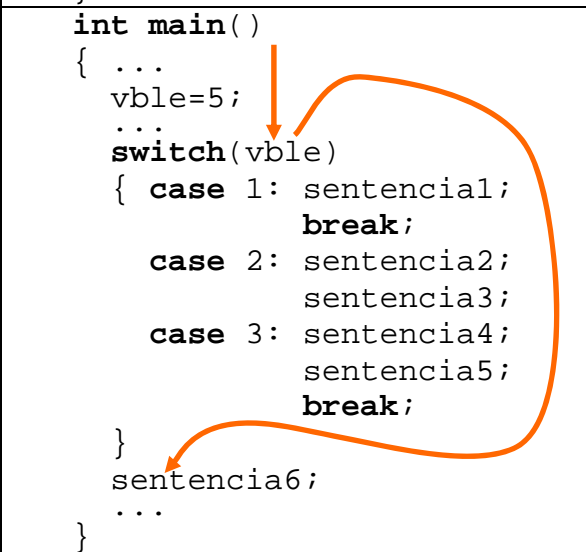
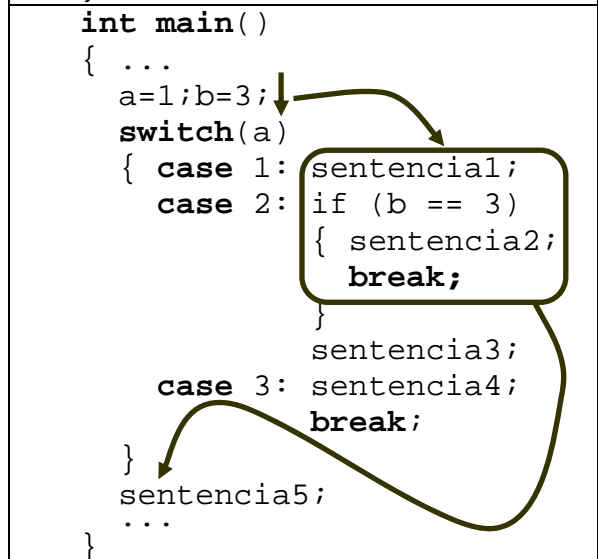
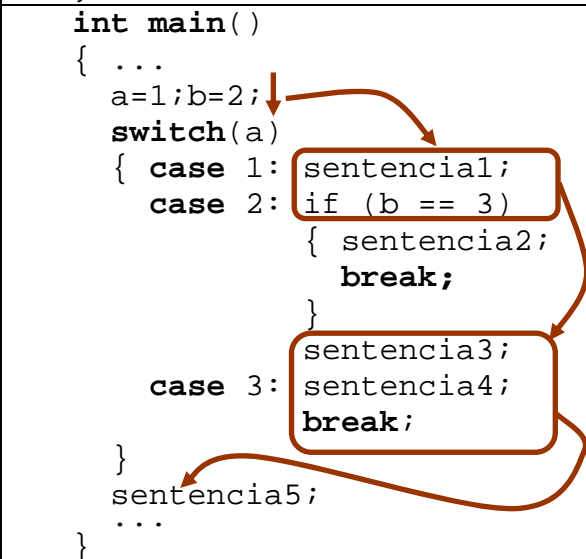


- valor es un valor o expresión constante entera.
- **default** es opcional, se ejecuta si no se cumple ningún **case**.
- Un **switch** puede tener anidada otro **switch** en algún **case**.

