Fundamentos de Programación

TEORÍA Nº 5

UNIDAD 2: INTRODUCCIÓN A LA PROGRAMACIÓN

Facultad de Ingeniería
Universidad Nacional de Jujuy

ÍNDICE

- Estructuras Repetitivas
  - PARA, MIENTRAS, REPETIR
  - Equivalencias entre estructuras repetitivas
  - Finalización de bucles
    - ✓Por contador
    - ✓Por valor centinela
    - ✓Por bandera
- Anidamiento de Control
- Prueba de escritorio

ESTRUCTURAS REPETITIVAS (1)

- Las soluciones basadas en pasos secuenciales o la selección de 2 o más caminos de acción pueden construirse mediante estructuras secuenciales y/o selectivas.
- Los problemas cuya solución consiste en la repetición de conjuntos de acciones requieren estructuras especiales llamadas **BUCLES**.

ESTRUCTURAS REPETITIVAS (2)

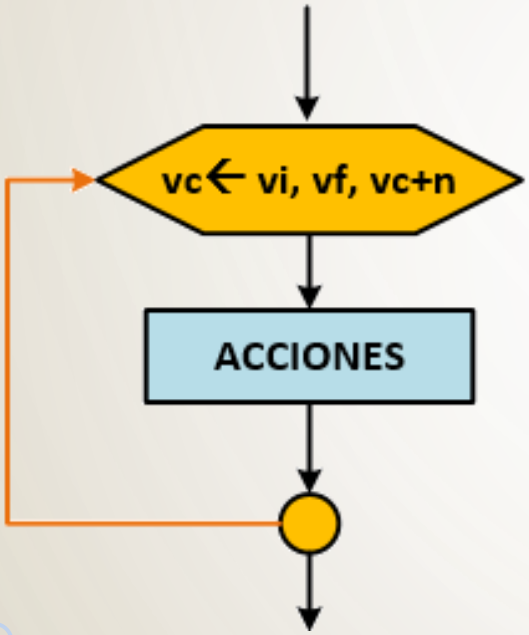
- Un **bucle** o *loop* es un conjunto de acciones que deben repetirse.
- El número de repeticiones (iteraciones) puede ser conocido a priori o no.
- En PE, las estructuras **PARA**, **MIENTRAS** y **REPETIR** permiten especificar el conjunto de acciones que deben ejecutarse en forma repetida.

ESTRUCTURA PARA (1)

- La estructura **PARA/FIN_PARA** se aplica cuando el **número de repeticiones** a realizar es **conocido**.
- La estructura utiliza una variable de control que **cuenta** las repeticiones realizadas.
- La **variable de control** varía entre *valor_inicial* y *valor_final*.
- El **incremento/decremento** de la variable de control puede especificarse (por defecto es 1).

ESTRUCTURA PARA (2)

- PARA (for)



PARA vc DESDE vi HASTA vf CON PASO n HACER
acciones
FIN_PARA

```
for(vc=vi;vc<=vf;vc=vc+n)
  accion_simple;
```

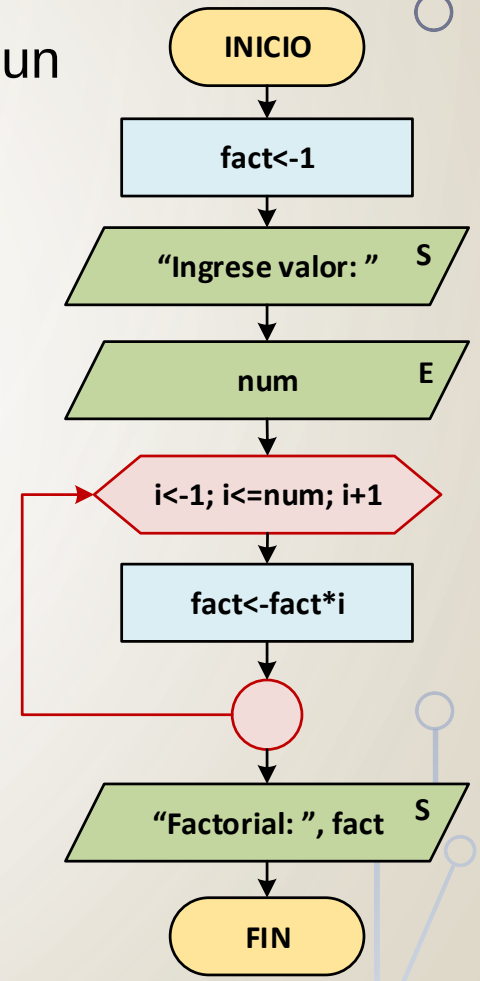
```
for(vc=vi;vc<=vf;vc=vc+n)
{
  bloque_accion;
}
```

- ✓ **vc**: ***variable de control*** del bucle
- ✓ **vi**: ***valor inicial*** de la variable de control
- ✓ **vf**: ***valor final*** de la variable de control
- ✓ **n**: ***incremento*** de la variable de control

ESTRUCTURA PARA (3)

- Diseñe un algoritmo que calcule el factorial de un número ingresado por el usuario.

```
PROGRAMA factorial
VARIABLES
  i,num, fact: Entero
INICIO
  fact<-1
  ESCRIBIR "Ingrese valor: "
  LEER num
  PARA i DESDE 1 HASTA num CON PASO 1 HACER
        fact<-fact*i
  FIN_PARA
  ESCRIBIR "Factorial: ", fact
FIN
```



ESTRUCTURA PARA (4)

Programa que calcula el factorial de un número ingresado por el usuario utilizando estructuras PARA.

PARA	

para *v* desde *vi* hasta *vf* hacer con paso *n* hacer
acciones
fin_para
	for (v=*vi*; v<=*vf*; *v*++)
	{
	acciones;
	}

```
#include <iostream>
#include <stdlib.h>

using namespace std;

main()
{ int num,i, fact=1;
  cout << "Ingrese valor: ";
  cin >> num;
  for(i=1;i<=num;i++)
     fact=fact*i;
  cout << "Factorial: " << fact << endl;
  system("pause");
}
```

PARA

ESTRUCTURA MIENTRAS (1)

- La estructura **MIENTRAS** repite un conjunto de acciones en tanto la condición de repetición sea VERDADERA.
- No necesita conocerse a priori el número de iteraciones a realizar.
- **MIENTRAS** es **pre-condicional**: la condición se evalúa antes de ejecutar el bloque de acciones (0 o más veces).
- Se aplica en cálculos aritméticos.

ESTRUCTURA MIENTRAS (2)

- MIENTRAS (while)

MIENTRAS condición HACER
acciones
FIN_PARA

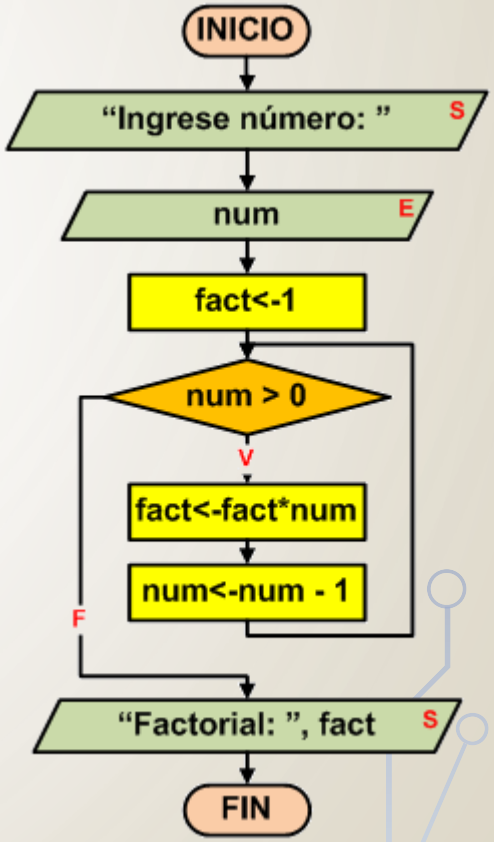
```
while (condición)
  accion_simple;
```

```
while (condición)
{
  bloque_accion;
}
```

ESTRUCTURA MIENTRAS (3)

- Diseñe un algoritmo que calcule el factorial de un número ingresado por el usuario.

```
PROGRAMA factorial
VARIABLES
  num, fact: Entero
INICIO
  ESCRIBIR "Ingrese numero: "
  LEER num
  fact<-1
   MIENTRAS num > 0 HACER
         fact<-fact*num
     num<-num-1
  FIN_MIENTRAS
  ESCRIBIR "Factorial: ", fact
FIN
```



ESTRUCTURA MIENTRAS (4)

Programa que calcula el factorial de un número ingresado por el usuario, utilizando estructuras MIENTRAS.

