Introducción a la Programación en C
–Control de Flujo–

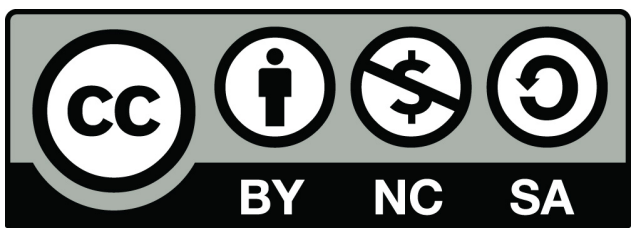
Christopher Expósito-Izquierdo
cexposit@ull.edu.es

Airam Expósito-Márquez
aexposim@ull.edu.es

Israel López-Plata
ilopezpl@ull.edu.es

Belén Melián-Batista
mbmelian@ull.edu.es

José Marcos Moreno-Vega
jmmoreno@ull.edu.es



Contenidos

Introducción a las Sentencias de Control de Flujo:

Flujo de Ejecución

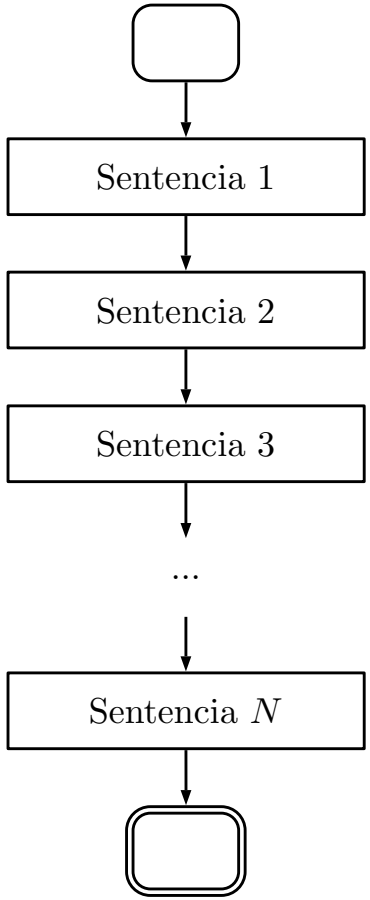
Indica el orden en que se ejecutan las sentencias en un programa

Hasta ahora las sentencias se ejecutaban en el orden en que venían definidas en el código fuente. Esto plantea ciertos problemas:

- Existen contextos en que queremos que se ejecute un conjunto de sentencias únicamente si se cumple una determinada condición
- Es tedioso ejecutar sentencias que se repiten una y otra vez

Introducción a las Sentencias de Control de Flujo:

```
int a = 4;
double b = 7.87;
char c = 'x';
...
a = a * 9;
```



Sentencia If:

Una sentencia condicional se define tal como sigue:

```
if (expresión) {
        secuencia de sentencias 1
} else {
        secuencia de sentencias 2
}
```

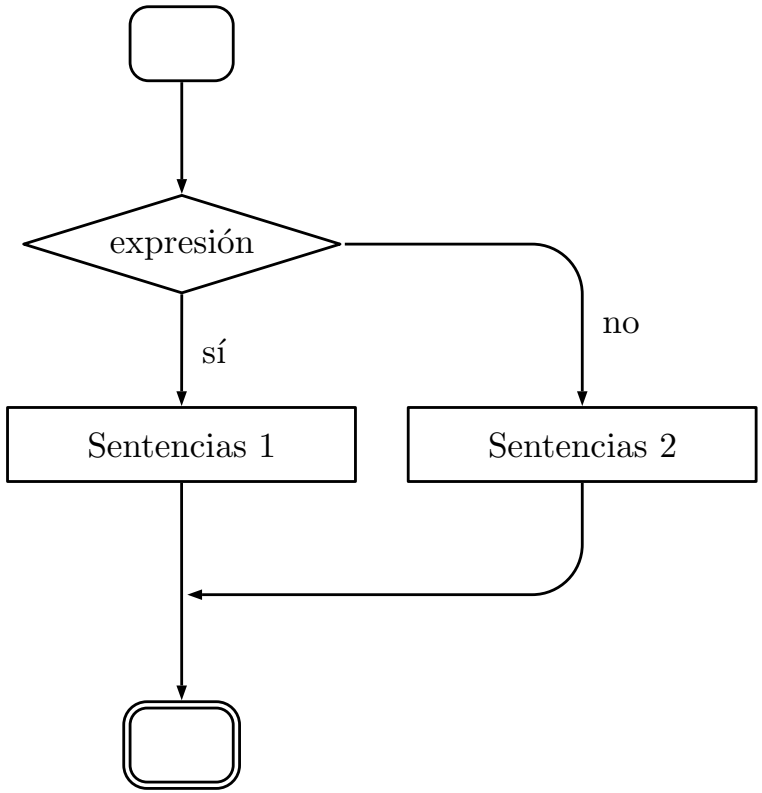
La sentencia condicional tiene el siguiente comportamiento:

