



# Tema 2

- Identificación de los elementos de un programa
- Estructuras de control



# IDENTIFICADORES

- Son palabras creadas por el programador para referenciar los datos y otros elementos del programa
  - Pueden estar constituidos por letras, dígitos, el carácter subrayado Y el carácter \$
  - No deben comenzar por un dígito.
  - No deben contener espacios
  - El nombre asignado debe tener relación con la información que contiene.
  - No se pueden utilizar palabras reservadas como identificadores.



# LOS DATOS

- Son los objetos sobre los que opera un ordenador
- Llevan asociados tres atributos:
  - **nombre**
  - **tipo**
  - **valor.**



# TIPOS DE DATOS: Según almacenamiento

- Datos simples
  - Numéricos
    - Enteros (byte, short, int, long)
    - Reales (float, double)
  - No numéricos
    - Carácter
    - Lógico
- Datos Estructurados
  - Internos
    - Estáticos
    - Dinámicos
  - Externos
    - Ficheros



# **TIPOS DE DATOS: Según la permanencia en memoria**

- **CONSTANTES**
- **VARIABLES**



# OPERADORES

- Son símbolos que sirven para conectar los datos haciendo diversas clases de operaciones dependiendo del lenguaje de programación que se utilice.

# TIPOS DE OPERADORES

- Paréntesis
- Aritméticos (+, -, \*, /, %)
- Relacionales (<, >, >=, <=, !=, ==)
- Lógicos (&&, ||, !)
- Unitarios (++ , -- )
- de asignación (=, \*=, /=, %=, +=, -=)

Ej.-  $x \% = 3 \iff x = x \% 3$

# Unitarios

---

- Ejemplo:

```
int x = 0, n = 10, i = 1;
```

```
n++; //Incrementa el valor de n en 1 “n vale 11”
```

```
++n; //Incrementa el valor de n en 1 “n vale 12”
```

```
x = ++n; /*Primero incrementa n en 1 y después asigna el  
resultado a x, en este momento “n vale 13” y “x vale  
13”.*/
```

```
x = n++; /*Equivale a realizar las dos operaciones siguientes  
x = n; n++;. Por lo tanto “x vale 13” y “n vale 14”. Es  
decir, primero asigna el valor de n a x y después  
incrementa el valor de n.*/
```





# CONTADORES

- variable numérica cuyo valor se incrementa o decrementa en cantidades fijas.
- deben inicializarse con un valor
- La función mas usual de un contador es controlar el número de iteraciones que se van a realizar en un bucle y determinar como salir de él



# ACUMULADORES

- es una variable cuyo valor se incrementa o decrementa con cantidades variables.
- Es importante tener en cuenta dos reglas:
  - Aquellos casos en los que se pretende obtener el total como suma de distintas cantidades es necesario inicializarlo a 0.
  - En aquellos casos en los que se pretende obtener el total como producto de distintas cantidades será necesario inicializarlo a 1.



# SWITCH O INTERRUPTORES

- variables que solo pueden tomar dos valores diferentes durante el desarrollo de un programa, sus dos posibles valores los elige el programador y pueden ser un 0 o 1 un si o no, 1 o -1
- deben ser inicializados con cualquiera de los dos únicos valores que podrán tomar durante la ejecución del programa con antelación a cualquier uso que se haga de ellos.

# Programación Estructurada

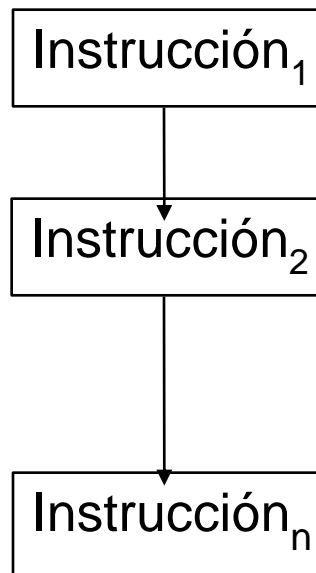
---

- La Programación Estructurada es un conjunto de técnicas de programación que están basadas en la utilización de tres tipos de sentencias
  - Secuenciales
  - Condicionales
  - Iterativas (bucles)
- Con la programación estructurada realizaremos programas que cumplan con las siguientes características:
  - Tienen un solo punto de entrada y uno de salida
  - Existen caminos de la entrada a la salida que se pueden seguir y pasan por todas las partes del programa.
  - No existen bucles infinitos
- La programación estructurada prohíbe el uso de la sentencia incondicional GOTO.