MIENTRAS	

mientras condición hacer
acciones
fin_mientras
	while (*condición*)
	{
	acciones;
	}

```
#include <iostream>
#include <stdlib.h>

using namespace std;

main()
{ int num, fact=1;
  cout << "Ingrese numero: ";
  cin >> num;
  while (num>0)
   { fact=fact*num;
     num=num-1;
   }
  cout << "Factorial: " << fact << endl;
  system("pause");
}
```

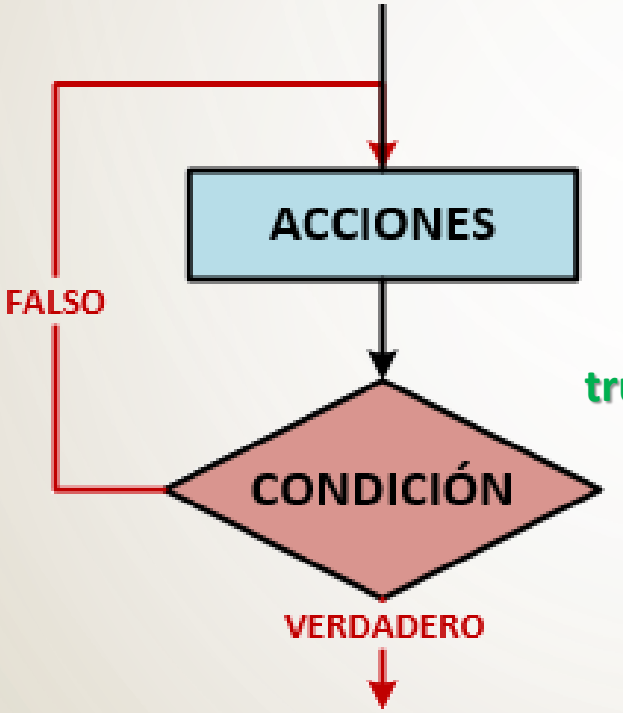
MIENTRAS

ESTRUCTURA REPETIR (1)

- La estructura **REPETIR** repite un conjunto de acciones en tanto la condición de repetición sea FALSA.
- No necesita conocer a priori el número de iteraciones a realizar.
- **REPETIR** es **pos-condicional**: el bloque de acciones se ejecuta antes de evaluar la condición; si ésta es FALSA, el bloque de acciones se ejecuta nuevamente (1 o más veces).
- Se utiliza en el ingreso de datos.

ESTRUCTURA REPETIR (2)

- REPETIR (do-while)



REPETIR
acciones
HASTA QUE condición

```
do
{
    acciones
} while (condición_opuesta);
```

true

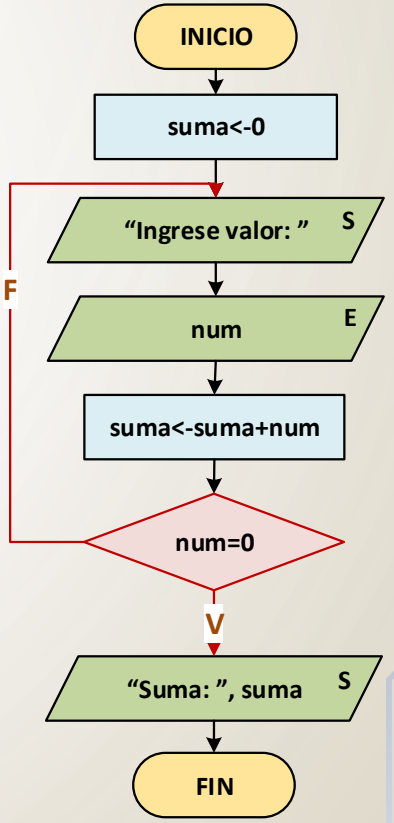
false

La estructura do-while repite el bloque de acciones con condición VERDADERA y finaliza con condición FALSA, contrario a la REPETIR de diseño.

ESTRUCTURA REPETIR (3)

- Diseñe un algoritmo que calcula la suma de valores ingresados por el usuario hasta que se introduce un CERO.

```
PROGRAMA suma_valores
VARIABLES
  num,suma: ENTERO
INICIO
  suma<-0
  REPETIR
        ESCRIBIR "Ingrese un valor: "
        LEER num
        suma<-suma+num
  HASTA_QUE num=0
  ESCRIBIR "La suma de valores es: ", suma
FIN
```



ESTRUCTURA REPETIR (4)

Programa que calcula la suma de valores ingresados por el usuario hasta que se introduce un CERO.

REPETIR	
repetir acciones hasta_que *condición*	**Do** **{** acciones; **} while** (*condición*);

```
#include <iostream>
#include <stdlib.h>

using namespace std;

main()
{ int num,suma=0;
  do
  { cout << "Ingrese un valor: ";
    cin >> num;
    suma=suma+num;
  } while (num!=0);
  cout << "La suma de valores es: " << suma << endl;
  system("pause");
}
```

REPETIR

ESTRUCTURA REPETIR (5)

- Modifique el algoritmo anterior de modo que utilice el concepto de bandera para finalizar el bucle.

```
#include <iostream>
#include <stdlib.h>
using namespace std;
main()
{ float num, suma=0;
  bool seguir;
  do
  { cout << "Ingrese valor: ";
    cin >> num;
    suma=suma+num;
    if (num==0)
      seguir=false;
    else
      seguir=true;
  } while(seguir==true);
  cout << "La suma es: " << suma << endl;
  system("pause");
}
```

- La variable lógica ***seguir*** permite detectar en qué momento se ingresa un dato cero.
- ***seguir*** es VERDADERA si el dato ingresado es distinto de cero.
- ***seguir*** es FALSA cuando el valor ingresado es igual a cero.
- el bucle finaliza cuando ***seguir*** es FALSA (num=0), ya que la condición de repetición no se cumple.

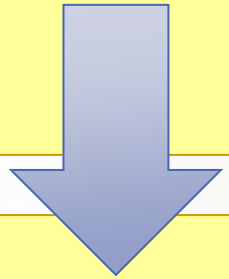
MIENTRAS VS REPETIR

- Evaluación de la condición de repetición
  - **MIENTRAS** evalúa la condición antes de ejecutar el bloque de acciones (0 o más veces)
  - **REPETIR** evalúa la condición luego de ejecutar el bloque de acciones (1 o más veces)
- Finalización de bucle
  - **MIENTRAS** finaliza con condición **FALSA**
  - **REPETIR** finaliza con condición **VERDADERA**

PARA, MIENTRAS Y REPETIR (1)

- Equivalencia entre **PARA** y **MIENTRAS**

```
PARA k DESDE 1 HASTA valor CON PASO n HACER
      Bloque de Acciones
FIN_PARA
```



```
k<-1 //inicialización de la var. de control
MIENTRAS k <= valor HACER //control valor final
      Bloque de Acciones
      k<-k+n //incremento
FIN_MIENTRAS
```

PARA, MIENTRAS Y REPETIR (2)

- Equivalencia entre **PARA** y **REPETIR**

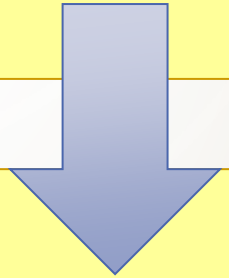
```
PARA k DESDE 1 HASTA valor CON PASO n HACER
      Bloque de Acciones
FIN_PARA
```

```
k<-1 //inicialización de la var. de control
REPETIR
      Bloque de Acciones
      k<-k+n //incremento
HASTA_QUE k > valor //control de valor final
```