- Si `expresión` es un valor distinto de 0 se ejecuta `secuencia de sentencias 1`
- Si `expresión` es 0 se ejecuta `secuencia de sentencias 2`

Sentencia If:

```
int a = ...;
int b = ...;

if (a > b) {
        printf("a es mayor que b");
} else {
        printf("a no es mayor que b");
}
```



Sentencia switch:

La sentencia **switch** permite decidir qué secuencia de código se va a ejecutar a continuación. Ésta se define tal como sigue:

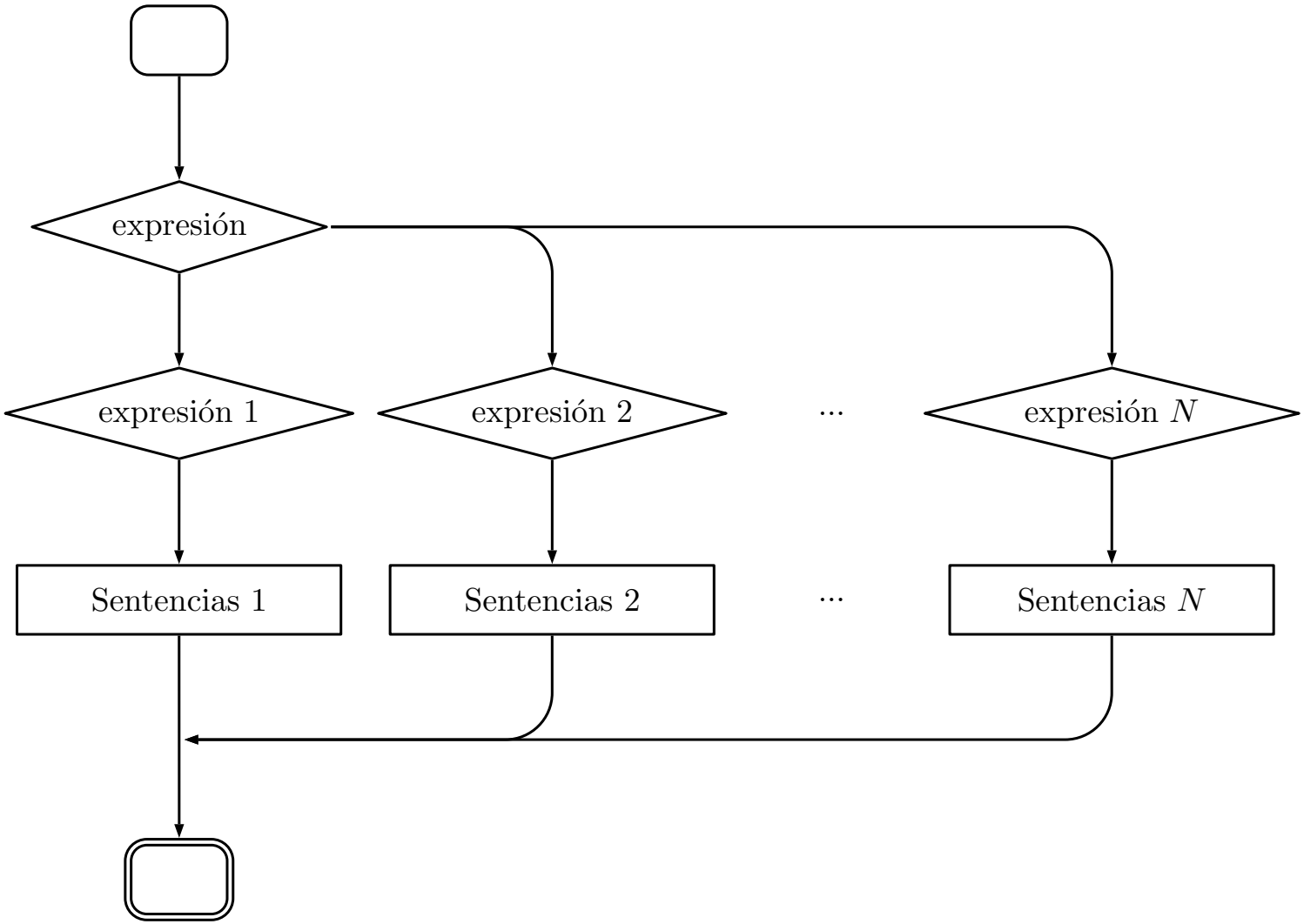
```
switch (expresión) {
      case expresión 1:
            secuencia de sentencias 1
            break;
      ...
      case expresión N:
            secuencia de sentencias N
            break;
      default:
            secuencia de sentencias por defecto
}
```