# Estructura Secuencial

---

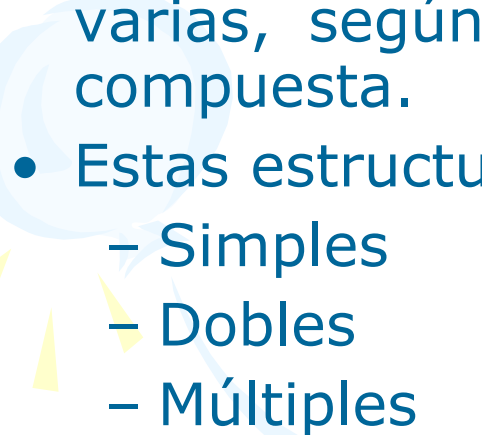

- Es una estructura con una entrada y una salida, donde figuran una serie de acciones que se suceden de tal forma, que la salida de una es la entrada de la siguiente y así sucesivamente.





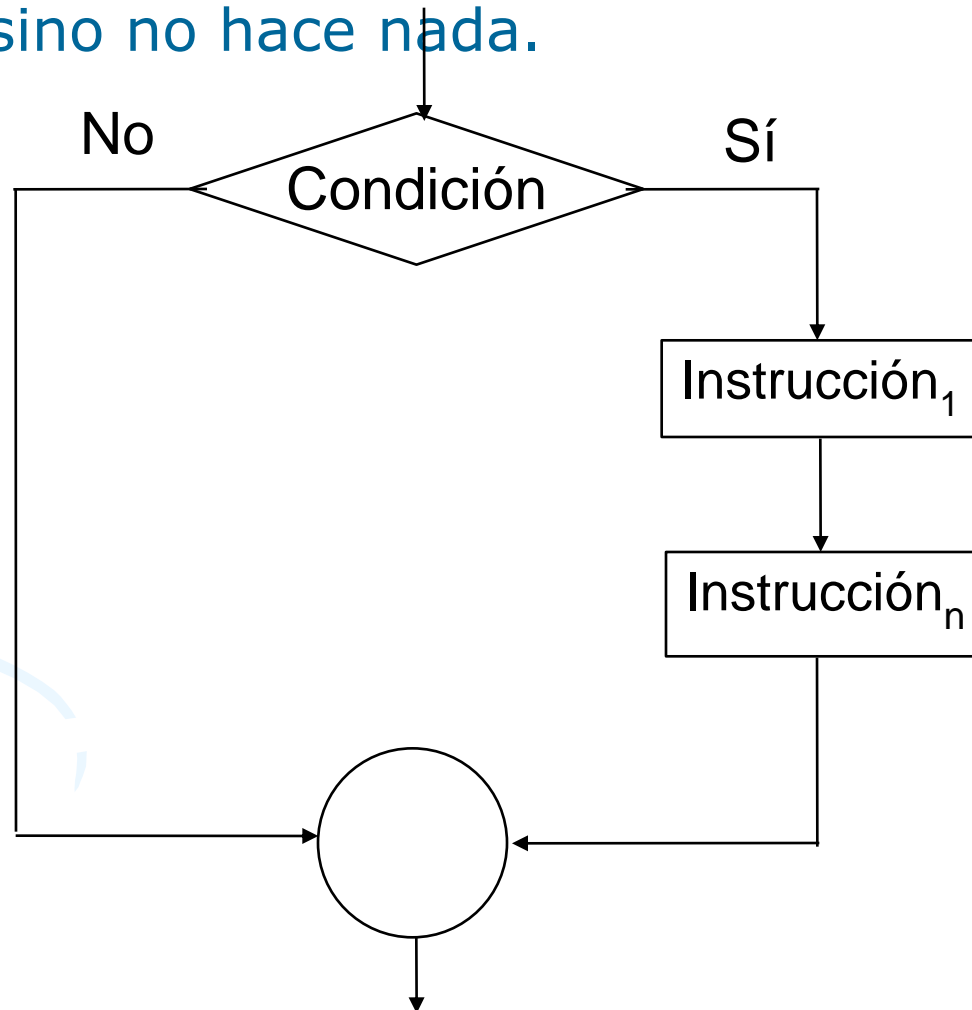
# Estructura Alternativa

---

- Es una estructura de una sola entrada y una sola salida en donde se realiza una acción o acciones de entre varias, según una condición que puede ser simple o compuesta.