PARA, MIENTRAS Y REPETIR (3)

- Equivalencia entre **MIENTRAS** y **REPETIR**

```
MIENTRAS condición HACER
        Bloque de Acciones
        Acción que modifica la condición
FIN_MIENTRAS
```



```
REPETIR
        Bloque de Acciones
        Acción que modifica la condición
HASTA_QUE condición
```

FINALIZACIÓN DE BUCLES (1)

- ¿Qué ocurre cuando en un programa un conjunto de acciones se repite sin control?
  - Bucles infinitos

- ¿Cuáles son los criterios para finalizar las iteraciones de un bucle?
  - Por *Valor Centinela*
  - Por *Bandera*
  - Por *Contador*

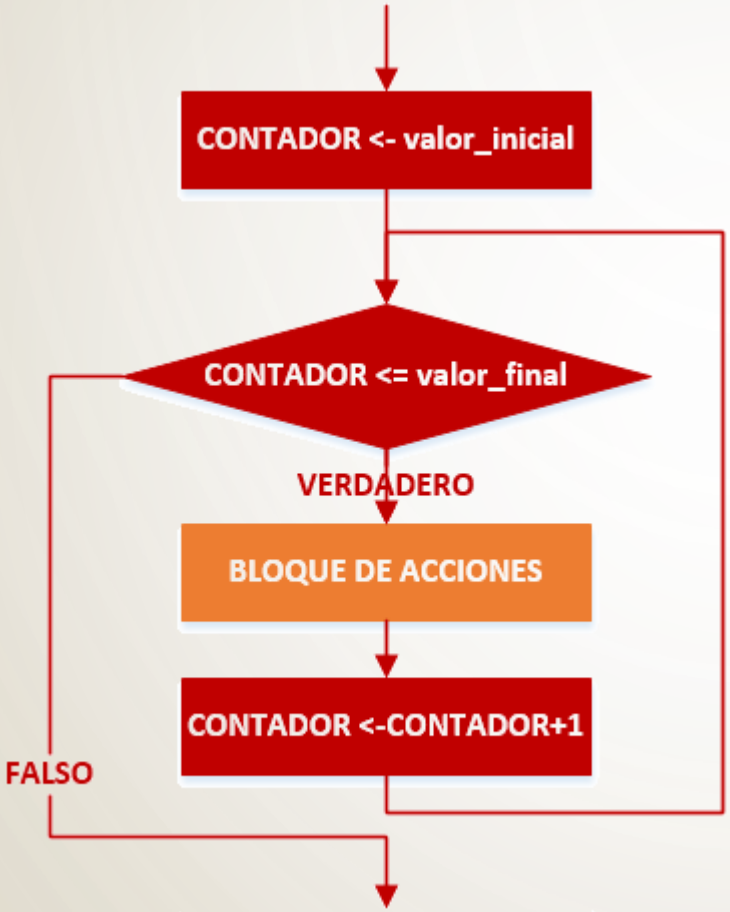
FINALIZACIÓN DE BUCLES (2)

- Los **bucles infinitos** no alcanzan la condición de finalización y por tanto se repiten indefinidamente.
- Se deben a errores de diseño: incorrecta formulación de la condición de finalización, omisión o errores en las instrucciones que modifican la condición de salida.

```
bandera<-VERDADERO
MIENTRAS bandera=VERDADERO HACER
        ESCRIBIR "BUCLE INFINITO"
FIN_MIENTRAS
```

```
contador<-1
MIENTRAS contador < 20 HACER
        ESCRIBIR "BUCLE INFINITO"
        contador<-contador - 1
FIN_MIENTRAS
```

MIENTRAS: CONTROLADO POR CONTADOR (1)



- **Inicialización**: se asigna el valor inicial al contador
- **Condición de repetición**: se verifica que el contador se encuentre entre el valor_inicial y valor_final
  - Cond. Verdadera repite
  - Cond. Falsa finaliza
- **Modificación del contador**: se incrementa o decrementa el valor del contador.

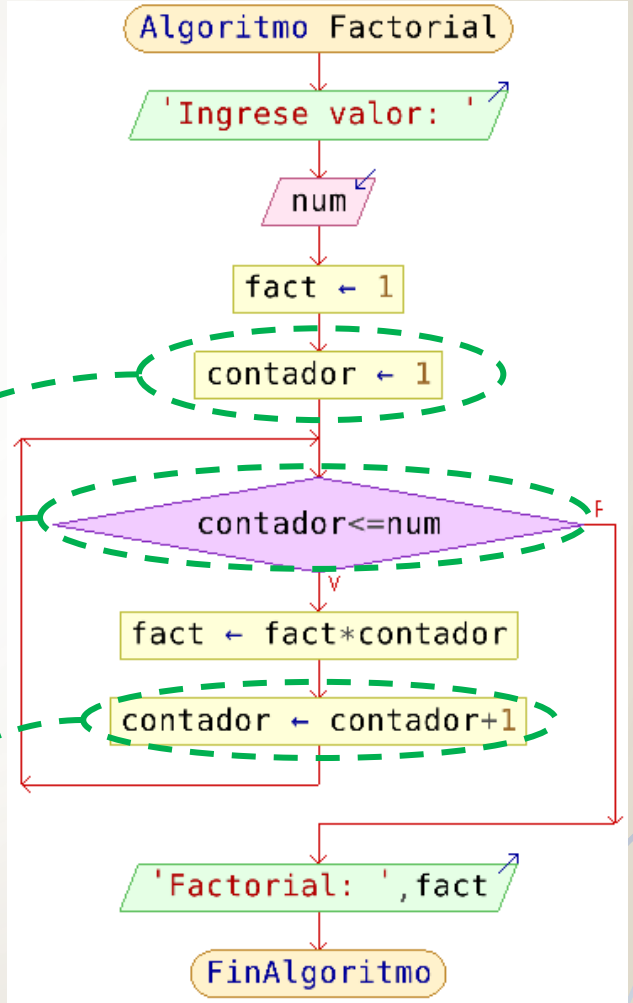
MIENTRAS: CONTROLADO POR CONTADOR (2)

- Diseñe un algoritmo que calcule el factorial de un número ingresado por el usuario. Implemente el bucle de cálculo con estructuras MIENTRAS y utilice la finalización por contador.
- Inicialización:

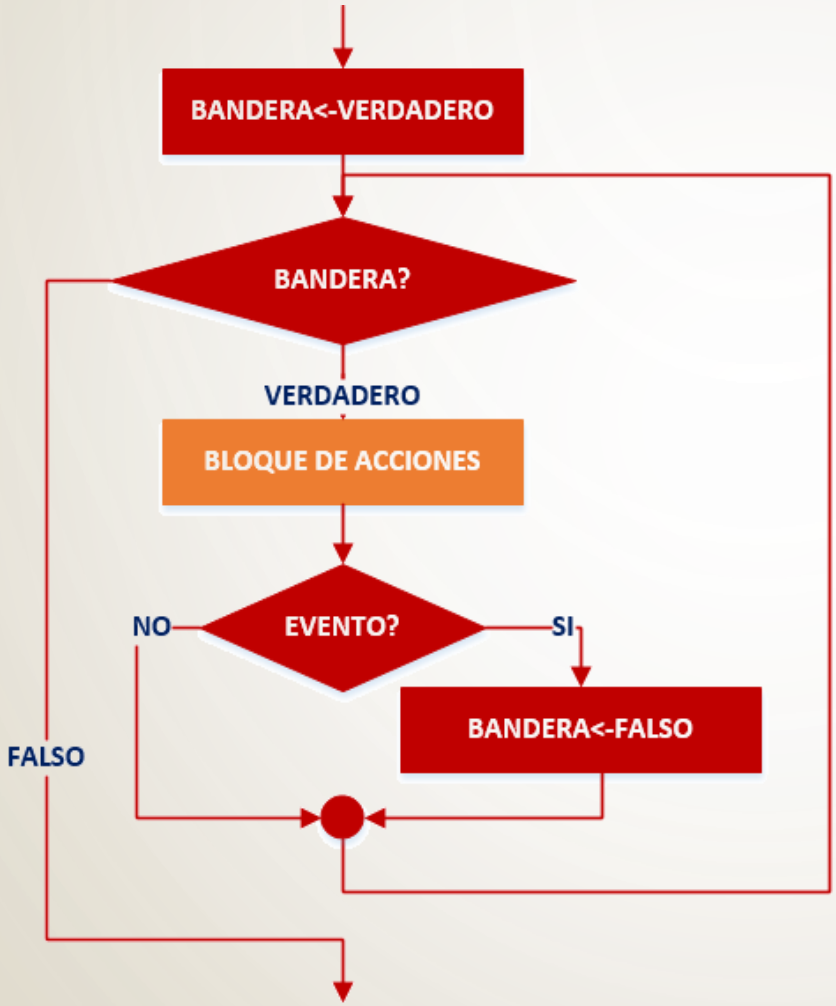
  contador ← 1
- Condición de repetición:

  contador <= num
- Modificación:

  contador ← contador + 1



MIENTRAS: CONTROLADO POR BANDERA (1)



