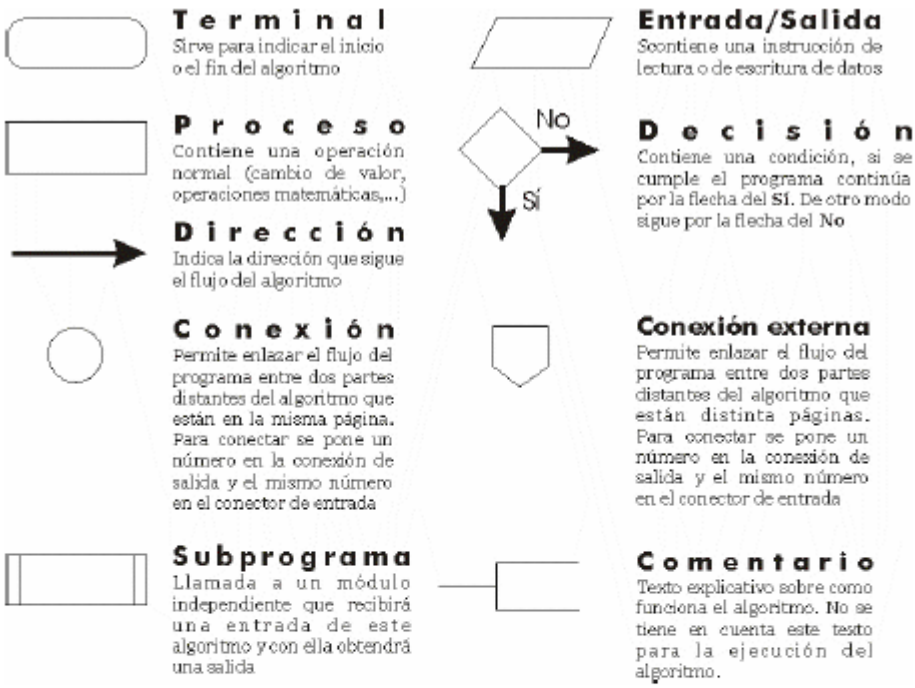
**Diagrama de Flujo:**

Bibilografía: Fundamentos de Programación 2006-2007 – Metodología de la programación. Jorge Sanchez.




Ejemplo:

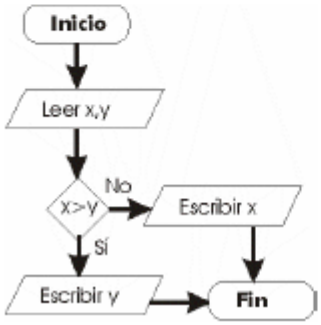



**Ilustración 5, Diagrama de flujo que escribe el mayor de dos números leídos**

## TIPOS DE DATOS

- **entero.** Permite almacenar valores enteros (sin decimales).
- **real.** Permite almacenar valores decimales.
- **carácter.** Almacenan un carácter alfanumérico.
- **lógico** (o **booleano**). Sólo permiten almacenar los valores **verdadero** o **falso.**
- **texto.** A veces indicando su tamaño (texto(20) indicaría un texto de hasta 20 caracteres) permite almacenar texto. Normalmente en cualquier lenguaje de programación se considera un tipo compuesto.

## INSTRUCCIONES PRIMITIVAS

- Asignaciones (←)
- Operaciones (+, -, * /,...)
- Identificadores (nombres de variables o constantes)
- Valores (números o texto encerrado entre comillas)
- Llamadas a subprogramas

En el pseudocódigo se escriben entre el inicio y el fin. En los diagramas de flujo y tablas de decisión se escriben dentro de un rectángulo

## ASIGNACION

```
identificador←valor
```

Los valores pueden ser:

- **Números.** Se escriben tal cual, el separador decimal suele ser el punto (aunque hay quien utiliza la coma).
- **Caracteres simples.** Los caracteres simples (un solo carácter) se escriben entre comillas simples: 'a', 'c',etc.
- **Textos.** Se escriben entre comillas doble "Hola"
- **Lógicos.** Sólo pueden valer **verdadero** o **falso** (se escriben tal cual)
- **Identificadores.** En cuyo caso se almacena el valor de la variable con dicho identificador. Ejemplo:

## OPERADORES ARITMETICOS Y PRIORIDADES

| | |
|---|---|
| + | Suma |
| - | Resta o cambio de signo |
| * | Producto |
| / | División |
| **mod** | Resto. Por ejemplo 9 mod 2 da como resultado 1 |
| **div** | División entera. 9 div 2 da como resultado 4 (y no 4,5) |
| ↑ | Exponente. 9↑2 es 9 elevado a la 2 |

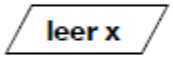
Hay que tener en cuenta la prioridad del operador. Por ejemplo la multiplicación y la división tienen más prioridad que la suma o la resta. Sí 9+6/3 da como resultado 5 y no 11. Para modificar la prioridad de la instrucción se utilizan paréntesis. Por ejemplo 9+(6/3)

## ENTRADA / SALIDA

Es la instrucción que simula una lectura de datos desde el teclado. Se hace mediante la orden leer en la que entre paréntesis se indica el identificador de la variable que almacenará lo que se lea. Ejemplo (pseudocódigo):

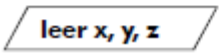
```
leer(x)
```

El mismo ejemplo en un diagrama de flujo sería:



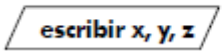

En ambos casos *x* contendrá el valor leído desde el teclado. Se pueden leer varias variables a la vez:

```
leer(x,y,z)
```




Funciona como la anterior pero usando la palabra escribir. Simula la salida de datos del algoritmo por pantalla.

```
escribir(x,y,z)
```




## CONTROL

## OPERADORES LOGICOS

Todas las instrucciones de este apartado utilizan expresiones lógicas. Son expresiones que dan como resultado un valor lógico (verdadero o falso). Suelen ser siempre comparaciones entre datos. Por ejemplo x>8 da como resultado verdadero si x vale más que 8. Los operadores de relación (de comparación) que se pueden utilizar son:

| | |
|---|---|
| > | Mayor que |
| < | Menor que |
| ≥ | Mayor o igual |
| ≤ | Menor o igual |
| ≠ | Distinto |
| = | Igual |

También se pueden unir expresiones utilizando los operadores Y (en inglés AND), el operador O (en inglés OR) o el operador NO (en inglés NOT). Estos operadores permiten unir expresiones lógicas. Por ejemplo:

| Expresión | Resultado verdadero si... |
|---|---|
| a>8 Y b<12 | La variable *a* es mayor que 8 y (a la vez) la variable *b* es menor que 12 |
| a>8 Y b<12 | O la variable *a* es mayor que 8 o la variable *b* es menor que 12. Basta que se cumpla una de las dos expresiones. |
| a>30 Y a<45 | La variable *a* está entre 31 y 44 |
| a<30 O a>45 | La variable *a* no está entre 30 y 45 |
| NO a=45 | La variable *a* no vale 45 |
| a >8 Y NO b<7 | La variable *a* es mayor que 8 y *b* es menor o igual que 7 |
| a>45 Y a<30 | Nunca es verdadero |

## ALTERNATIVA SIMPLE

```
si expresión_lógica entonces
  instrucciones
fin_si
```

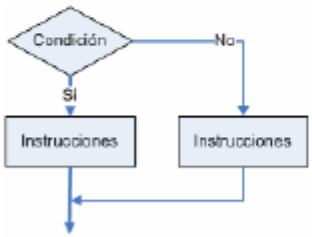

Las instrucciones sólo se ejecutarán si la expresión evaluada es verdadera.

