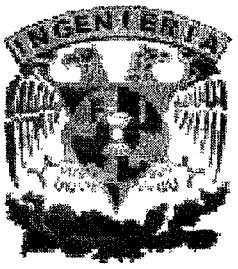
Algoritmos numéricos

Jesús Javier
Cortés Rosas

Miguel Eduardo
González Cárdenas

Víctor Damián
Pinilla Morán



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Algoritmos Numéricos

Jesús Javier Cortés Rosas

Miguel Eduardo González Cárdenas

Víctor Damián Pinilla Morán

P R E S E N T A C I Ó N

La Facultad de Ingeniería ha decidido realizar una serie de ediciones provisionales de obras recientemente elaboradas por académicos de la institución, como material de apoyo para sus clases, de manera que puedan ser aprovechadas de inmediato por alumnos y profesores. Tal es el caso de la obra *Algoritmos numéricos*, elaborada por los profesores Jesús Javier Cortés Rosas, Miguel Eduardo González Cárdenas y Víctor Damián Pinilla Morán.

Se invita a los estudiantes y profesores a que comuniquen a los autores las observaciones y sugerencias que mejoren el contenido de la obra, con el fin de que se incorporen en una futura edición definitiva.

INTRODUCCIÓN

Este material didáctico está integrado por un conjunto de algoritmos numéricos representados mediante diagramas de flujo. Algunos algoritmos se acompañan de una serie de comentarios para facilitarle al estudiante su aprendizaje y el entendimiento de los mismos. Para tal efecto, se inicia la presentación con algoritmos sencillos y conforme se avanza en su estudio se incrementa el grado de dificultad.

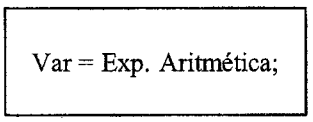
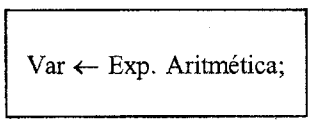
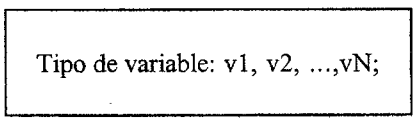
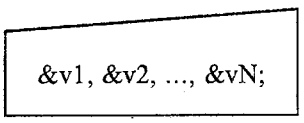
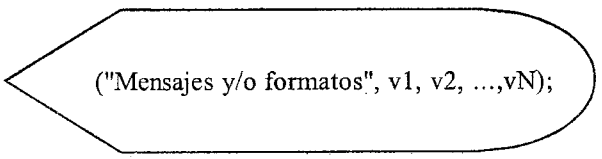
Otra característica de los algoritmos aquí presentados radica en darles un enfoque hacia el lenguaje de programación C, sin perder la generalidad; característica intrínseca de un diagrama de flujo; es decir, se utiliza parte de la sintaxis de este lenguaje de programación.

Cabe mencionar que si bien este trabajo no es la panacea para el estudiante, sí representa un apoyo fundamental para su aprendizaje, sobre todo para el iniciado en la programación, debido a que existen pocos libros que tratan a detalle el tema de algoritmos.

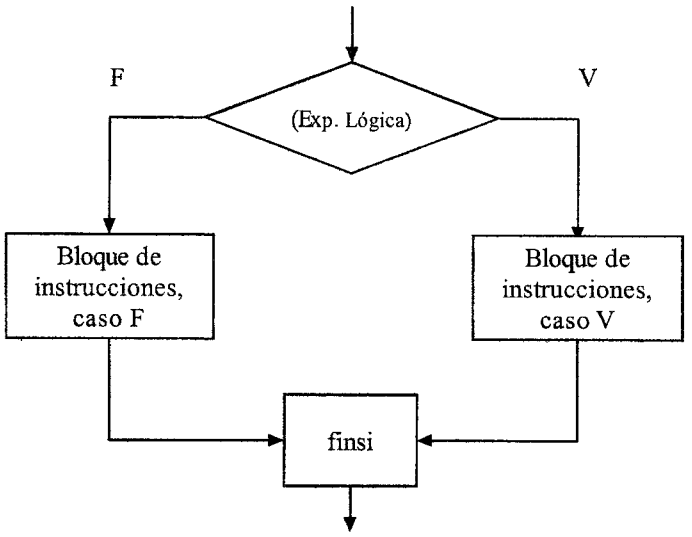
Los autores estamos conscientes de que todo es perfectible y este trabajo también lo es, por lo que agradeceremos nos hagan llegar cualquier comentario u observación, personalmente o al correo electrónico: miguele@cancun.fi-a.unam.mx

Agradecemos a *David Francisco Jiménez Román* su colaboración en la edición de este material y por sus observaciones que lo enriquecieron en gran medida.

SIMBOLOGÍA UTILIZADA EN DIAGRAMAS DE FLUJO

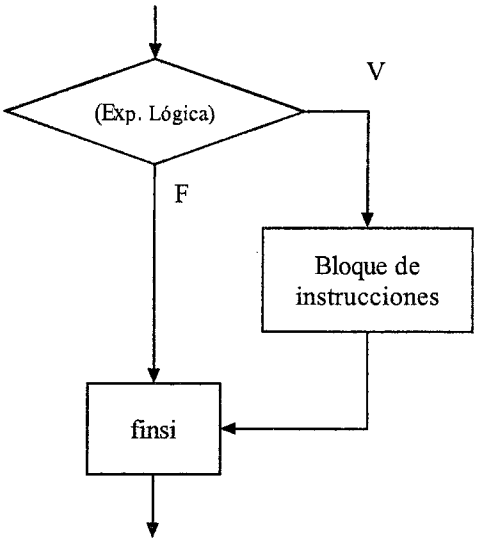
SÍMBOLO	PSEUDOCÓDIGO
	Inicio
	Fin
 **ó** 	Var = Expresión Aritmética; **(INSTRUCCIÓN DE ASIGNACIÓN)**
	Tipo de Variable: v1, v2, ..., vN; **(DECLARACIÓN DE VARIABLES)**
	Lee: &v1, &v2, ..., &vN; **(INSTRUCCIÓN DE ENTRADA DE DATOS)**
	Desplegar("**Mensaje**", v1, v2, ..., vN); **y/o formato** **(INSTRUCCIÓN DE SALIDA DE DATOS)**

SÍMBOLO

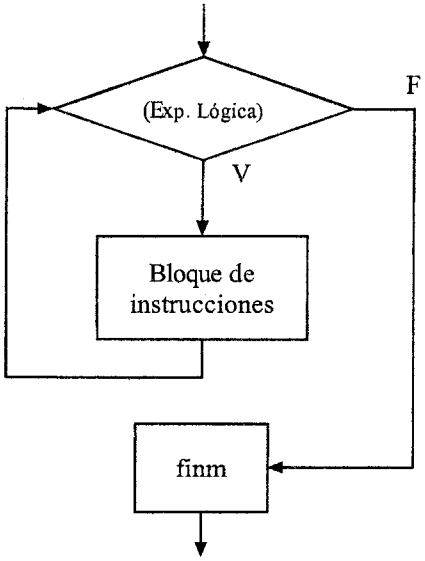


PSEUDOCÓDIGO

```
Si (Expresión Lógica) entonces
{
        Bloque de intrucciones
        /* caso verdadero */
}

caso contrario
{
        Bloque de instrucciones
        /* caso falso */
}
Finsi
```

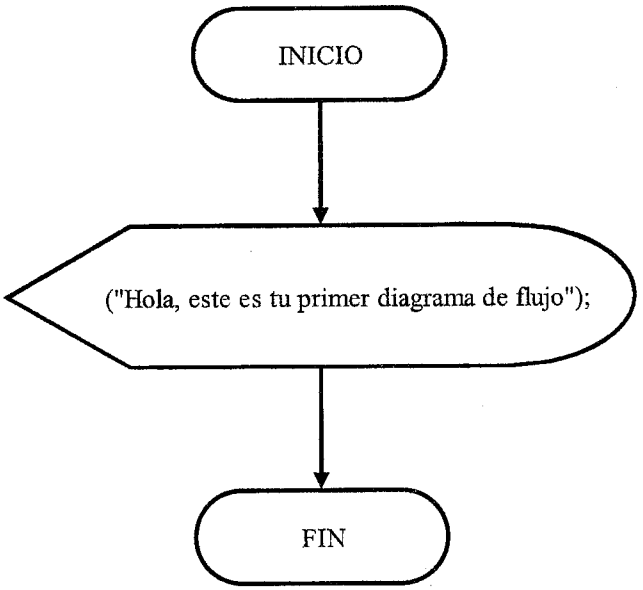



```
Si (Expresión Lógica) entonces
{

        Bloque de instrucciones
        /* caso verdadero */


}
Finsi
```




```
Mientras (Expresión Lógica)
{


        Bloque de instrucciones
        /* caso verdadero */


}
Finm
```

Elaborar un diagrama de flujo para desplegar en pantalla el mensaje: Hola, este es tu primer diagrama de flujo.



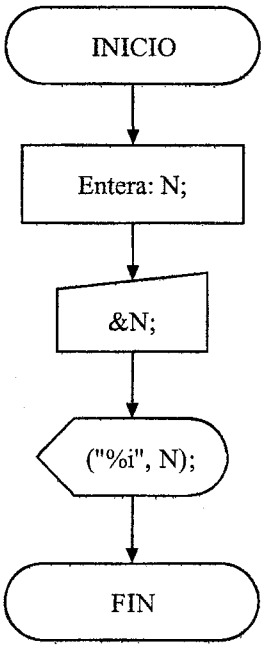
Inicio del programa

Se despliega el mensaje:
Hola, este es tu primer diagrama de flujo en la pantalla

Fin del programa

NOTA: En el lenguaje de programación C, cualquier comentario debe ir encerrado entre comillas, por lo tanto las comillas al inicio y al final del comentario no se despliegan.

Elaborar un diagrama de flujo que lea un entero N cualquiera y lo despliegue en la pantalla.



Inicio del programa

Declaración de la variable N de tipo entera

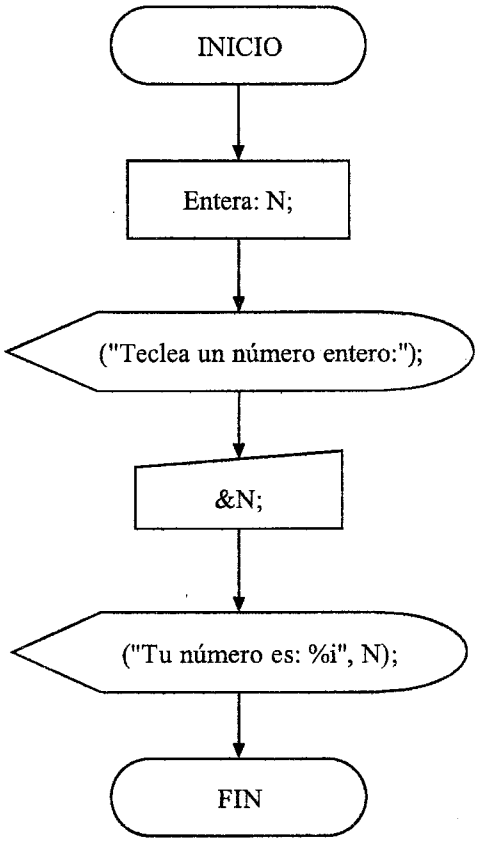
El programa espera un valor; y éste será leido y guardado en la variable N

Se despliega en pantalla el valor de N

Fin del programa

NOTA: En el lenguaje de programación C para desplegar el contenido de una variable se utiliza un formato que sea del mismo tipo que la variable utilizada. En este caso, %i es el formato - para una variable de tipo entera.

Elaborar un algoritmo que lea un entero N y lo despliegue en pantalla, utilizando los mensajes: Teclea un número entero: y Tu número es:



Inicio del programa

Declaración de N como variable de tipo entero

Mensaje para el usuario, solicitándole teclee un número

Se lee y se guarda su valor en N

Se despliega el mensaje: **Tu número es:** , seguido del valor tecleado con anterioridad (variable N)

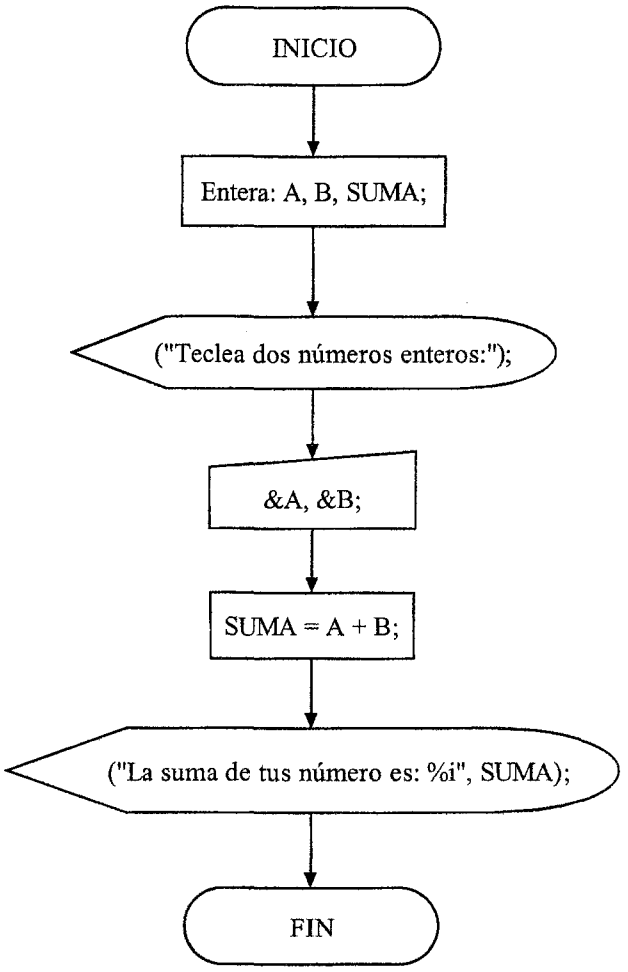
Fin del programa

NOTA: Observe que los mensajes en el diagrama de flujo se encierran entre comillas. En el caso del segundo mensaje: Tu nombre es:, los símbolos %i corresponden a un formato para números enteros y se encuentran después del mensaje con la finalidad de desplegar también el contenido de N (valor tecleado anteriormente)

Ejemplo: Si el usuario teclea 3, entonces este valor se almacena en N y posteriormente aparecerá en pantalla:

Tu número es: 3

Elaborar un algoritmo que solicite al usuario ingresar dos números enteros y después despliegue la suma de ambos valores (utilizar tres variables, una para cada sumando y una para la suma)



Inicio del programa

Declaración de las variables A (para el primer sumando), B (para el segundo sumando) y SUMA (para la suma) todas enteras.

Mensaje que solicita al usuario teclear dos números enteros

El programa espera hasta que sean ingresados los dos números (con su respectivo enter). Entonces se leen y se guardan los -- valores en las variables; el primero en A y el segundo en B

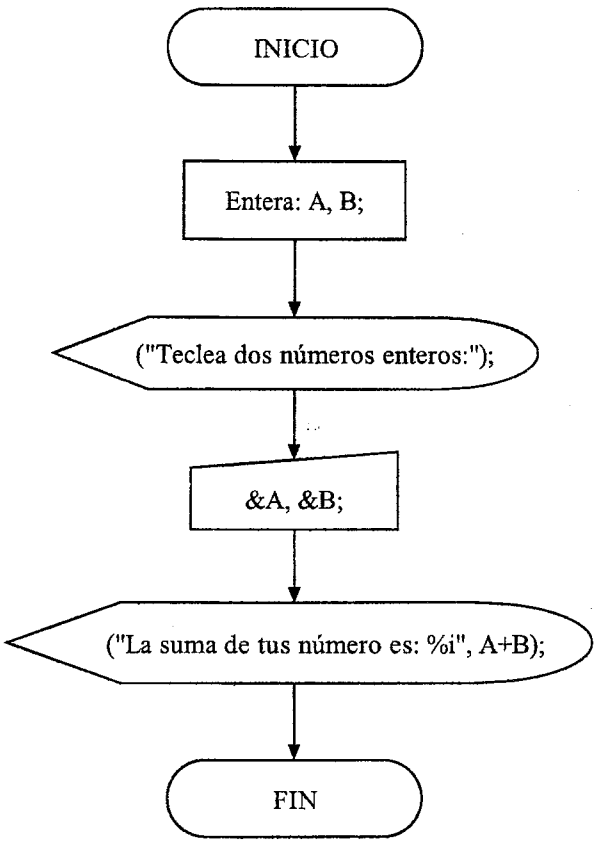
Expresión aritmética que realiza la operación A+B y guarda el resultado en la variable SUMA

Se despliega:
La suma de tus números es: seguido del contenido de SUMA (suma de los dos enteros)

Fin del programa

NOTA: Como se mencionó en un algoritmo anterior los caracteres **%i** no se despliegan en pantalla al colocarse dentro de un mensaje, **%i** es un formato establecido para números enteros que permite desplegar en pantalla un valor referenciado después de cerrar las comillas del mensaje. En este caso, el valor de A + B almacenado en la variable SUMA.

Elaborar un algoritmo que solicite al usuario ingresar dos números enteros y después despliegue la suma de ambos valores (utilizar únicamente dos variables)



Inicio del programa

Declaración de las variables A (para el primer sumando), B (para el segundo sumando), ambas como enteras

Mensaje que solicita al usuario teclear dos números enteros

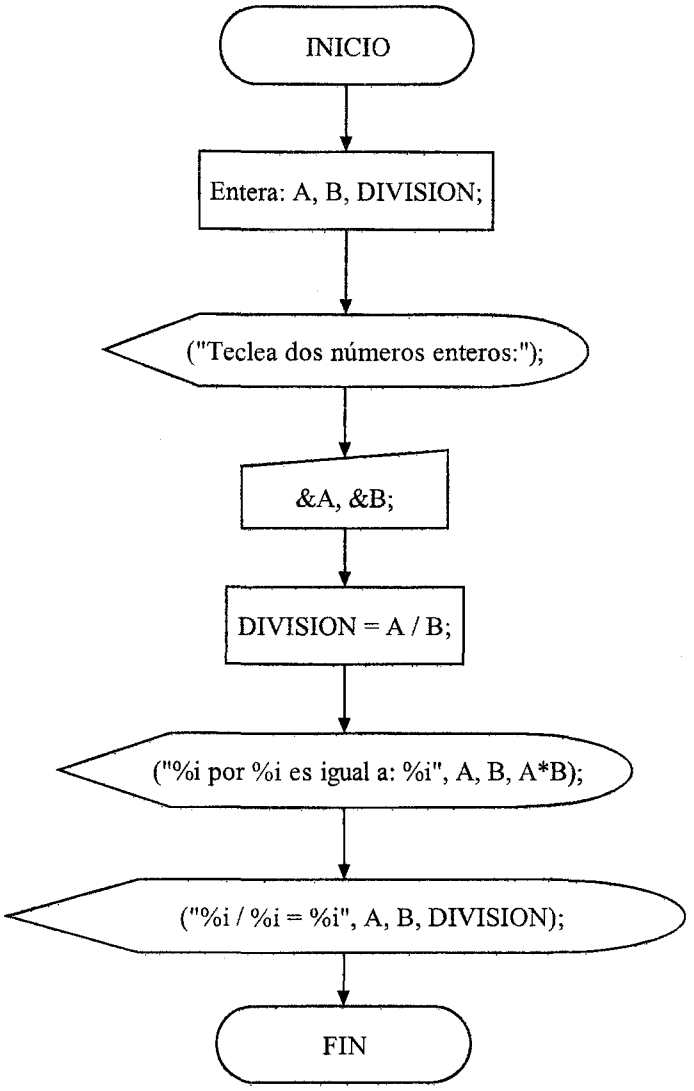
El programa espera hasta que sean ingresados los dos números (con su respectivo enter). Entonces se leen y se guardan los -- valores en las variables; el primero en A y el segundo en B

Se despliega:
La suma de tus números es: seguido del valor obtenido de la suma de los dos enteros

Fin del programa

NOTA: Observe que la utilización del formato **%i** es igual que en el ejemplo anterior. Lo único que cambia es que el valor al que have referencia el formato no está indicado por una sola variable como en el algoritmo anterior, sino que después de efectuarse la suma se despliega directamente el resultado sin necesidad de asignarlo a otra variable. Finalmente, ambos algoritmos realizan exactamente lo mismo.

Elaborar un algoritmo que solicite al usuario dos números enteros cualesquiera y los despliegue con los resultados tanto de su multiplicación como de su división.



Inicio del programa

Declaración de variables: A para el primer operando, B para el segundo y DIVISION para la operación de división, todas de tipo entero.

Mensaje para el usuario, Teclea dos números enteros:

Se leen y se guardan los valores en A y B respectivamente

Expresión aritmética para la operación A entre B. El resultado se guarda en la variable DIVISION.

Se despliega:
___ **por** ___ **es igual a:** ___ , donde en los espacios aparecerán el valor de A, el valor de B y el valor del resultado de la operación A*B, respectivamente.

Se despliega:
___ / ___ = ___, donde en los espacios se mostrarán los valores las variables A, B y DIVISION, respectivamente.

Fin del programa.

NOTA: En este algoritmo se puede apreciar que además de los mensajes pueden utilizarse más de un formato y que los valores contenidos en las variables se mostrarán en pantalla en el orden en que aparece después de cerrar las comillas.

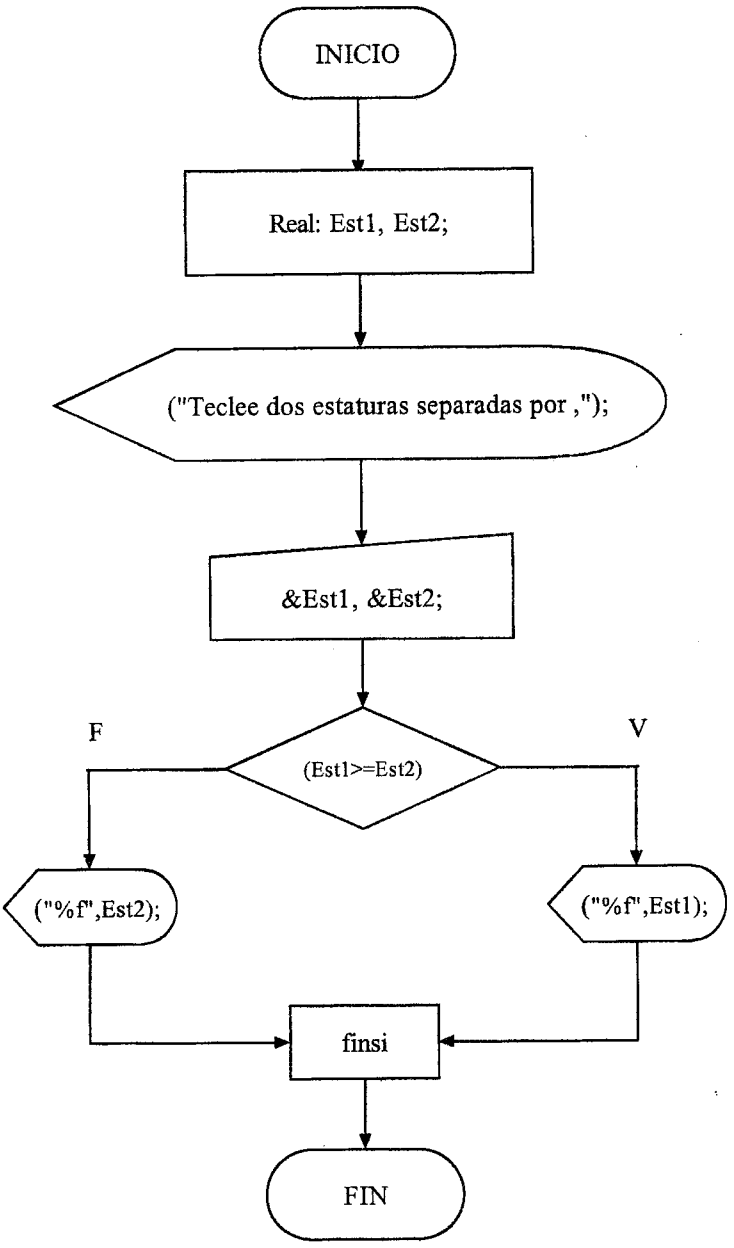
Por ejemplo, si fueron leídos los valores 6 y 3, en ese orden, se tendrá:

6 por 3 es igual a: 18
6 / 3 = 2

Cabe mencionar además que al ser DIVISION una variable de tipo entero, un ejemplo con valores como 5 y 2, respectivamente, generaría:
5 por 2 es igual a: 10
5 / 2 = 2

debido a que la operación es entre dos enteros el resultado 5 / 2 es también entero..

