

Estructuras de Control Selectivas

Notación:

BloqueInstruccion significa un **bloque** o una **instrucción**, ejemplo:

Bloque: `{printf("Ingrese el valor de n: "); scanf("%d", &n);}`

Instruccion: `scanf("%d", &n);`

Una estructura de control selectiva es **una instrucción compleja** compuesta por:

Una **condición** de selección

bloqueInstruccion que se ejecuta de acuerdo al valor (verdad o falso) de la **condición**.

Hay 2 tipos de estructuras de control selectivas.

Selectiva IF

Decide, de acuerdo a una **condición**, ejecutar un **bloque** o **instrucción**.

Sintaxis:

```
if (condición)    bloqueInstruccionVerdad
[else            bloqueInstruccionFalso]    // atento: esta parte es opcional, está encerrada entre [ ]
```

Si la **condición** es verdadera se ejecuta el **bloqueInstruccionVerdad**; si es falsa se ejecuta el **bloqueInstruccionFalso**

Ejemplo: si asignamos `n = 0` y preguntamos:

```
if(n==0) printf("Atento n ES 0");           // == es el operador relacional de igualdad
else     printf("Atento n NO ES 0");
```

Salida: Atento n ES 0

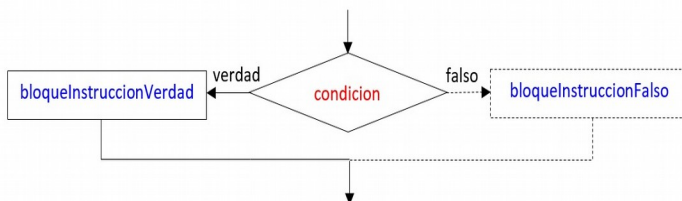
Ejemplo: si asignamos `n = 5` y preguntamos:

```
if(n!=0) {
    printf("n NO ES 0: \n");
    printf("Todo va bien\n");
} // En este caso se usó un bloque de instrucciones para el if( ) y se omitió la parte else.
```

Salida:

n NO ES 0
Todo va bien

Diagrama de flujo



Ejercicio: Escribirá el programa el cual lee un número 0 ó 1 y lo escribe en letras:

0: cero

1: uno

Programa 02_01.c:

```
// Leer un número 0 ó 1 y escribirlo en letras
#include<stdio.h>
void main(void){
    int n;
    printf("Números\n");
    printf("Escriba un número 0 ó 1: ");
    scanf("%d", &n);    // Ingrese 0
    if(n==0) printf("0: cero\n");
    else     printf("1: uno\n");
}
```

Salida:

0: cero

Atento a la condición:

La condición es una expresión lógica, que puede ser sencilla o compleja, ejemplo:

```
if(0) ...;           // 0 evalúa a falso, un número distinto de 0 evalúa a verdadero. Recuerde que no hay variable booleana.
if(expresión compleja con operadores aritméticos, relacionales y lógicos) ...;
```

Opcionalmente puede tener una asignación (=), ejemplo:

```
if(x=0) ...;           // asigna 0 a x, luego evalúa if(0) ...;
if(x= 2+1) ...         // opera 2 +1 → asigna 3 a x, luego evalúa if(3) ...;
```

Ejemplo:

```
n = 1;
if(n) printf("Verdad\n");
else printf("Falso\n");
```

Salida:

Verdad

El operador de igualdad es ==; pero si se equivoca y escribe = obtendrá algo inesperado:

```
n = 6;
if(n=5) printf("Verdad\n");    // asigna 5 a n; finalmente pregunta if(5) ...
```

Salida:

Verdad

El operador de desigualdad es !=; pero si se equivoca y lo escribe al revés =! obtendrá algo inesperado:

```
n = 6;
if(n!=5) printf("Verdad\n");    // evalúa !5 y da 0 (falso), luego asigna 0 a n; finalmente pregunta if(0) ...
```

Salida: // No escribe nada.

Operadores relacionales: ==, <, <=, >, >=

Ejemplos:

```
n !=0
n == 0 && m > 1
n == 0 || m > 1
```

Operadores lógicos: La condición es una expresión lógica, puede utilizar los operadores:

```
!    not    (no)
&&   and    (y)
||    or     (o)
```

Ejemplos:

```
n !=0
n == 0 && m > 1
n == 0 || m > 1
```

Prioridad de ejecución:

una condición **compleja** puede contener operadores:

aritméticos (+, -, *, /, %);	de relación (==, <, <=, >=);	lógicos (!, &&);	de asignación (=)
(1)	(2)	(3)	(4)

Los cuales se ejecutan en el orden indicado; ejemplo:

```
int m = 1, n = 2, k = 3, j = 5;
if(n = m == n-1 && 4 || k-j) printf("Hola\n");
```

Reemplazando valores se tiene:

```
if(n = 1 == 2-1 && 4 || 3-5) printf("Hola\n");
```

Aplicando las reglas de prioridad, la condición se ejecutó así:

$n = 1 == 2-1 \ \&\& \ 4 \ || \ 1 \rightarrow n = 1 == 1 \ \&\& \ 4 \ || \ 1 \rightarrow n = 1 \ \&\& \ 4 \ || \ 1 \rightarrow n = 1 \ || \ 1 \rightarrow n = 1$ → asigna 1 a n y evalúa a verdad

Por lo tanto al ejecutar:

```
if(n = m == n-1 && 4 || k-j) printf("Hola\n");
```

Salida:

Hola

Recuerde que también hay prioridad dentro de un mismo tipo de operadores:

Aritmético: `*` `/` `%` se ejecutan de izquierda a derecha, `+` `-` se ejecutan de izquierda a derecha

Relación : se ejecutan de izquierda a derecha

lógicos : `!`, `&&` `||`

Ejercicio: buscar en internet **prioridad de operadores**.