Sentencia switch:

La sentencia **switch** tiene el siguiente comportamiento:

- Primeramente se evalúa la expresión del switch. Ésta debe dar como resultado un valor entero o un carácter
- Después el programa empezará a ejecutar a partir de la sentencia **case** cuya expresión coincida con el resultado obtenido de la expresión del **switch**. Se ejecutarán todas las sentencias que haya a continuación hasta encontrar una sentencia **break** o se llegue al final de la sentencia **switch** completa. Si en el camino se encuentran más sentencias **case**, éstas serán ignoradas
- Si no se encuentra ninguna sentencia **case** cuya expresión coincida con el resultado del **switch**, pueden ocurrir dos cosas:
  - Que no exista una sentencia **default**, en cuyo caso finalizará la sentencia **switch**
  - Que exista una sentencia **default**, en cuyo caso se ejecutarán todas las sentencias que haya a continuación hasta encontrar una sentencia **break**, o se llegue al final de la sentencia **switch** completa

Sentencia switch:



Sentencia switch:

```
#include <stdio.h>

void main(void) {
        int input = ...;
        switch (input) {
                case 0:
                        printf("El valor introducido es 0");
                        break;
                case 1:
                case 20:
                        printf("El valor introducido es 1 o 20");
                        break;
                case 3:
                        printf("El valor introducido es 3");
                        break;
                default:
                        printf("El valor introducido no es 0, 1, 3 o 20");
        }
}
```

Sentencia switch:

```
enum TipoMes {enero, febrero,...}
typedef enum TipoMes Mes;

void main(void) {
        Mes mes = ...;
        int dias;
        switch (mes) {
                case abril:
                case junio:
                case septiembre:
                case noviembre:
                        dias = 30;
                        break;
                case febrero:
                        dias = 28;
                        break;
                default:
                        dias = 31;
        }
}
```

Introducción a los Bucles:

- Un bucle permite la repetición controlada de un conjunto de sentencias
- A cada repetición se le conoce con el nombre de *iteración*
- Los bucles son ampliamente empleados en todos los lenguajes de programación

Utilidades:

- Repetir algo un número determinado de veces
- Recorrer estructuras de datos (*e.g.*, arrays)
- Repetir una secuencia de sentencias mientras se cumpla una condición
- ...

Introducción a los Bucles: for

Un bucle **for** se define tal como sigue:

```
for (expresión1; expresión2; expresión3) {
      secuencia de sentencias
}
```

El funcionamiento del bucle **for** es como sigue:

- La primera vez que se ejecuta la sentencia **for**:
  - Se ejecuta **expresión1**
  - Se evalúa **expresión2** y se compara el resultado:
    - Si el resultado es verdadero se ejecuta la secuencia de sentencias y se vuelve a empezar la sentencia completa
    - Si el resultado es falso, finaliza la ejecución de la sentencia

Introducción a los Bucles: for

Un bucle **for** se define tal como sigue:

```
for (expresión1; expresión2; expresión3) {
      secuencia de sentencias
}
```