- **Inicialización**: se asigna el valor inicial a la bandera
- **Condición de repetición**: se analiza el valor de la bandera
  - Cond. Verdadera repite
  - Cond. Falsa finaliza
- **Detección de un evento**: se verifica si determinado evento ocurrió o no en el programa. La ocurrencia del evento implica modificar la bandera.

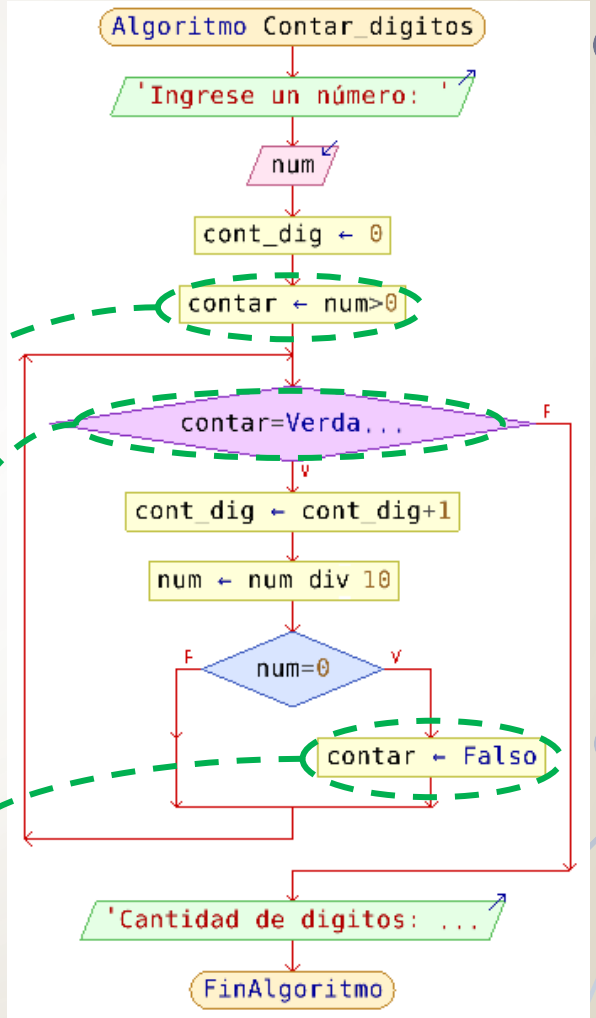
MIENTRAS: CONTROLADO POR BANDERA (2)

- Diseñe un algoritmo que calcule cuántos dígitos tiene un número ingresado por el usuario. Implemente el bucle de cálculo con estructuras MIENTRAS y utilice la finalización por bandera.
- Inicialización:

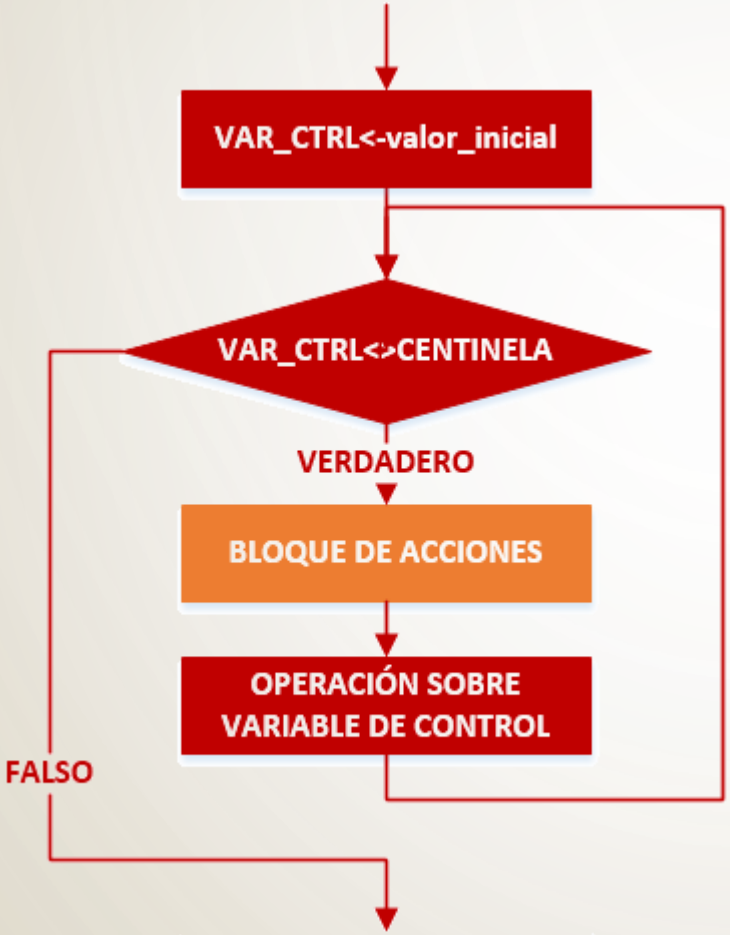
  contar ← num > 0
- Condición de repetición:

  contar = Verdadero
- Modificación:

  contar ← Falso



MIENTRAS: CONTROLADO POR CENTINELA (1)



- **Inicialización**: se asigna el valor inicial a la variable de control del bucle
- **Condición de repetición**: se verifica que la variable de control sea distinta al valor centinela.
  - Cond. Verdadera repite
  - Cond. Falsa finaliza
- **Modificación de la variable de control**: se realiza alguna operación que modifica la variable de control.

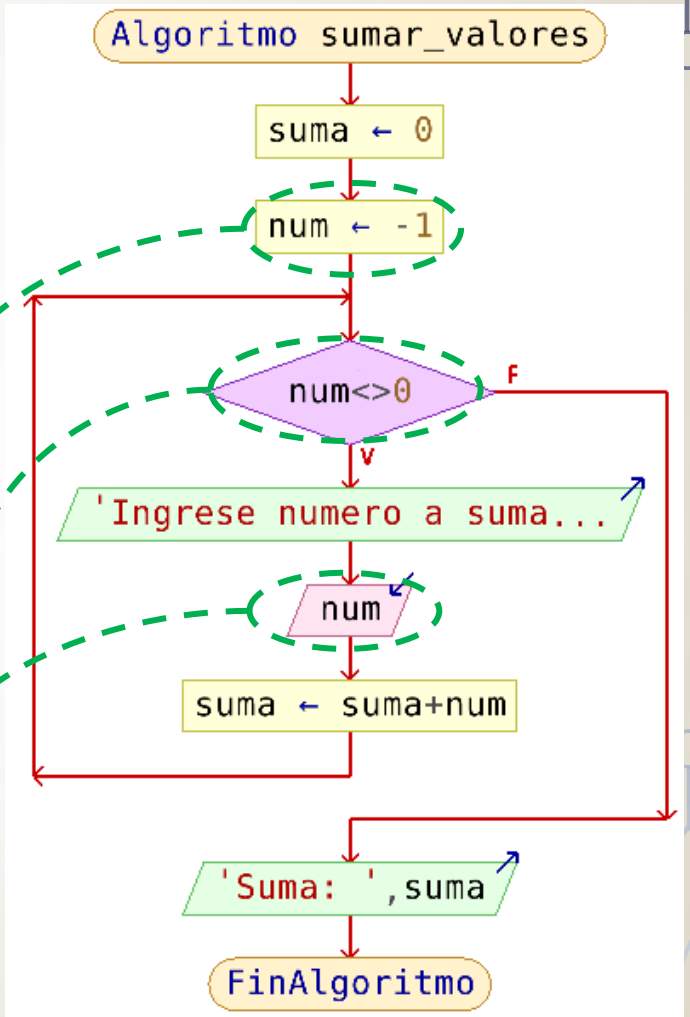
MIENTRAS: CONTROLADO POR CENTINELA (2)

