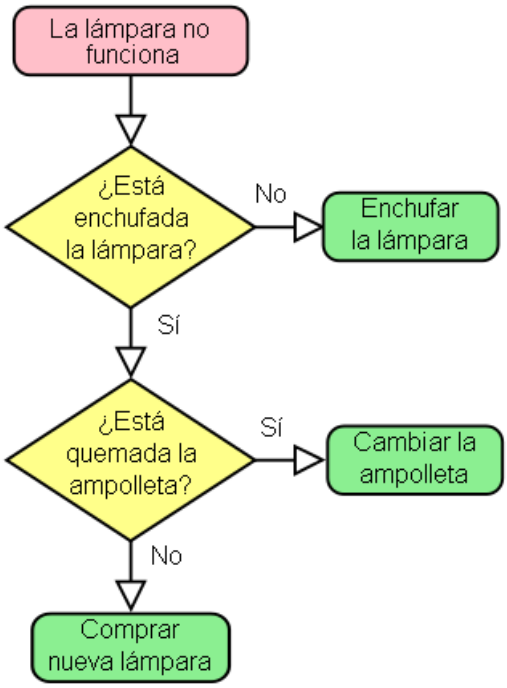
UNIDAD III: PROGRAMACIÓN ESTRUCTURADA



El diseño de algoritmos constituye una de las fases más importantes en la resolución de problemas basados en computadora. En esta fase se indican las acciones que se llevarán a cabo para dar solución a un problema específico. El paradigma de la programación estructurada utiliza 3 estructuras de control (secuenciales, condicionales y repetitivas) para representar los pasos de resolución de un algoritmo. A continuación se aborda este paradigma.

Programación Estructurada

La *Programación Estructurada* se refiere al conjunto de técnicas empleadas para aumentar la productividad de los programas, reduciendo el tiempo necesario para escribir, verificar, depurar y mantener los programas. El paradigma de la Programación Estructurada utiliza un número limitado de estructuras de control que minimizan la complejidad de los problemas, y por consiguiente, reducen los errores.

Este paradigma hace que los programas sean más fáciles de escribir, verificar, leer y mantener. Los programas deben estar dotados de estructura.

La Programación Estructurada es el conjunto de técnicas que incorporan:

- Diseño descendente
- Recursos Abstractos
- Estructuras Básicas

Diseño Descendente

El *diseño descendente* (*TOP-DOWN*) es el proceso mediante el cual un problema se descompone en una serie de niveles o pasos sucesivos de refinamiento (*stepwise*). La metodología descendente consiste en efectuar una relación entre las sucesivas etapas de estructuración, de modo que se relacionen unas con otras mediante entradas y salidas de información. Es decir, se descompone el problema en etapas o estructuras jerárquicas, de modo que se puede considerar

cada estructura desde dos puntos de vista: ¿lo que hace? (Figura 2) y ¿cómo lo hace? (Figura 3)

Figura 2. Diseño Top Down para el cálculo del factorial de un número.

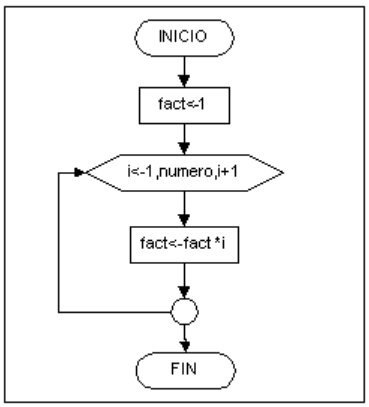


Figura 3. Algoritmo (diagrama de flujo) para el cálculo del factorial de un número.

El diseño descendente comienza con un problema general y se van diseñando soluciones especificas para cada uno de los subproblemas en los que ha sido divido el programa general.

Recursos Abstractos

La Programación Estructurada se auxilia de *recursos abstractos* en lugar de los recursos concretos de que se dispone (un determinado lenguaje de programación).

Descomponer un programa en términos de recursos abstractos, consiste en descomponer una determinada acción compleja en términos de un número de acciones más simples capaces de ser ejecutadas por una computadora y que constituirán sus instrucciones.

Estructuras Básicas: Teorema de la Programación Estructurada

El *Teorema de la Programación Estructurada* establece que un programa propio puede ser escrito utilizando solamente las siguientes estructuras de control:

- Secuenciales
- Selectivas
- Repetitivas

Un programa se define como propio si cumple con las siguientes características:

- tiene exactamente una entrada y una salida para control del programa,
- existen caminos que se pueden seguir desde la entrada hasta la salida que conducen por cada parte del programa, es decir, no existen lazos infinitos ni instrucciones que no se ejecutan.