  - Estas estructuras pueden ser:
    - Simples
    - Dobles
    - Múltiples
- 
- 

# Alternativa Simple

- Si la condición es verdadera se ejecuta la acción o acciones, sino no hace nada.





# Alternativa Simple

---

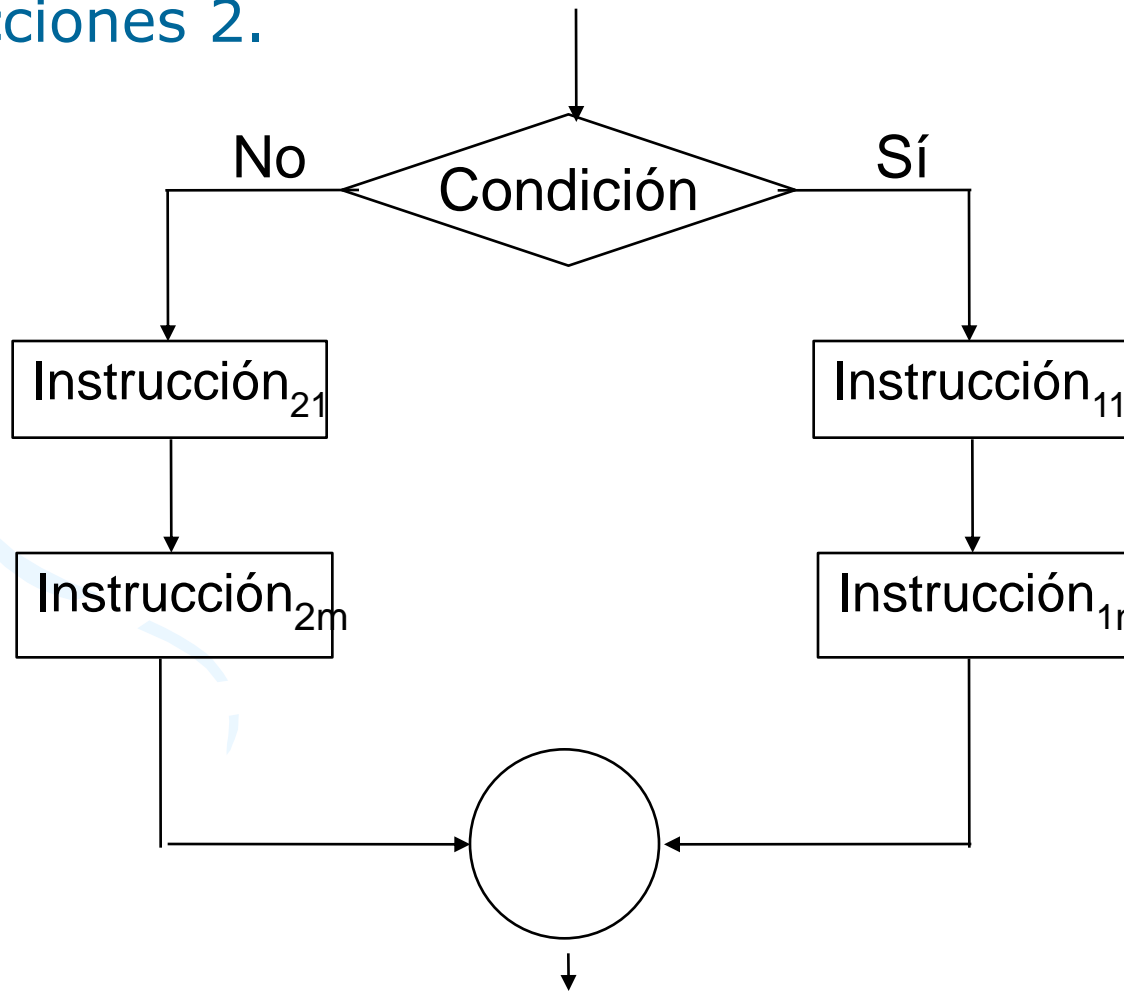
- Que traducido a un lenguaje de programación como C y Java equivale a:

```
if (condicion)
{
    instrucción1;
    instrucción2;
    ...
    instrucciónn;
}
```