Elaborar el diagrama de flujo para leer dos valores e indicar cual es el mayor de ambos



Inicio del programa

Declaración de las variables Est1 y Est2 como reales o punto flotante

Mensaje para el usuario solicitándole proporcione dos estaturas

Se leen y se guardan los valores en Est1 y Est2 respectivamente

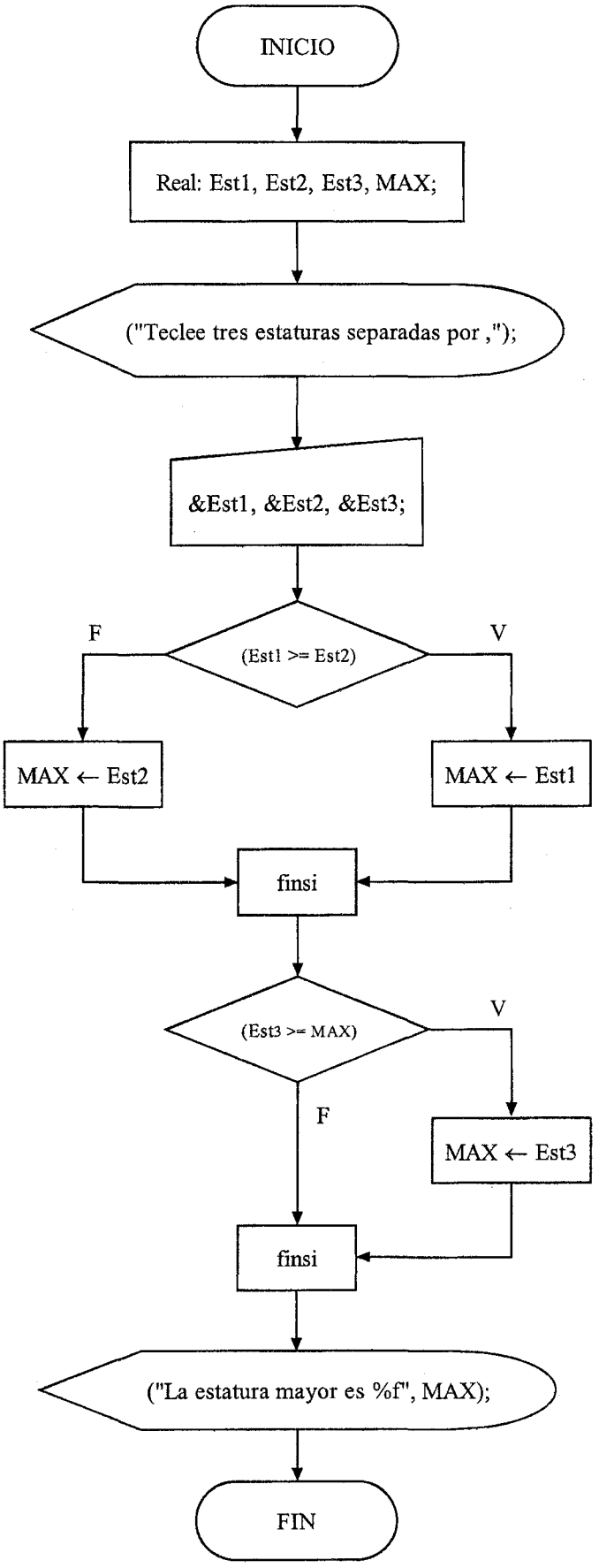
Se compara la estatura 1 con la estatura 2

Se despliega la estatura mayor

Fin de la condición

Fin del programa

Elaborar el diagrama de flujo para leer tres estaturas e indicar cual de las tres es mayor



Inicio del programa

Declaración de las variables Est1, Est2, Est3 (para las estaturas), como reales

Mensaje para el usuario pidiéndole que teclee tres estaturas

Se leen y se guardan los valores en Est1, Est2 y Est3 respectivamente

Se comparan Est1 y Est2

La estatura mayor se guarda en la variable MAX

Fin de esta condicion

Se compara Est3 con la estatura mayor (MAX) de las dos anteriores

Si Est3 es mayor, este valor reemplaza el anterior valor de MAX

Fin de esta condición

Se despliega el comentario **La estatura mayor es** , seguida del valor contenido en MAX

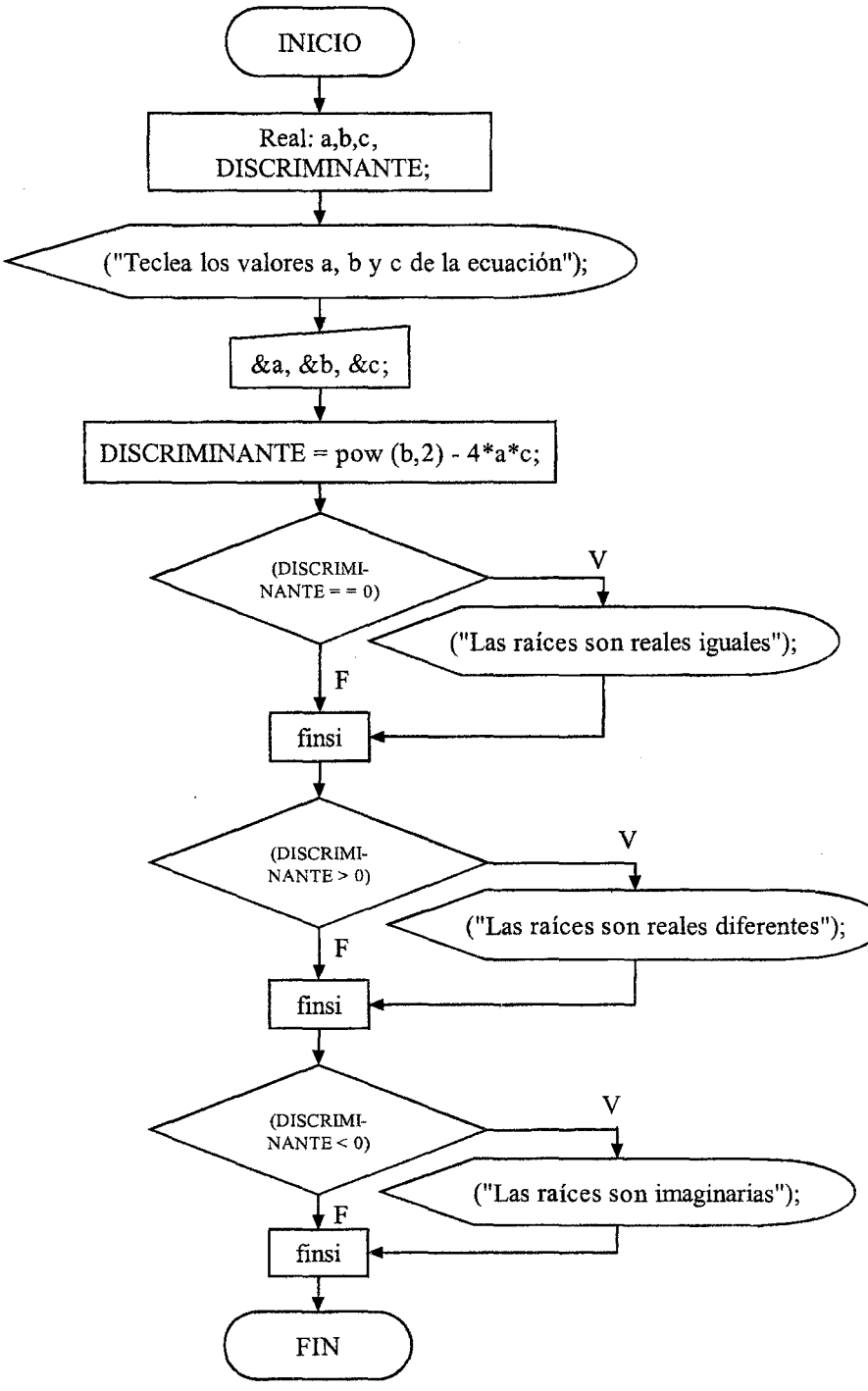
Fin del programa

Elaborar un algoritmo para indicar el tipo de raíces: reales iguales, reales diferentes o imaginarias, - en una ecuación de la forma ax^2+bx+c. Para tal efecto se obtendrán los valores de las raíces utilizan- la fórmula general para una ecuación de segundo grado:

$$x_{1,2} = [- b \pm (b^2 - 4ac)^{1/2}] / 2a$$

$$x_{1,2} = (- b / 2a) \pm (b^2 - 4ac)^{1/2} / 2a$$

Al valor $(b^2 - 4ac)$ se le llama discriminante porque permite identificar (discriminar) el tipo de raíces de las que se trata.
Utilizando una variable llamada DISCRIMINANTE y asignándole el valor de $(b^2 - 4ac)$ puede observar:
Si DISCRIMINANTE = 0 entonces las raíces son reales iguales.
Si DISCRIMINANTE > 0 entonces las raíces son reales diferentes.
Si DISCRIMINANTE < 0 entonces las raíces son imaginarias.
Utilice únicamente estructuras de control de la forma **Si entonces**



Inicio del programa

Declaración de las variables a, b y c (coeficientes de la ecuación) y DISCRIMI --- NANTE, de tipo real

Se despliega un mensaje al usuario

Se leen y se guardan los valores en a, b y c, respectivamente

Se obtiene el valor de DISCRIMINANTE

Si entonces1

Se despliega el mensaje:
Las raíces son reales iguales, cuando el valor de DISCRIMINANTE = 0

Fin del Si entonces1

Si entonces2

Se despliega el mensaje:
Las raíces son reales diferentes, cuando el valor de DISCRIMINANTE > 0

Fin del Si entonces2

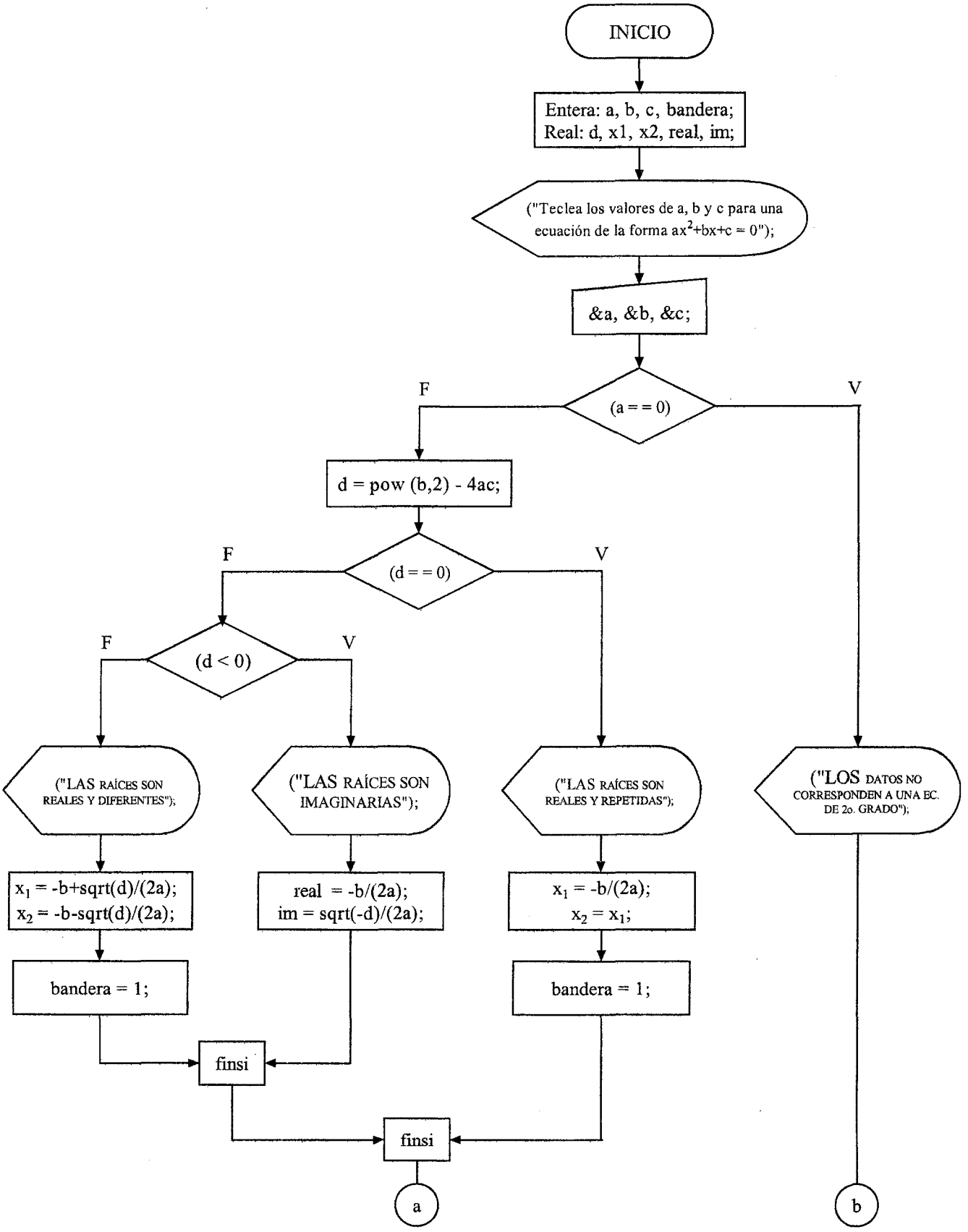
Si entonces3

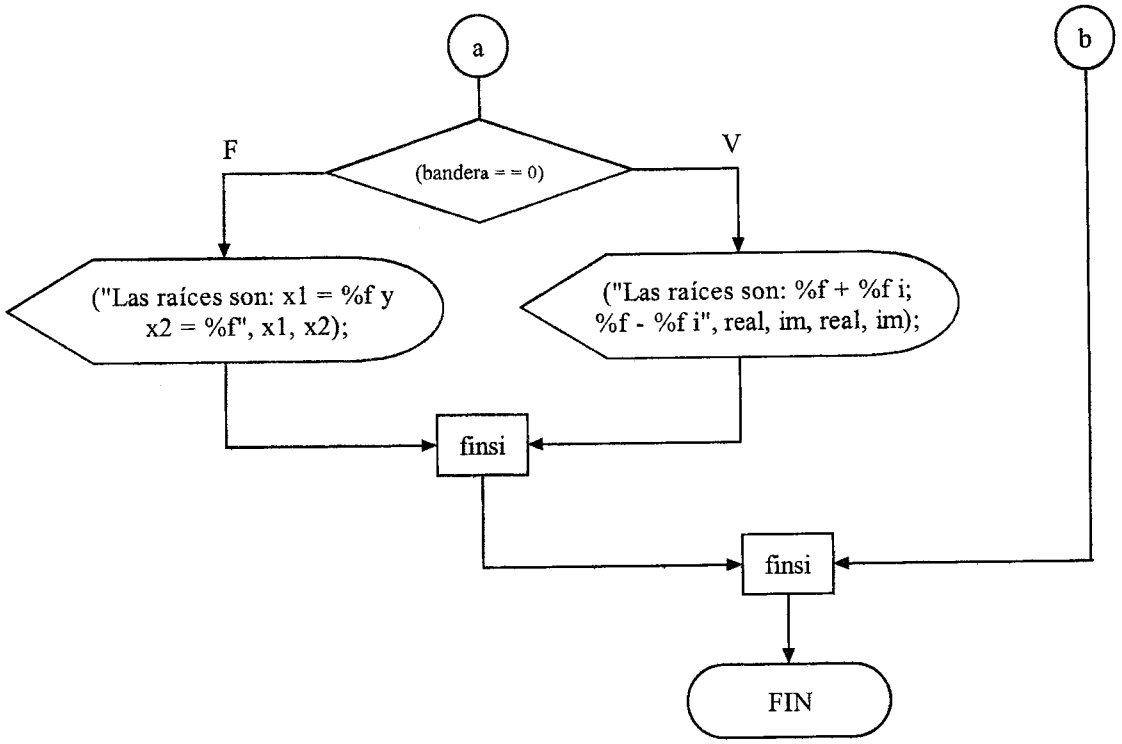
Se despliega el mensaje:
Las raíces son imaginarias, cuando el valor de DISCRIMINANTE < 0

Fin del Si entonces3

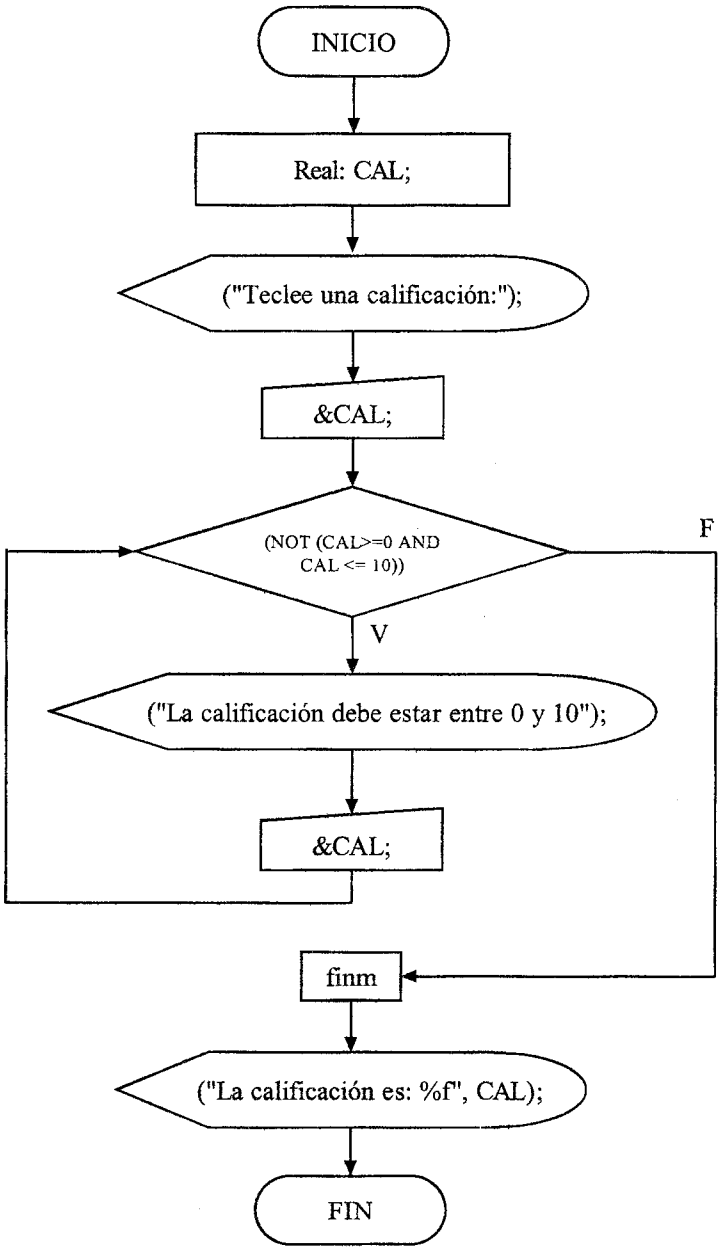
Fin del programa

Elaborar un algoritmo para determinar las raíces de una ecuación de segundo grado de la forma $ax^2+bx+c = 0$




a
b
F
(bandera = = 0)
V
("Las raíces son: x1 = %f y
x2 = %f", x1, x2);
("Las raíces son: %f + %f i;
%f - %f i", real, im, real, im);
finsi
finsi
FIN

Elaborar un diagrama de flujo que solicite una calificación , que tenga un filtro que no permita valores fuera de [0, 10]; y lo despliegue en pantalla.



Inicio del programa

Declaración de variable CAL como real o punto flotante (para la calificación)

Mensaje para el usuario solicitándole teclear una calificación

Se lee y se guarda el valor en la variable CAL

Mientras la calificación no sea mayor o igual a cero y menor o igual a diez; es decir, el valor -- tecleado este fuera del intervalo [0, 10]

Se solicita nuevamente la calificación

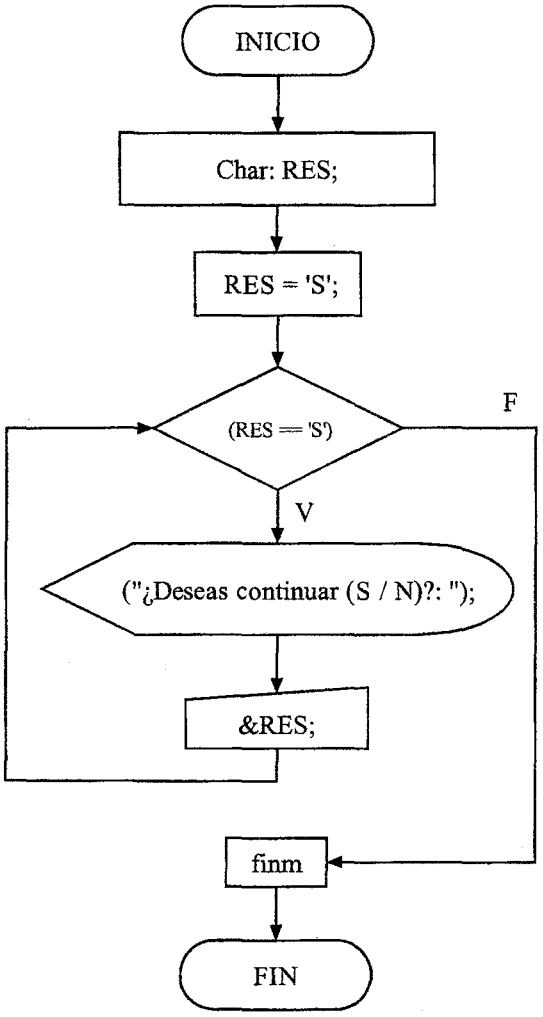
Se lee y se guarda el valor en CAL

Fin del mientras, esto es cuando la calificación está entre 0 y 10

Se despliega el mensaje: **La calificación es:**, seguido de la calificación.

Fin del programa

Elaborar un algoritmo que entre en un ciclo y que permanezca en él mientras el usuario así lo indique con su respuesta.



Inicio del programa

Declaración de la variable RES de tipo carácter para la respuesta del usuario

Se inicializa RES con el valor S (mayúscula)

Ciclo:
Mientras RES sea igual a S

Se pregunta al usuario:
¿Deseas continuar (S / N)?:

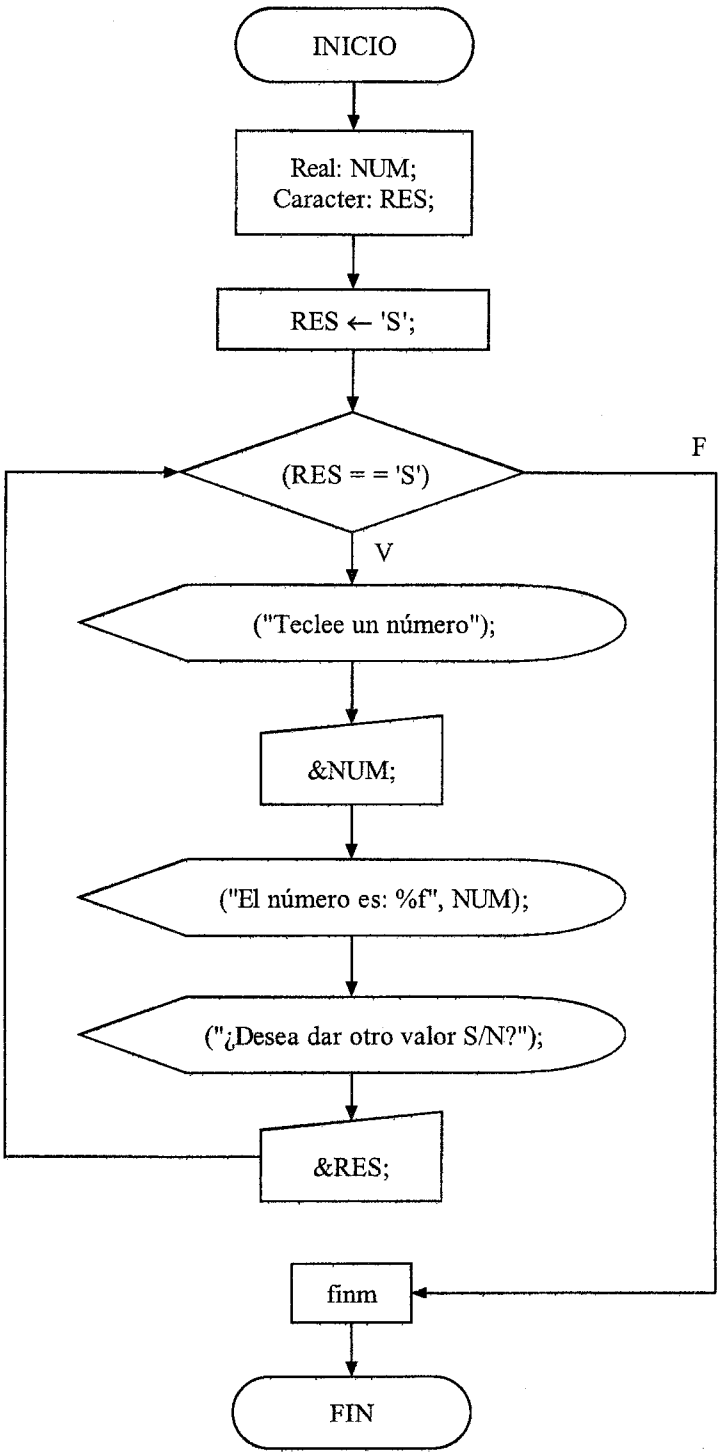
Se lee y se guarda su respuesta en la variable RES
Regresa al ciclo mientras.

Fin del ciclo; cuando RES es diferente del caracter S

Fin del programa

NOTA: Para continuar el proceso dentro del ciclo se tiene que presionar la tecla etiquetada con S (mayúscula), si el usuario presiona s (minúscula) entonces se termina el ciclo while.

Elaborar un diagrama de flujo que solicite un valor, lo despliegue en el monitor y se pregunte al usuario si desea teclear o no otro valor, de tal forma que se solicite un nuevo valor y se despliegue mientras el usuario presione la tecla 'S' como respuesta.



Inicio del programa

Declaración de las variables NUM, de tipo real o punto flotante, y RES de tipo carácter (para la respuesta)

Se asigna el valor S a la variable RES

Mientras variable RES tenga el valor S

Mensaje al usuario: **Teclee un número**

Se lee y se guarda el valor en variable NUM

Se despliega: **El número es:** , seguido del valor contenido en NUM

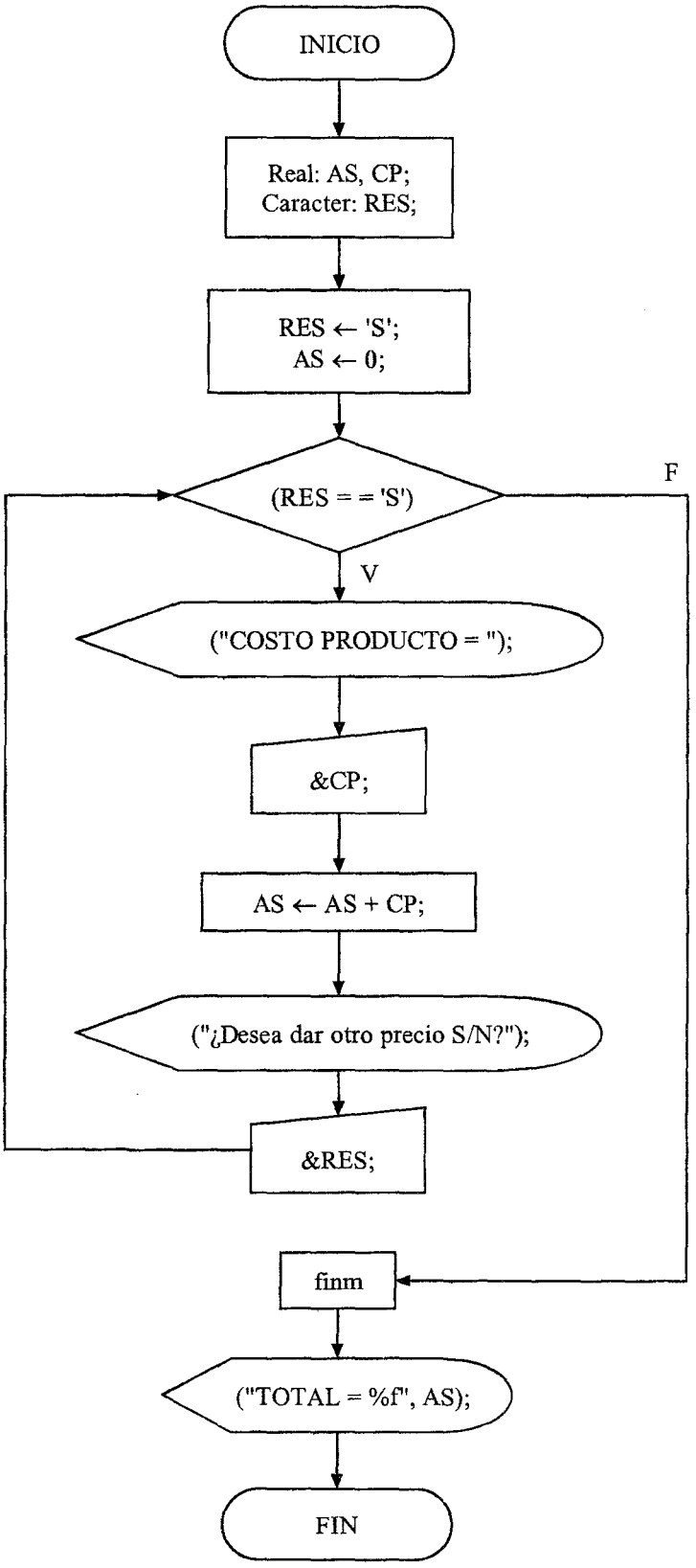
Se pregunta si desea dar otro valor

Se lee y se guarda la respuesta en la variable RES, y regresa el flujo al **mientras**, es decir; se pregunta nuevamente si el valor guardado en RES es igual a S

Fin del ciclo mientras, cuando RES es diferente de S

Fin del programa

Elaborar un algoritmo que calcule el costo total de productos adquiridos, mientras el usuario así lo desee.