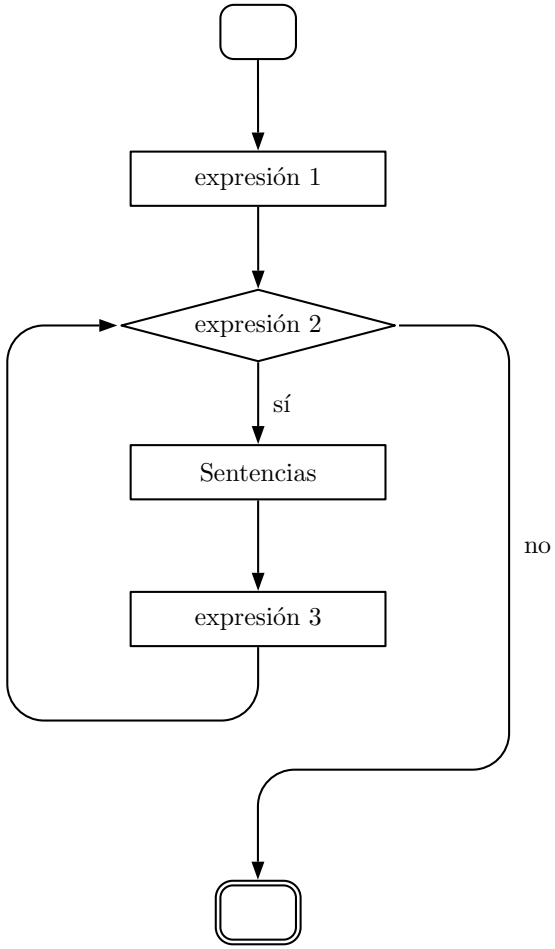
El funcionamiento del bucle **for** es como sigue:

- La segunda y sucesivas veces que se ejecuta la sentencia **for**:
  - Se ejecuta **expresión3**
  - Se evalúa **expresión2** y se compara el resultado:
    - Si el resultado es verdadero se ejecuta la secuencia de sentencias y se vuelve a empezar la sentencia completa
    - Si el resultado es falso, finaliza la ejecución de la sentencia

Introducción a los Bucles: for

```
int i;

for (i = 0; i < 10; i++) {
        printf("Iteracion %d", i);
}
```



Introducción a los Bucles: for

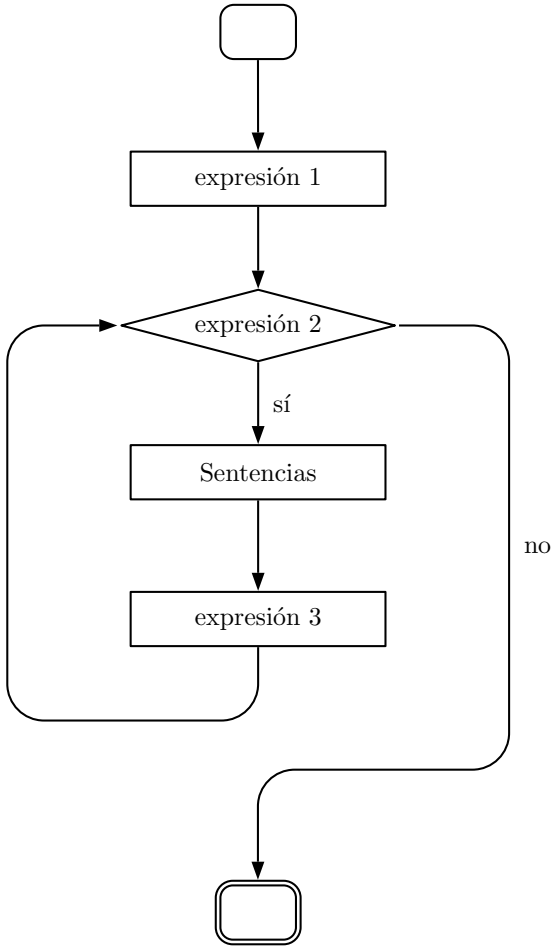
```
#define ROWS 5
#define COLUMNS 10

...

int matrix[ROWS][COLUMNS] = ...;

int i;
int j;

for (i = 0; i < ROWS; i++) {
        printf("Row %d:", i);
        for (j = 0; j < COLUMNS; j++) {
                printf("\t %d", matrix[i][j]);
        }
        printf("\n");
}
```



Introducción a los Bucles: while

Un bucle **while** se define tal como sigue:

```
while (expresión) {
      secuencia de sentencias
}
```