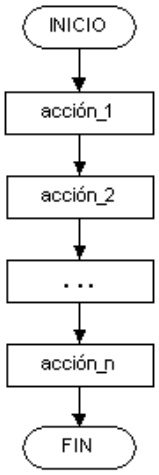
Secuenciales

La estructura de control más simple está representada por una sucesión de operaciones, en la que el orden de ejecución coincide con el orden físico de aparición de las instrucciones. La representación en pseudocódigo y diagrama de flujo de una estructura secuencial se presenta a continuación:

Pseudocódigo

```
INICIO
    acción_1
    acción_2
    . . .
    acción_n
FIN
```

Diagrama de Flujo



Observe que *INICIO* y *FIN* indican el punto de entrada y punto de salida, respectivamente, del algoritmo.

El siguiente ejemplo muestra un algoritmo (pseudocódigo y diagrama de flujo) que resuelve la suma de dos valores (*valor_a* y *valor_b*) ingresados por el usuario a través de instrucciones secuenciales (lectura, asignación, escritura).

El primer paso en la resolución de este problema es establecer el conjunto de datos de entrada, los resultados de salida y el objetivo del algoritmo. Para ello, se elaboran:

- diccionarios de datos: en él se registran las variables primarias, variables secundarias (auxiliares) y constantes de entrada identificadas en el problema.
- diccionario de resultados: en él se registran las variables primarias y constantes de salida identificadas en el problema.
- diccionario de condiciones vinculantes: en él se indican los pasos generales de la solución requerida

Para el ejemplo planteado se generan los siguientes diccionarios.

DICC. DE DATOS		Identificador	Formato	Descripción
Variables	Primarias	Valor_a, Valor_b	Entero	Valores numéricos
	Secundarias	-	-	-
Constantes		-	-	-

DICC. DE RESULTADOS	Identificador	Formato	Descripción
Variables Primarias	Suma	Entero	Valor numérico que representa la suma de los valores de entrada
Constantes	-	-	-

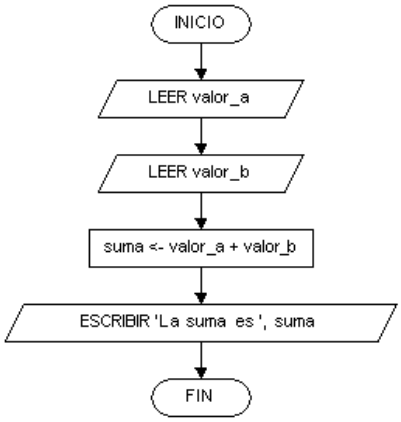
CONDICIONES VINCULANTES	
#	**Descripción**
1	Lectura de los valores a y b
2	Suma aritmética de los valores a y b
3	Presentación del resultado

El siguiente paso es el diseño del algoritmo que representa la solución del problema en un lenguaje algorítmico. En el ejemplo puede observarse el uso de las instrucciones LEER, ESCRIBIR y ASIGNAR (←); así como los símbolos de diagrama de flujo correspondientes a cada instrucción.

Pseudocódigo

```
INICIO
    LEER valor_a
    LEER valor_b
    suma ← valor_a + valor_b
    ESCRIBIR 'La suma es:', suma
FIN
```

Diagrama de Flujo



Selectivas, Condicionales o de Decisión

En general, los problemas que se resuelven utilizando computadoras requieren más que la ejecución de instrucciones una a continuación de la otra. Es por ello, que es prácticamente imposible que las instrucciones sean secuenciales puras. Es necesario tomar decisiones en función de los datos del problema.

Las estructuras selectivas se utilizan para tomar decisiones lógicas. Estas estructuras evalúan una condición y en función del resultado de ésta se realiza una acción u otra. Una *condición* es una expresión lógica que al evaluarse devuelve un valor lógico VERDADERO o FALSO para tomar la decisión.