Inicio del programa

Declaración de variables: AS (costo total) y CP (costo de c/ producto) de tipo real, y - RES (respuesta del usuario) como variable de tipo carácter

Se asignan los valores S y 0 en las variables RES y AS, respectivamente.

Ciclo:
Mientras variable RES sea igual a S (Si)

Despliega mensaje al usuario, pidiéndole el costo del producto.

Lee y guarda este valor en CP

Se incrementa el costo total (AS) con el costo de este producto (CP)

Pregunta para el usuario: **¿Desea dar otro precio S/N?**

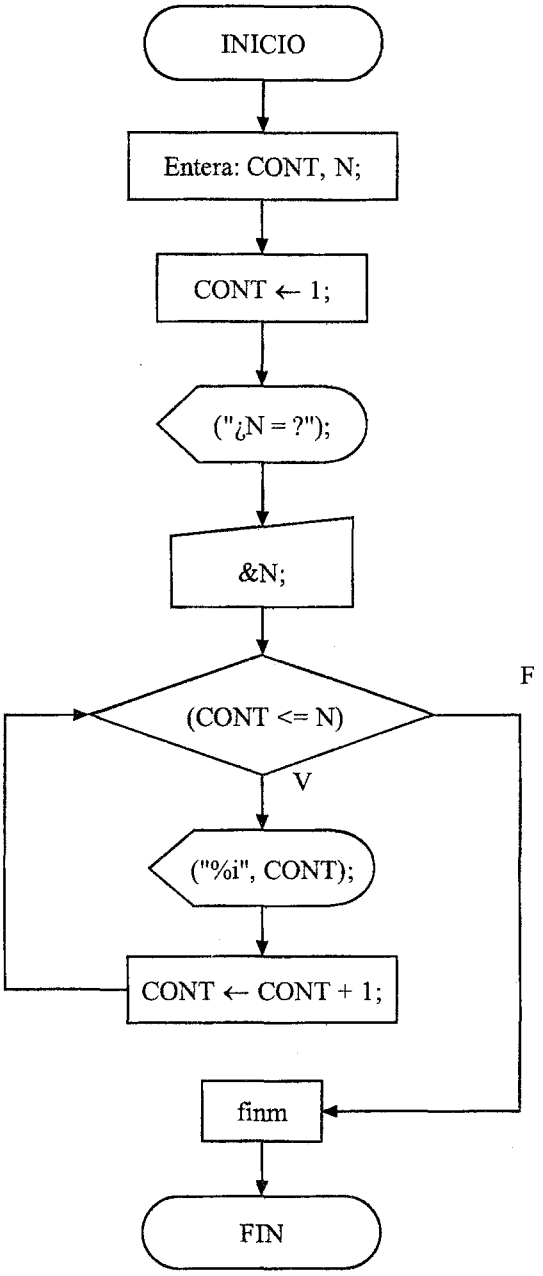
Se lee y se guarda su respuesta en RES
El flujo del programa regresa al ciclo mientras

Fin del ciclo; esto es cuando el usuario responde con una letra distinta de S

Se despliega el comentario: **TOTAL** = , seguido del valor correspondiente (contenido en AS)

Fin del programa

Elaborar un algoritmo que presente en el monitor los primeros N números enteros positivos.



Inicio del programa

Declaración de variables CONT y N como enteras

Se asigna el valor 1 a la variable CONT (contador)

Mensaje que solicita un número N al usuario

Se lee y se guarda el valor tecleado en la variable N (número de enteros positivos a generarse)

Ciclo:
Mientras el valor de la variable CONT sea menor o igual al valor de la variable N

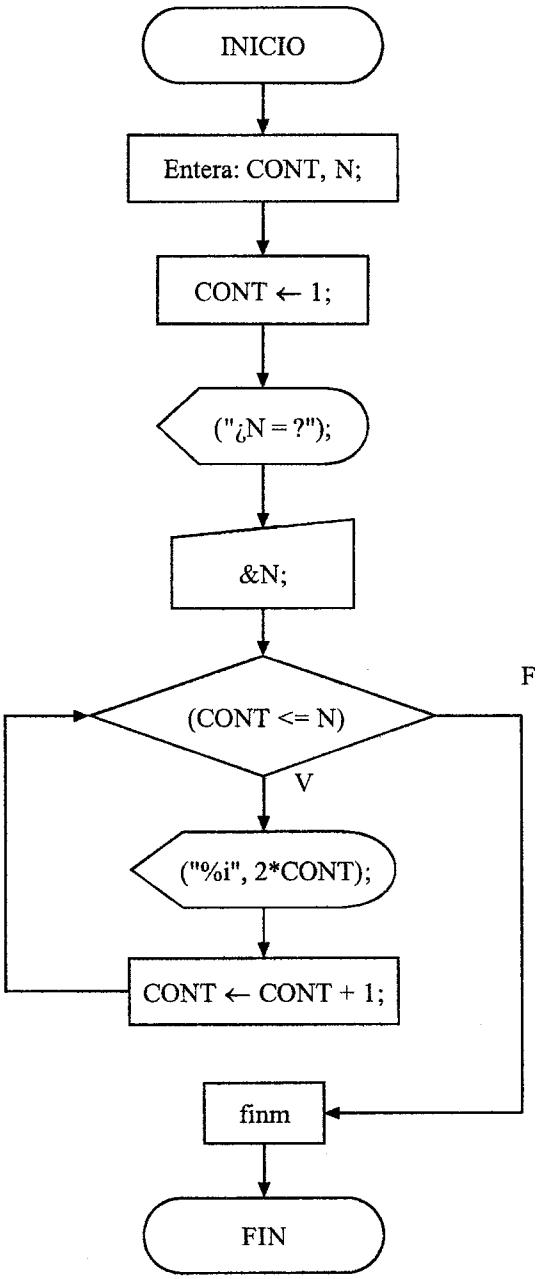
Se despliega en pantalla el valor actual de la variable CONT

Se incrementa en 1 el valor de CONT; y regresa el flujo al ciclo mientras.

Fin del mientras; cuando el valor de la variable CONT es mayor que el de la variable N

Fin del programa

Elaborar un algoritmo que presente en el monitor los primeros N números pares positivos.



Inicio del programa

Declaración de variables CONT (contador) y N (número de pares positivos a generarse), ambas como enteras

Se asigna el valor 1 a la variable CONT

Mensaje que solicita un número N al usuario

Se lee y se guarda el valor tecleado en la variable N

Ciclo:
Mientras el valor de la variable CONT sea menor o igual al valor de la variable N

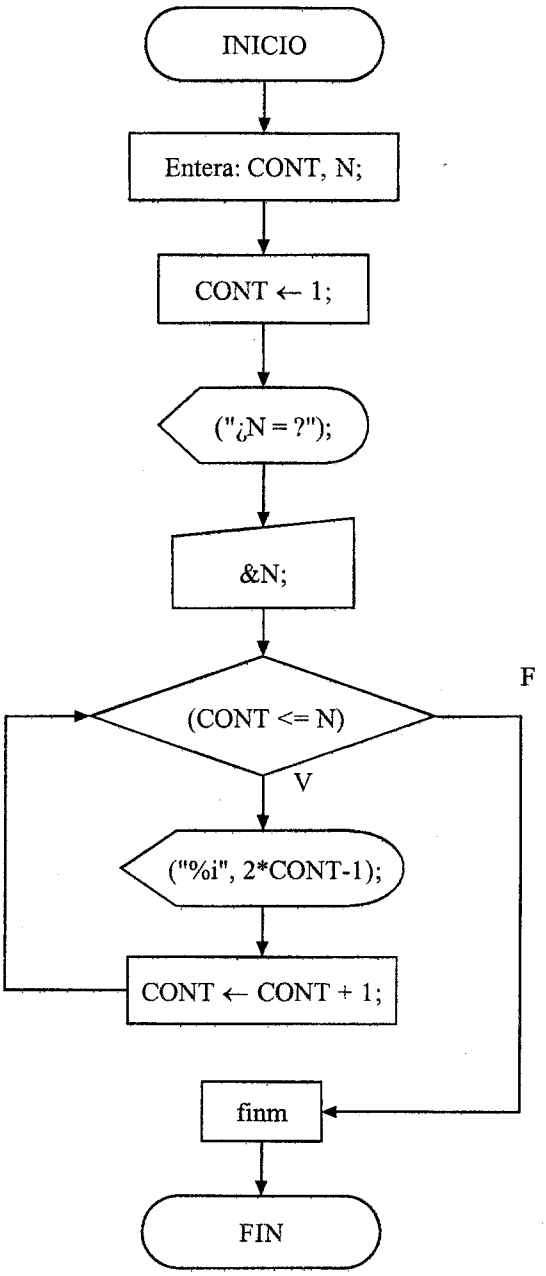
Se despliega en pantalla el valor del número par correspondiente con el valor de CONT
Si CONT es 1 entonces 2*CONT despliega **2**
Si CONT es 2 entonces 2*CONT despliega **4**
Si CONT es 3 entonces 2*CONT despliega **6**
NOTA: El cero no se está considerando como par

Se incrementa en 1 el valor de CONT; y regresa el flujo al ciclo mientras.

Fin del mientras; cuando el valor de la variable CONT es mayor que el de la variable N

Fin del programa

Elaborar un algoritmo que presente en el monitor los primeros N números impares positivos.



Inicio del programa

Declaración de variables CONT (contador) y N (número de impares positivos a generarse), ambas como enteras

Se asigna el valor 1 a la variable CONT

Mensaje que solicita un número N al usuario

Se lee y se guarda el valor tecleado en la variable N

Ciclo:
Mientras el valor de la variable CONT sea menor o igual al valor de la variable N

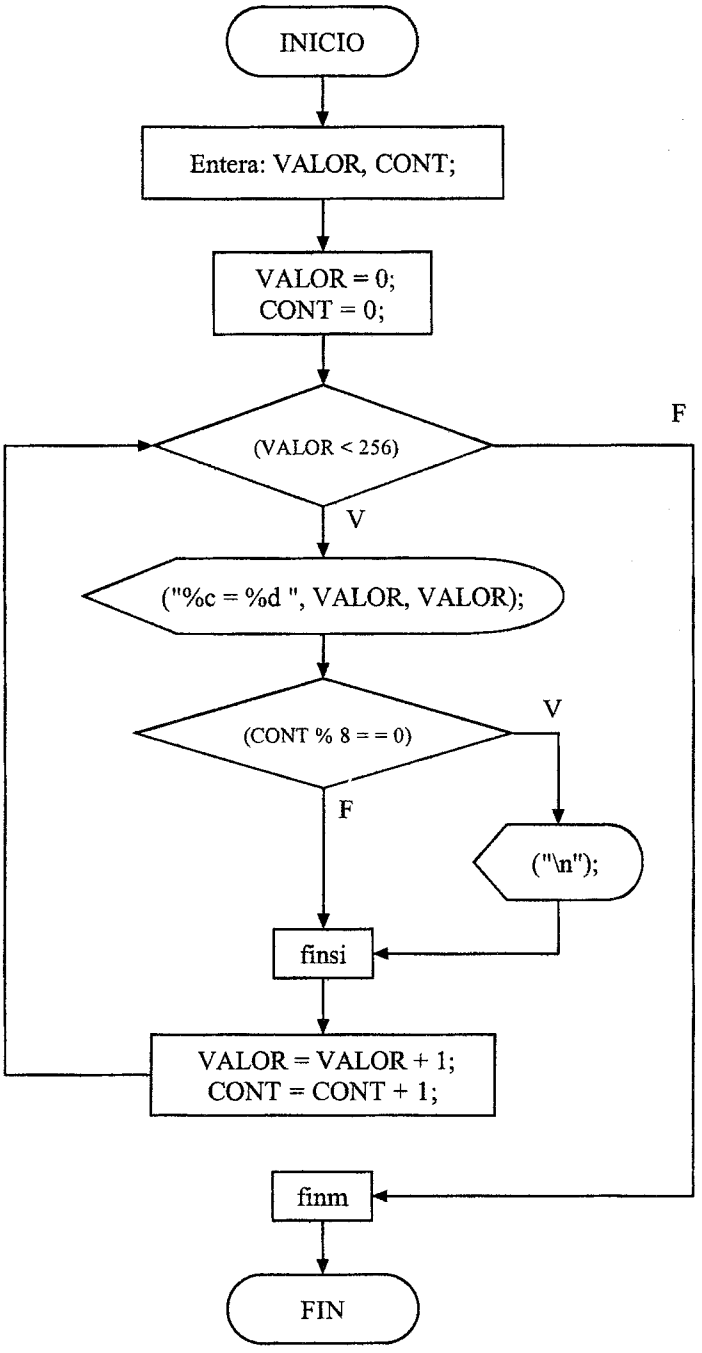
Se despliega en pantalla el valor del número impar correspondiente con el valor de CONT
Si CONT es 1 entonces 2*CONT-1 despliega **1**
Si CONT es 2 entonces 2*CONT-1 despliega **3**
Si CONT es 3 entonces 2*CONT-1 despliega **5**

Se incrementa en 1 el valor de CONT; y regresa el flujo al ciclo mientras.

Fin del mientras; cuando el valor de la variable CONT es mayor que el de la variable N

Fin del programa

Elaborar un algoritmo para generar el código ASCII. (Este código de representación de caracteres utiliza 8 bits; es decir, se pueden generar hasta 256 valores).



Inicio del programa

Declaración de variables: VALOR (para cada elemento del código ascii entre 0 y 255) y CONT (contador para que cada que despliegue 8 elementos realice un salto de renglón)

Se inicializan en cero ambas variables

Ciclo 1:
Mientras VALOR sea menor que 256

Se despliega: __ = __ ;
donde en el primer espacio se despliega el valor como carácter y en el segundo el valor como número que tiene la variable VALOR. Por ejemplo:
? = 63 @ = 64 A = 65 B = 66 C = 67 . . . F = 70

Si CONT módulo 8 es igual a cero **entonces**
/* Esto es cada que se despliegan 8 elementos */

Se despliega un salto de línea

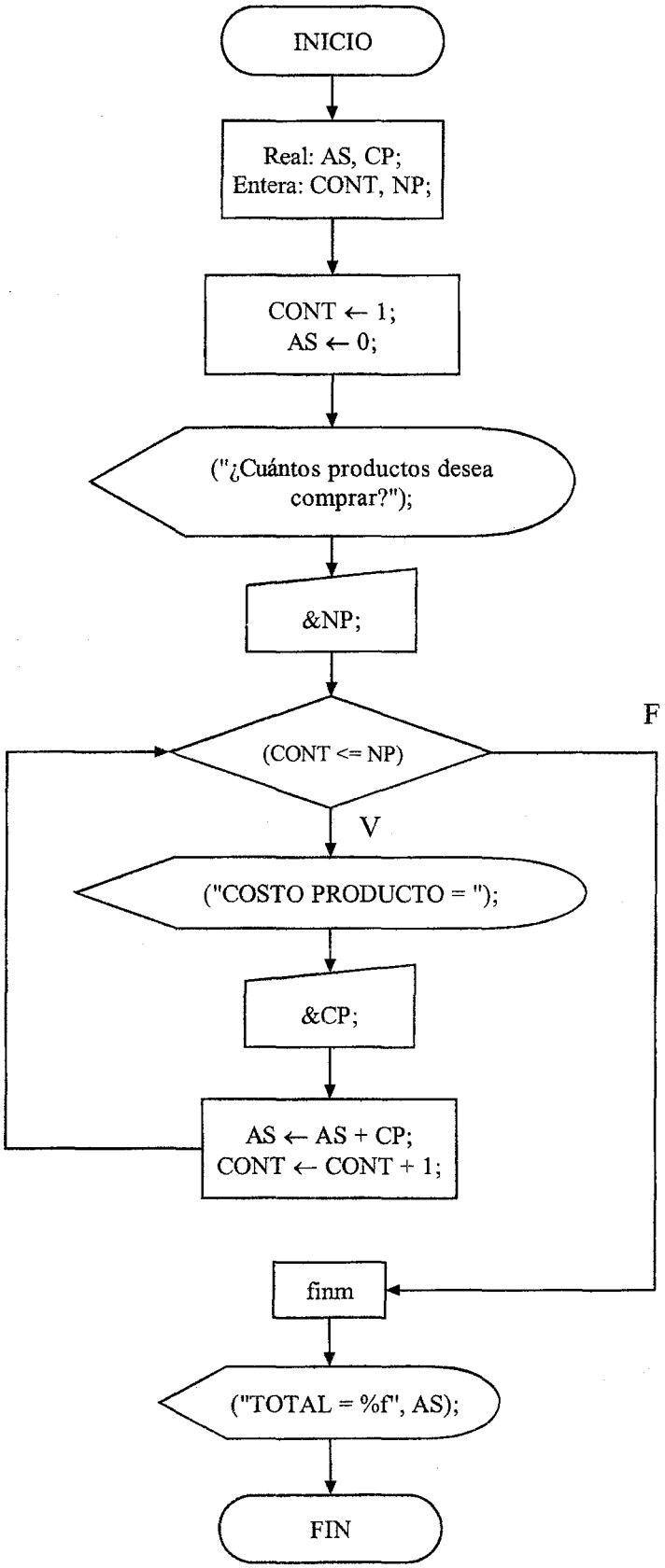
Fin del Si entonces

Se incrementan en 1 VALOR y CONT . El flujo regresa al ciclo 1

Fin de ciclo 1; cuando VALOR es igual a 256

Fin del programa

Elaborar un algoritmo que calcule el costo total del número de productos. Dicho número se pregunta al usuario.



Inicio del programa

Declaración de variables: AS y CP como reales; CONT y NP como enteras.

Se asigna a la variable CONT el valor 1, y a la variable AS el valor 0

Mensaje para el usuario: **¿Cuántos productos desea comprar?**

Se lee y se guarda el valor tecleado en la variable NP

Ciclo:
Mientras el valor de la variable CONT sea menor o igual al valor de la variable NP

Despliega mensaje:
COSTO PRODUCTO =

Se lee y se guarda el valor proporcionado en la variable CP

Se incrementa el valor de la variable AS (acumulador de costos) con el valor actual de la variable CP.
Se incrementa en 1 la variable CONT (contador de productos), y regresael flujo del programa al ciclo mientras

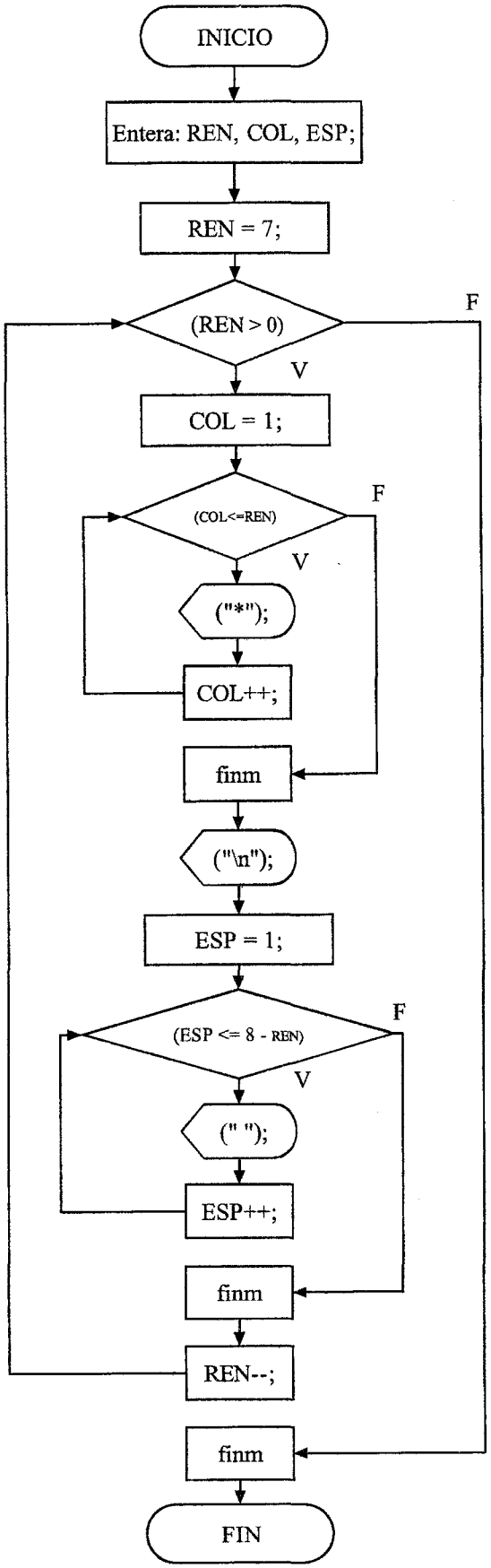
Fin del ciclo mientras; esto es cuando el valor de la variable CONT es mayor que el de la variable NP (número de productos)

Despliega el mensaje: **TOTAL =** , seguido del costo total de los productos (variable AS)

Fin del programa

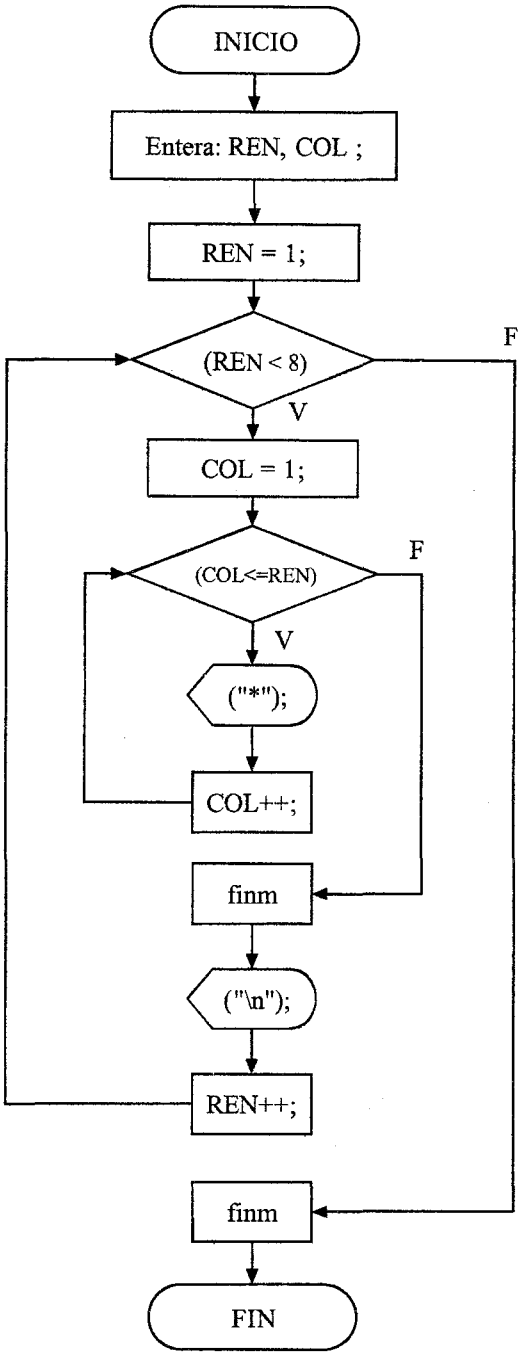
Realizar un algoritmo que despliegue en pantalla la siguiente figura.

```
*******
 ******
  *****
   ****
    ***
     **
      *
```



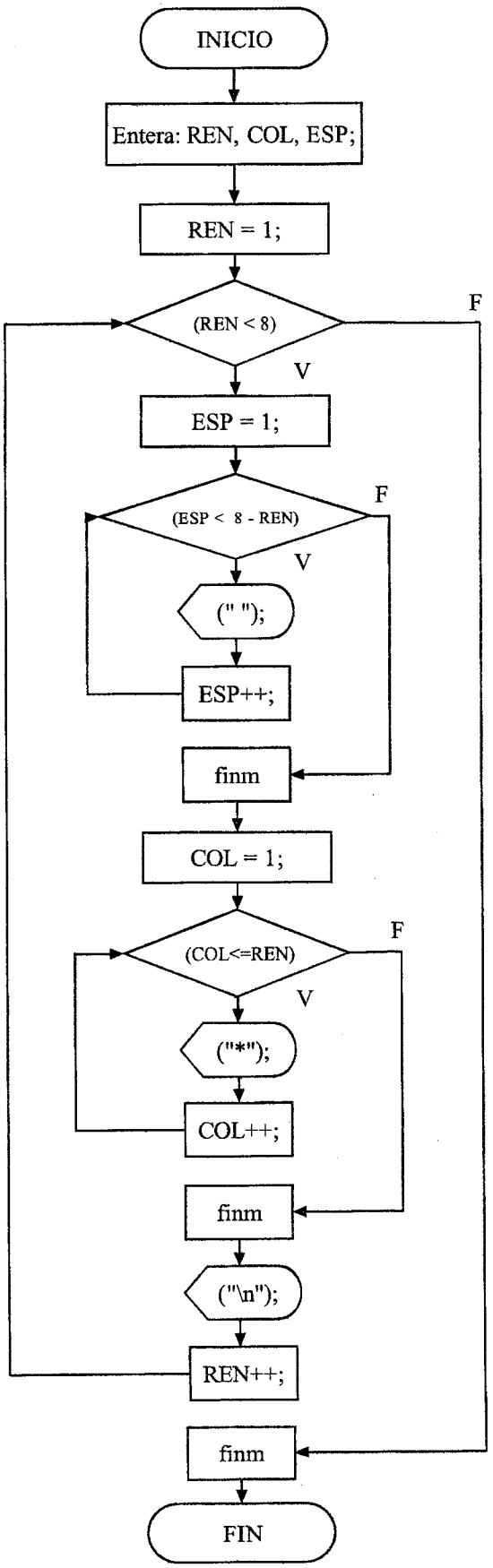
Realizar un algoritmo que despliegue en pantalla la figura siguiente.

```
*
**
***
****
*****
******
*******
```