Funcionamiento:

1. Se compara el valor de vble/expresión contenida en los paréntesis del **switch**, con los distintos valores constantes enteros **case**.
2. Se ejecutarán aquellas sentencias cuyo **case** tenga un valor igual al de la vble/expresión.
3. El resto de sentencias existentes en los **case** posteriores (y **default** si existe) también se ejecutan a menos que se encuentre una sentencia **break**.
4. Si ningún valor coincide con vble/expresión → Se ejecutarán las sentencias asociadas al apartado **default** (en caso que exista).
5. Si ningún **case** coincide y no hay **default** → no se ejecuta nada

► Funcionamiento switch

<pre> int main() { ... vble=1; switch(vble) { case 1: sentencia1; case 2: sentencia2; sentencia3; break; case 3: sentencia4; break; default: sentencia5; } sentencia6; ... } </pre> 	<pre> int main() { ... vble=2; switch(vble) { case 1: sentencia1; case 2: sentencia2; sentencia3; break; case 3: sentencia4; break; default: sentencia5; } sentencia6; ... } </pre> 
<pre> int main() { ... vble=7; switch(vble) { case 1: sentencia1; case 2: sentencia2; sentencia3; break; case 3: sentencia4; break; default: sentencia5; } sentencia6; ... } </pre> 	<pre> int main() { ... vble=5; switch(vble) { case 1: sentencia1; break; case 2: sentencia2; sentencia3; break; case 3: sentencia4; sentencia5; break; } sentencia6; ... } </pre> 
<pre> int main() { ... a=1;b=3; switch(a) { case 1: sentencia1; case 2: if (b == 3) { sentencia2; break; } sentencia3; case 3: sentencia4; break; } sentencia5; ... } </pre> 	<pre> int main() { ... a=1;b=2; switch(a) { case 1: sentencia1; case 2: if (b == 3) { sentencia2; break; } sentencia3; case 3: sentencia4; break; } sentencia5; ... } </pre> 

La sentencia **switch** se utiliza frecuentemente para procesar selecciones de menú, como la mostrada en el siguiente ejemplo:

```
void ver_menu( )
{ char c;
  printf("1. Añadir\n");
  printf("2. Modificar\n");
  printf("3. Eliminar\n");
  printf("\nIntroduzca la opción deseada: ");
  c = getche( );          // lee una tecla y la muestra en pantalla
  switch (c)
  { case '1': insertar( ); // si la tecla pulsada es 1 se ejecuta insertar( )
    break;
    case '2':          // si la tecla pulsada es 2 se ejecuta modificar( )
      modificar( ); break;
    case '3':          // si la tecla pulsada es 3 se ejecuta borrar( )
      borrar( );
      break;
    default:           // si no es ninguna de ellas muestra un mensaje
      printf("\nElija una opción correcta\n");
  }
}
```

El **switch** es una alternativa al uso de **else if**, pero no siempre:

La sentencia **switch** sólo puede comprobar la igualdad, mientras que el **if** puede evaluar cualquier expresión relacional o lógica.

Por ejemplo, no es posible realizar con la sentencia **switch** el ejemplo realizado con la sentencia **else if** que consistía en escribir en pantalla la calificación sacada por los alumnos en función de sus notas, ya que una sentencia **switch** sólo puede comprobar la igualdad, mientras que en ese ejemplo nos hace falta usar operadores relacionales (\geq) distintos a los de igualdad ($=$).

Para ilustrar la equivalencia entre el uso del **switch** y el uso del **else if** veamos un ejemplo realizado de las dos formas:

Uso de la estructura else if:	Uso de la estructura switch:
<pre> int x, y, op, suma, resta; float cociente; printf("1. Sumar \n"); printf("2. Dividir \n"); printf("3. Calcular Cociente \n"); printf("4. Restar \n"); printf("\nElija opcion: "); scanf("%i", &op); if (op == 1) suma = x + y; else if (op == 2 op == 3) { if (y == 0) printf("Error: división por 0\n"); else { cociente = x / (float)y; printf("%g \n", cociente); } } else if (op == 4) resta = x - y; else printf("Opción incorrecta \n"); ... </pre>	<pre> int x, y, op, suma, resta; float cociente; printf("1. Sumar \n"); printf("2. Dividir \n"); printf("3. Calcular Cociente \n"); printf("4. Restar \n"); printf("\nElija opcion: "); scanf("%i", &op); switch (op) { case 1 : suma = x + y; break; case 2 : case 3 : switch (y) { case 0: printf("Error: division por 0 \n"); break; default: cociente = (float)x / y; printf("%g \n", cociente); } break; case 4: resta = x - y; break; default : printf("Opción incorrecta \n"); } ... </pre>