## ALTERNATIVA DOBLE Y ANIDADAS

Se trata de una variante de la alternativa en la que se ejecutan unas instrucciones si la expresión evaluada es verdadera y otras si es falsa. Funcionamiento:

```
si expresión_lógica entonces
  instrucciones //se ejecutan si la expresión es
verdadera
si_no
  instrucciones //se ejecutan si la expresión es falsa
fin_si
```

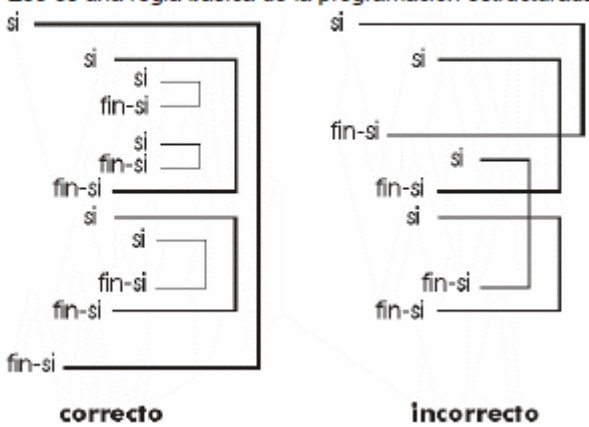
Sólo se ejecuta unas instrucciones dependiendo de si la expresión es verdadera. El diagrama de flujo equivalente es:




Hay que tener en cuenta que se puede meter una instrucción si dentro de otro si. A eso se le llama alternativas anidadas.

Al anidar estas instrucciones hay que tener en cuenta que hay que cerrar las instrucciones si interiores antes que las exteriores.

Eso es una regla básica de la programación estructurada:



**ALTERNATIVA COMPUESTA**

```
según_sea expresión hacer
  valor1:
       instrucciones del valor1
  valor2:
       instrucciones del valor2
  ...
  si-no
       instrucciones del si_no
fin_según
```

**EJEMPLOS**

```
programa pruebaSelMultiple
var
  x: entero
inicio
  escribe("Escribe un número del 1 al 4 y te diré si es par o impar")
  lee(x)
  según_sea x hacer
       1:
            escribe("impar")
       2:
            escribe("par")
       3:
            escribe("impar")
       4:
            escribe("par")
       si_no
            escribe("error eso no es un número de 1 a
4")
  fin_según
fin
```

El según sea se puede escribir también:

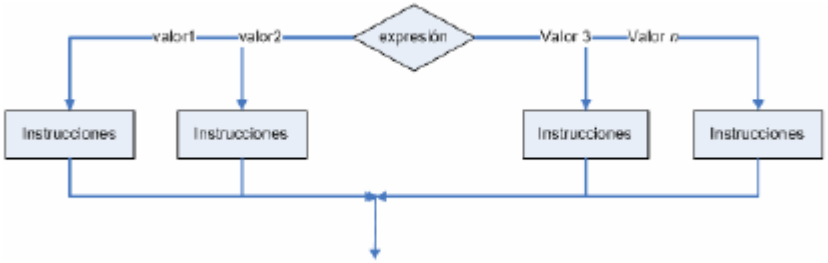
```
  según_sea x hacer
       1,3:
            escribe("impar")
       2,4:
            escribe("par")
       si_no
            escribe("error eso no es un número de 1 a
4")
  fin_según
```

Es decir el valor en realidad puede ser una lista de valores. Para indicar esa lista se pueden utilizar expresiones como:

| | |
|---|---|
| 1..3 | De uno a tres (1,2 o 3) |
| >4 | Mayor que 4 |
| >5 Y <8 | Mayor que 5 y menor que 8 |
| 7,9,11,12 | 7,9,11 y 12. Sólo esos valores (no el 10 o el 8 por ejemplo) |

Sin embargo estas últimas expresiones no son válidas en todos los lenguajes (por ejemplo el C no las admite).

En el caso de los diagramas de flujo, el formato es:




## ITERATIVA MIENTRAS

```
mientras condición hacer
  instrucciones
fin_mientras
```

Significa que las instrucciones del interior se ejecutan una y otra vez mientras la condición sea verdadera. Si la condición es falsa, las instrucciones se dejan de ejecutar. El diagrama de flujo equivalente es:

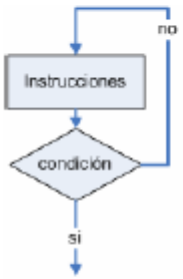

```
x←1
mientras x<=10
  escribir(x)
  x←x+1
fin_mientras
```

Las instrucciones interiores a la palabra mientras podrían incluso no ejecutarse si la condición es falsa inicialmente.