Un viejo truco

El siguiente if:

```
if(cond1 && cond2 && cond3 ....) bloqueInstruccionVerdad
```

Es equivalente a:

```
if(cond1)
    if(cond2)
        if(cond3)
            bloqueInstruccionVerdad
```

Esta segunda forma se usa cuando es complicado calcular `cond1`, `cond2`, por ejemplo cuando se valida varios datos:

```
if(validar1(dato1)) // si hay error en el dato 1 se reporta
if(validar2(dato2)) // si hay error en el dato 2 se reporta
if(validar3(dato3)) //.....
if(valid... ) printf("Todos los datos fueron ingresados correctamente");
```

Operador ternario:

Supongamos que tenemos el código:

```
int m = 3, n;
if (m==3) n = 1;
else     n = 2;
```

La instrucción if/else anterior es muy frecuente, por tal motivo tiene un atajo:

```
int m = 3, n;
n = (m==3) ? 1 : 2; // operador ternario, con dos operadores y tres operandos.
```

Anidamiento: una instrucción if puede contener (anidar) otros if; ejemplo:

```
if(n==0)
    if (m == 0 ) printf("n = 0 y m = 0");
    else        printf("n = 0 y m != 0");
else
    if (m == 4 ) printf("n != 0 y m = 4");
    else        printf("n != 0 y m != 4");
```

Indente las instrucciones con claridad, esto permite "visualizar" la lógica. Los anidamientos pueden llevar a confusiones. Personalmente yo prefiero:

- Alinear los bloques de modo que se "visualice" la lógica.
- No uso `{ }` si no es necesario. Atento cuando quita o aumenta instrucciones.

A continuación otras dos formas -que no me gustan- de escribir el mismo código anterior:

Forma 1:

```
if(n==0) {
    if (m == 0 ) printf("n = 0 y m = 0");
    else        printf("n = 0 y m != 0");
} else {
    if (m == 4 ) printf("n != 0 y m = 4");
    else        printf("n != 0 y m != 4");
}
```

Forma 2:

```
if(n==0)
{
    if (m == 0 )
    {
        printf("n = 0 y m = 0");
    }
    else
    {

```

```

        printf("n = 0 y m != 0");
    }
}
else
{
    if(m == 4) printf("n != 0 y m = 4");
    else      printf("n != 0 y m != 4");
}

```

// ya me cansé de usar { }

El programador debe utilizar la forma más clara, tanto para él, como para las otras personas que lo revisan.

Ejemplo: Lea 4 números i, j, k y m; encuentre el mínimo. Hay varias formas de hacerlo, acá van dos sabores:

Programa 02_02.c:

Inicio	<pre> #include<stdio.h> void main(void){ int i, j, k, m, min; printf("Ingrese 4 números enteros: "); scanf("%d %d %d %d", &i, &j, &k, &m); // ingrese: 8, 5, 1, 7 </pre>	
	<pre> if(i < j) min = i; else min = j; </pre>	<pre> min = i; </pre>
Proceso	<pre> if(min > k) min = k; if(min > m) min = m; </pre>	<pre> if(min > j) min = j; if(min > k) min = k; if(min > m) min = m; </pre>
Fin	<pre> printf("El mínimo de: %d %d %d y %d es: %d\n", i, j, k, m, min); } </pre>	

Salida en ambos sabores:

Ingrese 4 números enteros: **8, 5, 1, 7**

El mínimo de: **8, 5, 1, 7** es: **1**

Observe los 3 pasos en que se dividen frecuentemente los procesos.

Ejercicio: Escriba el programa **numeros.c**; el cual lee un número entre 0 y 5 y lo escribe en letras:

0: cero

1: uno

.....

Este ejercicio lo podemos programar utilizando la sentencia if ... else en modo anidado; pero sería muy engorroso; Será más fácil utilizar otra estructura de decisión.

Selectiva switch

Decide las instrucciones a ejecutar según el **valor** de una **variable**:

```

switch(variable) {
    case valor1: bloqueInstruccion1 // si variable = valor1, se ejecuta bloque1
                                     // continúa al siguiente case
    case valor2: bloqueInstruccion2 // si variable = valor2, se ejecuta bloque2
                                     // continúa al siguiente case
    ...
    [default: bloqueInstruccionDefault] // se ejecuta bloqueInstruccionDefault
}

```

Ejercicio: Escribirá el programa **numeros.c**; el cual lee un número entre 0 y 3 y lo escribe en letras:

0: cero

1: uno

.....