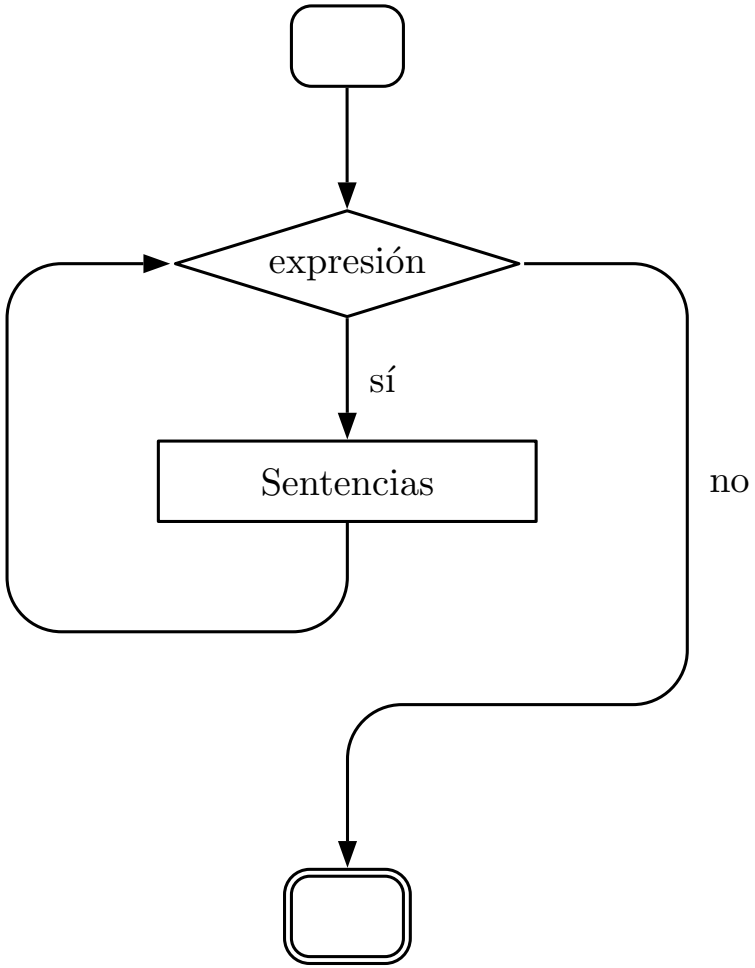
El funcionamiento del bucle while es como sigue:

- Mientras **expresión** sea un valor verdadero (valor distinto de 0), se ejecuta **secuencia de sentencias**
- En cada iteración se evalua **expresión**
- Si **expresión** devuelve un valor falso (valor igual a 0), se finaliza la ejecución del bucle

Introducción a los Bucles: while

```
int contador = 10;
int i = 0;

while (i < contador) {
       printf("Iteracion %d", i);
       i = i + 1;
}
```



Introducción a los Bucles: do-while

Un bucle **do-while** se define tal como sigue:

```
do {
        secuencia de sentencias
} while (expresion);
```

El funcionamiento del bucle **do-while** es como sigue:

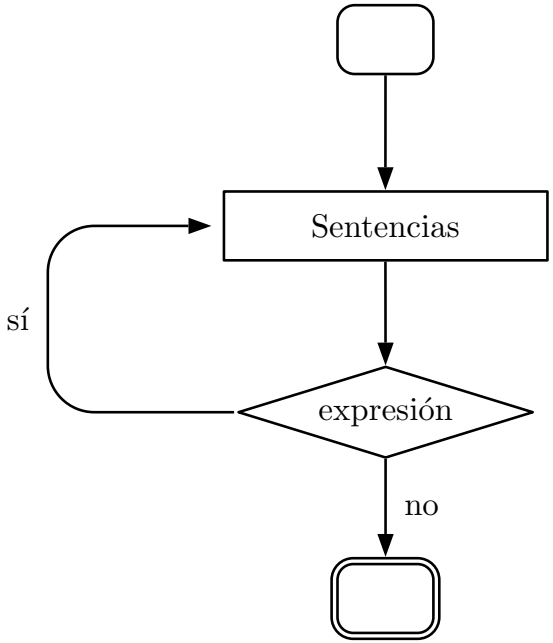
- Se ejecuta la secuencia de sentencias
- Se ejecuta **expresión** y se comprueba el resultado. Si es verdadero se repite la ejecución de la secuencia de sentencias. Si es falso, finaliza dicha ejecución

Introducción a los Bucles: do-while

```
int suma = 0;
int i = 0;

do {
        suma = suma + i;
        i = i + 1;
} while (i < 10);

printf("Suma %d: ", suma);
```



Sentencias break y continue:

La sentencia **`break`** afecta a las sentencias **`switch`**, **`while`**, **`for`** y **`do-while`** de la forma siguiente:

- Si aparece una sentencia **`break`** dentro de la secuencia de sentencias de cualquiera de las anteriores, dicha sentencia termina inmediatamente
- Si aparece en el interior de un bucle anidado sólo finaliza la sentencia de iteración más interna, el resto se ejecuta de forma normal