# Alternativa Doble

- Se ejecuta la acción o conjunto de acciones 1 si se cumple la condición, sino ejecuta la acción o conjunto de acciones 2.





# Alternativa Doble

---

- Que traducido a un lenguaje de programación

if (condicion)

{

instrucción<sub>11</sub>;

instrucción<sub>12</sub>;

instrucción<sub>1n</sub>;

}

else

{

instrucción<sub>21</sub>;

instrucción<sub>22</sub>;

instrucción<sub>2m</sub>;

...

}



# Alternativa Doble

---

- Ejemplo:

```
int num1, num2, resultado;
```

```
num1 = 10;
```

```
num2 = 5;
```

```
resultado = 0;
```

```
if (num2>0)
```

```
{
```

```
    resultado= num1/num2;
```

```
}
```

```
else
```

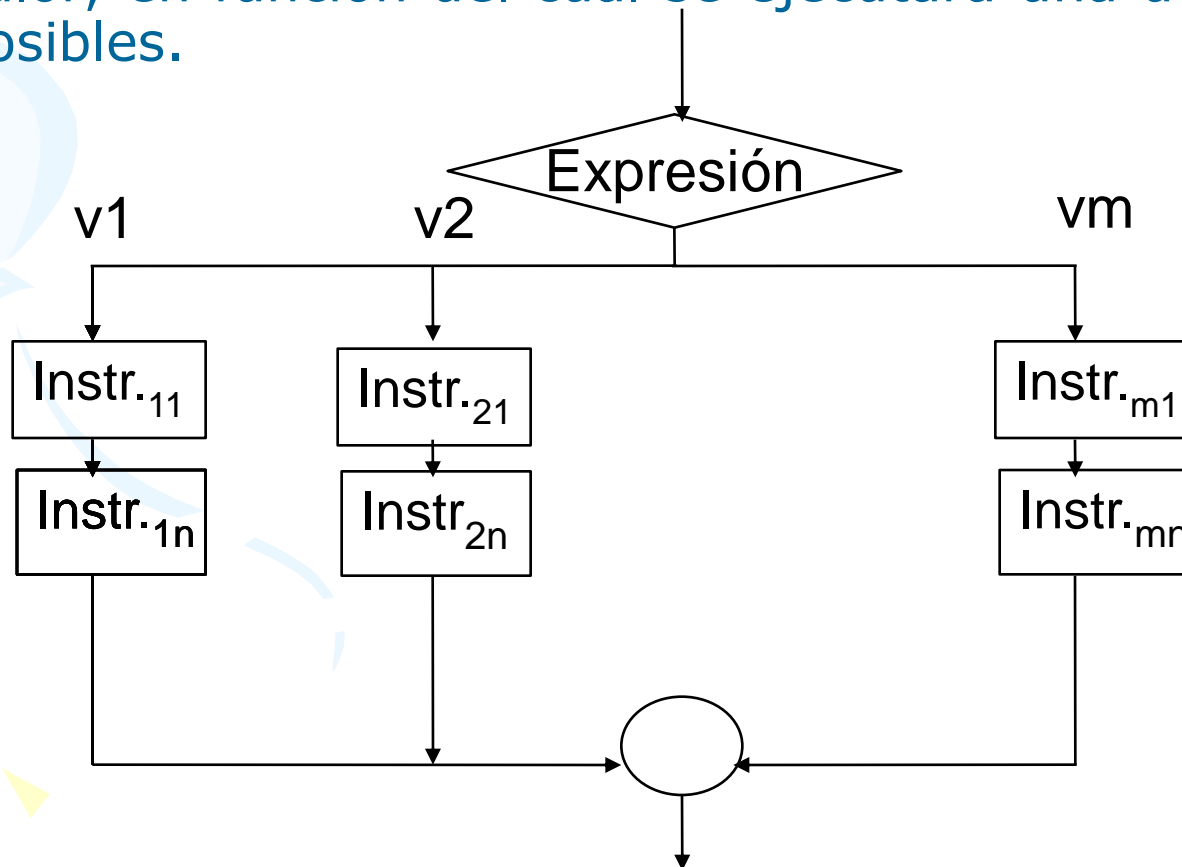
```
{
```

```
    resultado = num1;
```

```
}
```