- Diseñe un algoritmo que sume valores ingresados por el usuario, hasta que se introduzca un 0. Implemente el bucle de cálculo con estructuras MIENTRAS y utilice la finalización por centinela.
- Inicialización:

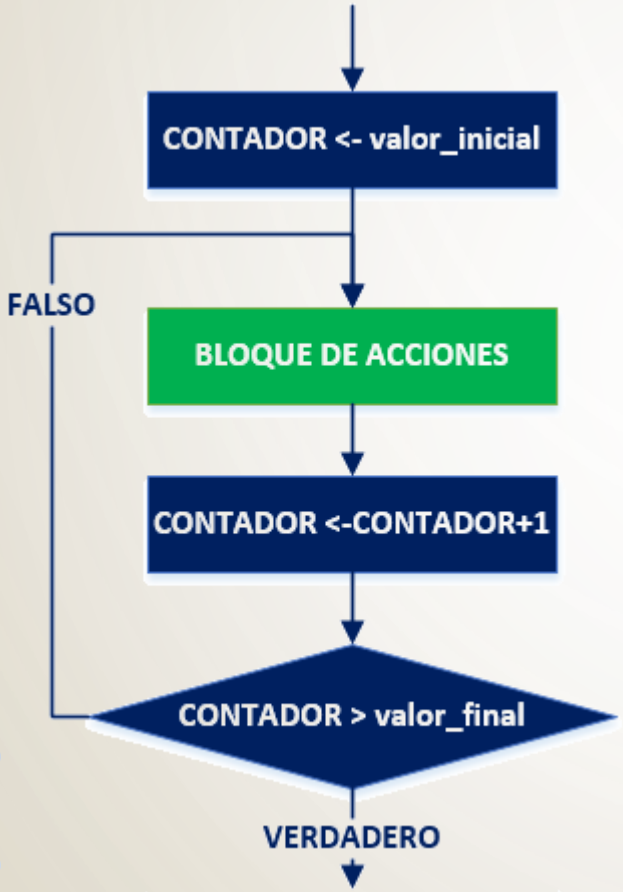
  num ← -1 (arbitrario)
- Condición de repetición:

  num <> 0
- Modificación:

  LEER num



REPETIR: CONTROLADO POR CONTADOR (1)



- **Inicialización**: se asigna el valor inicial al contador
- **Condición de repetición**: se verifica que el contador no supere el valor_final
  - Cond. Falsa repite
  - Cond. Verdadera finaliza
- **Modificación del contador**: se incrementa o decrementa el valor del contador.

REPETIR: CONTROLADO POR CONTADOR (2)

- Diseñe un algoritmo que calcule el producto, mediante sumas, de 2 números ingresados por el usuario. Implemente el bucle de cálculo con estructuras REPETIR y use la finalización por contador.
- Inicialización:

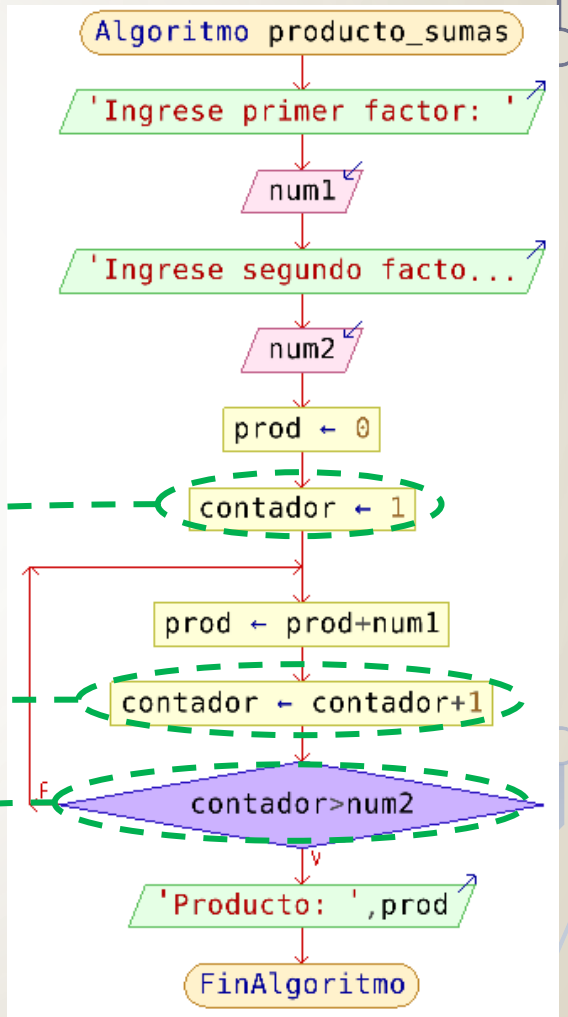
  contador ← 1

- Modificación:

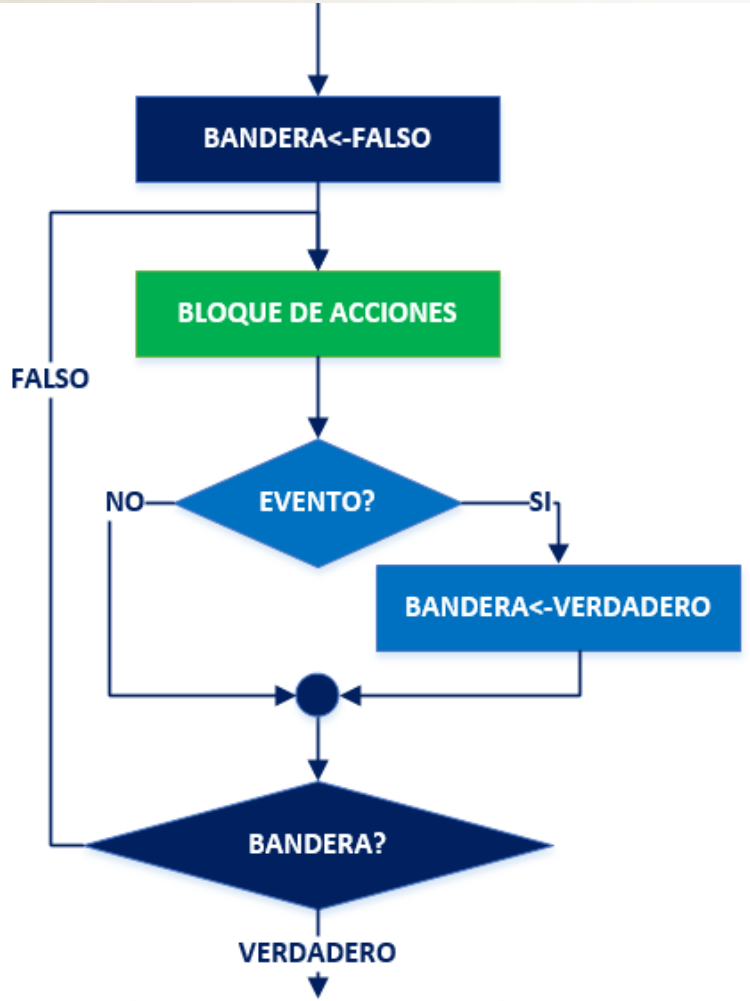
  contador ← contador + 1

- Condición de repetición:

  contador > num2



REPETIR: CONTROLADO POR BANDERA (1)



- **Inicialización**: se asigna el valor inicial a la bandera
- **Condición de repetición**: se analiza el valor de la bandera
  - Cond. Falsa repite
  - Cond. Verdadera finaliza
- **Detección de un evento**: se verifica si determinado evento ocurrió o no en el programa. La ocurrencia del evento implica modificar la bandera.

