

Versión 10 de octubre de 2023

1 ORGANIGRAMAS

Símbolo	Descripción
→	Indica el orden de las operaciones del proceso
○	Marca el inicio o el final de un proceso
□	Representa un conjunto de instrucciones
◇	Representa una bifurcación del proceso, en función de un valor lógico

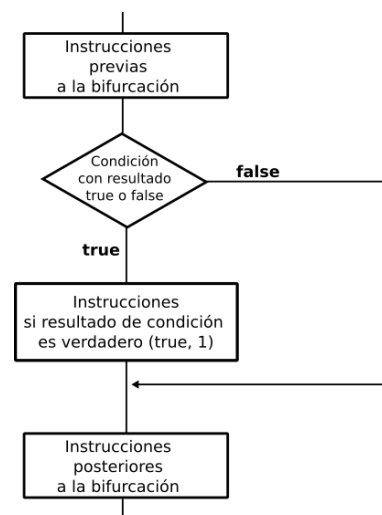


2 BIFURCACIÓN if SIMPLE (SIN RAMA else)

```
// Instrucciones antes de la bifurcación

if(condición_logica) {
    // Instrucciones si resultado de
    // condición_logica es verdadero
}

// Instrucciones tras la bifurcación
```



Ejemplo 1 Bifurcación if simple (sin rama else)

```
#include <stdio.h>

int main() {
    int num = 0;
    scanf("%d", &num);

    if(num < 0) {
        num = num * -1;
    }

    printf("num= %d\n", num);
}
```

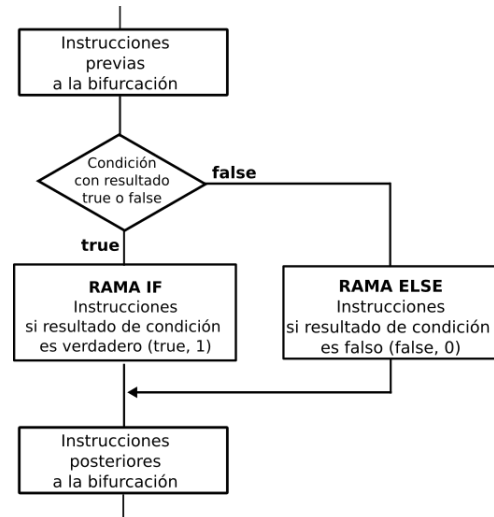
Ejemplo: El programa del Ejemplo 1 pide al usuario un número entero y lo imprime en pantalla. Si el número que teclea el usuario es negativo, lo convierte en positivo antes de imprimir.

3 BIFURCACIONES if...else

```
// Instrucciones previas a la bifurcación

if(condición_logica) {
    // Instrucciones si resultado de
    // condición_logica es (1, true)
} else {
    // Instrucciones si resultado de
    // condición_logica es (0, false)
}

// Instrucciones posteriores a la bifurcación
```



Ejemplo 2 Bifurcación if ... else

```
#include <stdio.h>

int main() {
    int num = 0;
    scanf("%d", &num);

    int resto = num % 2;

    if(resto == 0) {
        printf("PAR\n");
    } else {
        printf("IMPAR\n");
    }
}
```

Ejemplo: El programa del Ejemplo 2 pide al usuario un número entero, determina si es par o impar, e imprime el resultado en pantalla.