# Alternativa Múltiple

- En esta situación la condición no toma un único valor de verdadero o falso. Sino que suele ser una variable que se evaluará a cierto o falso comparándola con un determinado valor, en función del cual se ejecutará una de las  $n$  acciones posibles.





# Alternativa Múltiple

---


- Que traducido a un lenguaje de programación

```
switch ( expresión )  
{  
  case expresion_constante1: instrucción11;  
    ... instrucción1m;  
    break;  
  case expresion_constante2: instrucción21;  
    ... instrucción2m;  
    break;  
  default: instrucciónn1;  
    instrucciónnm;  
    break;  
  ...  
}
```

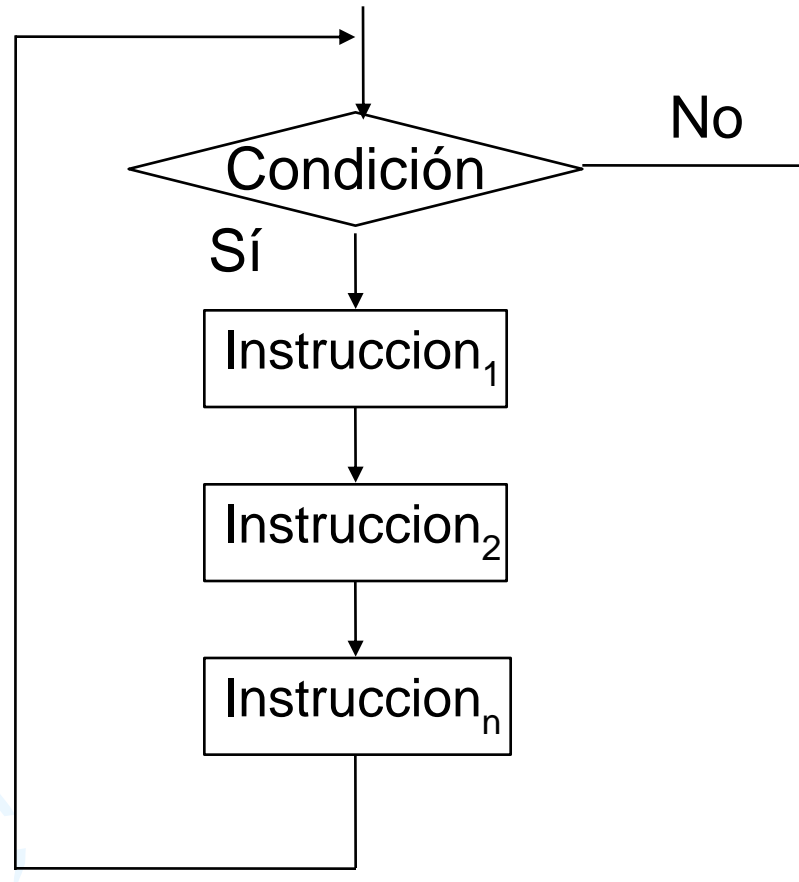


# Estructura Repetitiva

---

- Es una estructura de una sola entrada y una sola salida, donde se repite una acción un número de veces que dependerá de una condición.
  - A estas estructuras se las conoce como **bucles**, y a la acción o acciones que repiten **iteración**.
  - En la ejecución de los bucles se utiliza una condición de parada, que puede estar situada en el interior del bucle, aunque normalmente suele ponerse al principio o al final del mismo.
  - Existen 3 tipos de instrucciones repetitivas:
    - Mientras (While)
    - Repetir mientras (do while)
    - Para (For)
- 