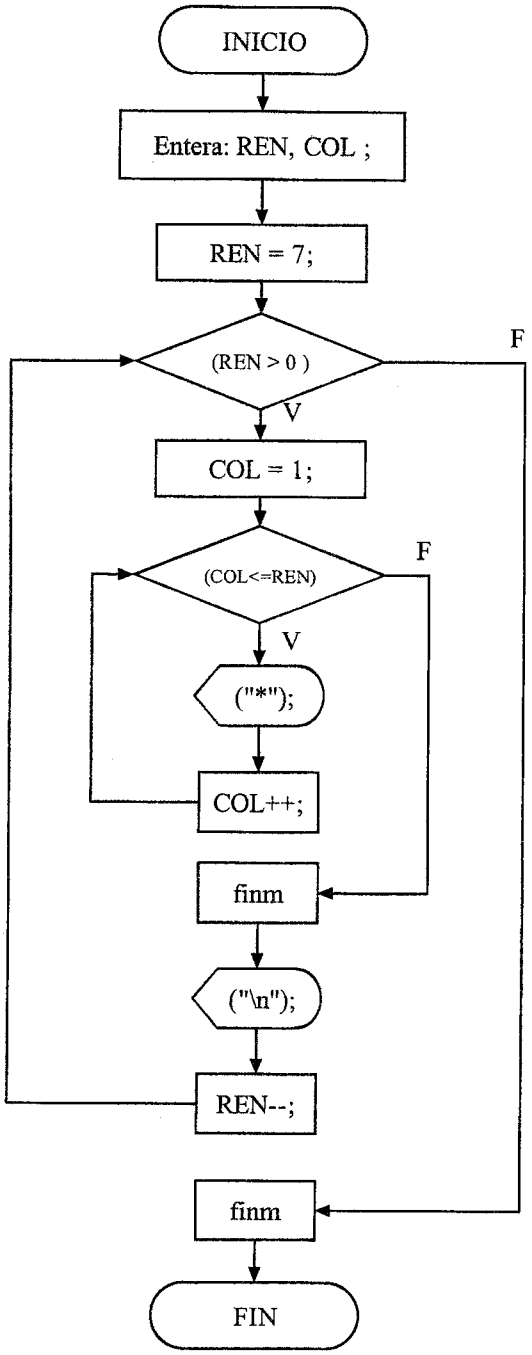
Realizar un algoritmo que despliegue en pantalla la siguiente figura.

```
*
**
***
****
*****
******
*******
```

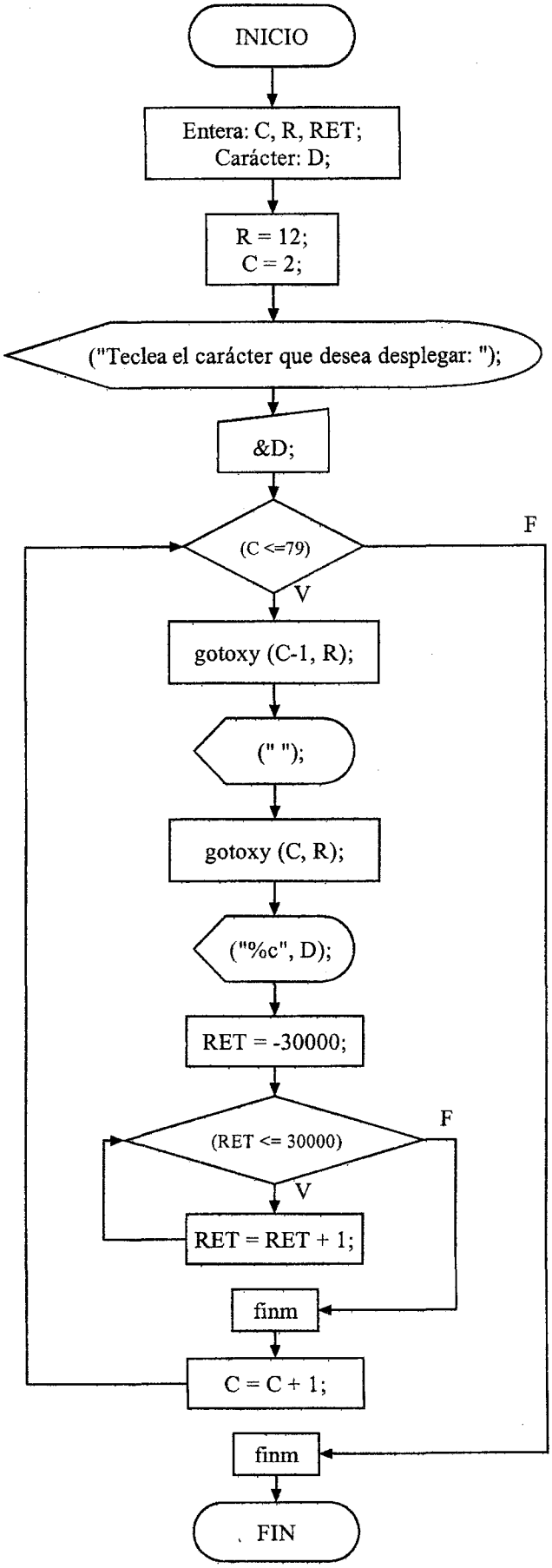


Realizar un algoritmo que despliegue en pantalla la figura siguiente.

```
*******
******
*****
****
***
**
*
```



Elaborar un algoritmo para simular el movimiento de un carácter cualquiera a través de la pantalla, en forma horizontal.



Inicio del programa

Declaración de variables: C (columna), R (renglón) y RET (que se utilizará como retardo) de tipo enteras; D (carácter que se desplegará) de tipo carácter

Se inicializan el renglón en 12 y la columna en 1

Mensaje al usuario

Se lee y se guarda el carácter en D

Ciclo 1:
Mientras C sea menor o igual a 79 (ancho de la pantalla)

Se sitúa el cursor en la columna C-1, y renglón R

Se despliega un carácter en blanco

Se sitúa el cursor en la columna C y renglón R

Se despliega el caracter guardado en D

Se inicializa RET con -30000

Ciclo 2:
Mientras RET sea menor o igual a 30000

/* Este es un ciclo vacío para generar un retardo, este ciclo únicamente cuenta de -30000 hasta 30000 */
Se incrementa en 1 la variable RET. El flujo regresa al ciclo 2.

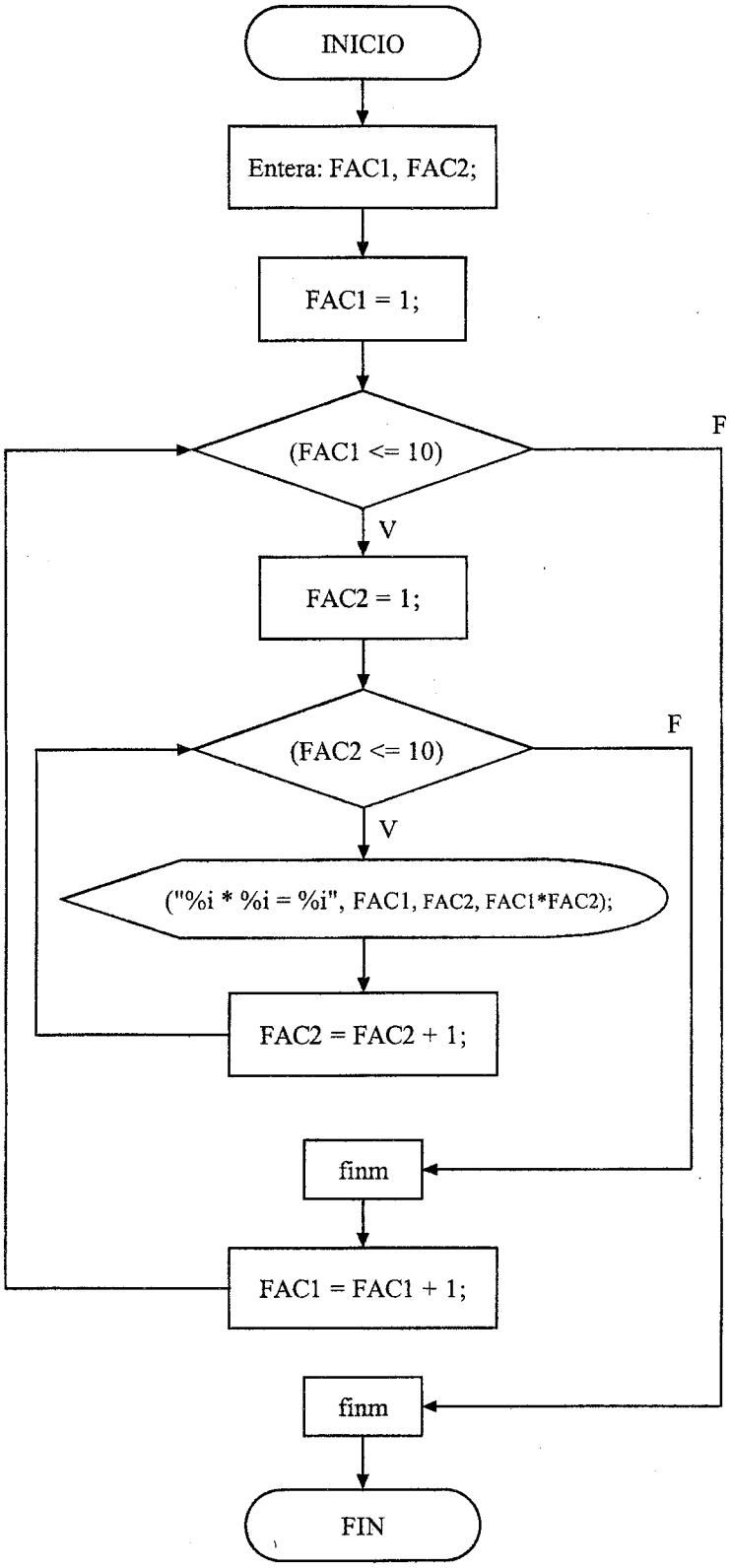
Fin del ciclo 2

Se incrementa en 1 la columna. Regresa al ciclo 1
/* Esto es lo que aparentará el movimiento del carácter */

Fin del ciclo 1, esto es cuando el carácter recorrió toda la pantalla

Fin del programa

Elaborar un algoritmo que despliegue en pantalla las tablas de multiplicar del 1 al 10



Inicio del programa

Declaración de variables FAC1 y FAC2 (factor 1 y factor 2 de las tablas de multiplicar, respectivamente) como enteras

Se asigna el valor 1 a FAC1

Ciclo 1:
Mientras FAC1 es menor o igual a diez

Se asigna a FAC2 el valor 1

Ciclo 2:
Mientras FAC2 es menor o igual a diez

Se despliega en pantalla la tabla de multiplicar del 1 al 10 para el factor 1. Ejemplo:
1 * 1 = 1
1 * 2 = 2
: * : = :
1 * 10 = 10

Se incrementa en 1 el valor del factor 2

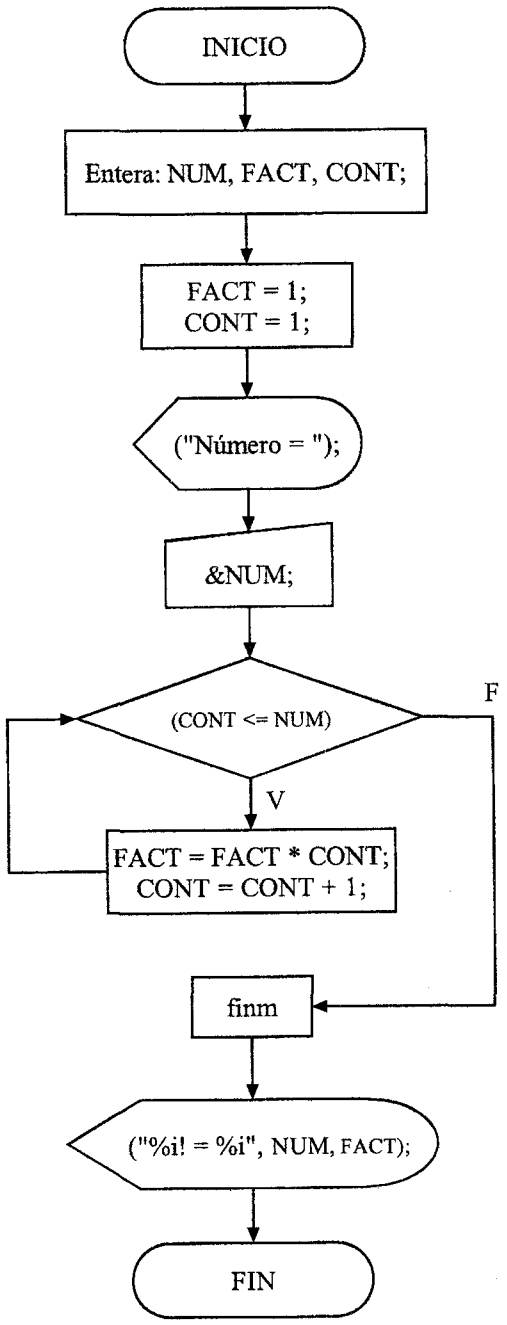
Fin del ciclo 2

Se incrementa en 1 el valor del factor 1

Fin del ciclo 1

Fin del programa

Elaborar un algoritmo que calcule el factorial de un número.



Inicio del programa

Declaración de variables: NUM (número del que se obtendra su factorial), FACT (para el factorial), y - CONT, que es un contador; todas de tipo entero.

Se inicializan en 1 las variables FACT y CONT

Mensaje que le pide un número al usuario:
Número =

Se lee y se guarda el número tecleado en NUM

Ciclo:
Mientras el contador sea menor o igual que el número

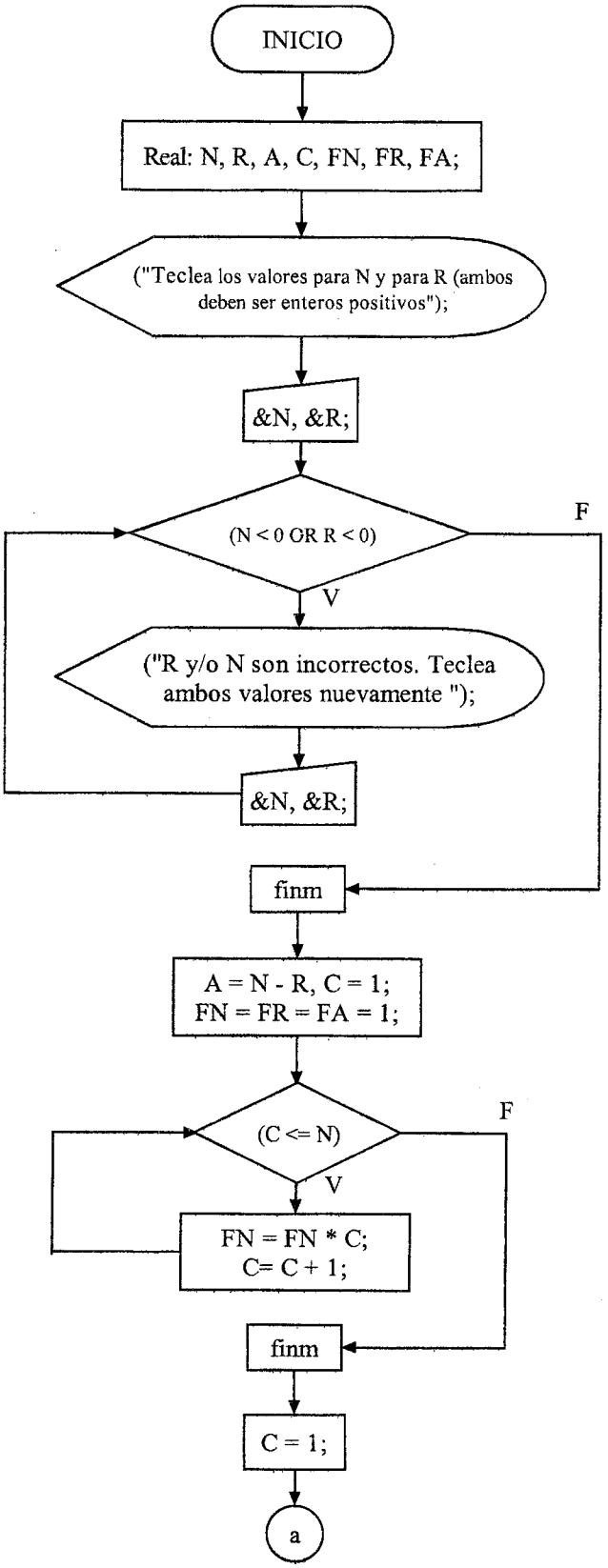
Se multiplica FACT por el valor actual del contador, y se reemplaza el valor del factorial con este sultado.
Se incrementa en 1 el valor de CONT

Fin del ciclo; esto es cuando CONT es mayor que NUM, o sea, el contador es más grande que el número al que se va a calcular el factorial

Se despliegan en pantalla los respectivos valores para el siguiente formato: NUM! = FACT. Por - ejamplo, para 5, se tendría: 5! = 120

Fin del programa

Elaborar un algoritmo para calcular las combinaciones de N elementos tomados en grupos de R elementos, (${}_NC_R = N! / [(N-R)! * R!]$)



Inicio del programa

Declaración de variables: N, R, A (que será igual a N-R), C (contador), FN, FR y FA (para los factoriales de N, R y A=N-R, respectivamente

/*

Esta parte del algoritmo solicita los valores para N y R. Si estos valores son menores que cero se pedirán hasta que ambos sean positivos

*/

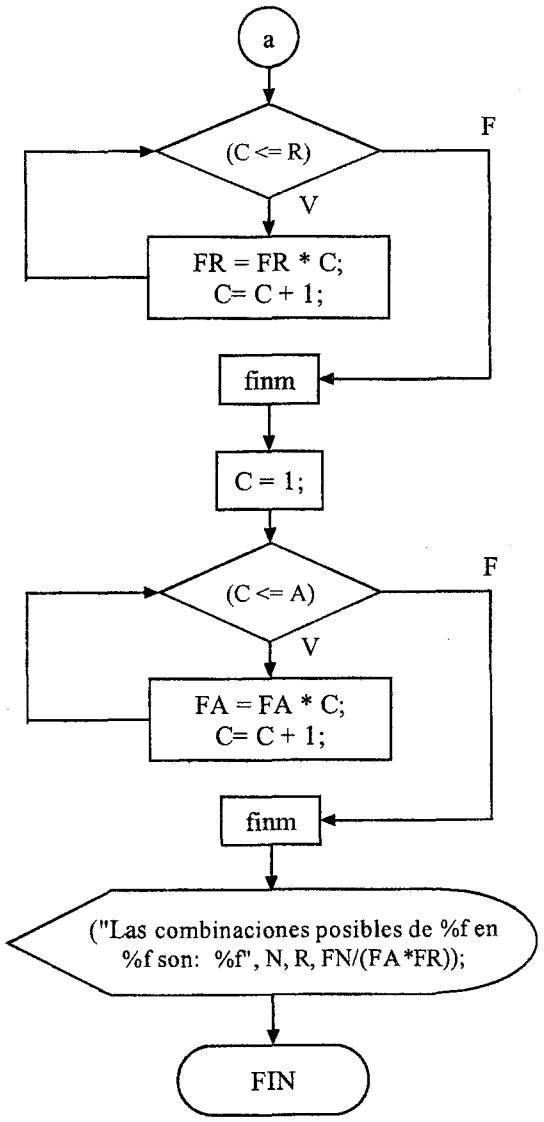
Se asigna a A el valor de N menos R, y se inicializan en 1 el contador y los factoriales de N, de R y de A.

/*

Se calcula el factorial de N

*/

/*



Calculo del factorial de R

*/

/*

Se calcula el factorial de A, donde A = N - R

*/

Se despliega el resultado de las combinaciones posibles

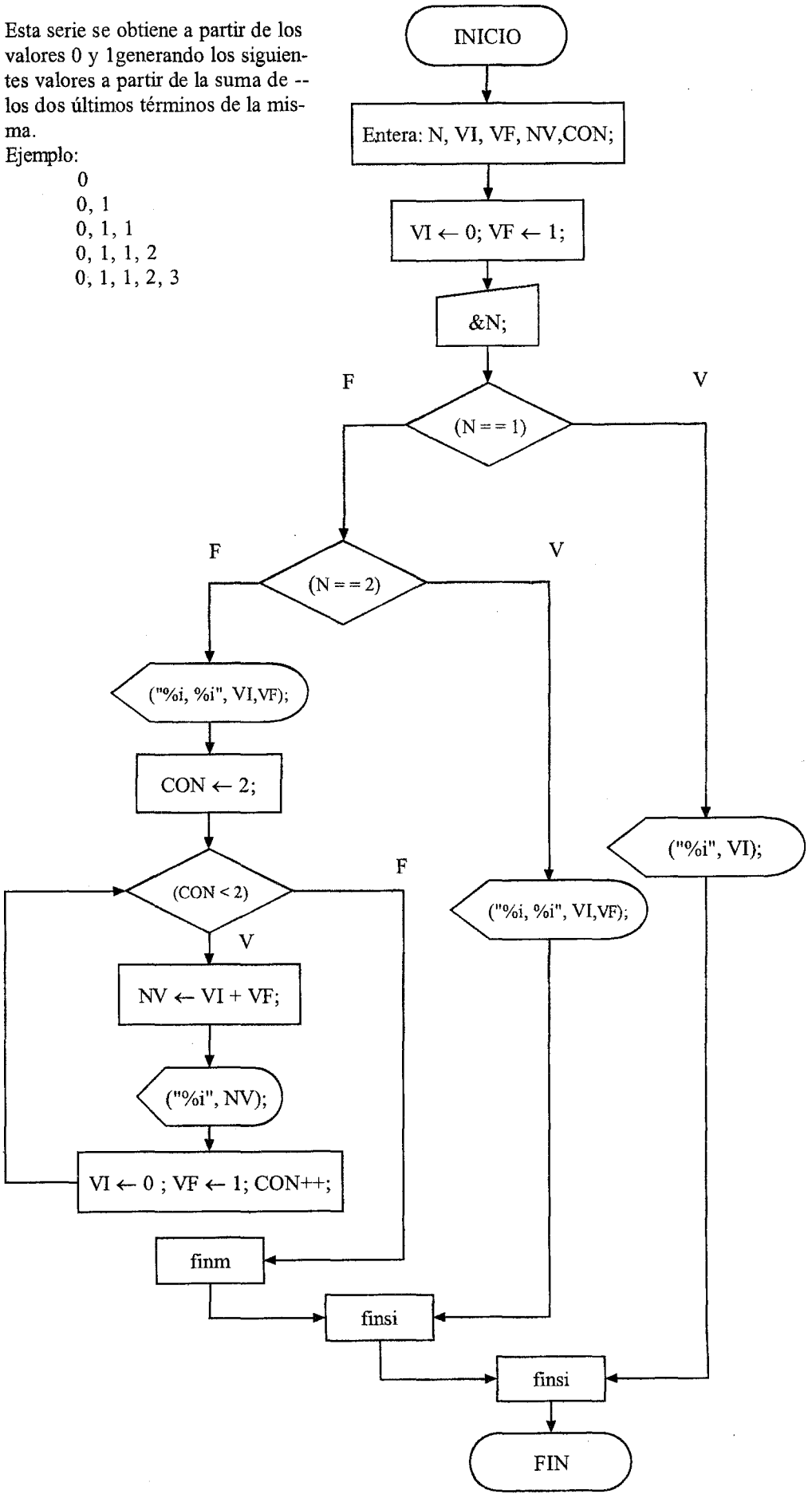
Fin del programa

Elaborar un algoritmo que genere la serie de Fibonacci

Esta serie se obtiene a partir de los valores 0 y 1generando los siguientes valores a partir de la suma de -- los dos últimos términos de la misma.

Ejemplo:

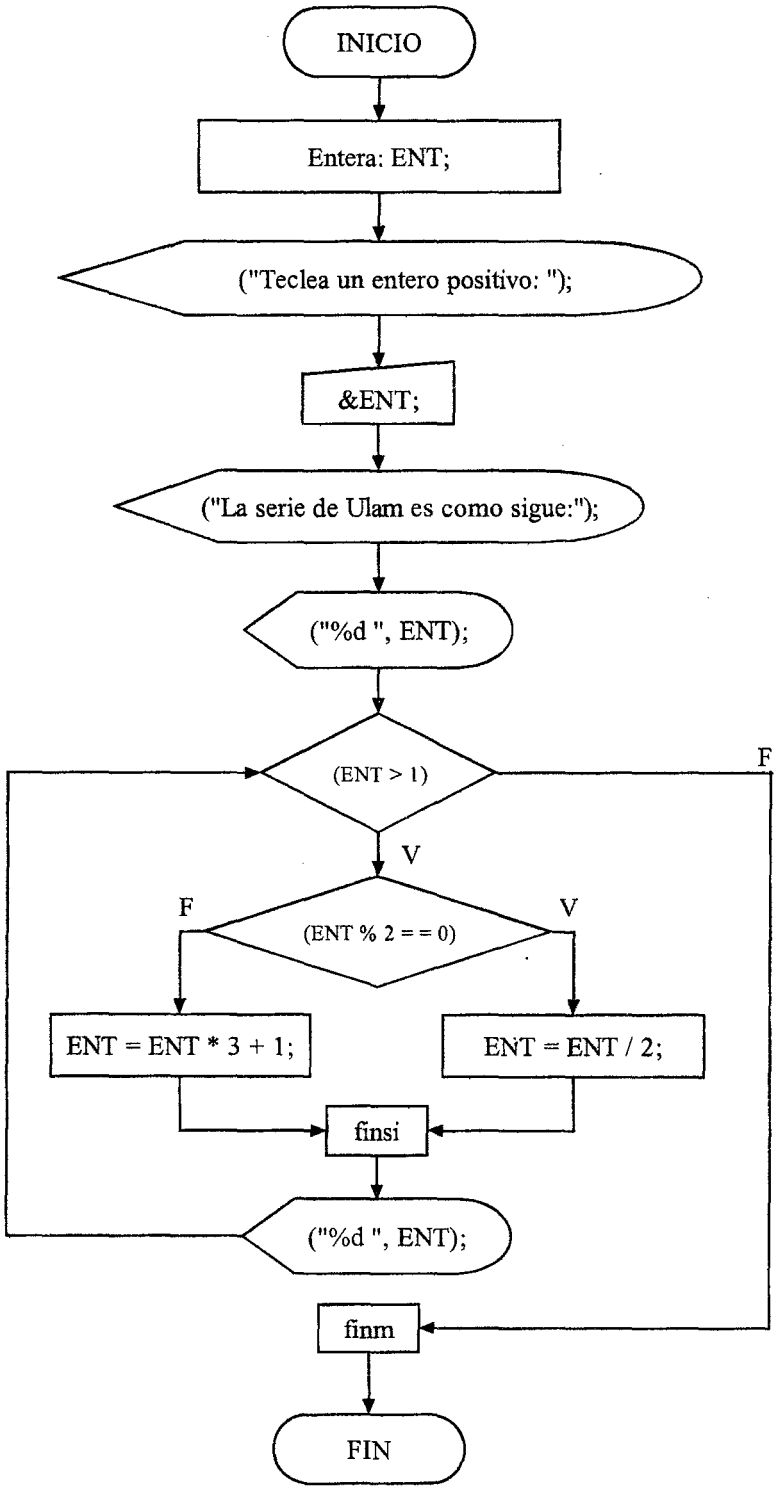
0
0, 1
0, 1, 1
0, 1, 1, 2
0, 1, 1, 2, 3



Elaborar un algoritmo para obtener la conjetura (serie) de Ulam.
Se llama conjetura de Ulam en honor del matemático S. Ulam a lo siguiente:

1. Empiece con cualquier entero positivo
2. Si es par, divídase entre 2; si es impar, multiplíquese por 3 y agréguese 1.
3. Obtenga enteros sucesivamente repitiendo el proceso hasta que el entero en curso sea 1.