Del análisis de este ejemplo, se puede ver lo siguiente:

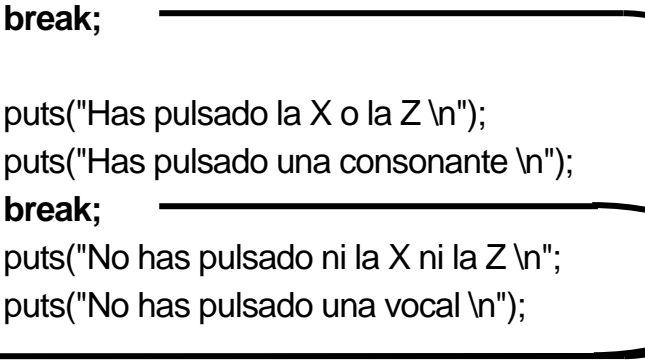
- Pueden existir etiquetas **case** vacías
- Un conjunto de sentencias dentro de un **case** no necesitan '{' y '}'.
- Una sentencia **switch** puede tener anidada otra sentencia **switch**.
- Sin la sentencia **break** al final del grupo de sentencias de **case**, el conjunto de sentencias contenidas en el resto de etiquetas **case** también se ejecutarían.

Para comprender mejor estas observaciones veamos este ejemplo:

```
int tecla;
...
tecla = toupper( getch( ) ); // lee una tecla sin hacer eco en
                             // pantalla y la pasa a mayusculas

switch (tecla) {
    case 'A':    puts(" Has pulsado la A \n");
    case 'E': case 'I':
    case 'O':    puts("No has pulsado una consonante \n");
    case 'U':    puts("Has pulsado una vocal \n");
                 break;
    case 'Z':
    case 'X':    puts("Has pulsado la X o la Z \n");
                 puts("Has pulsado una consonante \n");
                 break;
    default:     puts("No has pulsado ni la X ni la Z \n");
                 puts("No has pulsado una vocal \n");
}

```



Si pulsamos la tecla A aparece en pantalla los siguientes mensajes:

```
Has pulsado la A
No has pulsado una consonante
Has pulsado una vocal
```

Si pulsamos la I:

```
No has pulsado una consonante
Has pulsado una vocal
```

Si pulsamos la Z:

```
Has pulsado la X o la Z
Has pulsado una consonante
```

5.4 Estructuras de repetición

Permiten iterar el flujo de ejecución de un programa dentro de un bloque de sentencias mientras se verifique una condición.

Las estructuras de repetición del lenguaje C son las siguientes:

- **while, do while, for**

¿Conocemos cuántas veces hay que repetir un conjunto de instrucciones (o una sola)?:

- ☞ **Sí** → Usamos un bucle **for**.
- ☞ **No** → ¿Esas instrucciones debe ejecutarse al menos una vez ?:
 - ☞ **Sí** → Usamos un bucle **do while**.
 - ☞ **No** → Usamos un bucle **while**.