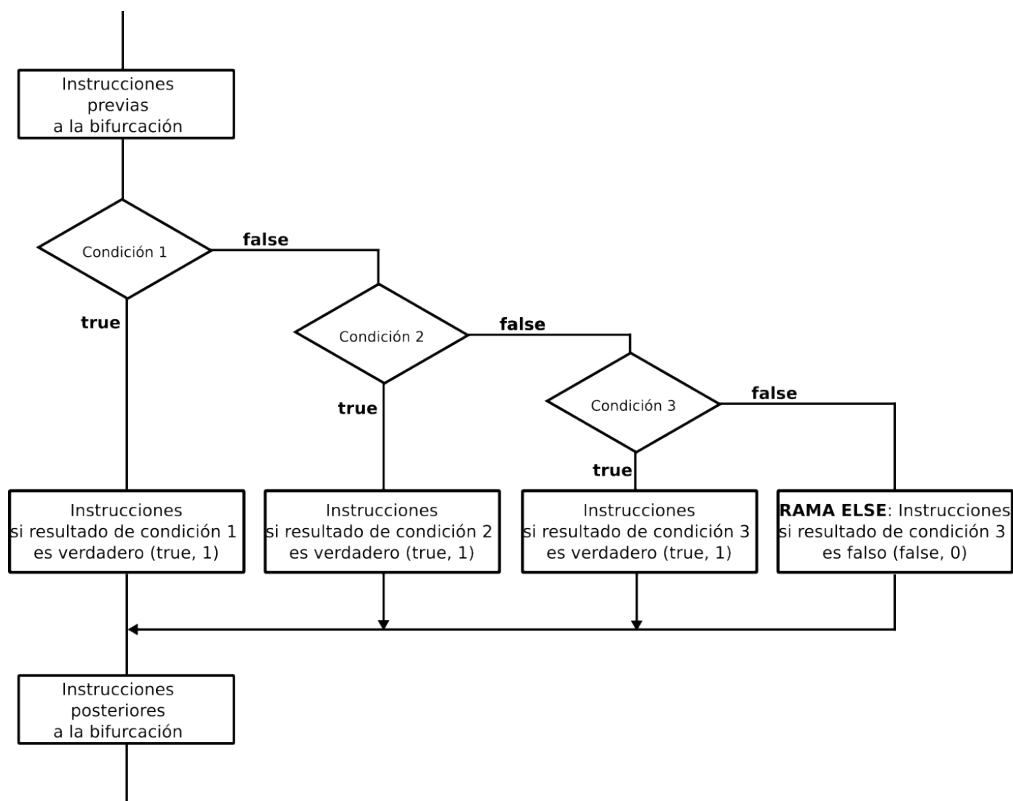
Nota: Se recuerda al lector que un número es *par* cuando el resto de la división entre 2 da 0. El resto se obtiene con el operador %.

4 BIFURCACIÓN MULTICONCONDICIONAL: if...else if... else

```
// Instrucciones previas a la bifurcación

if(condición_1) {
    // Instrucciones si condición_1 es true
} else if(condición_2) {
    // Instrucciones si condición_2 es true
} else if(condición_3) {
    // Instrucciones si condición_3 es true
} else {
    // Instrucciones si condición_3 es false
}

// Instrucciones posteriores a la bifurcación
```



Ejemplo 3 Bifurcación multicondicional if ... else if ... else

```

#include <stdio.h>

int main( void ){
    char talla;
    int medida;

    printf("Medida: ");
    scanf("%d", &medida);

    if(medida>60) {
        talla = 'L';
    } else if(medida>50) {
        talla = 'M';
    } else if(medida>40) {
        talla = 'S';
    } else {
        talla = 'X';
    }

    printf("Talla: %c\n", talla);

    return 0;
}
  
```

Ejemplo: El código del Ejemplo 3 asigna una talla de ropa en función de determinada medida. Observe que, si se cumple una de las condiciones, se ejecuta el bloque correspondiente y el programa continua en la instrucción posterior al bloque if. Solo uno de los bloques se ejecuta. Si no se cumple ninguna de las condiciones, se ejecuta el bloque de la cláusula else.

Observe también que, por ejemplo la talla M, corresponde a medidas del intervalo [51, 60]; pero como no se cumplió la condición anterior, `medida>60`, solo es necesario comprobar que la medida es mayor que 50, con lo que estará garantizado que la medida está entre 50 y 60.

5 COMPROBACIÓN DE QUE UN NÚMERO ESTÁ EN UN INTERVALO

Un caso frecuente es tener que comprobar que determinado valor x está comprendido en un intervalo (a, b) . En escritura algebraica la notación que se suele utilizar es:

$$a < x < b$$

Si se utiliza esta expresión en programación, se obtendrán resultados incorrectos. Compruebe el lector el siguiente código:

Ejemplo 4 Resolución incorrecta de pertenencia a intervalo

```
#include <stdio.h>

int main( void ){
    int a = 0;
    int b = 10;

    int x = -3;
    printf("%d \n", a<x<b); // Muestra 1, verdadero
    x = 5;
    printf("%d \n", a<x<b); // Muestra 1, verdadero
    x = 15;
    printf("%d \n", a<x<b); // Muestra 1, verdadero

    return 0;
}
```

Si ejecuta el programa anterior, obtendrá que cualquier valor de x está comprendido en el intervalo. Por ejemplo, cuando $x = -3$, la operación $a < x < b$ se ejecuta en dos pasos: primero se hace $a < x$, que devuelve un cero (*false*); a continuación se hace $0 < b$, que devuelve un uno (*true*). El lector puede comprobar el resultado de los demás casos.