Al final se obtendrá el número 1, independientemente del entero inicial. Por ejemplo, cuando el entero inicial es 26, la secuencia será: 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.



Inicio del programa

Declaración de variable ENT (entero inicial de la serie) de tipo entero

Se despliega un comentario al usuario:
Teclea un entero positivo:

Se lee y se guarda el valor en ENT

Se despliega en pantalla:
La serie de Ulam es como sigue:

Se despliega el valor del primer elemento de la serie

Ciclo 1:
Mientras ENT sea mayor que 1

Si ENT módulo 2 es igual a cero, entonces
/* Esto verifica si el entero es par o impar */

Si es verdadero (entero par), ENT se divide entre dos.
Caso contrario (entero impar), ENT se multiplica por tres y se le agrega 1

Fin del Si entonces caso contrario

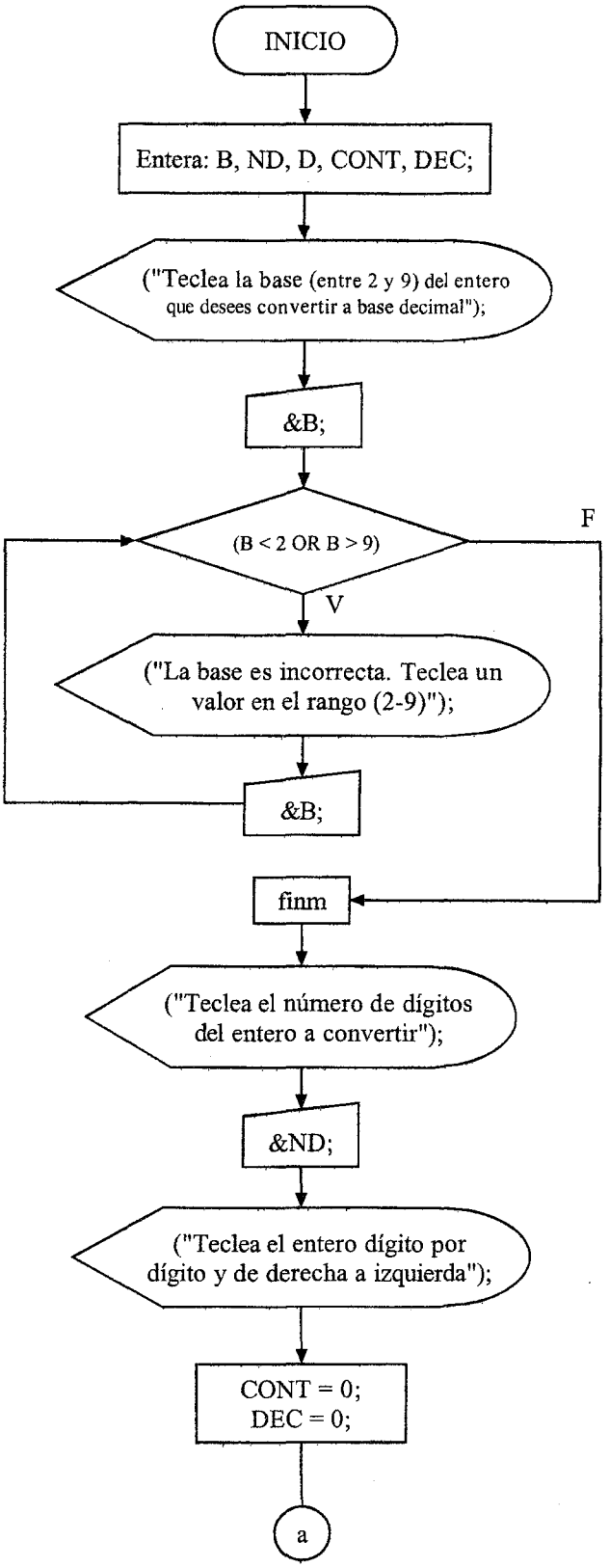
Se despliega el nuevo valor de la variable ENT (siguiente elemento de la serie)
Regresa el flujo al Ciclo 1

Fin del Ciclo 1; esto es cuando ENT llego al valor de 1 (último elemento de la serie)

Fin del programa

NOTAS: En este algoritmo se da por supuesto que el usuario proporciona un valor válido para iniciar la serie. La operación módulo es una operación válida sólo entre enteros que devuelve el residuo al efectuar una división entre dos de estos. En este algoritmo se verifica que el módulo del entero y dos sea cero para saber si el primero es par (puesto que un número par dividido entre dos no deja residuo). Si la operación devuelve un número diferente de cero, entonces el entero es impar.

Elaborar un algoritmo para convertir un número entero en base B, a su número equivalente en base decimal (base 10), donde $2 \leq B \leq 9$, tecleando los dígitos del entero a transformar uno por uno.



Inicio del programa

Declaración de variables: B (para la base), ND (No. de dígitos del entero), D (para guardar temporalmente c/ dígito), - CONT (contador) y DEC(para ir guardando el valor del número convertido)

Se despliega un comentario al usuario

Se lee y se guarda el valor de la base tecleada en B

Ciclo1:
Mientras la base sea menor que 2 o mayor que 9 (esto es, mientras se encuentre fuera de los valores permitidos para B)

Se despliega un comentario de error al usuario

Se lee y se guarda el nuevo valor en B. El flujo regresa al ciclo 1

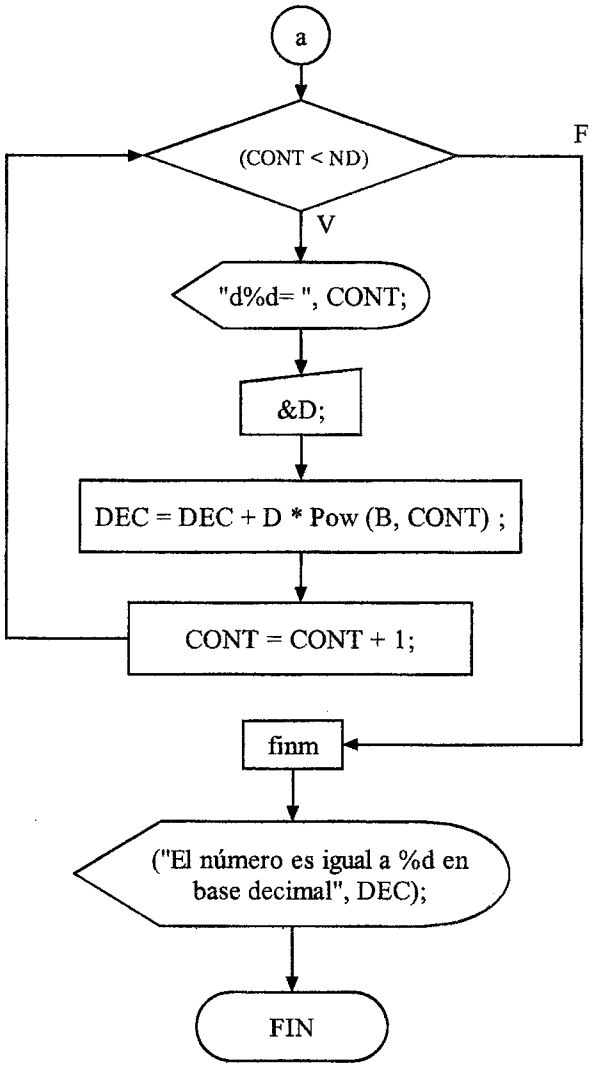
Fin del ciclo 1; esto es cuando el valor de la base B esté entre 2 y 9

Se pide al usuario el número de dígitos del entero

Se lee y se guarda el valor correspondiente en ND

Se despliega otra instrucción para el usuario

Se inicializan las variables CONT y DEC con un valor cero



Ciclo 2:
Mientras CONT sea menor que el número de dígitos

Se despliega: **d__**, donde en el espacio se presenta el no. de dígito en curso. Por ejemplo, el usuario verá:
d0=
d1=
:

Se lee y guarda el dígito en D

Se actualiza el valor de la conversión al considerar el siguiente dígito del entero original, donde Pow (B, CONT) calcula B elevada a la potencia CONT.

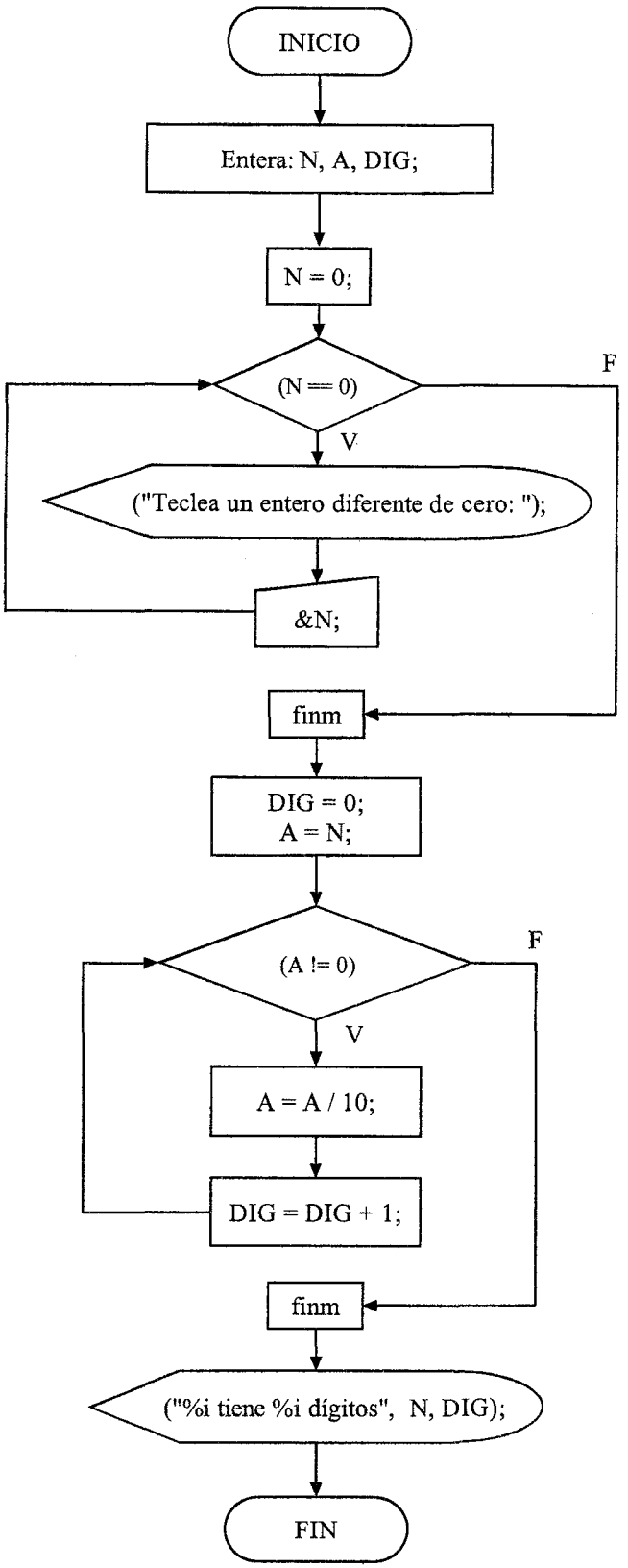
Se incrementa en 1 el contador. El flujo regresa al ciclo 2.

Fin del ciclo 2; esto sucede cuando el contador es igual a ND

Se despliega :
El número es igual a __ en base decimal; donde en el espacio se despliega el valor de la conversión del entero a base decimal

Fin del programa

Elaborar un algoritmo para determinar el número de dígitos de un entero N cualquiera



Inicio del programa

Declaración de variables: N (para guardar el entero dado por el usuario), A (variables temporal) y DIG (contador del número de dígitos).

Se inicializa en cero la variable N

Ciclo 1:
Mientras N sea igual a cero

/* Este ciclo es un filtro que verifica que el usuario te-- clee un entero diferente de cero */

Se despliega un comentario

Se lee y se guarda en N el valor solicitado. El flujo regresa al ciclo 1.

Fin del ciclo 1; esto es cuando el usuario ha tecleado un entero distinto de cero

Se inicializan las variables DIG con el valor cero y A con el valor de N

Ciclo 2:
Mientras A sea diferente de cero

Se actualiza el valor de A dividiéndolo entre 10

Se incrementa en 1 el contador DIG. El flujo regresa al ciclo 2.

Fin del ciclo 2; esto es cuando A toma el valor 0

Se despliega:
__ **tiene** __ **dígitos**. En el primer espacio se despliega el valor del entero N, y en el segundo el número de dígitos que tiene (DIG)

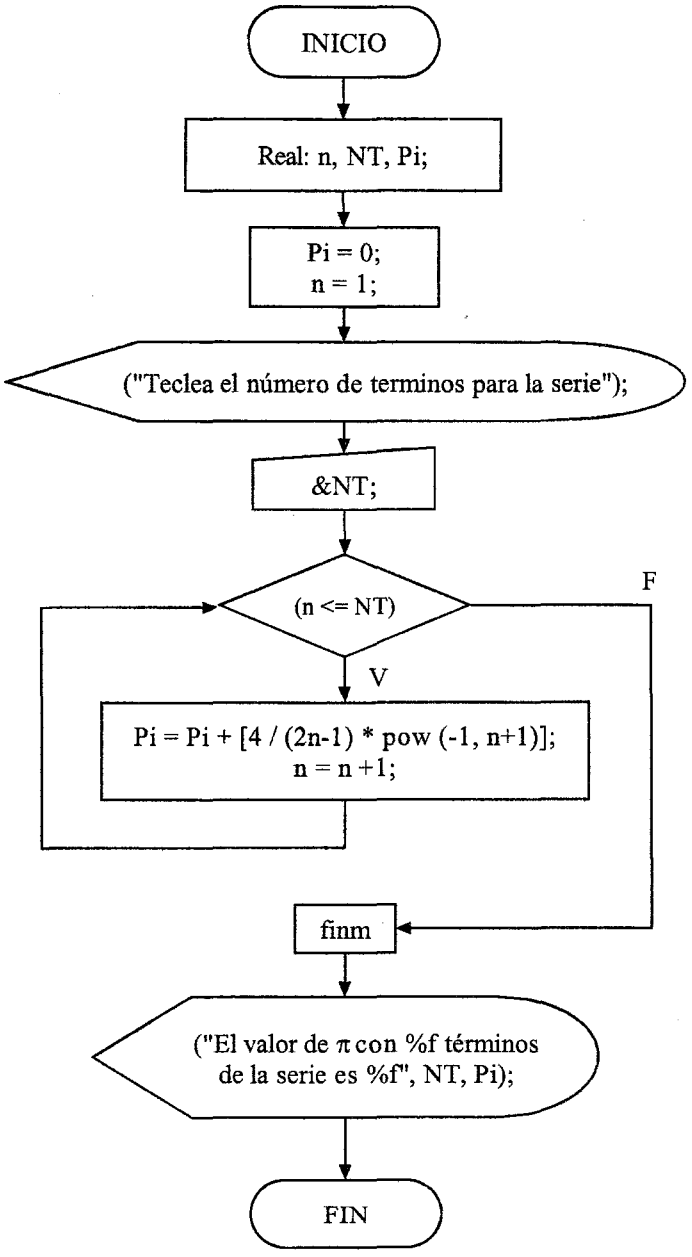
Fin del programa

Calcular el valor de π (Pi) por medio de la serie infinita:

$$\pi = 4 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + 4/13 - \ldots$$

El usuario deberá proporcionar el número de términos de la serie con el cual desea obtener el valor aproximado de π.
Para resolver este problema puede utilizar la siguiente expresión, la cual es equivalente a la serie -- mencionada anteriormente:

$$\pi = \sum_{n=1}^{NT} [4/(2n-1) * (-1)^{n+1}]$$



Inicio del programa

Declaración de variables: n (para obtener el valor de cada tér-- mino de la serie y que a su vez funciona como un contador), NT (número de términos para la serie infinita), y Pi (para ir guardando el valor de π), todas de tipo real o punto flotante.

Se inicializa Pi con el valor 0 y n con el valor 1

Éste es un mensaje para el usuario

Se lee y se guarda el valor proporcionado en la variable NT

Ciclo:
Mientras n sea menor o igual que NT

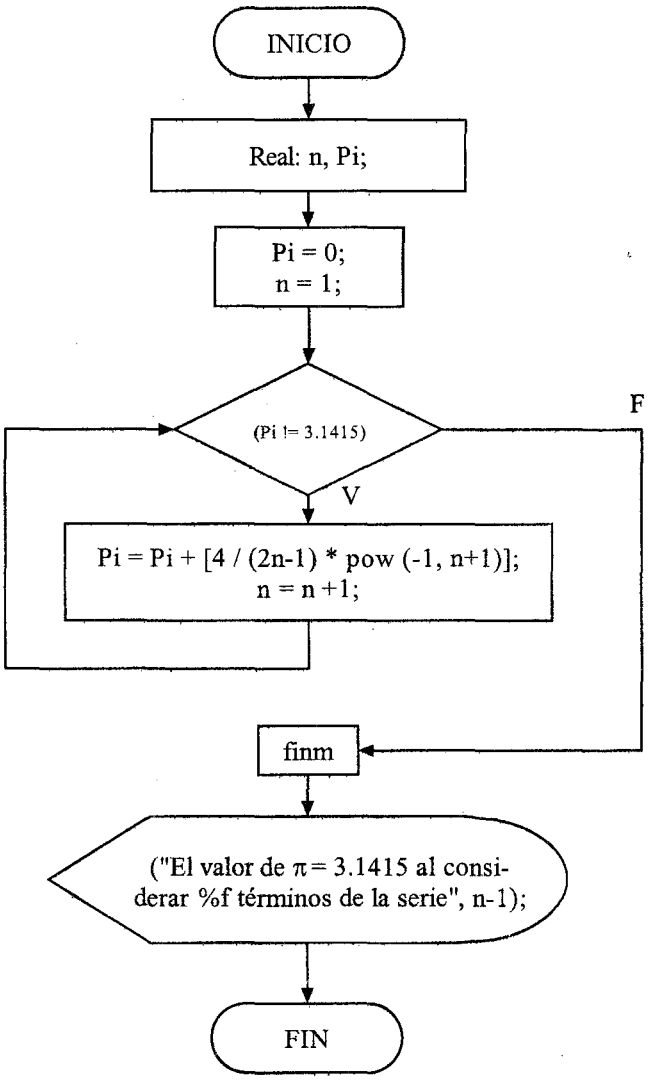
El valor de Pi se actualiza al considerarse un término más de la serie.
Se incrementa en 1 la variable n cada vez que se toma otro - término de la serie

Fin del ciclo mientras; cuando n es mayor que NT

Se despliega:
El valor de π con __ términos de la serie es __ , donde en el primer espacio se despliega el valor de la variable NT y en el segundo el valor calculado en Pi

Fin del programa

Modificar el algoritmo anterior para calcular el valor de Pi, para que ahora determine cuantos términos de la serie se requieren para que π tome el valor de 3.1415.
Observe que para este caso el ciclo del programa deberá ser controlado por el valor que vaya tomando π, y no por un contador para el número de términos de la serie.



Inicio del programa

Declaración de variables: n (para obtener el valor de cada término de la serie y que a su vez funciona como un contador) y Pi (para ir guardando el valor de π), ambas de tipo real o punto flotante.

Se inicializa Pi con el valor 0 y n con el valor 1

Ciclo:
Mientras el valor de Pi sea diferente de 3.1415

El valor de Pi se actualiza al considerarse un término más de la serie.
Se incrementa en 1 la variable n cada vez que se toma otro término de la serie. El flujo regresa al ciclo mientras

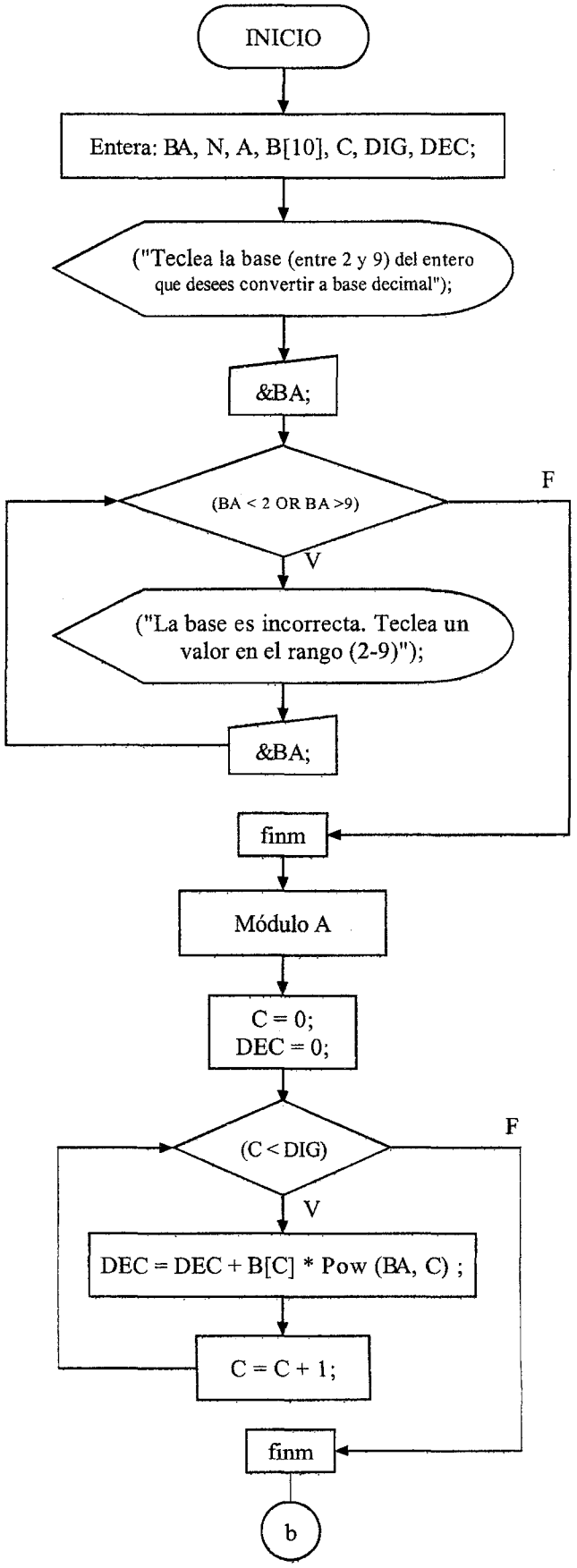
Fin del ciclo mientras; cuando Pi = 3.1415

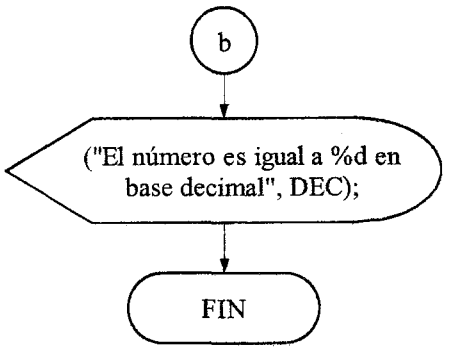
Se despliega:
El valor de π = 3.1415 al considerar __ términos de la serie donde en el espacio se desplegará el número de términos que se utilizaron (valor del contador menos 1, ya que se incrementa después de calcular Pi)

Fin del programa

NOTA: Debe apreciar que el número de términos de la serie que se utilizan para obtener el valor 3.1415 no es en primera instancia el valor de n, puesto que después de calcular Pi se incrementa esta variable. Así cuando el flujo regrese al ciclo mientras y Pi haya alcanzado el valor pedido ya no entrara a éste, y n tendrá un término más que el que realmente tomó el valor establecido para Pi. Es por esta razón que el número que buscamos estará dado por n-1

Elaborar un algoritmo para convertir un número entero en base B (donde $2 \leq B \leq 9$), al entero equivalente en base decimal (base 10).





La declaración de variables es: BA (base del entero a convertir), N, A, B[10], C y DIG (se utilizan igual que en el algoritmo de Dígitos) y DEC (para almacenar el entero convertido en base 10)

/* La primera parte de este algoritmo solicita la base del entero a transformar; y después hay un "filtro" para verificar que ese valor este sólo entre 2 y 9, solicitando al usuario nuevamente el valor, si es que éste es no válido en el rango especificado */

/* En la parte del algoritmo Módulo A se debe solicitar al usuario un entero diferente de cero (var N), verificar esto, y guardar cada uno de los dígitos del mismo en el arreglo B[10].
Lo anterior se realizó en el algoritmo Dígitos, y como se puede observar se están utilizando las mismas variables para hacer este algoritmo más comprensible */

/* Después que se han guardado los dígitos del entero en el arreglo (estos quedan de B[0] al elemento B[DIG-1]) donde DIG es el número de dígitos del entero, se procede al cálculo del valor en base decimal (DEC) */

Se inicializan el contador C y la variable DEC con cero

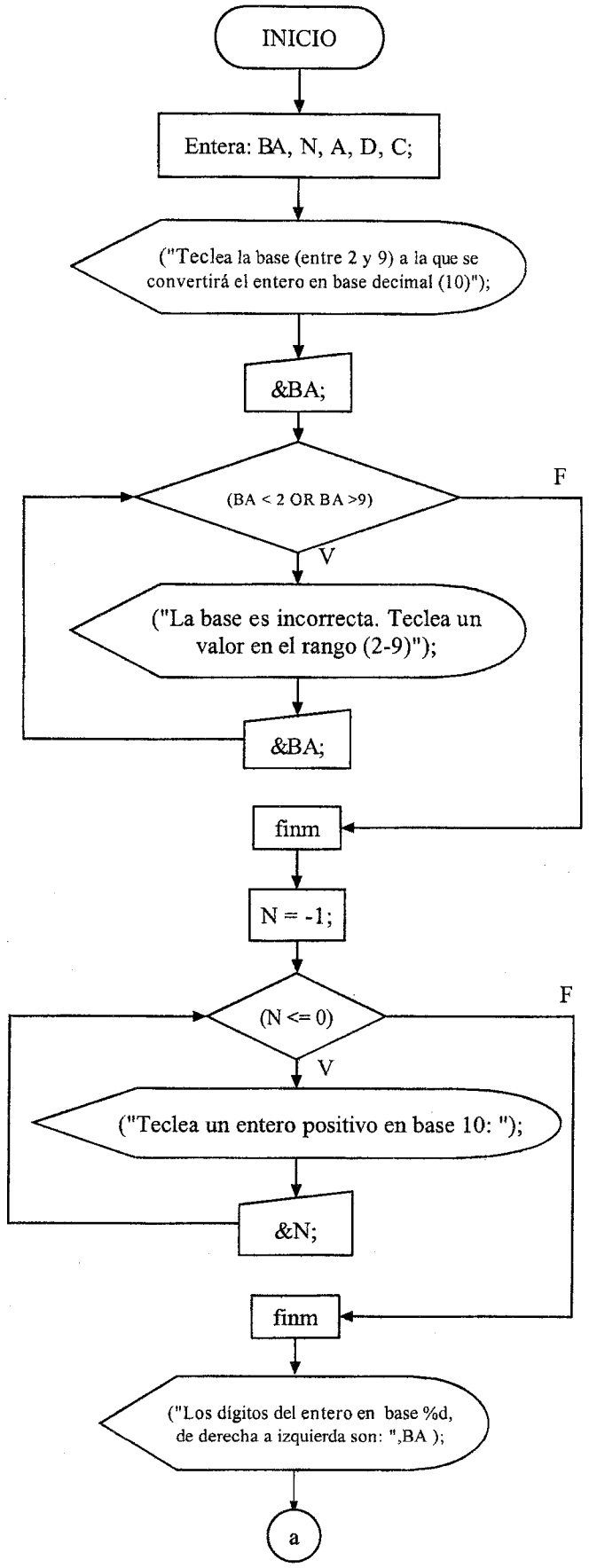
Mientras C sea menor que DIG

Se actualiza el valor de la conversión al considerar el siguiente dígito del entero original (arreglo B) y donde Pow de (BA, C) calcula BA elevada a la potencia C.