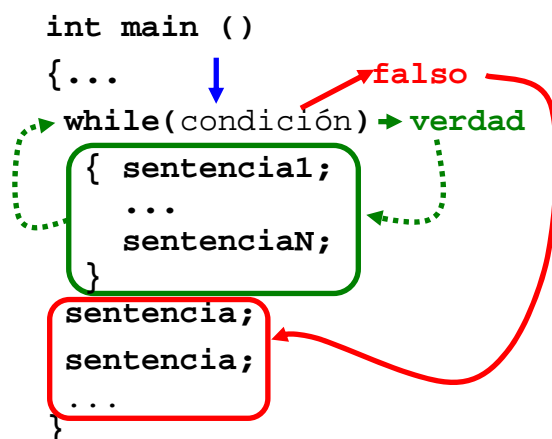
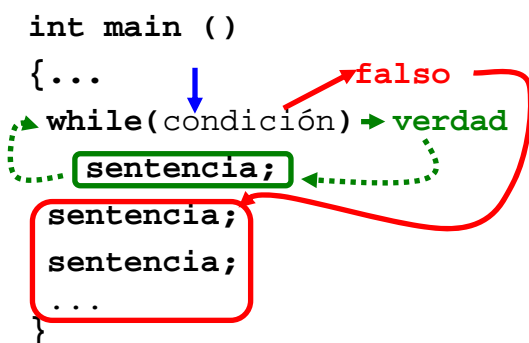
5.4.1 while

La sentencia **while** se utiliza para expresar bucles.

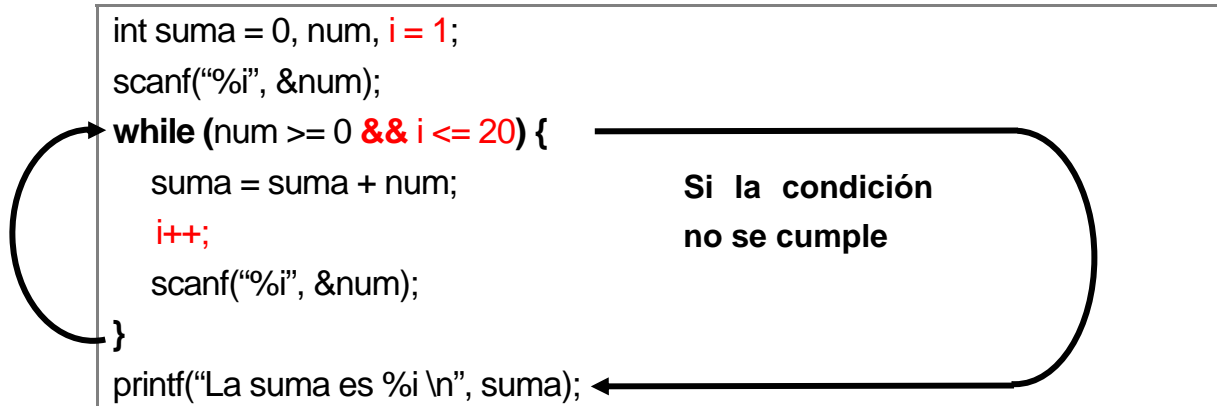
```
while (condición)
    sentencia;
```

```
while (condición)
{
    sentencia1;
    ...
    sentencia1;
}
```

► Funcionamiento while



Como la expresión es evaluada al principio, si no se verifica la expresión, no se ejecutan las sentencias del bucle **while**.



Este programa pide números por teclado y los va sumando hasta que introduzca un número negativo o lleve pedidos 20 números. Si al principio mete un número negativo, no suma ninguno.

5.4.2 do while

Similar al **while**, excepto que 1º se ejecutan las sentencias y luego se evalúa la condición (las sentencias se ejecutan al menos 1 vez).

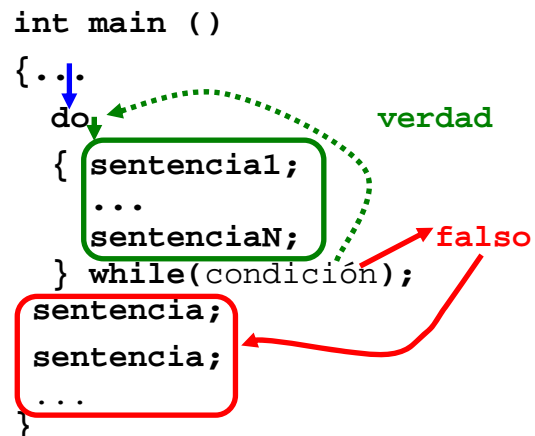
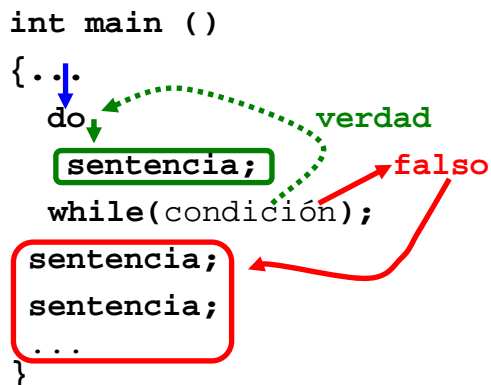
```
do
    sentencia;
while (condición);
```