# Bucle Mientras (While)



# Bucle Mientras (While)

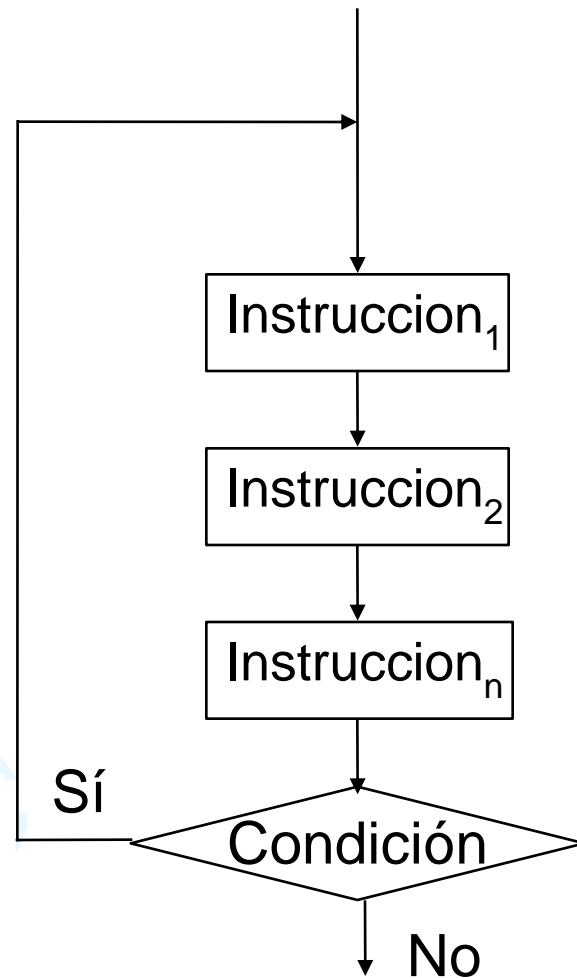
---

- En este bucle primero se evalúa la condición y si se cumple se repite el conjunto de instrucciones hasta que la condición se evalúe como false.
- Luego si al principio la condición no se cumple las acciones del interior del bucle **NO** se ejecutarán nunca.
- Traducido a un lenguaje de programación imperativo:

```
while ( condicion )  
{  
    instrucción1;  
    instrucción2;  
    instrucciónn;  
}
```



# Bucle Repetir Mientras (do while)



# Bucle Repetir Mientras (do while)

---

- Ejemplo:

```
int cont, num;  
cont = 0;  
num = 2;
```

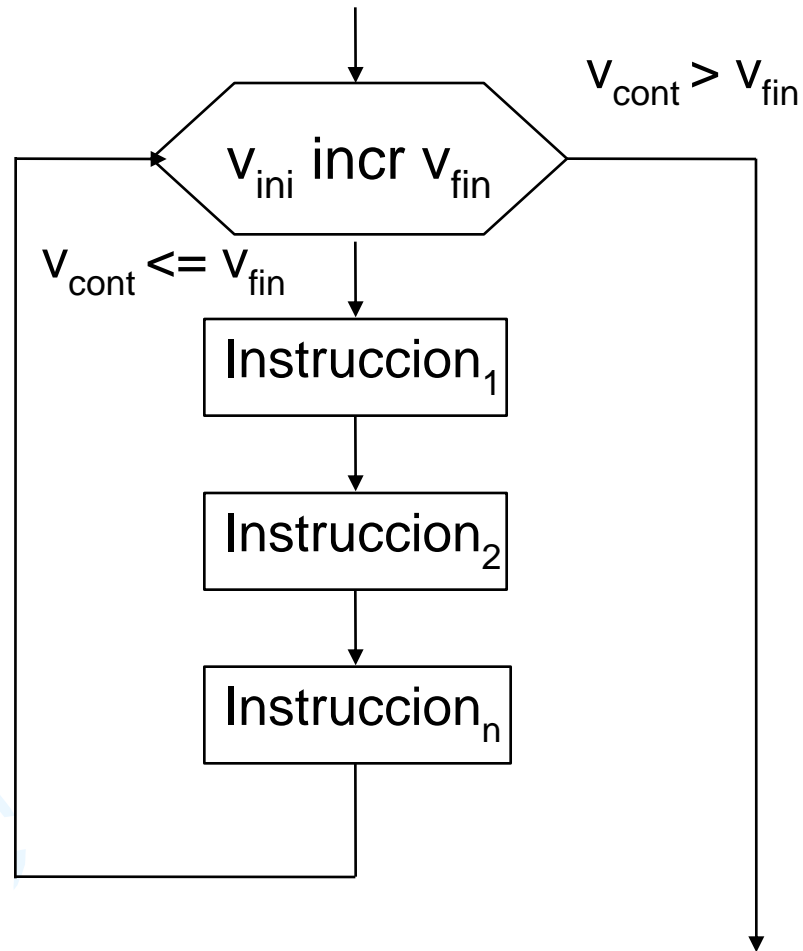
```
do  
{
```

```
    num = num + num;  
    cont = cont + 1;
```

```
}
```

```
while( cont<5 ) ;
```