Las estructuras selectivas pueden ser:

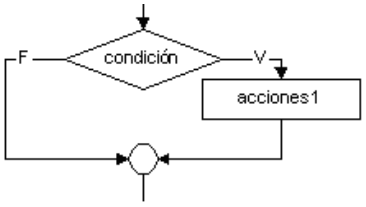
- Simples
- Dobles
- Múltiples

Las estructuras *selectivas simples* (SI-ENTONCES-FIN_SI) permiten realizar un conjunto de acciones si la condición que se evalúa es VERDADERA, caso contrario, dichas acciones no se realizan. La representación en pseudocódigo y diagrama de flujo de una estructura selectiva simple se presenta a continuación:

Pseudocódigo

```
SI condición ENTONCES
    acciones
FIN_SI
```

Diagrama de Flujo

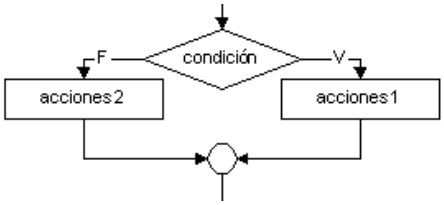


Las estructuras *selectivas dobles* (SI-ENTONCES-SINO-FIN_SI) presentan 2 caminos alternativos de acción, los que se eligen de acuerdo al valor de una determinada condición (VERDADERA o FALSA). La representación en pseudocódigo y diagrama de flujo de una estructura selectiva doble se presenta a continuación:

Pseudocódigo

```
SI condición ENTONCES
    acciones_1
SINO
    acciones_2
FIN_SI
```

Diagrama de Flujo



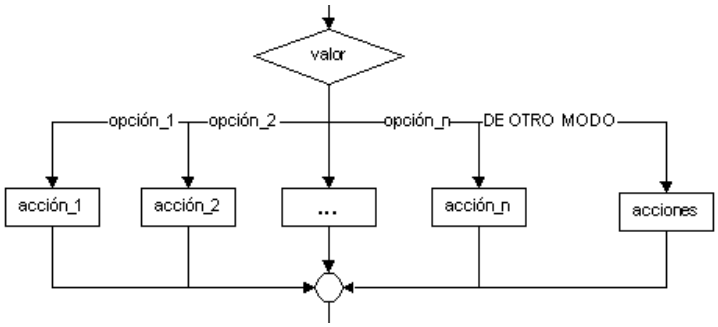
Las estructuras *selectivas múltiples* (SEGÚN-HACER-FIN_SEGÚN) permiten seleccionar entre más de 2 caminos alternativos de ejecución. Estas estructuras evalúan una expresión que puede tomar *n* valores distintos, y según se elija uno de estos valores se realizará una de las *n* acciones posibles. La representación en pseudocódigo y diagrama de flujo

de una estructura selectiva múltiple se presenta a continuación:

Pseudocódigo

```
SEGÚN valor HACER
        opción_1: acciones_1
        opción_2: acciones_2
        . . .
        opción_n: acciones_n
DE OTRO MODO
        acciones
FIN_SEGUN
```

Diagrama de Flujo



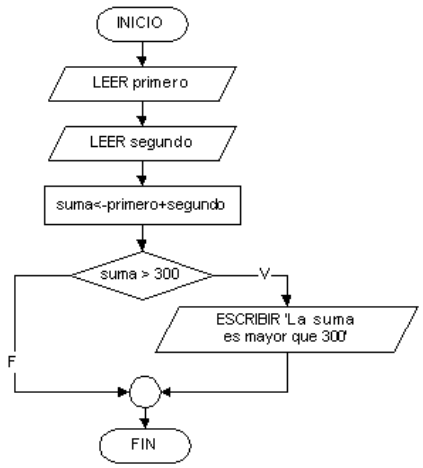
A continuación se presentan 3 algoritmos que ejemplifican el uso de las instrucciones selectivas en la resolución de problemas simples.

Ejemplo 1: Diseñe un algoritmo que determine si la suma de 2 valores ingresados por el usuario es mayor a 300.