Sentencias break y continue:

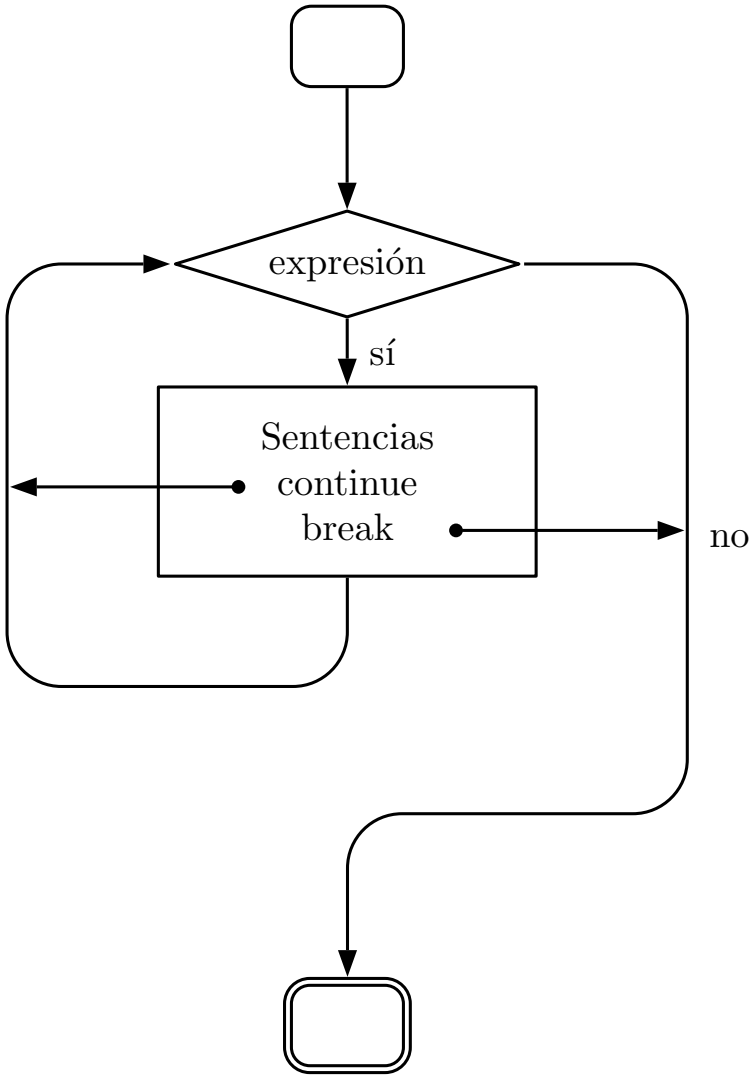
La sentencia `continue` afecta a las sentencias `while`, `for` y `do-while` de la forma siguiente:

- Si aparece una sentencia `continue` dentro de una secuencia de sentencias de las anteriores, dicha sentencia da por terminada la iteración actual y se ejecuta una nueva iteración, evaluando de nuevo la expresión condicional del bucle
- Si aparece en el interior de un bucle anidado sólo afecta a la sentencia de iteración más interna, el resto se ejecuta de forma normal

Sentencias break y continue:

```
int contador = 10;
int i = 0;

while (i < contador) {
        if (i == 5) {
                i = i + 1;
                continue;
        }
        printf("Iteracion %d", i);
        if (i == 7) {
                break;
        }
        i = i + 1;
}
```



Errores Habituales:

- **No utilizar secuencias delimitadas entre llaves**
- **Incluir un ; junto a la sentencia de control**
- **Confusiones entre asignación y comparación de igualdad**. Si se escribe `if (contador = 0)`, se está asignando `0` a `contador`, no se está haciendo una comparación sino una asignación
- **Bucles infinitos**. A veces no se controla bien la condición de control y se ejecuta un bucle infinito. Por ejemplo si se escribe:

```
int contador = 0;
int valor = 2;

while (contador < 20) {
        contador = contador * 2;
        valor = valor * 2;
}
```

Introducción a la Programación en C
–Control de Flujo–

Christopher Expósito-Izquierdo
cexposit@ull.edu.es

Airam Expósito-Márquez
aexposim@ull.edu.es

Israel López-Plata
ilopezpl@ull.edu.es

Belén Melián-Batista
mbmelian@ull.edu.es

José Marcos Moreno-Vega
jmmoreno@ull.edu.es