En realidad, la expresión algebraica $a < x < b$ incluye dos comprobaciones: $a < x$ Y $x < b$. Ese Y es el operador lógico *AND*, $\&\&$. La expresión correcta en lenguaje C es la siguiente:

$(a < x) \&\& (x < b)$

El lector puede comprobar que el siguiente código sí que arroja los resultados esperados:

Ejemplo 5 Resolución correcta de pertenencia a intervalo

```
#include <stdio.h>

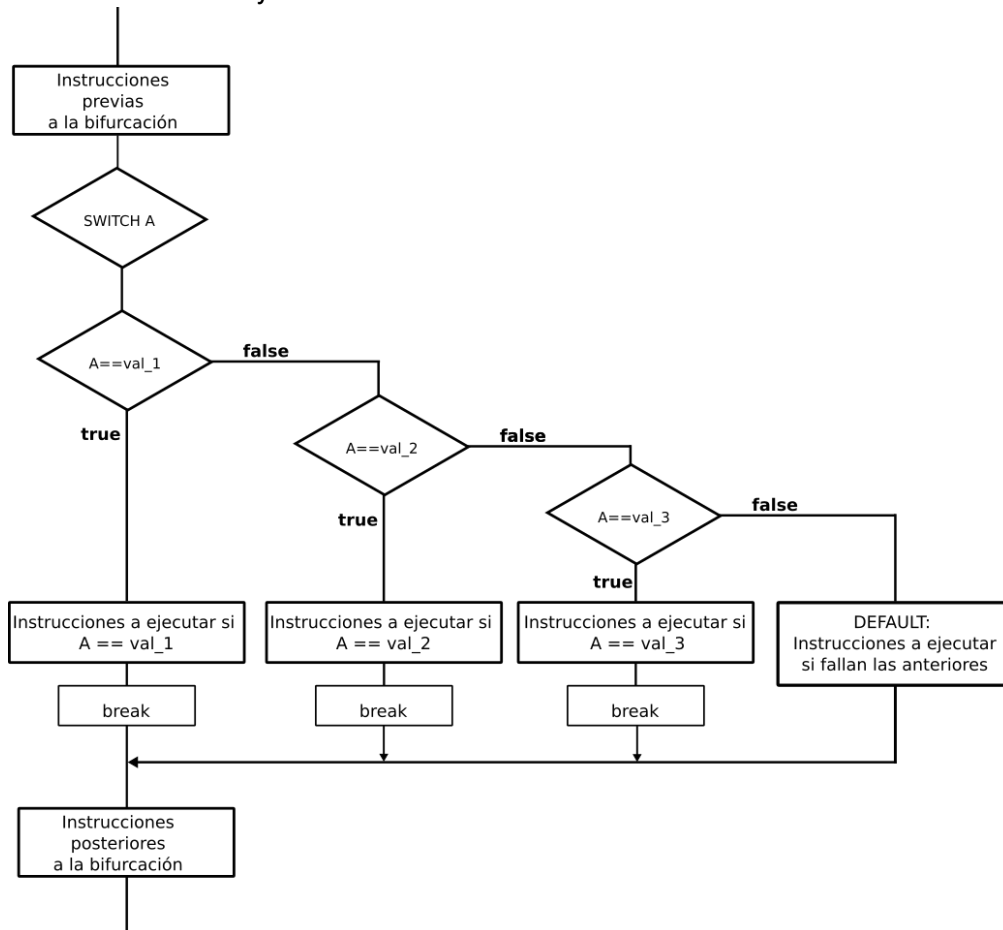
int main( void ){
    int a = 0;
    int b = 10;

    int x = -3;
    printf("%d \n", (a<x)&&(x<b)); // Muestra 0, falso
    x = 5;
    printf("%d \n", (a<x)&&(x<b)); // Muestra 1, verdadero
    x = 15;
    printf("%d \n", (a<x)&&(x<b)); // Muestra 0, falso

    return 0;
}
```