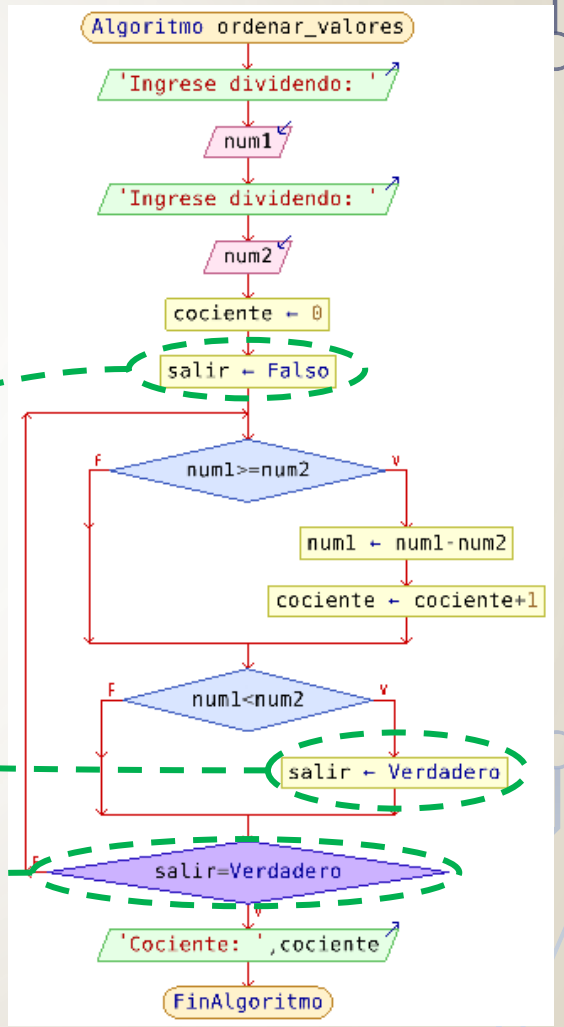
REPETIR: CONTROLADO POR BANDERA (2)

- Diseñe un algoritmo que calcule el cociente entero, mediante restas, de 2 números ingresados por el usuario. Implemente el bucle de cálculo con estructuras REPETIR y utilice la finalización por bandera.
- Inicialización:

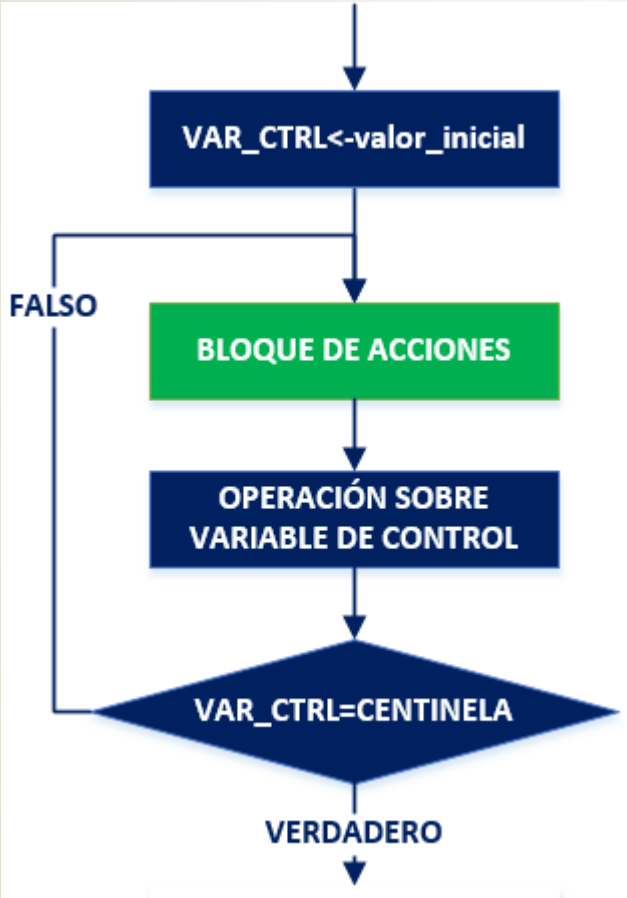
  salir ← Falso
- Modificación:

  salir ← Verdadero
- Condición de repetición:

  salir = Verdadero



REPETIR: CONTROLADO POR CENTINELA (1)



- **Inicialización**: se asigna el valor inicial a la variable de control del bucle (OPCIONAL)
- **Condición de repetición**: se verifica que la variable de control sea igual al valor centinela.
  - Cond. Falsa repite
  - Cond. Verdadera finaliza
- **Modificación de la variable de control**: se realiza alguna operación que modifica la variable de control.

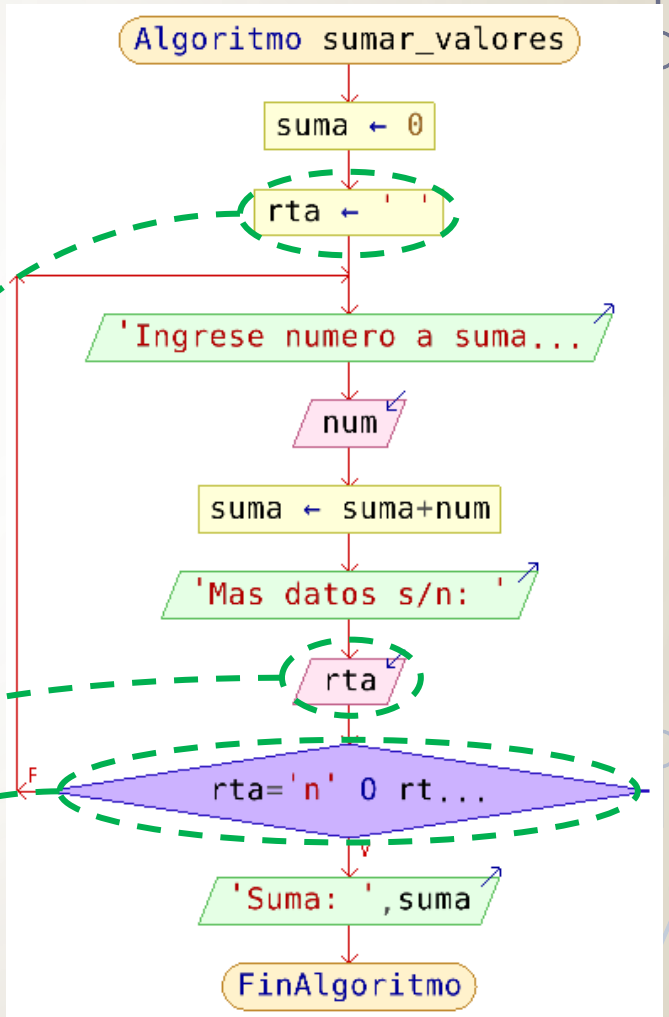
REPETIR: CONTROLADO POR CENTINELA (2)

- Diseñe un algoritmo que sume valores ingresados por el usuario. Considere que el ingreso finaliza a petición del usuario. Implemente el bucle de cálculo con estructuras REPETIR y utilice la finalización por centinela.
- Inicialización:

  rta ← ‘ ‘ (arbitrario y opcional)
- Modificación:

  LEER rta
- Condición de repetición:

  rta = ‘n’ O rta = ‘N’



RESUMEN (1)

- Estructura **PARA**
  - Utiliza una variable de control de bucle (contador) que lleva cuenta del número de repeticiones.
  - Se aplica cuando se conoce el número de repeticiones a realizar.
  - El incremento de la variable de control puede ser configurado, por defecto, es 1.
  - Es una estructura pre-condicional

RESUMEN (2)

- Estructura **MIENTRAS**
  - Pre-condicional: la condición de repetición se evalúa antes de iniciar cada iteración del bucle.
  - Repite con condición VERDADERA, finaliza con condición FALSA.
  - Se aplica cuando NO se conoce el número de repeticiones a realizar.
  - Siempre debe incluir alguna instrucción que modifique la condición de repetición (finalización del bucle).