¡Ojo!

```
do
{
    sentencia1;
    ...
    sentenciaN;
} while (condición);
```

¡Ojo!

► Funcionamiento do while



```
int suma = 0, num, i = 0;
num = 0;
do {
    suma = suma + num;
    i++;
    scanf("%i", &num);
} while (num >= 0 && i <= 20);
printf("La suma es %i \n", suma);
```

Si la condición se cumple

La diferencia entre usar **do while** y **while** es que con el **do while** al menos una vez se ejecuta el bucle, con **while** puede que no.

switch + do while ➔ Se utilizan para realizar MENÚS:

```
void ver_menu( ) {
    char c;
    do {
        printf("1. Añadir\n");
        printf("2. Modificar\n");
        printf("3. Eliminar\n");
        printf("4. Salir\n");
        printf("\nIntroduzca la opción deseada: ");
        c = getche();
        switch (c) {
            case '1': insertar( ); break;
            case '2': modificar( ); break;
            case '3': borrar( ); break;
            case '4': break;
            default: printf("\nElija una opción correcta\n");
        }
    } while (c != '4');
}
```

```
int c;

scanf("%i", &c);

case 1:
case 2:
case 3:
case 4:

} while (c != 4);
```

Este ejemplo es una mejora del hecho antes ya que muestra de nuevo el menú mientras el usuario no seleccione la opción salir (4).

5.4.3 for

La sentencia **for** se utiliza cuando el número de veces que debe ser repetido el bucle está definido. Si el número de repeticiones del bucle es indefinido es mejor usar **while** o **do while**.

```
for (expr_inic; condición; expr_incr)
    sentencia;
```

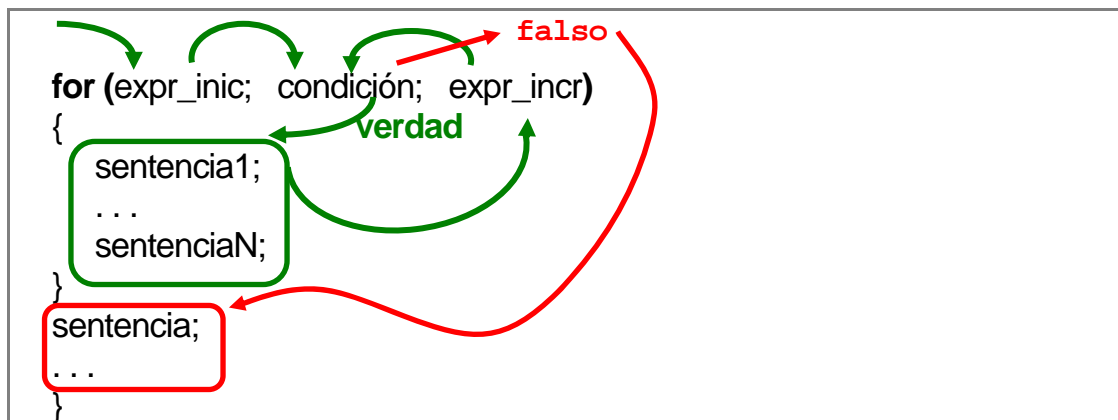
```
for (expr_inic; condición; expr_incr)
{
    sentencia1;
    ...
    sentenciaN;
}
```

donde:

- `expr_inic` es una expresión que inicializa el bucle.
- `condición` es una expresión que determina si el bucle se ejecuta o no.
- `expre_incr` es una expresión que incrementa (decrementa) el índice del bucle cada vez que este se itera.