Se incrementa C en 1. El flujo regresa al Mientras

Fin del mientras, esto es cuando C alcanza el valor del número de dígitos, lo que significa que el arreglo ya recorrió todos los dígitos del entero para efectuar la conversión.

Elaborar un algoritmo para transformar un número de base 10 a cualquier base B (donde $2 \leq B \leq 9$)



Inicio del programa

Declaración de variables: BA (base a la que se convertirá el entero), N (entero dado en base 10), A (variable para cálculos temporales), D (para almacenar temporalmente los dígitos del entero convertido) y C (contador)

/*

En esta parte del algoritmo se solicita al usuario el valor para la base a la que se convertira el entero proporcionado, y se verifica que dicha base esté en el rango especificado, solicitándola de nuevo en caso de ser necesario

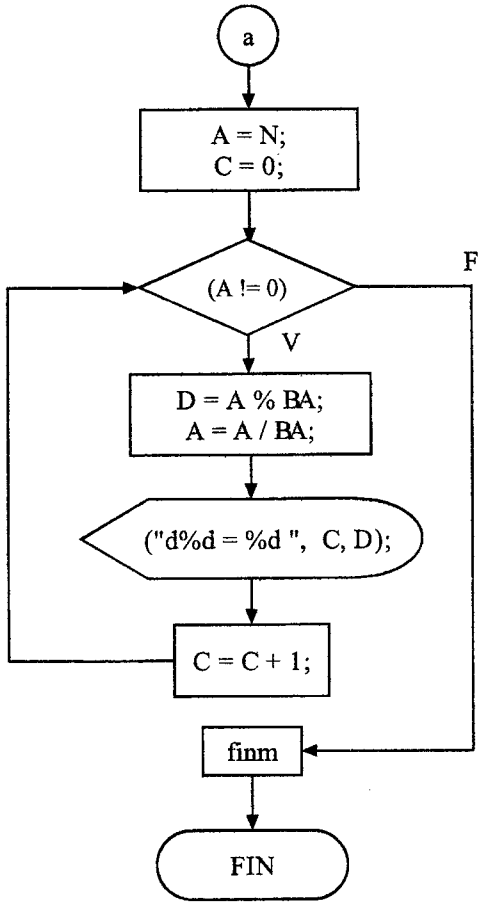
*/

/*

Esta parte del algoritmo solicita al usuario el entero en base 10 que se convertirá a la base que dió anteriormente y se revisa que dicho valor sea positivo, y si éste no lo es, se solicita tantas veces como sea necesario

*/

Se despliega un comentario para el usuario



/* En esta última parte se devuelven al usuario, uno por uno, los dígitos que componen el entero convertido a la base dada, del menos al más significativo (derecha a izquierda) */

Se asigna a A el valor de N y se inicializa en cero a C

Ciclo 3:
Mientras A sea diferente de cero

Se guarda en D al residuo de la división A / BA correspondiente al dígito en la posición C (C=0, 1, 2, ...)
En A se guarda el nuevo valor entero; es decir, al cociente de la división A / BA

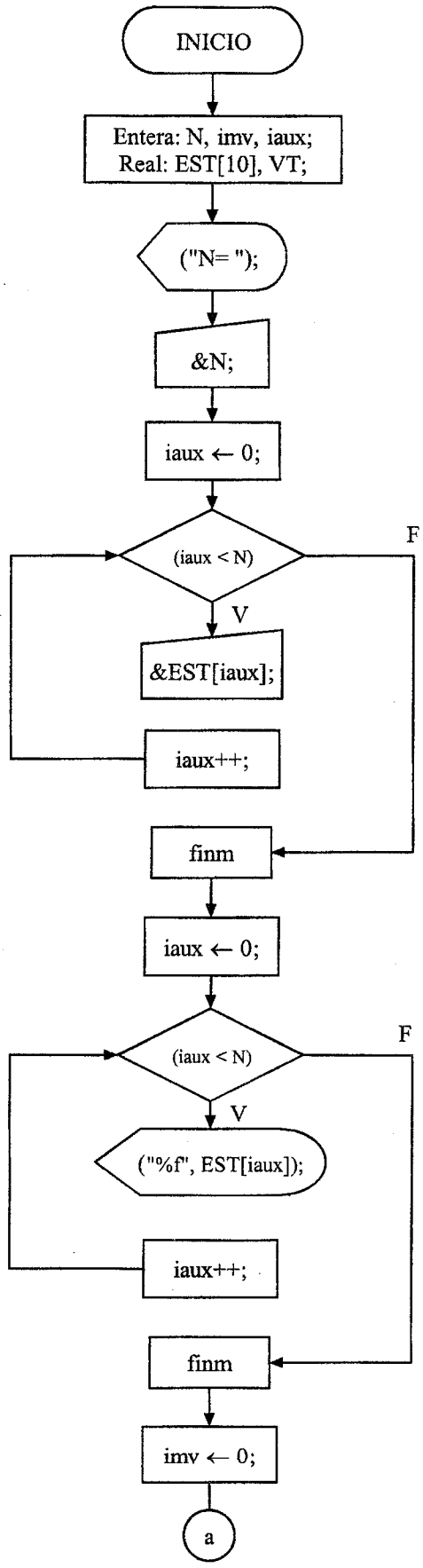
Se despliega:
d__ = __ , donde en el primer espacio se verá la posición del dígito del entero convertido, y en el segundo el valor de dicho dígito. Por ejemplo, el usuario podría ver algo como: **d0 = 0** ; **d1 = 1** ; **d2 = 0** ..... dN = 1.

Se incrementa en 1 el contador C. El flujo regresa al ciclo 3

Fin del ciclo 3, esto sucede cuando las divisiones sucesivas del entero terminan porque se llego al valor de cero

Fin del programa

Elaborar un algoritmo para leer un conjunto de estaturas, utilizando un arreglo unidimensional (vector); desplegarlo y ordenarlo de menor a mayor estatura.



Inicio del programa

Declaración de variables: N (número de estaturas), imv (índice del menor valor: indica la posición en la que se guarda siempre el menor valor del arreglo), iaux (índice auxiliar), EST[10] (arreglo donde se guardan las estaturas)

/* En todo momento se compara EST[imv] y EST[iaux] iaux have un recorrido de todo el arreglo, indicando siempre un valor arriba de imv. Durante el recorrido el menor valor, después de hacer la comparación, se guarda en --- EST[imv] */

/*

Lectura del arreglo

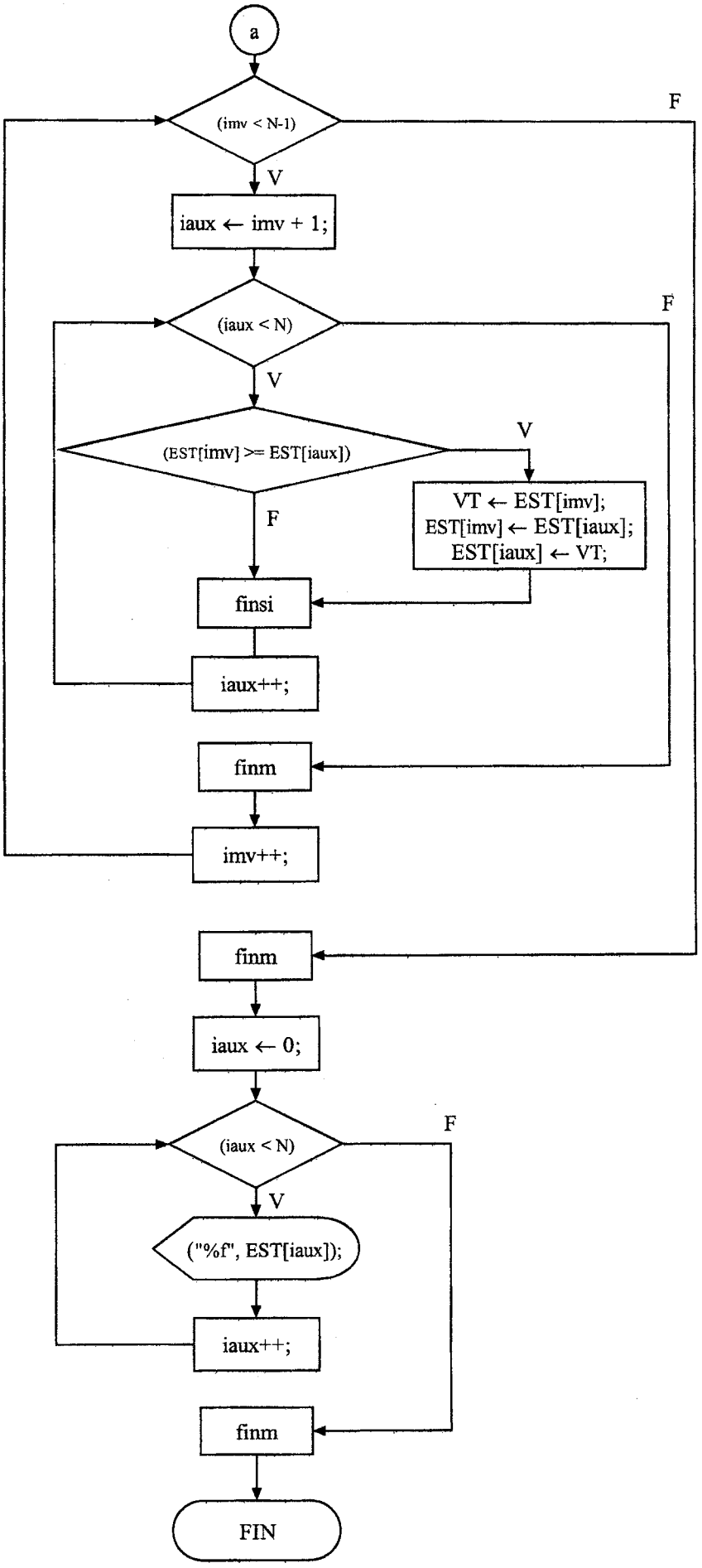
*/

/*

Se despliegan las estaturas

*/

imv señala la posición cero para guardar el menor valor de todo el arreglo en EST[0]



Ciclo **mientras** para dejar fijo imv y para que iaux haga todo el recorrido del arreglo.

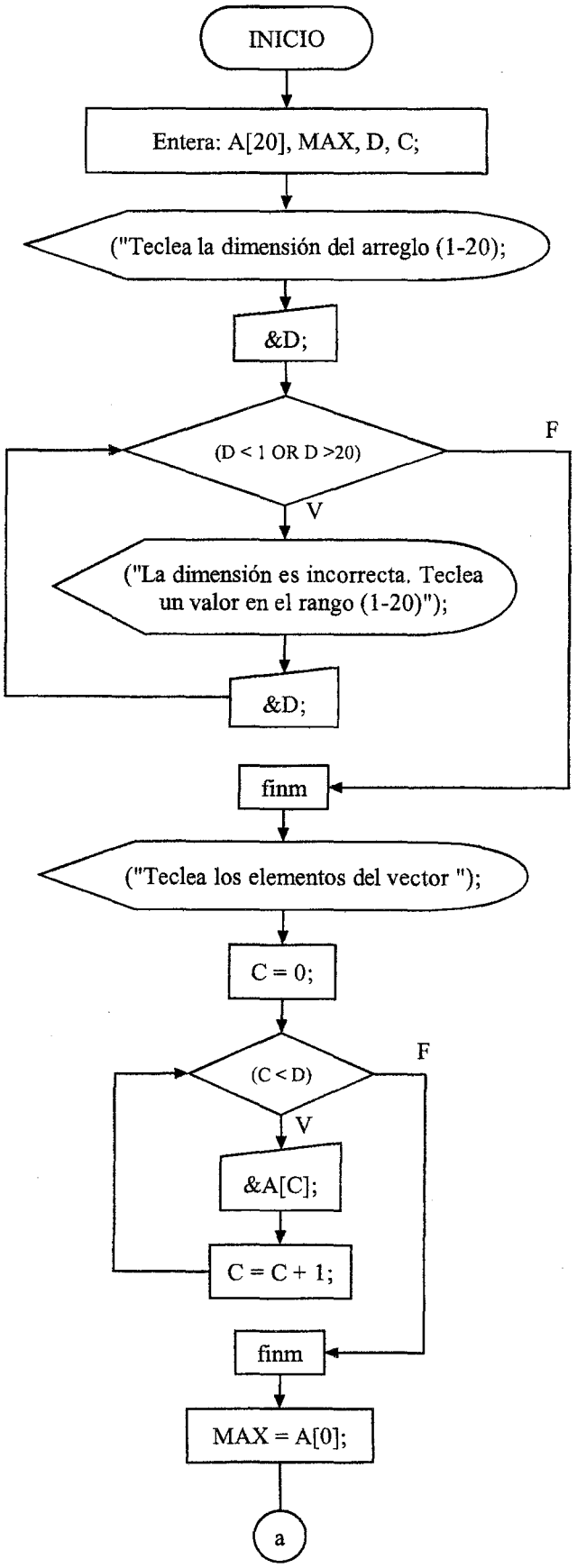
Condición que permite guardar en EST[imv] siempre el menor valor

/*

Se despliega el vector ordenado de menor a mayor

*/

Elaborar un algoritmo que lea un arreglo e indique cual es el mayor elemento dentro de éste.



Inicio del programa

Declaración de variables: A (arreglo de tamaño 20), MAX (para el elemento máximo), D (dimensión dada por el usuario para el arreglo a leer) y C (que se utiliza como contador)

/*

Esta parte del algoritmo solicita al usuario el valor para la dimensión (tamaño) que desea que tenga - el arreglo a leer, en un intervalo entre 1 y 20.
Si se teclea un valor fuera del rango se solicita nuevamente hasta que éste sea correcto.

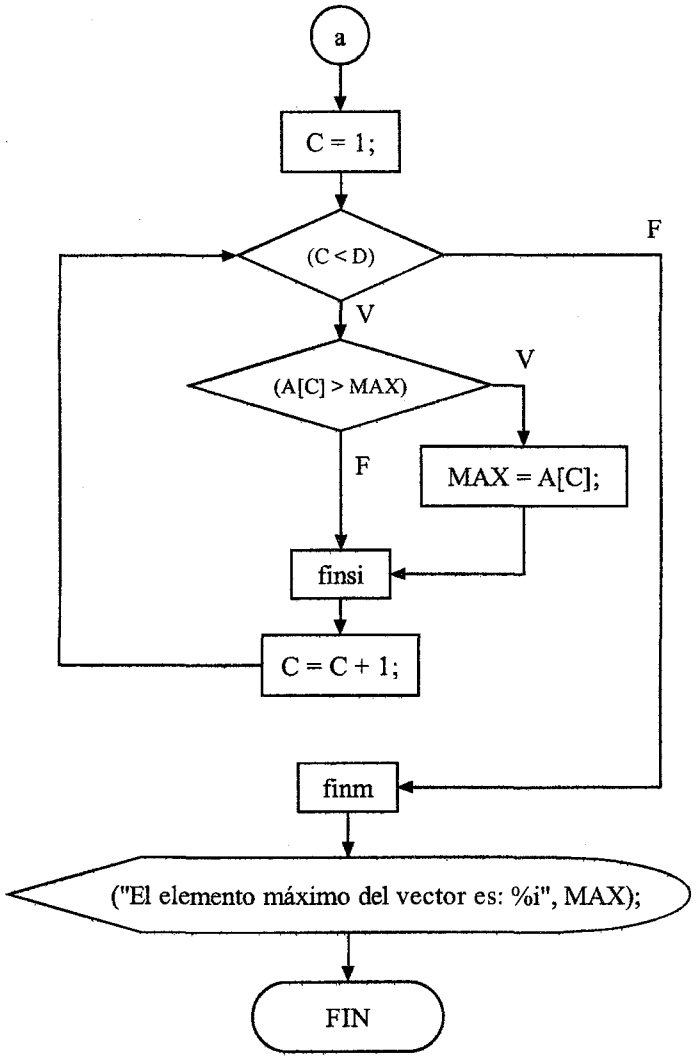
*/

/*

En esta parte se leen los elementos del vector, que corresponden al número dado para la dimensión -- del arreglo

*/

/*



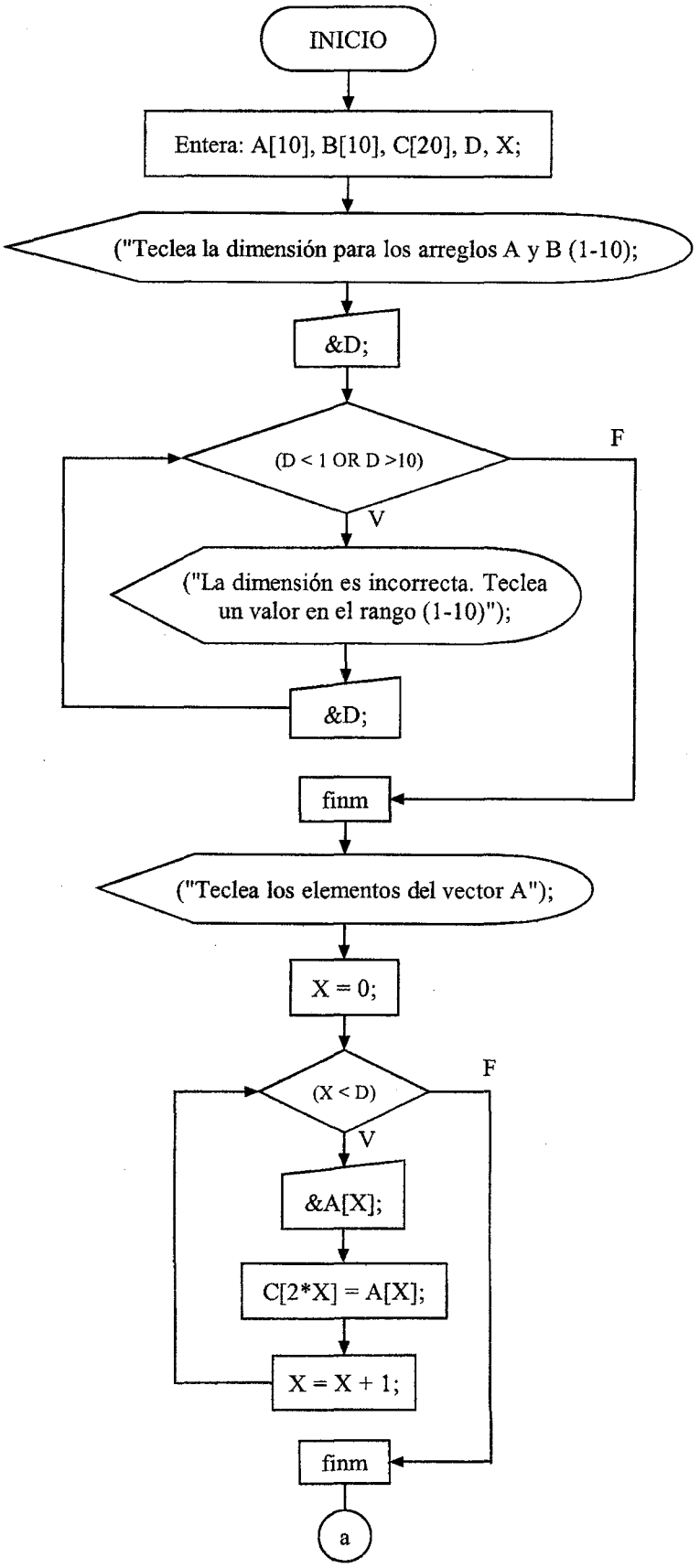
En esta última parte del algoritmo se determina cual es el mayor elemento dentro del arreglo.
Se asigna a MAX el valor del primer elemento y entonces se recorre el arreglo a partir del elemento 2 y hasta el final del mismo
Si algun elemento es mayor que MAX entonces ese valor se reasigna a MAX, haciéndose tantas veces como sea necesario hasta recorrer todo el arreglo.

*/

Se despliega el elemento máximo del arreglo (MAX)

Fin del programa

Elaborar un algoritmo que solicite al usuario los elementos de dos arreglos de enteros (de dimen-- sión dada por el usuario entre 1 y 10) y guarde ambos en un tercer vector (C), cuya dimensión será dos veces la proporcionada para los arreglos originales, siendo C = $[a_0, b_0, a_1, b_1, \ldots, a_n, b_n]$



Inicio del programa

Declaración de variables: A[10], B[10] y C[20] (arre-- glos A, B y C de tamaño 10, 10 y 20, respectivamente) D (dimensión dada por el usuario para los arreglos) y X (contador)

/*

Esta parte del algoritmo solicita al usuario la dimen--- sión para A y B en un intervalo de 1 a 10. Lee el valor y lo guarda en D.
Si el valor no está en el rango especificado, entonces se le solicita nuevamente hasta que éste sea correcto

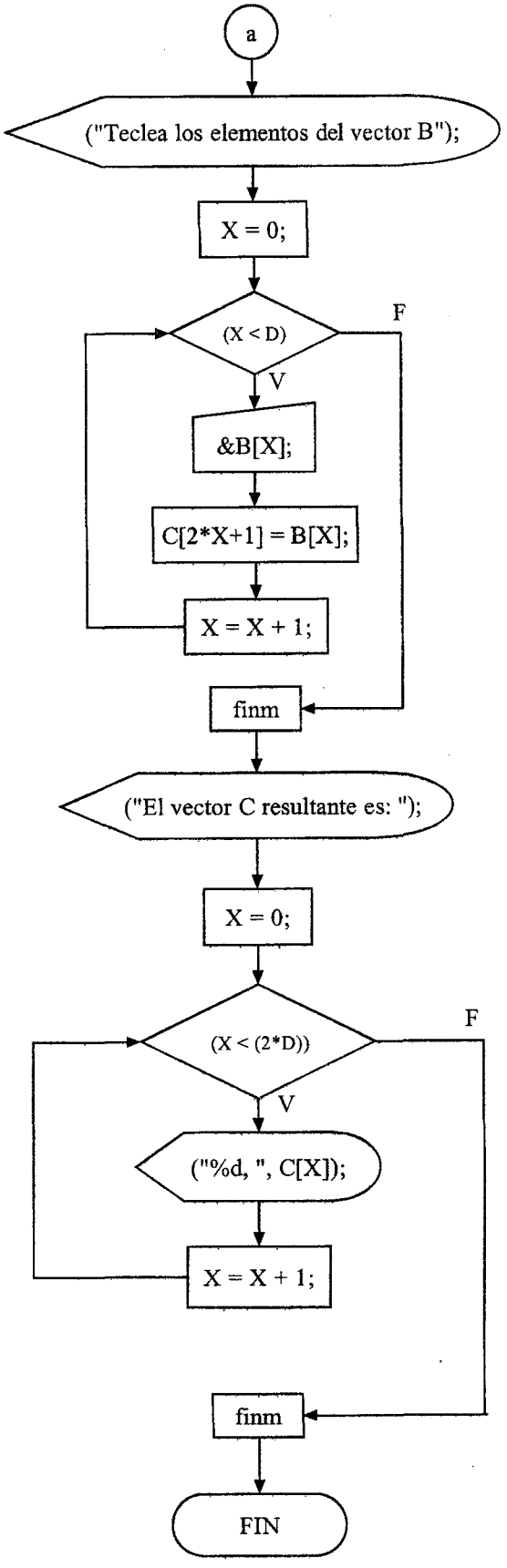
*/

/*

En esta parte se leen todos los elementos para el vec- tor A, y éstos se guardan además en el arreglo C, co- locándose (como se solicita en el algoritmo), en las - posiciones pares de este arreglo, para lo cual utiliza-- mos la expresión:

C[2*X] = A[X]

*/



/*

Esta sección del algoritmo lee y guarda todos los elementos del arreglo B, y los guarda además en el arreglo C en las posiciones impares de éste (como se --- señala en el enunciado del algoritmo) para lo cual utilizamos la expresión:

C[2*X+1] = B[X]

*/

/*

Esta parte despliega los elementos del vector C, los cuales debieron haber quedado en la forma:

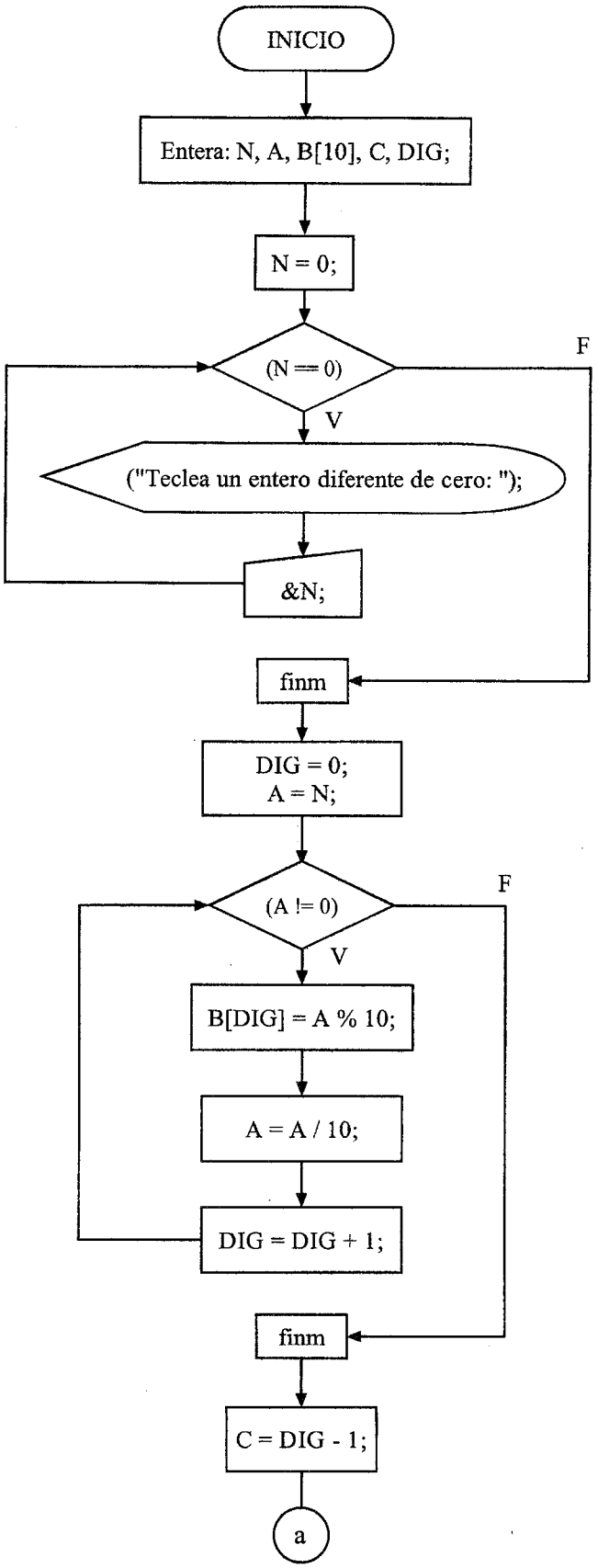
$C = [a_0, b_0, a_1, b_1, \ldots, a_n, b_n]$

donde las a´s son los elementos de A, y las b´s son - los elementos de B

*/

Fin del programa

Elaborar un algoritmo para que dado un entero N cualquiera, despliegue en pantalla cada uno de los dígitos que forman dicho número. Por ejemplo, si el usuario teclea el número 12345, el algoritmo deberá devolverle: 1 2 3 4 5



Inicio del programa

Declaración de variables: N (entero dado por el usuario), A (variable temporal), B[10] (arreglo para guardar los dígitos), C (contador auxiliar) y DIG (contador de dígitos)

Se inicializa en cero la variable N

Ciclo 1:
Mientras N sea igual a cero

/* Este ciclo se utiliza para verificar que el entero dado por el usuario sea diferente de cero */

Se despliega un comentario

Se lee y se guarda en N el valor dado. El flujo regresa al ciclo 1.