RESUMEN (3)

- Estructura **REPETIR**
  - Pos-condicional: la condición de repetición se evalúa luego de ejecutar cada iteración del bucle.
  - Repite con condición FALSA, finaliza con condición VERDADERA.
  - Se aplica cuando NO se conoce el número de repeticiones a realizar.
  - Siempre debe incluir alguna instrucción que modifique la condición de repetición (finalización del bucle).

ANIDAMIENTO (1)

- Consiste en combinar las estructuras de control básicas.
- Una estructura de control puede contener otra si es necesario (*anidamiento)*.
- Reglas para el anidamiento
  - la estructura interna debe quedar completamente incluida dentro de la externa, y
  - no puede existir solapamiento de estructuras.

ANIDAMIENTO VÁLIDO (2)

- Ejemplos

```
SI condición_1 ENTONCES
    SI condición_2 ENTONCES
        acciones
    SINO
        acciones
    FIN_SI
SINO
    SI condición_3 ENTONCES
        acciones
    FIN_SI
FIN_SI
```

```
MIENTRAS condición_1 HACER
    PARA i DESDE vi HASTA vf HACER
        SI condición_2 ENTONCES
            acciones_1
        SINO
            acciones_2
        FIN_SI
        acciones_3
    FIN_PARA
FIN_MIENTRAS
```

ANIDAMIENTO INVÁLIDO (3)

- Ejemplos

```
MIENTRAS condición_1 HACER
    SI condición_2 ENTONCES
        acciones
    FIN_MIENTRAS
FIN_SI
```

```
PARA i DESDE vi HASTA vf HACER
    MIENTRAS condición_1 HACER
        acciones
    FIN_PARA
FIN_MIENTRAS
```

```
REPETIR
    MIENTRAS condición_2 HACER
        SI condición_3 ENTONCES
            acciones
    FIN_MIENTRAS
        SINO
            SI condición_4 ENTONCES
                acciones
            FIN_SI
        FIN_SI
HASTA_QUE condición_1
```

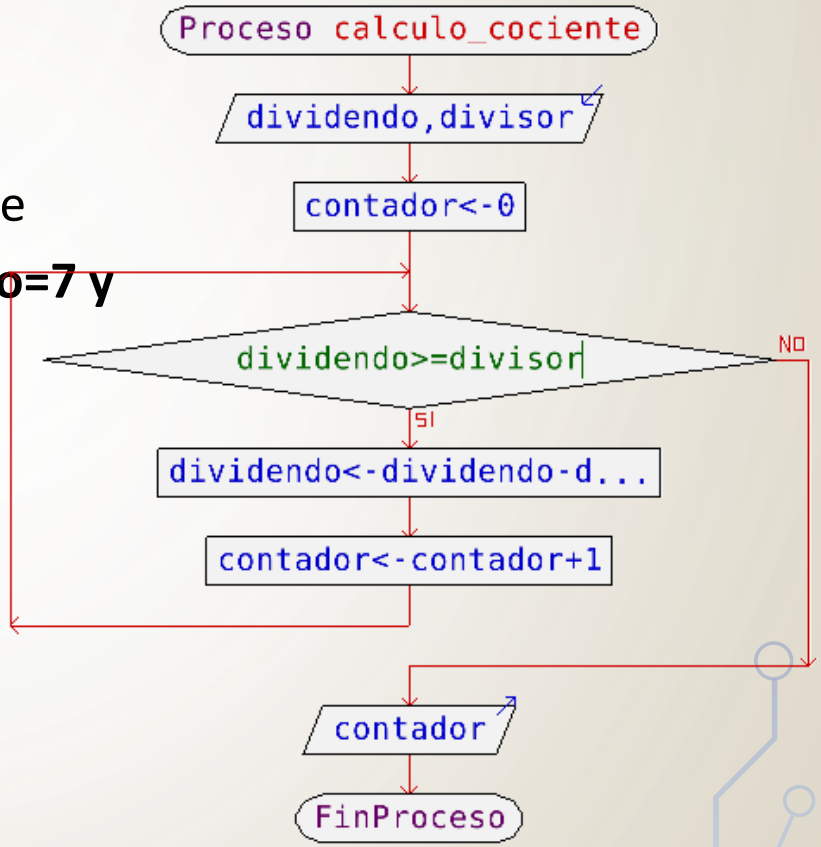
```
SI condición_1 ENTONCES
    PARA i DESDE vi HASTA vf HACER
        acciones
    FIN_PARA
REPETIR
SINO
    acciones
FINS_SI
HASTA_QUE condición_2
```

PRUEBA DE ESCRITORIO (1)

- Comprobación de un algoritmo en tiempo de diseño.
- Se analiza, paso a paso, el algoritmo y se indican los valores de las variables y condiciones.
- Se pueden probar tanto datos esperados como valores de excepción.

PRUEBA DE ESCRITORIO (2)

- Diseñe un algoritmo que calcule el cociente entero entre dos números ingresados por el usuario, aplicando restas sucesivas. Realice la prueba de escritorio para los valores: **dividendo=7 y divisor=2**



	VARIABLES			CONDICIONES
PASO	dividendo	divisor	contador	dividendo>=divisor
1	7			
2		2		
3			0	
4				VERDADERO
5	5			
6			1	
7				VERDADERO
8	3			
9			2	
10				VERDADERO
11	1			
12			3	
13				FALSO
RESULTADO: 3				

RESUMEN (1)

- Estructura **PARA**
  - Utiliza una variable de control de bucle (contador) que lleva cuenta del número de repeticiones.
  - Se aplica cuando se conoce el número de repeticiones a realizar.
  - El incremento de la variable de control puede ser configurado, por defecto, es 1.
  - Es una estructura pre-condicional

RESUMEN (2)

- Estructura **MIENTRAS**
  - Pre-condicional: la condición de repetición se evalúa antes de iniciar cada iteración del bucle.
  - Repite con condición VERDADERA, finaliza con condición FALSA.
  - Se aplica cuando NO se conoce el número de repeticiones a realizar.
  - Siempre debe incluir alguna instrucción que modifique la condición de repetición (finalización del bucle).

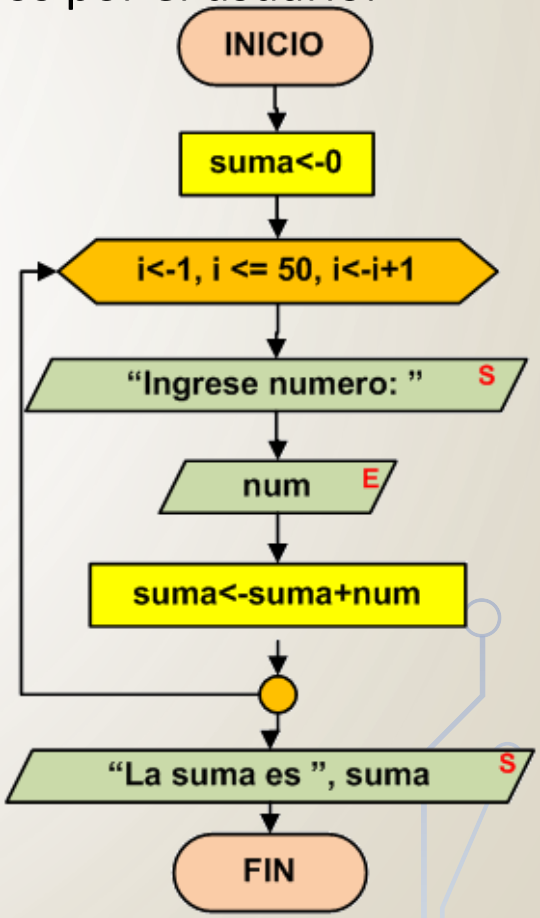
RESUMEN (3)

- Estructura **REPETIR**
  - Pos-condicional: la condición de repetición se evalúa luego de ejecutar cada iteración del bucle.
  - Repite con condición FALSA, finaliza con condición VERDADERA.
  - Se aplica cuando NO se conoce el número de repeticiones a realizar.
  - Siempre debe incluir alguna instrucción que modifique la condición de repetición (finalización del bucle).