Pseudocódigo

```
INICIO
    LEER primero
    LEER segundo
    suma←primero + segundo
    SI suma > 300 ENTONCES
        ESCRIBIR 'La suma es mayor a 300'
    FIN_SI
FIN
```

Diagrama de Flujo

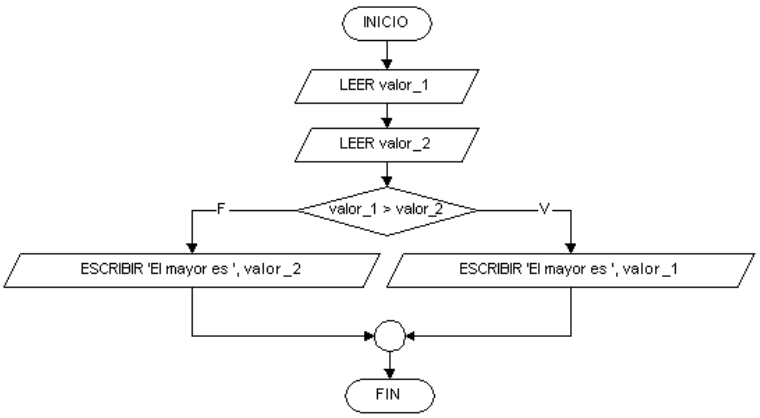


Ejemplo 2: Diseñe un algoritmo que determine cuál es el mayor de dos valores ingresados por el usuario.

Pseudocódigo

```
INICIO
    LEER valor_1
    LEER valor_2
    SI valor_1 > valor_2 ENTONCES
        ESCRIBIR 'El mayor valor es:', valor_1
    SINO
        ESCRIBIR 'El mayor valor es:', valor_2
    FIN_SI
FIN
```

Diagrama de Flujo



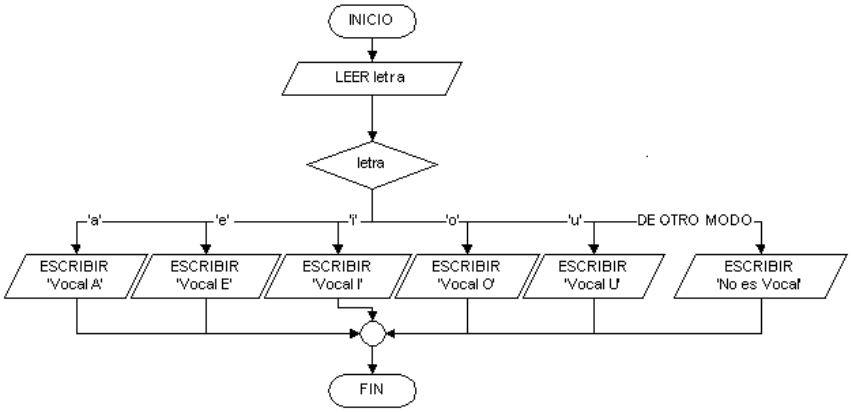
Ejemplo 3: Diseñe un algoritmo que muestre el nombre de la vocal ingresada por el usuario. Para ello, considere lo siguiente:

- Si la vocal es '*a*' mostrar *VOCAL A*
- Si la vocal es '*e*' mostrar *VOCAL E*
- Si la vocal es '*i*' mostrar *VOCAL I*
- Si la vocal es '*o*' mostrar *VOCAL O*
- Si la vocal es '*u*' mostrar *VOCAL U*
- Para cualquier otra letra o símbolo, mostrar *NO ES VOCAL*

Pseudocódigo

```
INICIO
    LEER letra
    SEGÚN letra HACER
        'a': ESCRIBIR 'VOCAL A
        'e': Escribir 'VOCAL E'
        'i': Escribir 'VOCAL I'
        'o': Escribir 'VOCAL O'
        'u': Escribir 'VOCAL U'
    DE OTRO MODO
        Escribir 'NO ES VOCAL'
    FIN_SEGÚN
FIN
```

Diagrama de Flujo



Repetitivas e Iterativas

Las computadoras están especialmente diseñadas para todas aquellas aplicaciones en las que una operación o conjunto de ellas deben repetirse muchas veces.

Las estructuras que permiten repetir una secuencia de instrucciones se denominan bucles, y se llama iteración al hecho de repetir la ejecución de una secuencia de acciones.

Las sentencias que permiten especificar la repetición de un conjunto de instrucciones son:

- Para (PARA-FIN_PARA)
- Mientras (MIENTRAS-FIN_MIENTRAS)
- Repetir (REPETIR-HASTA_QUE)

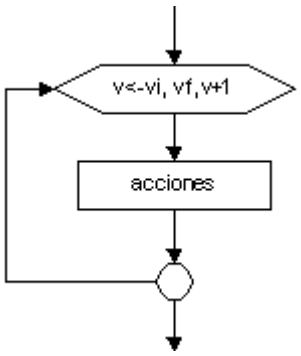
La estructura *iterativa PARA* permite repetir un conjunto de acciones un número conocido de veces. Esta estructura comienza con un *valor inicial* (vi) de la variable de control o índice y las acciones especificadas en el cuerpo del bucle se ejecutan hasta que el índice alcanza el *valor final* (vf). Por defecto, el incremento o decremento del índice se realiza en una unidad, aunque puede especificarse otro valor (PASO n). La variable de control debe ser tipo ordinal, y por lo general se la designa i, j, k.