6 BIFURCACIONES switch ... case

Se trata de una construcción muy similar a la bifurcación multicondicional if ... else if ... else.



```
// Instrucciones previas a la bifurcación

switch(A) {
  case val_1:
    // Instrucciones si A == val_1
    break;
  case val_2:
    // Instrucciones si A == val_2
    break;
  case val_3:
    // Instrucciones si A == val_3
    break;
  default:
    // Instrucciones por defecto
}

// Instrucciones posteriores a la bifurcación
```

Observaciones:

- Las expresiones que van en las cláusulas case tienen que ser constantes, no se permiten operadores o variables.
- Cada bloque case debe terminar con un break, para dirigir el flujo a la instrucción posterior al bloque switch.
- La rama default es opcional.
- Esta bifurcación es muy similar a la bifurcación multicondicional if ... else if ... else.

Ejemplo 6 Ejemplo de switch ... case con cláusulas break

```
#include <stdio.h>

int main( void ){
    printf("MENÚ:\n 1.- Opción 1\n 2.- Opción 2\n 3.- Opción 3\n\nSu opción: ");

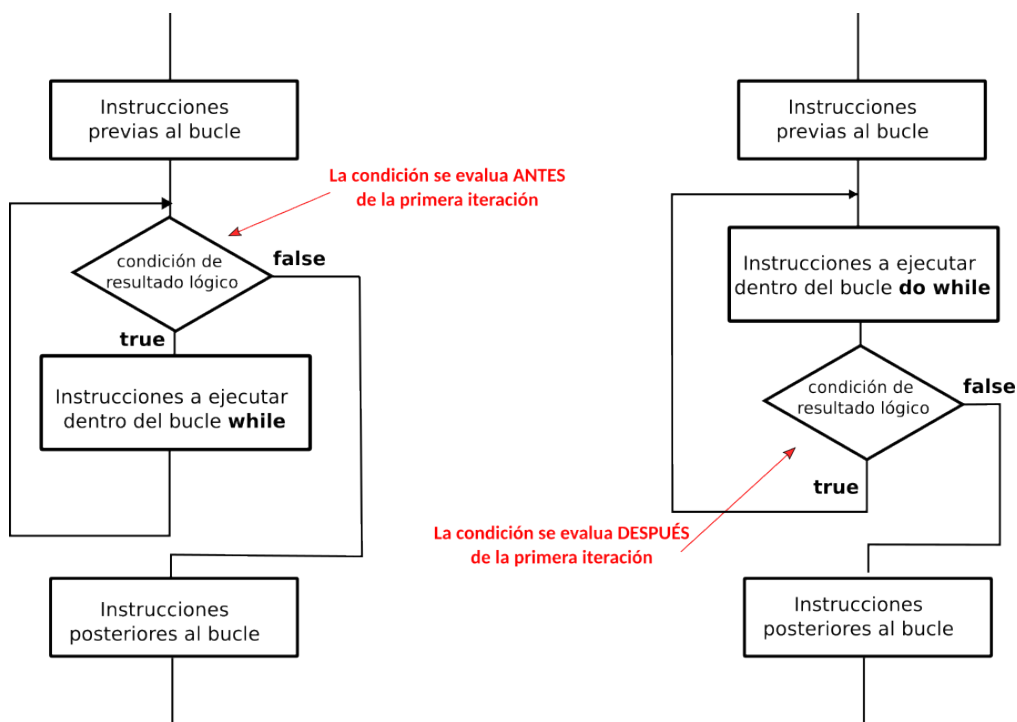
    int opcion = 0;
    scanf("%d", &opcion);

    switch(opcion) {
        case 1:
            printf("Ejecutando opción 1\n");
            break;
        case 2:
            printf("Ejecutando opción 2\n");
            break;
        case 3:
            printf("Ejecutando opción 3\n");
            break;
        default:
            printf("Fin del programa\n");
    }

    return 0;
}
```

Nota: no es obligatorio utilizar la cláusula break al final de los bloques case, pero no se va a tratar aquí el caso de no utilización de break.