Cualquiera de las tres partes del bucle **for** se puede omitir, aunque deben permanecer los puntos y comas.

► Funcionamiento for



El ejemplo anterior realizado con el **while** quedaría con el **for** así:

```
int suma = 0, num, i;
scanf("%i", &num);
for (i = 1; num >= 0 && i <= 20; i++)
{
    suma = suma + num;
    scanf("%i", &num);
}
printf("La suma es %i \n", suma);
```

Los bucles **for** y los demás tipos de bucles los podemos anidar.

Ejemplo: tabla de multiplicar de los números pares entre el 10 y el 2.

```
#include <stdio.h>
#include <conio.h>
int main(void)
{ int i, j;
  for ( i=10 ; i >= 2 ; i = i-2 )
  { printf("La tabla del %d \n\n", i);
    for ( j = 1 ; j <= 10 ; j++ )
      printf("%i x %i = %i \n", i , j , i * j);
    printf("\n\n");
    getch( );
  }
  return 0;
}
```

El **for** puede **no tener** expresión de inicialización y/o de incremento:

Ejemplo: Potencias de 2 que hay entre el 2 y el 1000000

<pre>#include <stdio.h> #include <conio.h> int main(void) { float i = 2; for (; i <= 1000000 ;) { printf("%7.0f \n", i); i = i * 2; } getch(); return 0; }</pre>	<pre>#include <stdio.h> #include <conio.h> int main(void) { float i; for (i = 2; i <= 1000000; i = i * 2) { printf("%7.0f \n", i); } getch(); return 0; }</pre>
--	---

Algunos bucles especiales son:

- **Bucles infinitos** ➔ Son bucles cuya condición **SIEMPRE** se cumple ➔ **No terminan nunca**.
- **Bucles vacíos** ➔ Una sentencia vacía consiste en un ; sin más. Un posible uso sería para producir retardos.

5.4.4 El bucle infinito

Los bucles **while**, **do while** y **for** pueden ser bucles infinitos.

En el bucle **for** se puede hacer un bucle sin fin teniendo la expresión de condición vacía, como se muestra en este ejemplo:

```
for( ; ; ) // al dejarlo en blanco la condición siempre se cumple
    printf("Este bucle se ejecuta de manera infinita");
```

Lo mismo podemos aplicar al bucle **while** y al **do while**:

```
while( 1 ) // al ser distinto de cero siempre se cumple
    printf("Este bucle se ejecuta de manera infinita");
```

5.4.5 Bucles vacíos

El cuerpo de cualquier bucle puede ser vacío (;).

El siguiente ejemplo muestra un bucle de retardo usando **for**:

```
for( i = 0; i < 100000; i++ );
```

El ';' al final del **for** representa a una sentencia vacía (nos indica que en el cuerpo del bucle no se realiza nada).

```
for( i = 1; printf("%i \n", i) && i < 10; i++)
    ;
```

Este ejemplo imprime en pantalla los 10 primeros n° naturales. La condición $i < 10$ no impide que este n° se imprima ya que la evaluación de la condición se realiza de izquierda a derecha y entonces antes de comprobar la condición $i < 10$ se ejecuta el **printf**.

El mayor peligro de la sentencia nula se muestra en este ejemplo:

```
i = 1;
while( i <= 10);
    printf("%d \n", i++);
```

Ambos son
Equivalentes

```
i = 1;
while( i <= 10)
    ;
    printf("%d \n", i++);
```

Este programa no imprime los números del 1 al 10 (como parece ser que ocurre), sino que nunca termina y no llega a imprimir nada.