La representación en pseudocódigo y diagrama de flujo de una estructura iterativa PARA se presenta a continuación:

Pseudocódigo

```
PARA v DESDE vi HASTA vf HACER CON PASO n
        acciones
FIN_PARA
```

Diagrama de Flujo



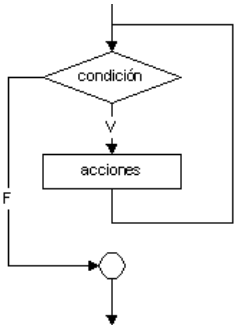
La estructura *iterativa MIENTRAS* permite repetir un conjunto de acciones en tanto la condición evaluada sea VERDADERA, no conociéndose de antemano el número de repeticiones. Esta estructura se clasifica en pre-condicional, ya que evalúa la condición y, si es VERDADERA, entonces ejecuta el bloque de acciones; por cuanto el bloque se puede ejecutar 0, 1 o más veces. En general, esta estructura se utiliza para cálculos aritméticos.

La representación en pseudocódigo y diagrama de flujo de una estructura iterativa MIENTRAS se presenta a continuación:

Pseudocódigo

```
MIENTRAS condición HACER
      acciones
FIN_MIENTRAS
```

Diagrama de Flujo



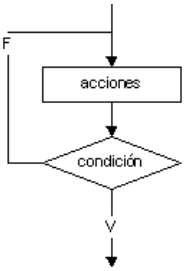
La estructura *iterativa REPETIR* permite repetir un conjunto de acciones en tanto la condición evaluada sea FALSA, no se conoce de antemano el número de repeticiones. Esta estructura se clasifica en pos-condicional, ya que primero ejecuta el bloque de acciones y luego se evalúa la condición; si ésta es FALSA, el bloque de acciones se ejecuta nuevamente. A diferencia de las estructuras pre-condicionales, el bloque de acciones se puede ejecutar 1 o más veces. En general, esta estructura se utiliza para el ingreso de datos.

La representación en pseudocódigo y diagrama de flujo de una estructura iterativa REPETIR se presenta a continuación:

Pseudocódigo

```
REPETIR
      acciones
HASTA_QUE condición
```

Diagrama de Flujo



<u>Diferencias entre las estructuras MIENTRAS y REPETIR</u>

- La estructura *mientras* finaliza cuando la condición es FALSA, en tanto que *repetir* termina cuando la condición es VERDADERA.
- En la estructura *repetir* el cuerpo del bucle se ejecuta al menos una vez, por el contrario, *mientras* es más general y permite la posibilidad de no ejecutar el bucle.

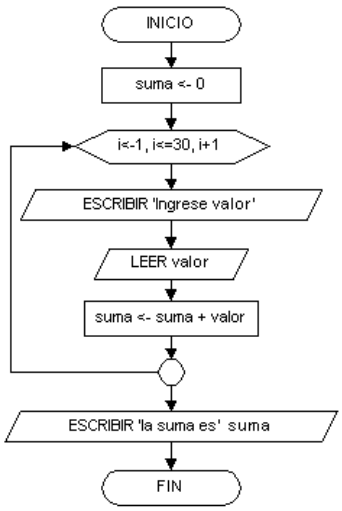
<u>Ejemplos</u>: A continuación se presentan 3 algoritmos que ejemplifican el uso de las instrucciones repetitivas en la resolución de problemas simples.

Ejemplo 1: Diseñe un algoritmo que sume 30 valores ingresados por el usuario.