7 BUCLES while Y do while



BUCLE WHILE

BUCLE DO WHILE

En ambos casos se trata de un bloque de instrucciones que se repite mientras se cumpla determinada condición lógica. La diferencia es que, en los bucles while, la condición se evalúa antes de la primera ejecución del

bloque de instrucciones, mientras que en los bucles `do while` la condición se evalúa después de haber realizado al menos una vez el contenido del bloque de instrucciones. En los bucles `while`, podría suceder que no llegue a ejecutarse ninguna vez el bloque de código.

Ejemplo 7 Ejemplo de bucles `while` y `do while`

```
#include <stdio.h>

int main( void ){
    int num = 0;

    while(num<5) {
        printf("%d ", num);
        num = num + 1;
    }
    printf("\n");

    num = 0;
    do {
        printf("%d ", num);
        num = num + 1;
    } while(num<5);
    printf("\n");

    printf("Pulse una tecla para finalizar\n");
    getchar();
    return 0;
}
```

8 BUCLES for

En los bucles `for`, se designa una variable cómo índice del bucle, con un valor inicial, un valor final y una regla de incremento de los valores.

Variable índice y valor inicial Condición del valor final Regla de incremento

for(int i=0; i<5; i++) {
 // Instrucciones del bucle
}

Ejemplo 8 Ejemplo de bucle `for`

```
#include <stdio.h>

int main(){

    for(int i=0; i<10; i=i+2) {
        printf("%d ", i);
    }
    printf("\n");

    getchar();
    return 0;
}
```

El código del ejemplo 8 muestra un bucle `for` que imprime en pantalla los números enteros que comienzan en 0 y son menores que 10, incrementando de 2 en 2.


9 SENTENCIAS break Y continue

Se utilizan para alterar la ejecución natural de los bucles.

La instrucción **break** fuerza la salida del bucle y que el flujo del programa pase a la instrucción siguiente al bloque de código del bucle.

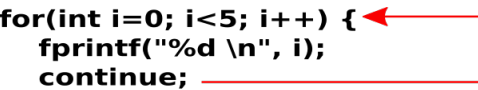
En el ejemplo de la figura siguiente, el bucle solo imprime el número 1 y la palabra FIN, pues la instrucción break fuerza la salida del bucle.

```
for(int i=0; i<5; i++) {  
    fprintf("%d \n", i);  
    break;  
}  
fprintf("FIN\n");
```



La instrucción **continue** hace que el programa vuelva a la cabecera del bucle y comience la siguiente iteración. Observe la figura siguiente, con una línea de código que no se ejecuta nunca.

```
for(int i=0; i<5; i++) {  
    fprintf("%d \n", i);  
    continue;  
    fprintf("Esta línea no se ejecuta nunca\n");  
}  
fprintf("FIN\n");
```



Observación: Las instrucciones break y continue se utilizan dentro de los bucles (for, while y do while). En los bloques if no tienen sentido (la instrucción break se utiliza también en los bloques switch).

10 BUCLES INFINITOS

Se denominan *bucles infinitos* los bucles del tipo while o do while en los que la condición es siempre verdadera. En estos casos, el bucle no terminaría nunca y, para salir, se utiliza una instrucción break que forzará la salida cuando se den las circunstancias adecuadas.

Se utilizan en situaciones en las que no se sabe de antemano cuándo será necesario salir del bucle. Un ejemplo sería para hacer la validación de la entrada de usuario. El código del ejemplo 9 pide un número entero par al usuario, y continúa pidiéndolo mientras el número tecleado no sea par:

Ejemplo 9 Ejemplo de bucle infinito para validación de entradas

```
#include <stdio.h>  
  
int main(){  
    int num_par = 0;  
    while(1) {  
        printf("Teclee un número par: ");  
        scanf("%d", &num_par);  
        if(num_par%2 == 0) {  
            break;  
        }  
    }  
    printf("Fin del programa\n");  
    getchar();  
    return 0;  
}
```


Otro caso habitual de utilización de los bucles infinitos se da cuando es el usuario el que elige una opción que fuerza la salida del bucle. El código del Ejemplo 10 muestra un menú al usuario con dos opciones. El bucle se ejecutará mientras el usuario no seleccione la opción 0, que pone fin a la ejecución del programa.