Intentemos: **Programa 02_03a.c:**

```

// Uso de switch: Escribir número en letras
#include<stdio.h>
void main(void){
    int n = 2;
    switch(n) {
        case 1: printf("uno\n");
        case 2: printf("dos\n");
    }
}

```

```

        case 3: printf("tres\n");
        default: printf("fuera de valor\n");
    }
}

```

Salida:

dos
fuera de valor // Esta salida cumple las reglas de la sintaxis; pero no resuelve el problema planteado; hagamos un cambio:

Programa 02_03b.c:

```

// Uso de switch: Escribir número en letras
#include<stdio.h>
void main(void){
    int n = 2;
    switch(n) {
        case 1: printf("uno\n"); break;           // break: Salta al fin del switch
        case 2: printf("dos\n"); break;           // break: Salta al fin del switch
        case 3: printf("tres\n"); break;           // break: Salta al fin del switch
        default: printf("Fuera de valor\n");
    }
}

```

Salida:

dos

Es el resultado que necesitamos; si hiciéramos **n = 4**; la salida sería:

Salida:

Fuera de valor.

Los **break** obligan a que se ejecute solo el **bloqueInstruccion** correspondiente al case; si no se cumple ningún case, se ejecuta el **bloqueInstruccionDefault**.

Anidamiento: Ahora podemos hacer varios tipos de anidamiento de instrucciones de decisión, por ejemplo:

```

if(n >= 0) {
    switch(n) {...}
} else if(...) {...}
else ....

```

Sea cuidadoso al usar {}, tanto para agrupar instrucciones, como para aclarar la lógica.

Ejercicio: Escriba un programa que lea un número n entero y haga lo siguiente:

si n está entre 0 y 5, escriba su valor en letras.

En caso contrario:

si $n < 0$, que escriba: n debe ser ≥ 0

en caso contrario, que escriba: n es muy grande.

Ejemplo: Este ejercicio ya fue presentado en el capítulo anterior, ahora lo modificaremos para presentar más opciones.

Escriba un programa que lea dos enteros m n mayores que 0, luego presente el menú:

Calculadora:

1) Sumar: $m + n$

2) Restar: $m - n$

3) Multiplicar: $m * n$

4) Dividir: m/n

Elija la operación: _

Ejecute la opción seleccionada.

Programa 02_04.c:

```

// Leer dos enteros m n mayores que 0, presentar un menú y ejecutar la operación seleccionada
#include<stdio.h>
void main(void){
    int m, n, op;
    printf("Ingrese un entero m = "); scanf("%d",&m);
    printf("Ingrese un entero n = "); scanf("%d",&n);
    printf("\nOperación que requiere:\n");
    printf("1) Sumar: m + n\n");
}

```

```

printf("2) Restar: m - n\n");
printf("3) Multiplicar: m * n\n");
printf("4) Dividir: m/n\n");
printf(" Elija su opción: ");      scanf("%d",&op);
switch(op){                        // Selección de opción
    case 1: printf("suma = %d\n", m+n);      break;
    case 2: printf("Resta = %d\n", m-n);      break;
    case 3: printf("Multiplicación = %d\n", m*n); break;
    case 4: if(n==0) printf("Denominador 0\n");
              else printf("División = %.2f\n", (float)m/n);
              break;
    default: printf("Opción errada\n");
}
}

```

Salida:

```

Ingrese un entero m = 3
Ingrese un entero n = 2

Operación que requiere:
1) Sumar: m + n
2) Restar: m - n
3) Multiplicar: m * n
4) Dividir: m/n
 Elija su opción: 1
suma = 5

```

Atento:

- Usar las indentaciones correctas “visualiza” la lógica
- Fíjese que se ha violado la recomendación de escribir una instrucción por línea, por ejemplo:
 case 2: printf("Resta = %d\n", m-n); break;
 Esto se hizo para visualizar mejor la lógica.

Matriz de prueba:

Caso	Entradas			Salida	Chequeo
	m	n	opción		
1	3	2	1	5	✓
2	3	2	2	1	✓
3	3	2	3	6	✓
4	3	2	4	1.50	✓
5	2	0	4	Denominador 0	✓
6			7	Opción errada	✓

Ejercicios: Escriba, compile y ejecute programas que hagan lo siguiente:

- 1) Pida ingresar un ángulo θ y calcule y muestre $\sin(\theta)$, $\cos(\theta)$ y $\tan(\theta)$.
- 2) Pida tres números y muestre el menor y mayor de ellos.
- 3) Lea un entero n entre 1 y 9 y escriba su equivalente en números romanos.
- 4) Lea un entero n y escriba si es par o impar.
- 5) Un estudiante ha dado 3 prácticas y obtuvo 16, 17 y 13, en base 20. La cuarta práctica debe cumplir: 1) Ser la máxima posible y 2) El promedio de las 4 prácticas debe ser entero.
- 6) Lea dos enteros j , k y calcule m de dos modos:
 - a) if($j < k$) $m = a$;
 else $m = b$;
 - b) $m = (j < k) ? a : b$;
 ¿Hay alguna diferencia?