Pseudocódigo

```
INICIO
      suma <- 0
      PARA i DESDE 1 HASTA 30 HACER
            ESCRIBIR 'ingrese valor:'
            LEER valor
            suma <- suma + valor
      FIN_PARA
      ESCRIBIR 'la suma es:' suma
FIN
```

Diagrama de Flujo

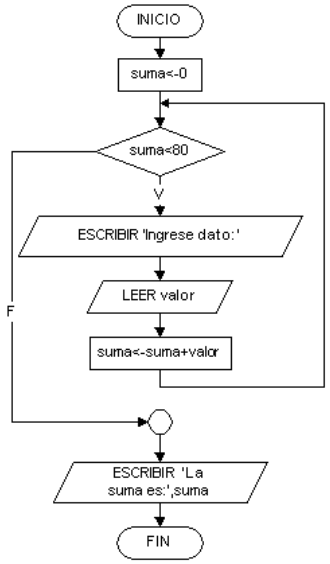


Ejemplo 2: Diseñe un algoritmo que calcule la suma de valores ingresados por el usuario en tanto la suma sea menor a 80.

Pseudocódigo

```
INICIO
    suma←0
    MIENTRAS suma < 80 HACER
        ESCRIBIR 'Ingrese dato:'
        LEER valor
        suma←suma+valor
    FIN_MIENTRAS
    ESCRIBIR 'La suma es:', suma
FIN
```

Diagrama de Flujo

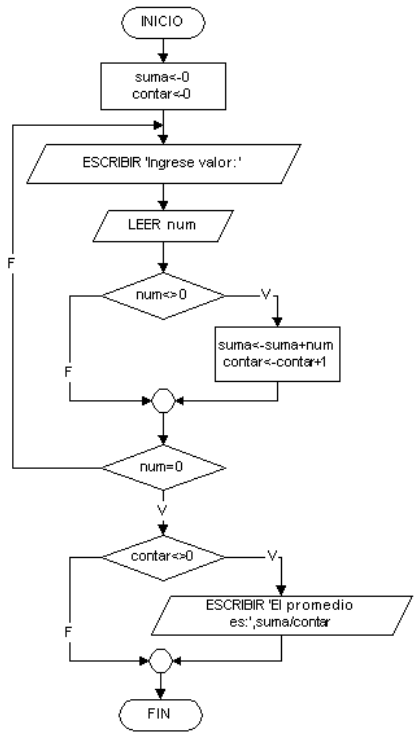


Ejemplo 3: Diseñe un algoritmo que calcule el promedio de valores introducidos por el usuario. El ingreso finaliza cuando el usuario introduce un valor cero.

Pseudocódigo

```
INICIO
    suma←0
    contar<-0
    REPETIR
        ESCRIBIR 'Ingrese valor:'
        LEER num
        SI num <> 0 ENTONCES
            suma←suma+num
            contar←contar+1
        FIN_SI
    HASTA_QUE num=0
    SI contar<> 0 ENTONCES
        ESCRIBIR 'El promedio es:' suma/contar
    FIN_SI
FIN
```

Diagrama de Flujo



Bucles Infinitos

Algunos bucles no exigen fin, sin embargo otros no encuentran fin por errores en su diseño. Un bucle que nunca termina se denomina bucle infinito o sin fin. Los bucles sin fin no intencionados son perjudiciales para la programación y siempre deben evitarse.

Finalización de Bucles

Si un algoritmo o programa ejecuta un bucle cuyas iteraciones dependen de una condición, este bucle puede estar controlado por un valor centinela, una bandera o un contador.

Un *bucle controlado por centinela* es aquel en el que un valor que se ingresa en el bucle permite finalizar las iteraciones. Este valor especial se denomina valor centinela.

Un *bucle controlado por bandera* es aquel en el que una variable lógica permite determinar la ocurrencia de un evento y de este modo finalizar o no las iteraciones del bucle. Esta variable lógica se denomina bandera.

Un *bucle controlado por contador* es aquel en el que una variable se utiliza para contabilizar la cantidad de veces que ocurre algún evento y de este modo finalizar o no las iteraciones del bucle. Esta variable se denomina contador.

Anidamiento de Estructuras de Control

Para diseñar algoritmos que solucionen diversos tipos problemas mediante programas de computadora, la Programación Estructurada permite al diseñador combinar las estructuras de control básicas de una manera flexible. En muchos casos es necesario colocar una estructura de control dentro de otra, a esta forma de combinar las estructuras se denomina anidamiento. El anidamiento debe cumplir con las siguientes reglas de construcción:

- la estructura interna debe quedar completamente incluida dentro de la externa, y
- no puede existir solapamiento.