## ITERATIVA REPETIR HASTA

```
repetir
  instrucciones
hasta que condición
```

El diagrama de flujo equivalente es:




Ejemplo (escribir números del 1 al 10):

```
x←1
repetir
  escribir(x)
  x←x+1
hasta que x>10
```

---

## ITERATIVA HACER MIENTRAS

Se trata de una iteración que mezcla las dos anteriores. Ejecuta una serie de instrucciones mientras se cumpla una condición. Esta condición se evalúa tras la ejecución de las instrucciones. Es decir es un bucle de tipo *mientras* donde las instrucciones al menos se ejecutan una vez (se puede decir que es lo mismo que un bucle repetir salvo que la condición se evalúa al revés). Estructura:

```
hacer
  instrucciones
mientras condición
```

Este formato está presente en el lenguaje C y derivados (C++, Java, C#), mientras que el formato de *repetir* está presente en el lenguaje Pascal.

El diagrama de flujo equivalente es:


---

Ejemplo (escribir números del 1 al 10):

```
x←1
hacer
  escribir(x)
  x←x+1
mientras x≤10
```

---

## ITERATIVA PARA

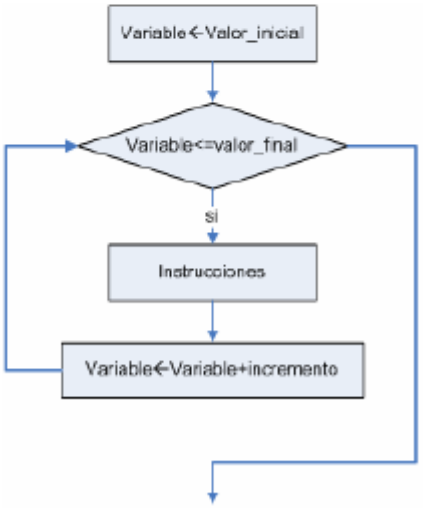
Existe otro tipo de estructura iterativa. En realidad no sería necesaria ya que lo que hace esta instrucción lo puede hacer una instrucción *mientras*, pero facilita el uso de bucles con contador. Es decir son instrucciones que se repiten continuamente según los valores de un contador al que se le pone un valor de inicio, un valor final y el incremento que realiza en cada iteración (el incremento es opcional, si no se indica se entiende que es de uno). Estructura:

```
para variable←valorInicial hasta valorfinal hacer
  instrucciones
fin_para
```

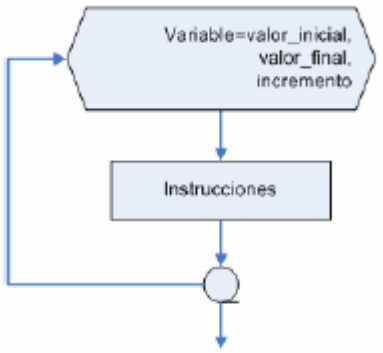
Si se usa el incremento sería:

```
para variable←vInicial hasta vFinal incremento valor
hacer
  instrucciones
fin_para
```

El diagrama de flujo equivalente a una estructura *para* sería:




También se puede utilizar este formato de diagrama:



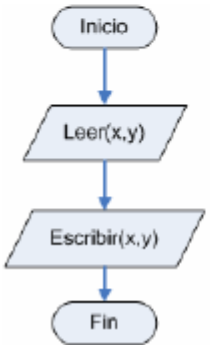

Otros formatos de pseudocódigo utilizan la palabra desde en lugar de la palabra para (que es la traducción de for, nombre que se da en el original inglés a este tipo de instrucción).

**EJEMPLO DE ALGORITMO**

El algoritmo completo que escribe el resultado de multiplicar dos números leídos por teclado sería (en pseudocódigo)

```
programa mayorDe2
var
  x,y: entero
inicio
  leer(x,y)
  escribir(x*y)
fin
```

En un diagrama de flujo:




-