EJERCICIOS A RESOLVER EN C++ (1)

- Diseñe un algoritmo que sume ***50*** **valores** ingresados por el usuario.

```
PROGRAMA ej_bucle_2
VARIABLES
    num, suma: REAL
    i: ENTERO
INICIO
    suma<-0 //inicialización de suma
    PARA i DESDE 1 HASTA 50 CON PASO 1 HACER
      ESCRIBIR "Ingrese numero"
      LEER num
      suma<-suma+num
    FIN_PARA
    ESCRIBIR "La suma es ", suma
FIN
```



EJERCICIOS A RESOLVER EN C++ (2)

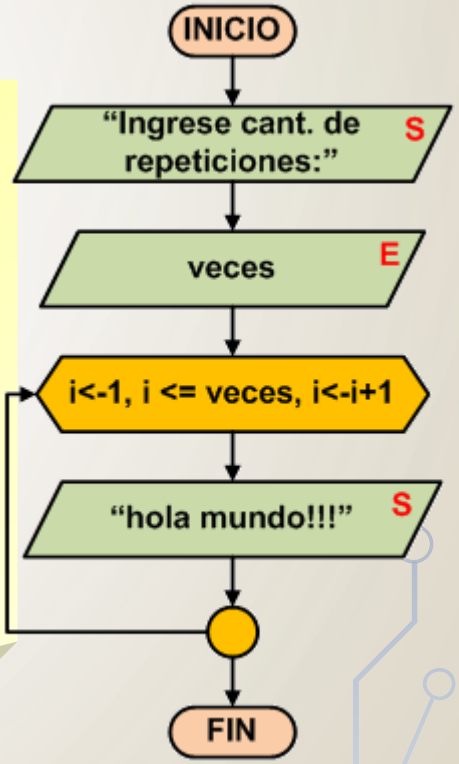
- Diseñe un algoritmo que calcule la potencia a^b mediante productos sucesivos.

```
PROGRAMA calculo_potencia
VARIABLES
   base, exp: ENTERO // variables de entrada
   i: ENTERO  // variable auxiliar
   potencia: ENTERO // variable de salida
INICIO
   ESCRIBIR "Ingrese base: "
   LEER base
   ESCRIBIR "Ingrese exponente: "
   LEER exp
   potencia<-1 // acumulador de productos (valor inicial 1)
   PARA i DESDE 1 HASTA exp CON PASO 1 HACER
      potencia<-potencia*base
   FIN_PARA
   ESCRIBIR "La potencia es: ", potencia
FIN
```

EJERCICIOS A RESOLVER EN C++ (3)

- Diseñe un algoritmo que muestre ***n* veces** el mensaje “hola mundo!!!”siendo ***n*** especificado por el usuario.

```
PROGRAMA ej_bucle_1
VARIABLES
   veces, i: ENTERO
INICIO
   ESCRIBIR “Ingrese cant. de repeticiones: ”
   LEER veces
   PARA i DESDE 1 HASTA veces CON PASO 1 HACER
     ESCRIBIR “hola mundo!!!”
   FIN_PARA
FIN
```



EJERCICIOS A RESOLVER EN C++ (4)

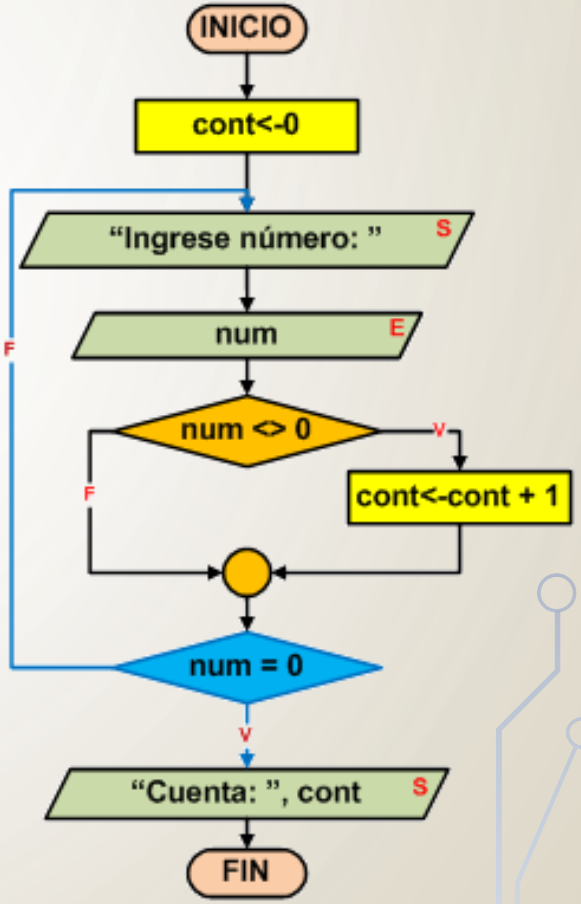
- Diseñe un algoritmo que calcule el máximo común divisor de 2 números ingresados por el usuario.

```
PROGRAMA calculo_mcd
VARIABLES
   num1, num2: ENTERO // variables de entrada
   i: ENTERO  // variable auxiliar
   mcd: ENTERO // variable de salida
INICIO
   ESCRIBIR "Ingrese valor: "
   LEER num1
   ESCRIBIR "Ingrese valor: "
   LEER num2
   mcd<-1
   i<-2 // contador (divisor de m y n)
   MIENTRAS i < num1 Y i < num2 HACER
      SI m mod i = 0 Y n mod i = 0 ENTONCES
        mcd<-i
      FIN_SI
      i<-i+1
   FIN_MIENTRAS
   ESCRIBIR "Max divisor comun: ", mcd
FIN
```

EJERCICIOS A RESOLVER EN C++ (5)

- Diseñe un algoritmo que cuente valores ingresados por el usuario hasta que se presente **un CERO**.

```
PROGRAMA ej_bucle_4
VARIABLES
   num, cont: ENTERO
INICIO
   cont<-0
   REPETIR
      ESCRIBIR "Ingrese numero"
      LEER num
      SI num<>0 ENTONCES
        cont<-cont+1
      FINS_SI
   HASTA_QUE num = 0
   ESCRIBIR "Cuenta: ", cont
FIN
```



EJERCICIOS A RESOLVER EN C++ (6)

- Diseñe un algoritmo que determine el mínimo de una serie de datos ingresados por el usuario. Finaliza con 0.

```
PROGRAMA calculo_minimo
VARIABLES
   num: ENTERO // variables de entrada
   primero: LOGICO // variable auxiliar (bandera)
   min: ENTERO // variable de salida
INICIO
   primero<-VERDADERO
   REPETIR
      ESCRIBIR "Ingrese valor: "
      LEER num
      SI primero = VERDADERO ENTONCES
         min<-num
         primero<-FALSO
      SINO
        SI min > num Y num <> 0 ENTONCES
           min<-num
        FIN_SI
      FIN_SI
   HASTA_QUE num=0
   ESCRIBIR "Mínimo: ", min
FIN
```

EJERCICIOS A RESOLVER EN C++ (7)

- Diseñe un algoritmo que sume valores hasta que el usuario ingrese un valor par. Utilice el concepto de centinela para controlar el bucle.

```
PROGRAMA centinela
VARIABLES
   num,suma:REAL
INICIO
   suma<-0
   REPETIR
      ESCRIBIR "Ingrese numero"
      LEER num
      suma<-suma+num
   HASTA_QUE num mod 2 = 0
   ESCRIBIR "La suma es ", suma
FIN
```

- Modifique el algoritmo anterior para utilizar el concepto de bandera para control del bucle.

BIBLIOGRAFÍA

- Sznajdleder, Pablo Augusto. Algoritmos a fondo. Alfaomega. 2012.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.
- Hernández, Roberto *et al.* Estructuras de datos y algoritmos. Prentice Hall. 2001.
- Fundamentos Básicos de Programación en C++. Martínez del Rio. Jaén. 2015
- Curso de C++. https://www.programarya.com/Cursos/C++