Los siguientes ejemplos muestran anidamientos válidos de sentencias selectivas.

a)
```
SI condición_1 ENTONCES
      SI condición_2 ENTONCES
            acciones
      FIN_SI
FIN_SI
```

b)
```
SI condición_1 ENTONCES
      SI condición_2 ENTONCES
            acciones
      FIN_SI
ELSE
      acciones
FIN_SI
```

c)
```
SI condición_1 ENTONCES
      SI condición_2 ENTONCES
            acciones
      ELSE
            acciones
      FIN_SI
ELSE
      SI condición_3 ENTONCES
            acciones
      FIN_SI
FIN_SI
```

d)
```
SI condición_1 ENTONCES
      SI condición_2 ENTONCES
            acciones
      ELSE
            acciones
      FIN_SI
ELSE
      SI condición_3 ENTONCES
            acciones
      ELSE
            acciones
      FIN_SI
FIN_SI
```

Obsérvese que para una mejor comprensión y lectura del algoritmo se utilizaron sangrías que permiten diferenciar las estructuras anidadas. Esta es una práctica recomendable al momento de escribir un programa.

Los siguientes ejemplos muestran anidamientos válidos de sentencias repetitivas.

a)
```
MIENTRAS condición_1 HACER
      MIENTRAS condición_2 HACER
            acciones
      FIN_MIENTRAS
FIN_MIENTRAS
```

b)
```
REPETIR
      REPETIR
            acciones
      HASTA_QUE condición_2
HASTA_QUE condición_1
```

c)
```
PARA i DESDE vi HASTA vf HACER
      PARA k DESDE vi_2 HASTA vf_2 HACER
            acciones
      FIN_PARA
FIN_PARA
```

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

e)

```
PARA i DESDE vi HASTA vf HACER
        REPETIR
                acciones
        HASTA_QUE condición_1
FIN_PARA
```

f)

```
REPETIR
        MIENTRAS condición_2 HACER
                acciones
        FIN_MIENTRAS
HASTA_QUE condición_1
```

En los bucles anidados, el bucle interno ejecuta todas sus repeticiones para cada una de las iteraciones del bucle externo.

Los siguientes ejemplos muestran anidamientos inválidos de sentencias selectivas y repetitivas.

a)

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

b)

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

c)

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

d)

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

En cada ejemplo se destacan en negritas las estructuras que están incorrectamente anidadas.

Estructura Básica de Programa

Cada vez que se escribe un programa se debe seguir en detalle ciertas reglas de estructura. La estructura básica de un programa es:

- Nombre del programa: es el nombre que se utilizará para designar al programa, este nombre debe estar relacionado con las acciones que realizará el programa. En pseudocódigo, antes del nombre del programa se escribe la palabra *programa*; en los lenguajes de programación se utiliza la palabra *program*. Por ejemplo: programa calcular_superficies, programa sumar_numeros, etc.
- Declaración de variables y constantes: en esta sección se definirán con nombre y tipo, las constantes y variables que se utilizarán dentro del programa.
- Inicio del programa: siempre se debe indicar el punto en el que comienzan las acciones correspondientes al programa. En pseudocódigo se denota con la palabra *inicio* y en los lenguajes de programación por lo general con la palabra *begin*.
- Cuerpo del programa: en esta parte están contenidos todos los procesos y acciones que realizará el programa.
- Fin del programa: así como se indica el inicio de las acciones del programa, también se debe dejar marcado el final de estas acciones, en pseudocódigo esto hace con la palabra *fin* y en lenguajes de programación con la palabra *end*.

Elementos Básicos de Programa

Los lenguajes de programación tienen elementos básicos que se utilizan como bloques constructivos, y reglas que permiten combinar tales elementos. Estas reglas se denominan sintaxis del lenguaje. Sólo las instrucciones sintácticamente correctas podrán ser interpretadas por la computadora.

Los elementos básicos constitutivos de un programa o algoritmo son:

- Palabras reservadas: son aquellas que tienen un significado especial y permiten indicar partes de un programa o instrucciones. Por ejemplo, las palabras INICIO y FIN delimitan el alcance del ámbito de un programa.
- Identificadores: los identificadores son los nombres con los que se distingue a las variables. Por ejemplo, 2

variables de tipo entero pueden identificarse como *suma* y *producto*.

- Caracteres especiales: los caracteres especiales son aquellos símbolos que tienen una significación propia para el lenguaje.
- Constantes: las constantes son objetos de datos que no varían su valor durante la ejecución de un programa.
- Variables: las variables son objetos de datos cuyo valor se modifica durante la ejecución de un programa.
- Expresiones: las expresiones combinan operandos (variables, constantes, funciones) y operadores (+, -, *, /).
- Instrucciones: las instrucciones son las operaciones que puede realizar la computadora.