Fin de ciclo 1; esto pasa cuando el usuario proporciona un entero diferente de cero

Se inicializa en cero a DIG, y se asigna a A el valor de N

Ciclo 2:
Mientras A sea diferente de cero

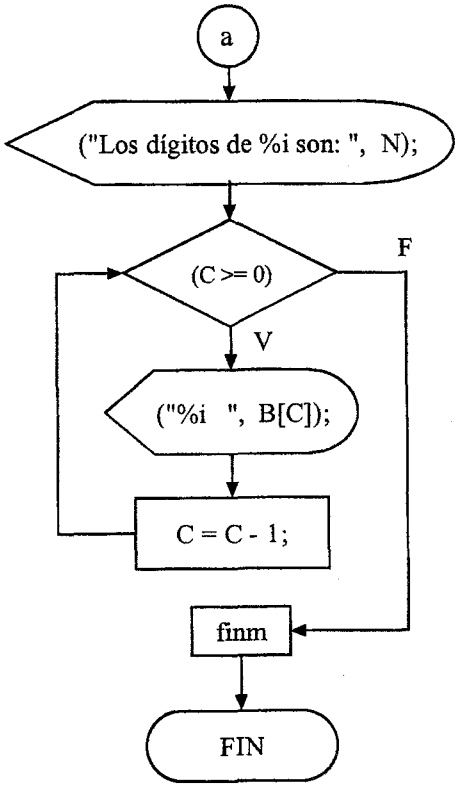
Se guarda el ultimo dígito del entero (el que corresponde a las unidades, posteriormente a las decenas, centenas, etc) en el arreglo B

Originalmente el número tiene n dígitos, con la división entre 10 se reduce el tamaño del entero en 1 dígito hasta llegar a cero.

Se incrementa el contador de dígitos en 1. El flujo regresa al ciclo 2.

Fin del ciclo 2; esto sucede cuando ya se tienen guardados todos los dígitos del entero en el arreglo B

Se asigna a C el valor del número de dígitos del entero menos 1.



/* Inicio del despliegue de los dígitos del entero*/

Se despliega un comentario

Ciclo 3:
Mientras C sea mayor o igual a cero

Se despliega el valor de cada dígito guardado en B (el usuario visualizará los dígitos en el mismo orden en que están en el entero original, separados por dos espacios)

Se decrementa en 1 el contador C. El flujo regresa al ciclo 3

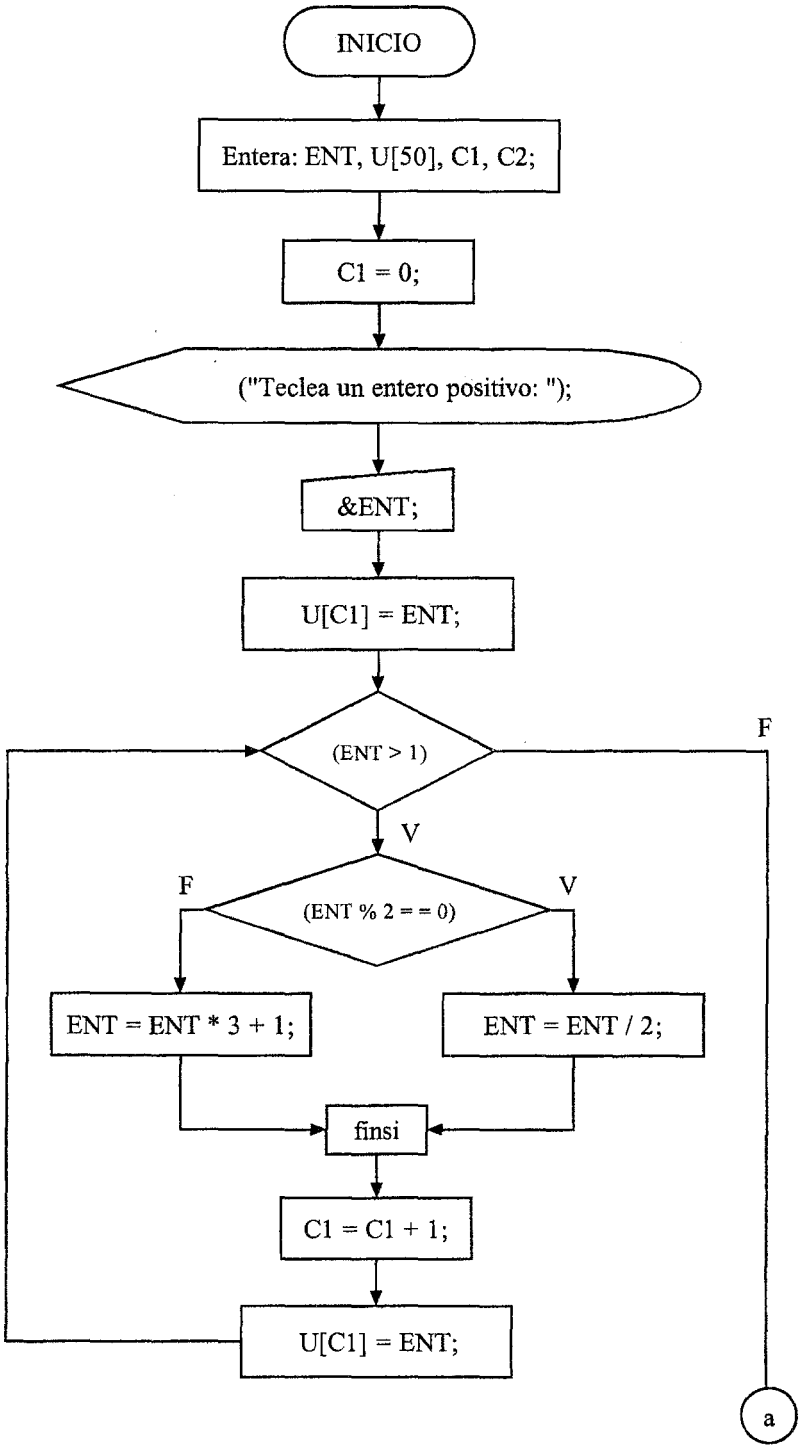
Fin del ciclo 3; esto es cuando ya fueron desplegados todos los dígitos del entero

Fin del programa

Elaborar un algoritmo para obtener la conjetura (serie) de Ulam.
Se llama conjetura de Ulam en honor del matemático S. Ulam a lo siguiente:

1. Empiece con cualquier entero positivo
2. Si es par, divídase entre 2; si es impar, multiplíquese por 3 y agréguese 1.
3. Obtenga enteros sucesivamente repitiendo el proceso hasta que el entero en curso sea 1.

Al final se obtendrá el número 1, independientemente del entero inicial. Por ejemplo, cuando el entero inicial es 26, la secuencia será: 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.



Inicio del programa

Declaración de variables: ENT (entero inicial de la serie), U[50] (arreglo para guardar todos los valores de la serie), C1 y C2 (contadores), todas como enteras

Se inicializa el primer contador en cero

Se despliega un comentario

Se lee y se guarda el valor en ENT

Ese mismo valor se guarda en la primera posición del arreglo de enteros U

Ciclo 1:
Mientras ENT sea mayor que 1

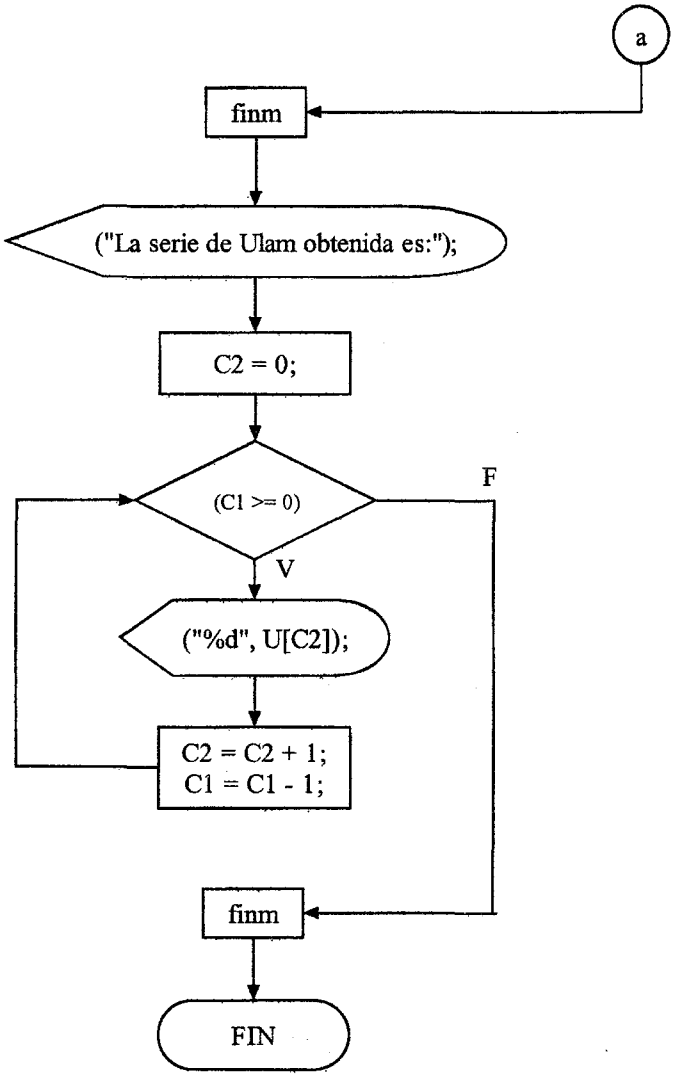
Si ENT módulo 2 es igual a cero, entonces
/* Esto verifica si el entero es par o impar */

Si es verdadero (entero par), ENT se divide entre dos.
Caso contrario (entero impar), ENT se multiplica por tres y se le agrega 1

Fin del Si entonces caso contrario

El contador 1 se incrementa en 1

El siguiente elemento de la serie (nuevo valor de ENT) se guarda en la siguiente posición del arreglo U (valor de C1)
Regresa el flujo al Ciclo 1



Fin del Ciclo 1; esto es cuando ENT llego al valor de 1 (último elemento de la serie)

Despliega el comentario:
La serie de Ulam obtenida es:

El contador dos se inicia en cero

Ciclo 2:
Mientras contador 1 sea mayor o igual a cero
/* Al llegar aquí C1 tendrá como valor el número de elementos que alcanzó la serie */

Se despliega el valor de la serie correspondiente
/* Éste estará controlado por C2 */

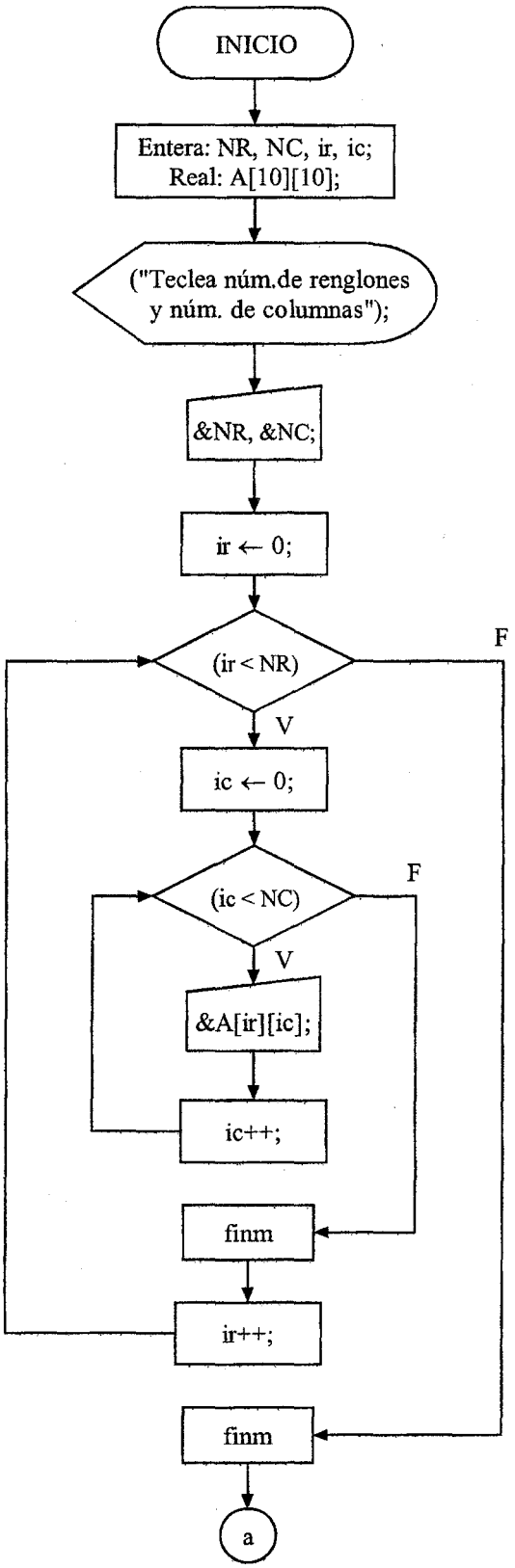
C2 se incrementa en 1. C1 se decrementa en 1
/* Cuando C1 llegue a cero, entonces se habrá - recorrido todo el arreglo que contiene la serie. Se utiliza C2 para poder imprimir en orden la serie - generada */

Fin del ciclo 2; cuando C1 es menor que cero. Esto es cuando el arreglo se recorrió por completo

Fin del programa

NOTAS: En este algoritmo se da por supuesto que el usuario proporciona un valor válido para iniciar la serie (entero positivo), y que el máximo número de elementos que ésta puede contener es 50.
La operación módulo es una operación válida sólo entre enteros que devuelve el residuo al efectuar una división entre dos de estos.
En este algoritmo se verifica que el módulo del entero y dos sea cero para saber si el primero es par (puesto que un número par dividido entre dos no deja residuo). Si la opera--- ción devuelve un número diferente de cero, entonces el entero es impar.

Elaborar un algoritmo para leer y desplegar los valores de una matriz de orden NR x NC.



Inicio del programa

Declaración de variables: NR (núm. de renglones), NC (núm. de columnas), ir, ic (índices para renglones y columnas respectivamente) y A (matriz de tamaño 10 x 10, con elementos de punto flotante)

Mensaje al usuario que pide introduzca el tamaño de la matriz (renglones y columnas)

Se leen y se guardan los dos valores dados por el usuario en las respectivas variables (NR y NC)

/* Inicio de la lectura de la matriz */

Se inicializa ir (índice para renglones)con un valor de cero

Ciclo 1:
Mientras ir sea menor que el número de renglones *NR* (dado que ir se inicializa con cero y no con uno)

Se inicializa con cero la variable ic (contador de col.)

Ciclo 2:
Mientras ic sea menor que el número de columnas *NC*

Se lee el elemento de la matriz correspondiente al indice actual de renglón y al índice actual de columna

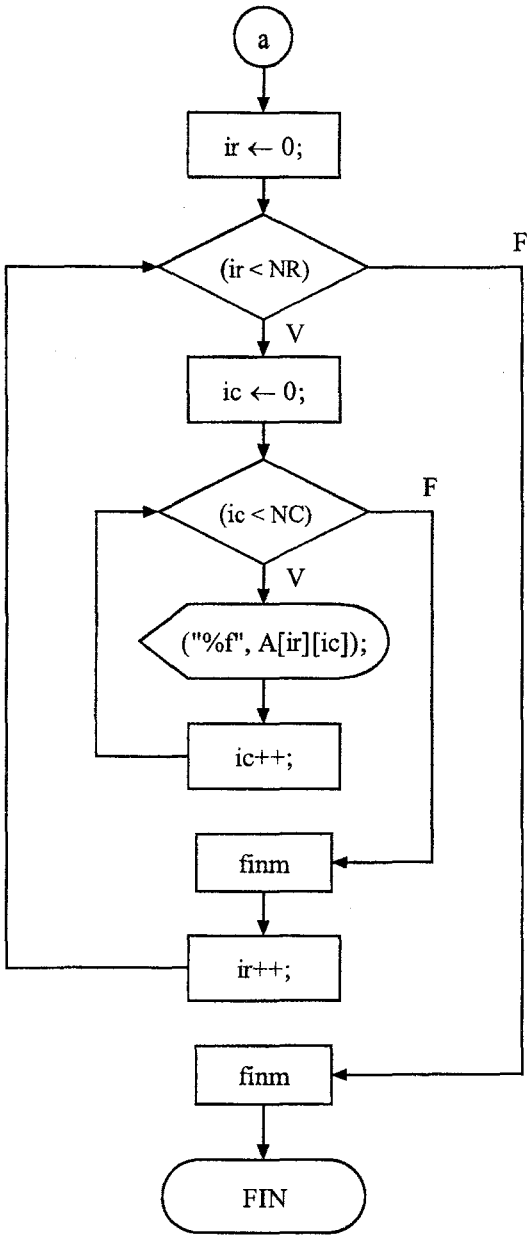
Se incrementa en 1 el contador de columnas y regresa el flujo al ciclo 2.

Fin del segundo ciclo; esto es cuando ic ya no es menor que el número de columnas (ic = = NC)

Se incrementa en 1 el valor de ir y regresa el flujo al ciclo 1.

Fin del primer ciclo; esto es cuando el contador de renglones alcanza el valor de NR (ir = = NR)

/* Fin de la lectura de la matriz */



/* Inicio del despliegue de la matriz */

Se inicializa el contador de renglones en cero

Ciclo 3:
Mientras ir sea menor que el número de renglones *NR*

Se inicializa en cero el indice de columnas

Ciclo 4:
Mientras ic sea menor que el número de columnas *NC*

Se despliega el valor actual del elemento de la matriz correspondiente al número del índice de renglón y número de índice de columna

Se incrementa en 1 el contador de columnas. El flujo regresa al ciclo 4

Fin del cuarto ciclo; esto es cuando el indice de columnas es igual al número de columnas (ic = = NC)

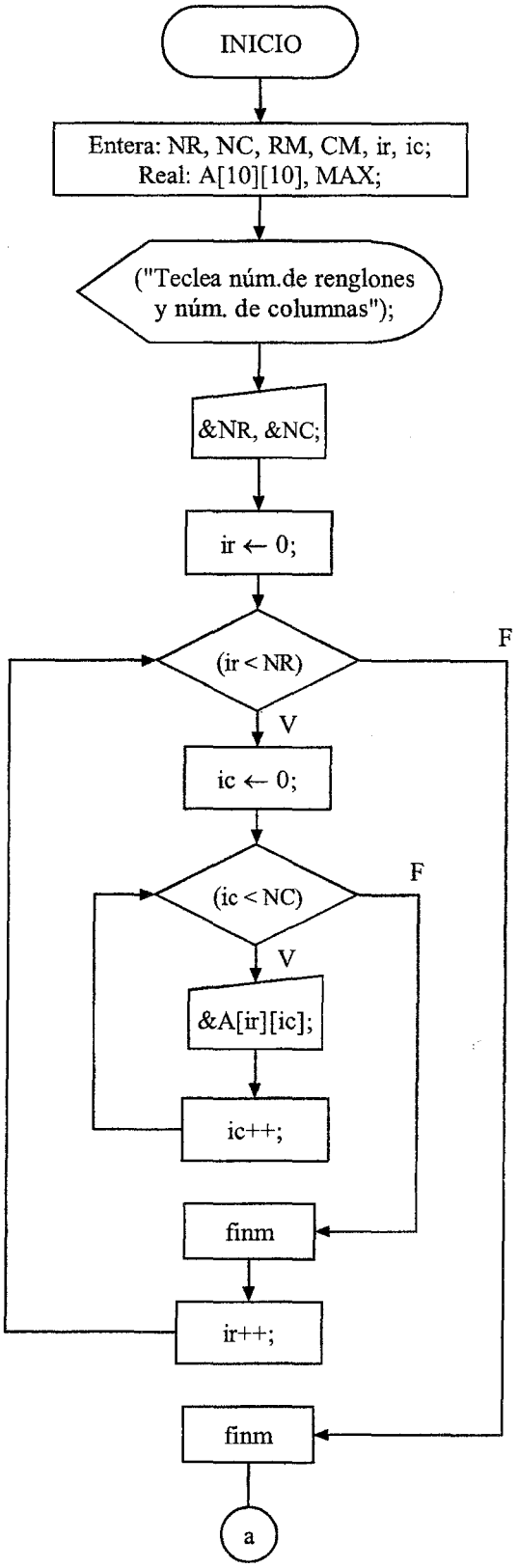
Se incrementa en 1 el contador de renglones y regresa el flujo al ciclo 3

Fin del tercer ciclo, o sea, cuando el indice de renglones sea igual que el número de renglones (ir = = NR)

/* Fin del despliegue en pantalla de la matriz */

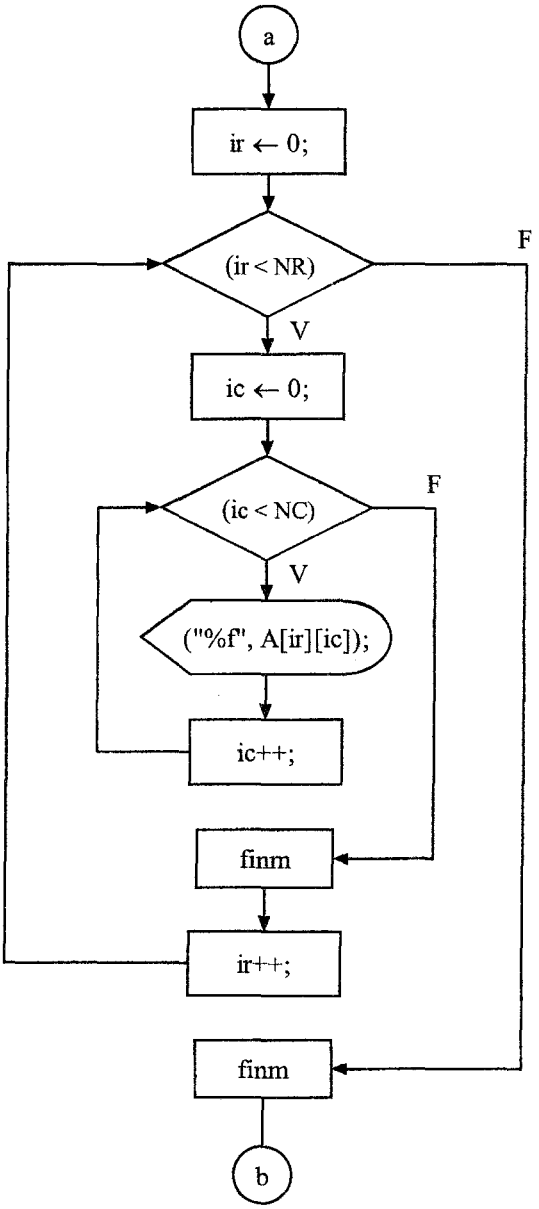
Fin del programa.

Elaborar un algoritmo para leer una matriz, desplegar sus valores e indicar cual es el valor máximo en ella, así como la posición de este último dentro del arreglo.

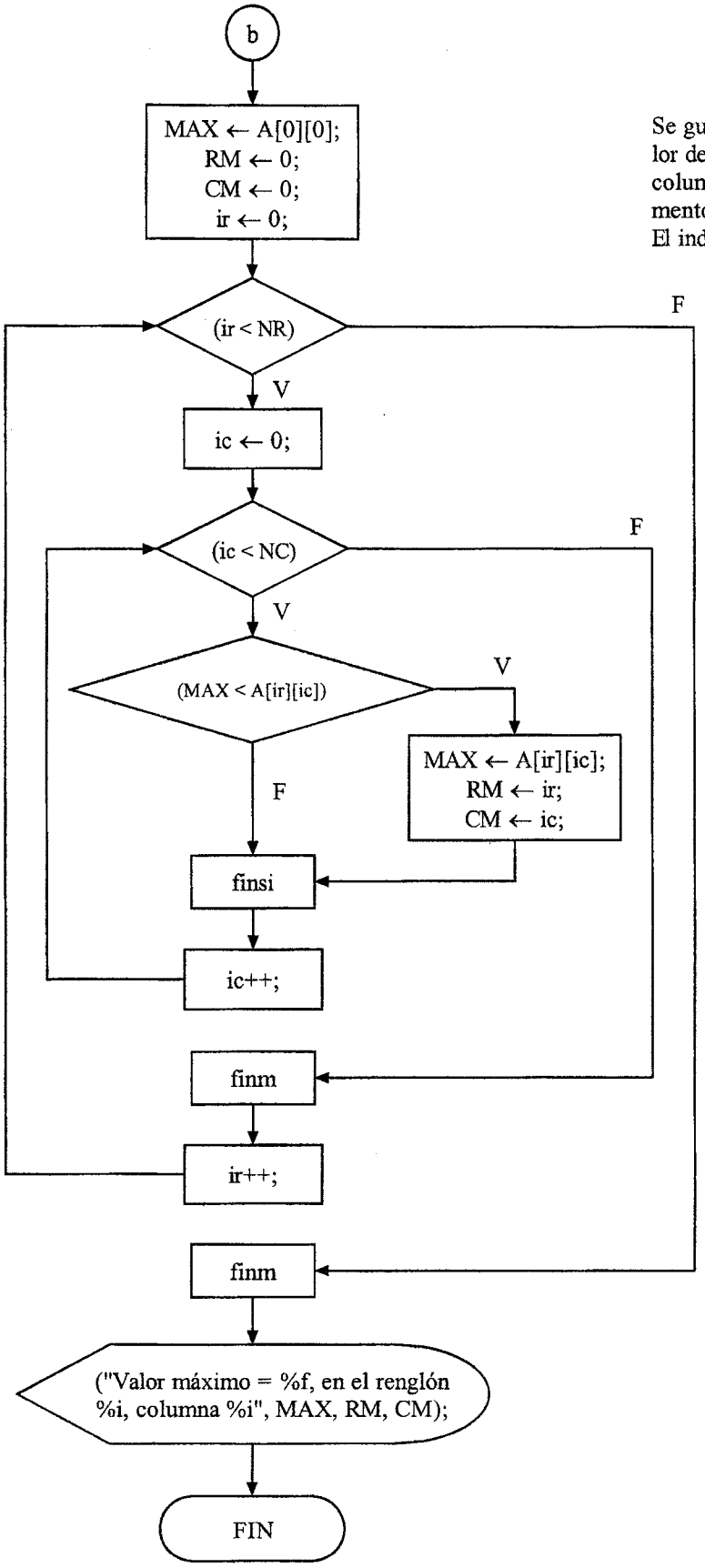


/* La lectura y despliegue de los elementos de la matriz es igual a la del algoritmo anterior. Sólo se comentarán las di-deferencias y las modificaciones en este algoritmo */

Se agregan las variables RM y CM para referenciar el número de renglón y número de columna correspondientes al elemento de máximo valor en la matriz (el cual se guardará en la variable MAX de tipo real, que también fue a --gregada)



/* Fin de la lectura y despliegue de la matriz */



Se guarda en la variable MAX (máximo elemento) el valor del primer elemento de la matriz (1er. renglón, 1era. - columna). Los valores para el renglón y columna del elemento máximo se inicializan con los del primer elemento El indice de renglones comienza en cero.