# Bucle Para (For)



# Bucle Para (For)

- En el caso del bucle for siempre se conoce a priori el número de veces que se ejecutará una instrucción.
- En este caso se utilizará un contador ( $v_{\text{cont}}$ ), que se inicializará a un valor ( $v_{\text{ini}}$ ) y se incrementará en una cantidad fija ( $\text{incr}$ ). La repetecición se lleva a cabo hasta que el contador consiga un valor determinado ( $v_{\text{fin}}$ ).
- Se evaluará en cada iteración si el contador ha conseguido alcanzar el valor final.
- Traducido a un lenguaje de programación imperativo:

```
for ( $v_{\text{ini}}$ ; condicion; incr)
{
    instruccion1;
    instruccionn
    ...
}
```



# Bucle Para (For)

---

- Ejemplo:

```
int cont, num;
```

```
num = 2;
```

```
for(cont = 0; cont < 5; cont++)
```

```
{
```

```
    num = num + num;
```

```
}
```



# Ejemplos de estructuras básicas

- La estructura if adopta una de las dos formas siguientes:

- if (expresión) sentencia;

- o bien

- if (expresión) sentencia;

- else sentencia;

- Ejemplo 1:

- o if (x > 0) escribe ("Positivo");

- o if (x) escribe ("Verdadero");

- else escribe ("Falso");

- o if (c >= 'a' && c <= 'z')

- {

- escribe ("La variable c almacena un carácter alfabético");

- escribe ("El carácter es una letra minúscula");

- }

# Ejemplos de estructuras básicas

- Inspecciona una variable y la va comparando con una lista de constantes. Cuando encuentra una coincidencia, se ejecuta la sentencia o grupo de sentencias asociado. Su estructura es:

```
switch (variable) {  
    case cte1: sentencia;  
        break;  
    case cte2: sentencia;  
        break;  
    ...  
    default:  sentencia;  
}
```

Donde variable es una variable o cualquier expresión que devuelva un valor.

Se compara la variable con cte1, cte2, ..., y si encuentra una coincidencia, ejecuta la sentencia correspondiente. Por sentencia entendemos tanto una sentencia simple como un grupo de sentencias, que no se agrupan entre llaves. Si no se encuentra ninguna coincidencia se ejecuta la sección default (que no es obligatoria).

# Ejemplos de estructuras básicas

---

- ❑ Ejemplo 8: Muestra en pantalla los números del 1 al 10 y sus cuadrados.

```
register int i;
```

```
...
```

```
for ( i = 1; i <= 10; i++)
```

```
{
```

```
    printf ("\nValor de i: %d", i);
```

```
    printf ("\nValor de i²: %d", i * i);
```

```
}
```

- ❑ Ejemplo 9: Muestra en pantalla las letras mayúsculas de la A a la Z.

```
char letra;
```

```
...
```

```
for (letra = 'A'; letra <= 'Z'; letra++) printf("\n%c", letra);
```

- ❑ **Puede ponerse un incremento/decremento diferente de 1.**