Además existen otros elementos que forman parte de los programas, éstos son: bucles, contadores, acumuladores, interruptores y estructuras de control.

Un *bucle* o *lazo* es un segmento de un algoritmo o programa, cuyas instrucciones se repiten un número determinado de veces o mientras se cumpla una determinada condición. Una condición puede ser VERDADERA o FALSA y se comprueba en cada iteración del bucle (total de instrucciones que se repiten en el bucle). Un bucle consta de tres partes: decisión, cuerpo del bucle, salida del bucle.

Los *contadores*, en general, se utilizan para controlar las iteraciones de un bucle. Un *contador* es una variable cuyo valor se incrementa o decrementa en una cantidad constante en cada iteración. El contador puede ser positivo o negativo.

Un *acumulador* es una variable cuya misión es almacenar cantidades variables resultantes de sumas o productos sucesivos. Realiza la misma función que un contador, con la diferencia que el incremento o decremento es variable en lugar de constante.

Un *interruptor* o *bandera* es, en general, una variable lógica que puede tomar valor VERDADERO o FALSO a lo largo de la ejecución de un programa y que permite comunicar información de una parte a otra del mismo.

Las *estructuras de control* secuenciales, selectivas y repetitivas ya fueron explicadas en detalle.

Comprobación de Algoritmos: Prueba de Escritorio

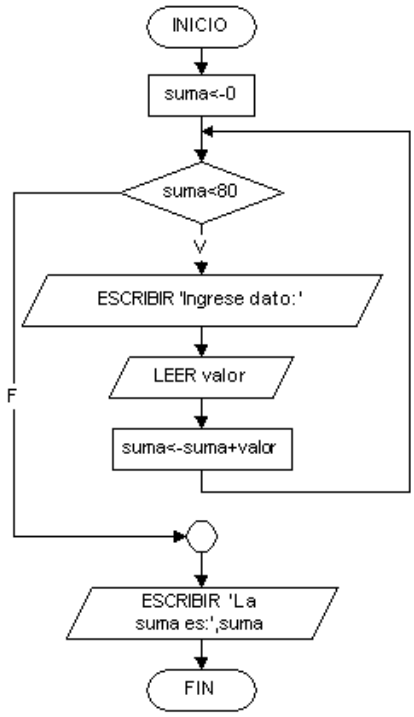
La *Prueba de Escritorio* es una herramienta útil para comprobar que un algoritmo realiza la tarea para la que fue diseñado. Esta prueba consiste en "ejecutar a mano" el algoritmo propuesto como solución del problema planteado. Para esto deben utilizarse datos representativos y anotarse los valores que toman las variables en cada paso. Es recomendable dar diferentes datos de entrada y considerar todos los posibles casos, aún los de excepción o no esperados, para asegurar que el programa codificado a partir del algoritmo no produzca errores en tiempo de ejecución al recibir tales entradas.

De este modo la prueba de escritorio permite comprobar, en tiempo de diseño, si el algoritmo es correcto o si es necesario realizar ajustes.

A continuación se muestra un ejemplo de prueba de escritorio del siguiente algoritmo:

Ejemplo: Diseñe un algoritmo que calcule la suma de valores ingresados por el usuario en tanto la suma sea menor a 80.

```
INICIO
    suma←0
    MIENTRAS suma < 80 HACER
        ESCRIBIR 'Ingrese dato:'
        LEER valor
        suma←suma+valor
    FIN_MIENTRAS
    ESCRIBIR 'La suma es:', suma
FIN
```



	Variables y Condiciones		
Pasos	suma	suma<80	valor
01	0	-	-
02		VERDADERO	-
03			30
04	30		
05		VERDADERO	
06			14
07	44		
08		VERDADERO	
09			25
10	69		
11		VERDADERO	
12			21
13	90		
14		FALSO	
15	LA SUMA ES: 90		

En la tabla anterior puede observarse cómo las *variables* (suma, valor) y la *condición* (suma<80) se van modificando al ejecutar, paso a paso, las instrucciones del algoritmo. Nótese además, que el número de repeticiones del bucle (que realiza la suma) depende de los datos ingresados por el usuario, finalizando éste en el momento en que la variable suma alcanza o supera el valor 80.