Ejemplo 10 Ejemplo de bucle infinito

```
#include <stdio.h>

int main(){
    printf("MENÚ\n1.- Opción 1\n2.- Opción 2\n0.- Salir\n\n");
    int opcion = 0;
    while(1) {
        printf("Su opción: ");
        scanf("%d", &opcion);
        if(opcion==1) {
            printf("Opción 1\n");
        } else if(opcion == 2) {
            printf("Opción 2\n");
        } else if(opcion==0) {
            break;
        }
    }
    printf("Fin del programa\n");
    getchar();
    return 0;
}
```

11 EL PROBLEMA DEL CERO EXACTO EN COMPUTACIÓN

Hay un viejo dicho en computación que dice: *nunca compares números float*. Cuando se trabaja con números double hay que tener en cuenta que se trata de números con una precisión de unos 15 decimales, pero no son números reales (pertenecientes a \mathbb{R}). Así, por ejemplo, la operación $1/3$ devolverá una aproximación al número real resultante de dividir 1 entre 3. Sucede lo mismo con todos los números irracionales como π o $\sqrt{2}$. Además, por la forma de guardar en memoria los números double, sucede que algunos números aparentemente sencillos, se guardan como una aproximación, no como su valor exacto.

Ejemplo 11 ¡Nunca compares números float!

```
#include <stdio.h>

int main(){
    double x = 0.15 + 0.15 + 0.15;
    double y = 0.10 + 0.20 + 0.15;

    printf("x = %.15lf\n", x);
    printf("y = %.15lf\n", y);

    printf("x==y -> %d\n", x==y);

    printf("1000x = %.15lf\n", 1000*x);
    printf("1000y = %.15lf\n", 1000*y);

    return 0;
}
```

```
x = 0.450000000000000
y = 0.450000000000000
x==y -> 0
1000x = 449.99999999999943
1000y = 450.00000000000057
```

El Ejemplo 11 muestra un código sencillo que permite ilustrar estos conceptos. Se declaran y asignan dos variables $x = 0.15 + 0.15 + 0.15$ e $y = 0.10 + 0.20 + 0.15$, cuyo contenido debería ser 0.45 en ambos casos. Si se imprimen las variables con 15 decimales, aparentemente guardan el valor 0.45 pero, si se comparan x e y con el operador $==$, el resultado es que no son iguales. Si se imprime el valor de $1000x$ y $1000y$ se ve que el valor que guardan las variables es sensiblemente diferente. La salida de pantalla del programa se muestra a la derecha del código.

A la vista del resultado del Ejemplo 11, al hacer comparaciones entre números `double` hay que tomar precauciones. Por ejemplo, si se quiere comprobar si determinado resultado es igual a 0, la forma correcta de operar es establecer un nivel de precisión y comprobar si el resultado, en valor absoluto, es menor que la precisión establecida. De manera análoga, para comparar dos números `double`, se puede comprobar si el valor absoluto de su diferencia es menor que la precisión que se desea obtener. Las expresiones podrían ser las siguientes:

$$|x| < precision \rightarrow x \approx 0$$
$$|x - y| < precision \rightarrow x \approx y$$

El código del Ejemplo 12 establece una precisión de 10^{-10} y con esa precisión, para las variables x e y del ejemplo anterior, comprueba que $x == y$ y $x - y == 0$.

Ejemplo 12 Método de comparación con números `double`

```
#include <stdio.h>
#include <math.h>

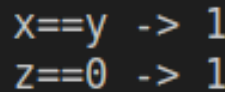
#define PRECISION 1e-10

int main(){
    double x = 0.15 + 0.15 + 0.15;
    double y = 0.10 + 0.20 + 0.15;

    printf("x==y -> %d\n", fabs(x-y)<PRECISION);

    double z = x-y;
    printf("z==0 -> %d\n", fabs(z)< PRECISION);

    return 0;
}
```



```
x==y -> 1
z==0 -> 1
```

Observe que la precisión se ha definido como una constante. Cada vez que se necesita usar en el código, se pone el nombre de la constante, no su valor. Así, si se quiere cambiar la precisión, con cambiar el valor de la constante, se actualizarán a la nueva precisión todos los sitios donde se haya usado.

Otro detalle de este código es que se ha hecho uso de la función `fabs()` (*Float ABS*) de la librería `math.h`, que devuelve el valor absoluto de un número `double`.