Ciclo 5:
Mientras ir sea menor que el número de renglones *NR*

El contador de columnas inicia en cero

Ciclo 6:
Mientras ic sea menor que el número de columnas *NC*

Condición:
Si el valor máximo (MAX) es menor que el elemento actual de la matriz

Este valor reemplaza al valor anterior de MAX
Los valores de los índices de renglón y de columna se asigan a las variables que contendrán el valor del renglón y columna del elemento máximo de la matriz, respectivamente

Fin del Si

Se incrementa en 1 el contador de columnas.
El flujo regresa al ciclo 6

Fin del ciclo 6; esto es cuando el valor de ic es igual al número de columnas (ic = = NC)

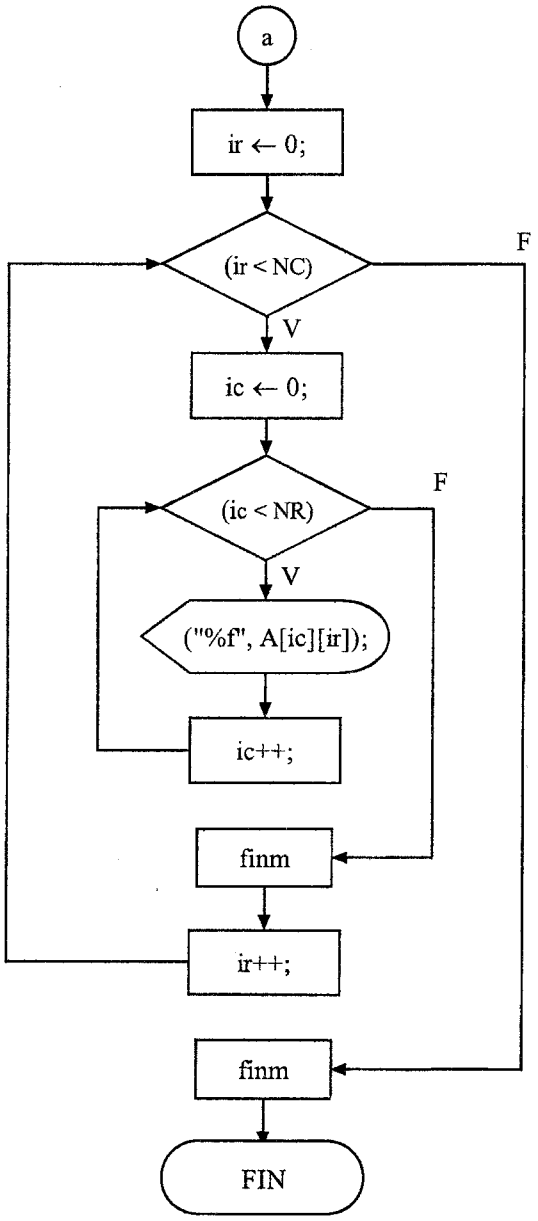
Se incrementa ir en 1. Regresa el flujo al ciclo 6

Fin del quinto ciclo, o sea, cuando el valor de - contador de renglones es igual a NR (ir = = NR)

Se despliega el mensaje **Valor máximo =** , seguido del valor del mayor elemento de la matriz (MAX), **en el renglón __, columna __** donde se desplegarán los valores del número de renglón y de columna, respectivamente, donde se encuentra el elemento anterior.

Fin del programa

Elaborar un algoritmo que lea y despliegue una matriz de orden NR x NC junto con su matriz transpuesta. NOTA: La lectura y despliegue de la matriz se omiten.



/* Después de la lectura y despliegue de la matriz */

Se inicializa el contador de renglones en cero

Ciclo 5:
Mientras ir sea menor que el número de columnas (esto es porque la matriz transpuesta será de orden inverso a la matriz dada, entonces tendrá como núm de renglones, el núm. de columnas que tiene la ma-- triz original)

Se inicializa en cero el índice para las columnas

Ciclo 6:
Mientras ic sea menor que el número de renglones (así, la matriz transpuesta tendrá como número de - columnas, el número de renglones que tenía la ma -- triz original)

Se despliega el valor del elemento de la matriz que - corresponde a los valores actuales de renglón y col. pero cambiados, o sea, A[columna][renglón]

Se incrementa en 1 el contador de columnas. Regresa el flujo al ciclo 6.

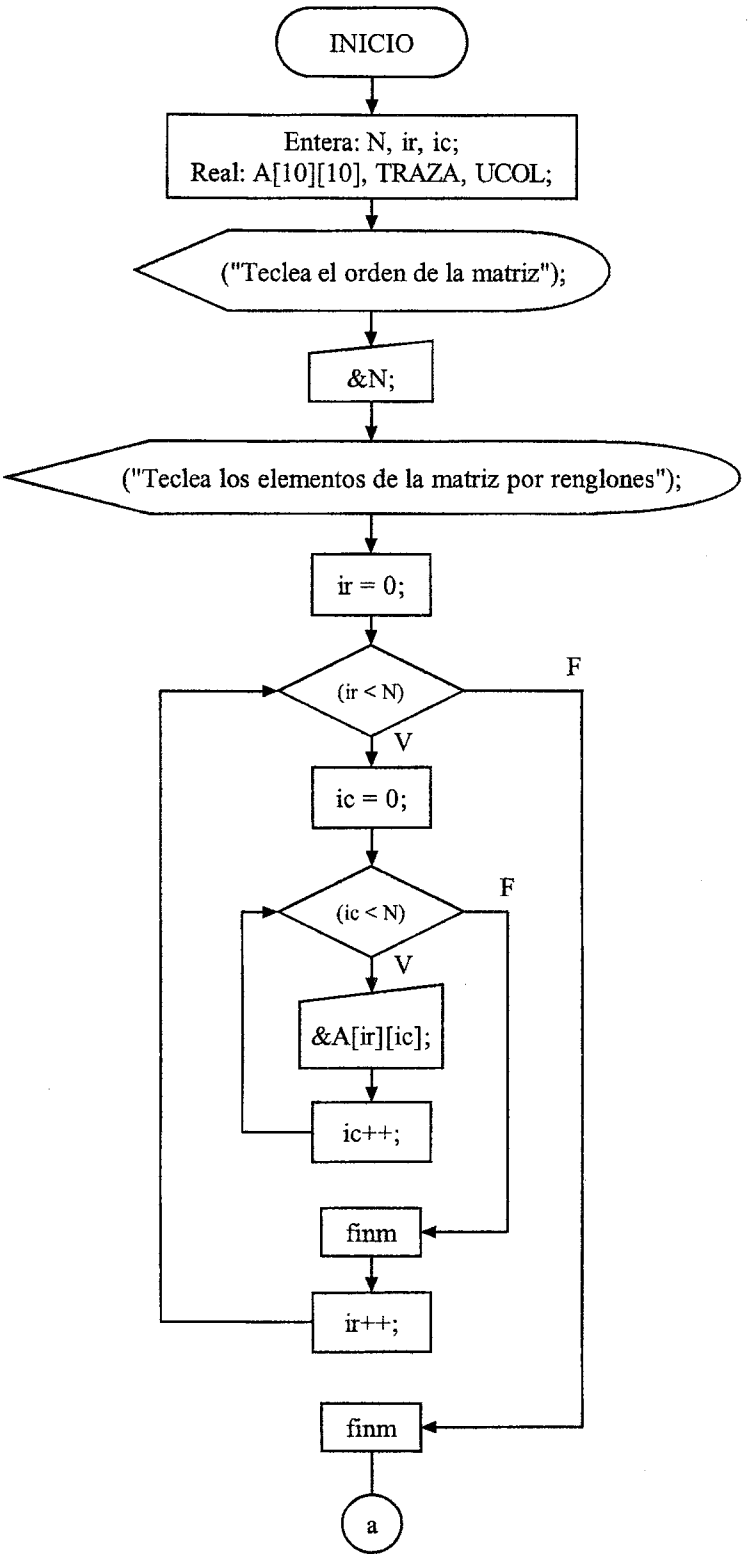
Fin del sexto ciclo, esto es, cuando ic es igual al número de renglones (ic = = NR)

Se incrementa en 1 el índice de renglones. El flujo - se va hacia el ciclo 5.

Fin del quinto ciclo, o sea, cuando ir es igual al número de columnas (ir = = NC)

Fin del programa

Realice un algoritmo que lea y despliegue una matriz de orden N (cuyos elementos son reales) y a partir de ésta, genere:
a) la traza de la matriz (La traza de una matriz es la suma de los elementos de su diagonal principal)
b) la suma de la última columna



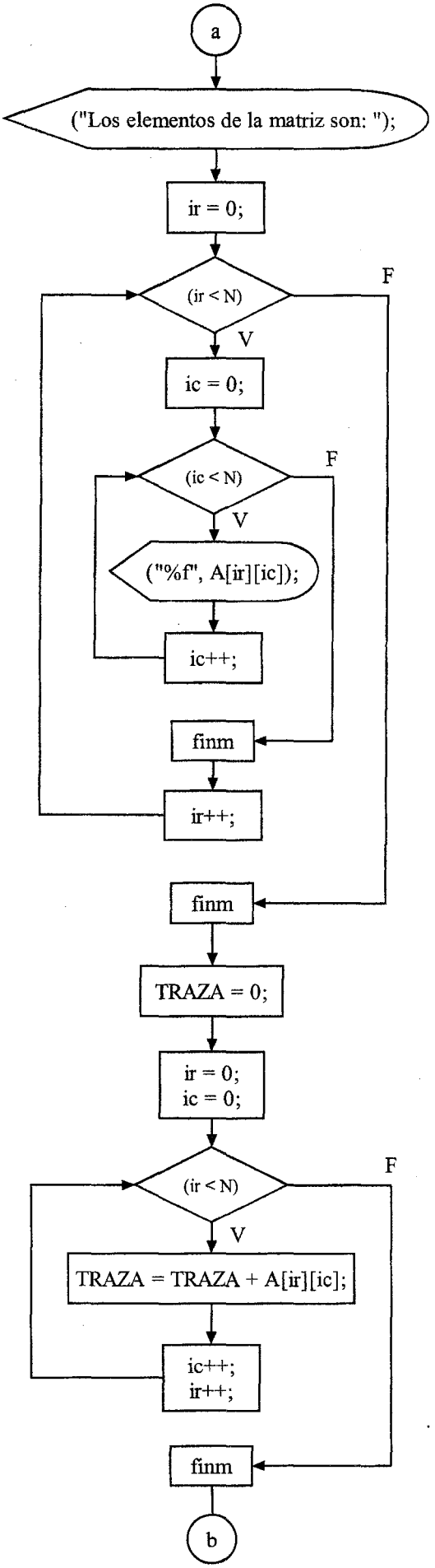
Inicio del programa

Declaración de las variables: N (orden de la matriz), ir, ic (contadores para renglones y columnas, respectivamente); y A (para la matriz), TRAZA (donde se guardará el valor de ésta) y UCOL (para la suma de los elementos de la última columna)

Se lee y se guarda el orden de la matriz en N

/* Inicio de la lectura de la matriz */

/* Fin de la lectura de la matriz */



/* Inicio del despliegue en pantalla de la matriz */

/* Fin del despliegue en pantalla de la matriz */

/* Inicio del cálculo de la traza de la matriz */

Se inicializa en cero el valor de la variable TRAZA

Se inicializan en cero los valores de los índices de renglón y de columna

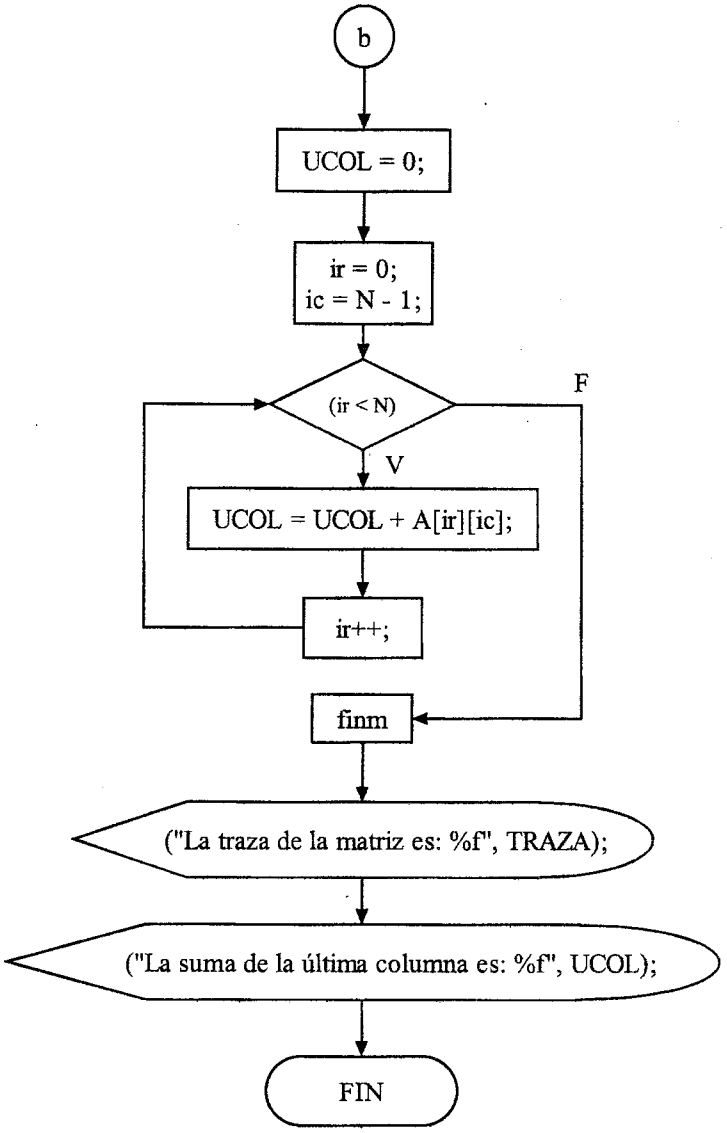
Ciclo 5:
Mientras ir sea menor que el número de renglones (orden de la matriz)

Se actualiza el valor de TRAZA sumando el elemento co-- rrespondiente del siguiente renglón.

Se incrementan en 1 los contadores ir e ic (esto debido a que en los elementos de la traza ambos índices, renglón y columna, siempre son iguales)

Fin del ciclo 5

/* Fin del cálculo de la traza */



/* Inicio de la suma de los elementos de la última columna */

Se inicializa en cero la variable UCOL

Se inicializa en cero el valor del contador de renglones
Se asigna al índice de columnas el valor N-1 (ver nota al final del algoritmo), y éste se mantiene constante ya que la columna no debe cambiar)

Ciclo 6:
Mientras ir sea menor al orden de la matriz

Se actualiza el valor de UCOL con el valor del elemento de la última columna en el siguiente renglón

Se incrementa en 1 el contador para los renglones. El flu- regresa al ciclo 6

Fin del ciclo 6. Esto sucede cuando ir alcanza el valor de N
/* Fin de la suma de la última columna */

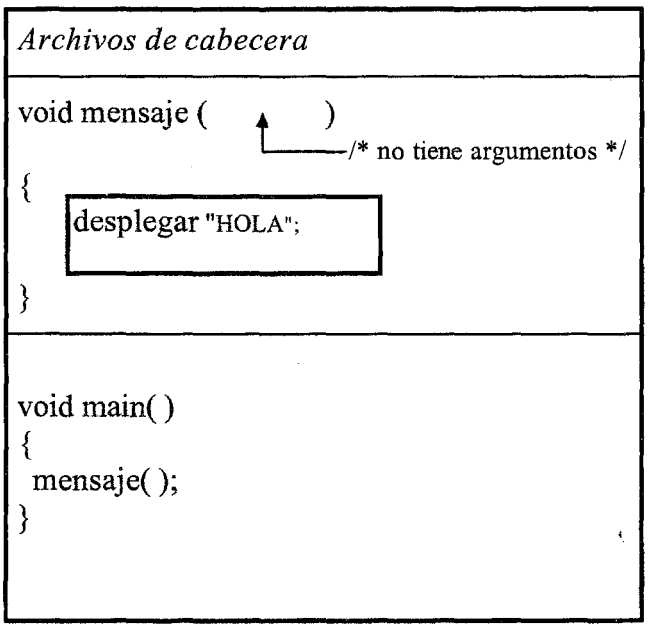
Mensaje:
La traza de la matriz es: __ , donde en el espacio se -- despliega el valor de la variable TRAZA

Mensaje:
La suma de la última columna es: __ , donde en el es- pacio se despliega la suma de los elementos de la última - columna (variable UCOL)

Fin del programa

NOTAS: 1. En el cálculo de la suma de la última columna el índice de columnas se coloca en el valor N-1 (orden de la matriz, menos uno) porque en C los arreglos inician en cero. Entonces - en C el valor real de un renglón o columna es ese valor menos uno.
2. De lo anterior también podemos entender el por que los ciclos se condicionan mientras el valor de los contadores es menor que el orden de la matriz (y no igual), ya que cuando estos - índices alcanzan el valor del orden menos uno, en realidad han contado el número real de ren- glones o de columnas, según el caso del que se trate.

Elaborar un algoritmo que llame a una función, la cual desplegará un mensaje en pantalla



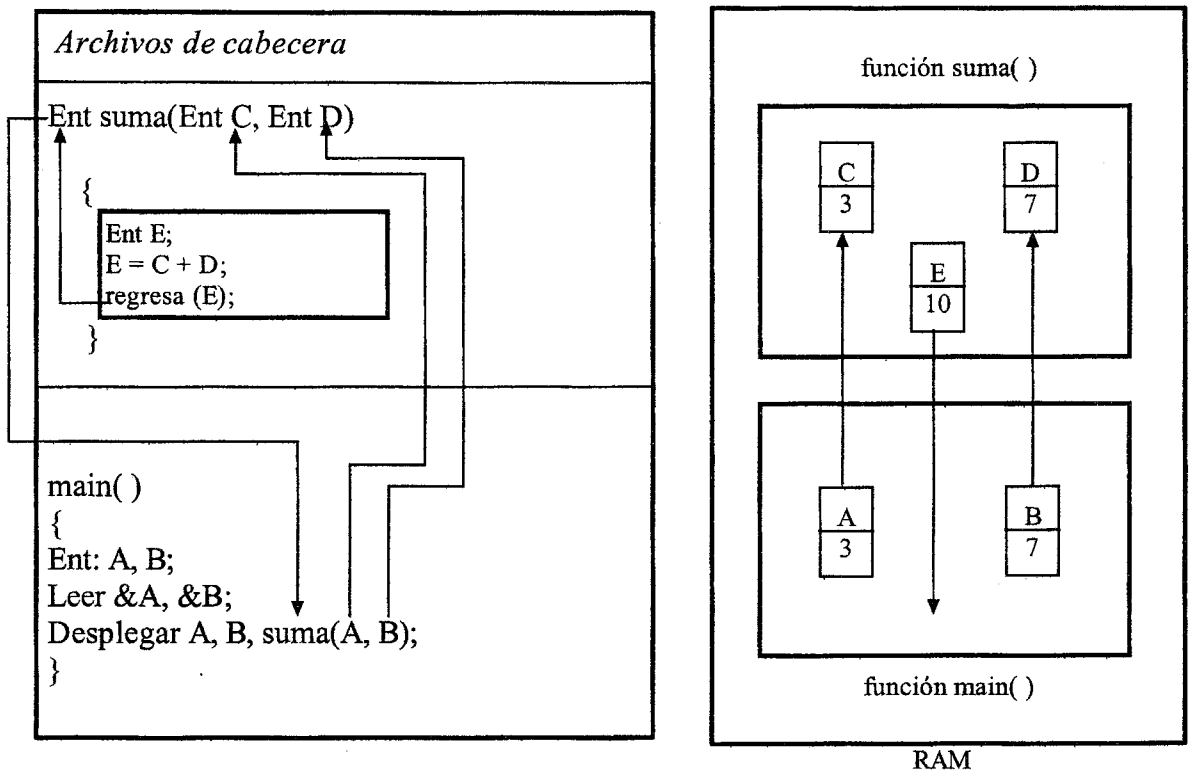
/* La función *mensaje* está definida como una fun-- ción de tipo void de tal forma que no regresa valores hacia la función principal, y en este caso, - tampoco está recibiendo parámetros provenientes de dicha función.

En la función main únicamente se manda llamar a la función mensaje, pero no se están pasando parámetros hacia ella.

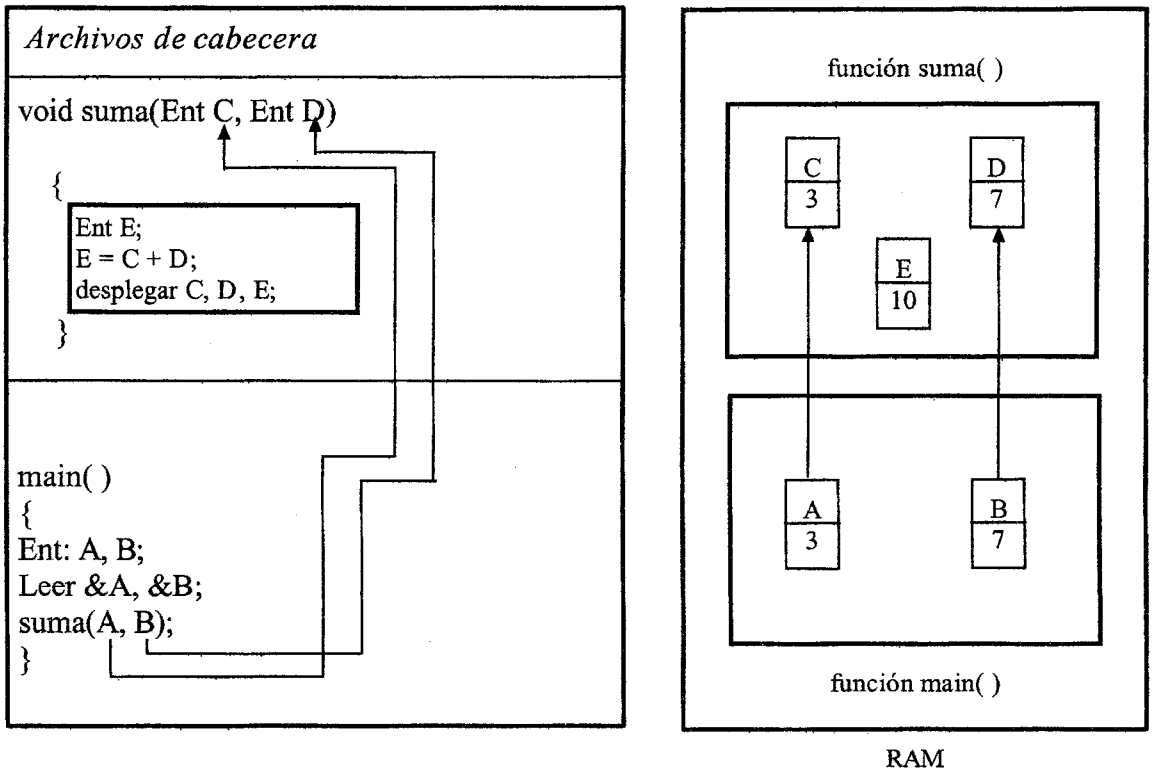
*/

Elaborar un algoritmo que lea dos valores y llame a la función *suma* que devuelve como resultado la suma de los dos valores. Se presentan distintos casos para ilustrar cómo esta función puede o no tener argumentos y regresar o no valores a la función main( ). En todos ellos se ejemplifica el paso de los valores a variables con pequeños bloques para facilitar su entendimiento.

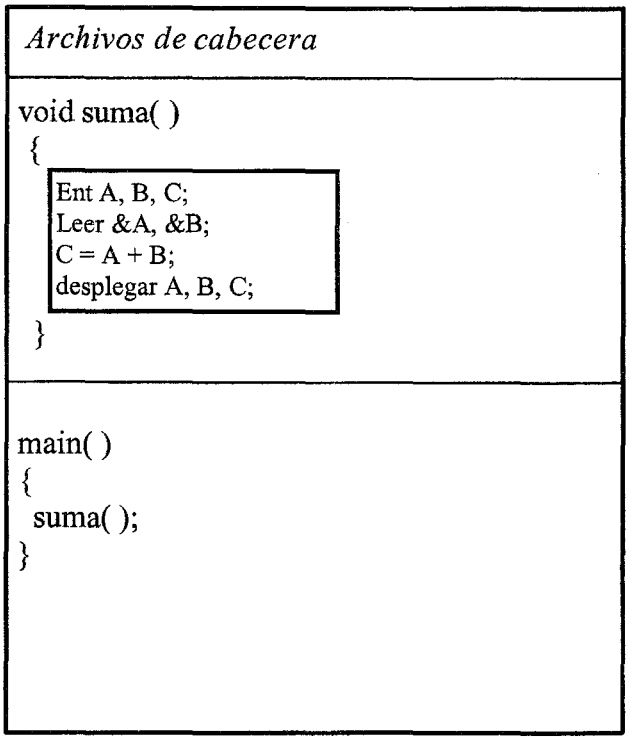
1er. Caso: La función *suma* recibe de la función main dos argumentos de tipo entero y al estar definida (suma) como una función de tipo entero también regresa a la función main un valor de tipo entero.

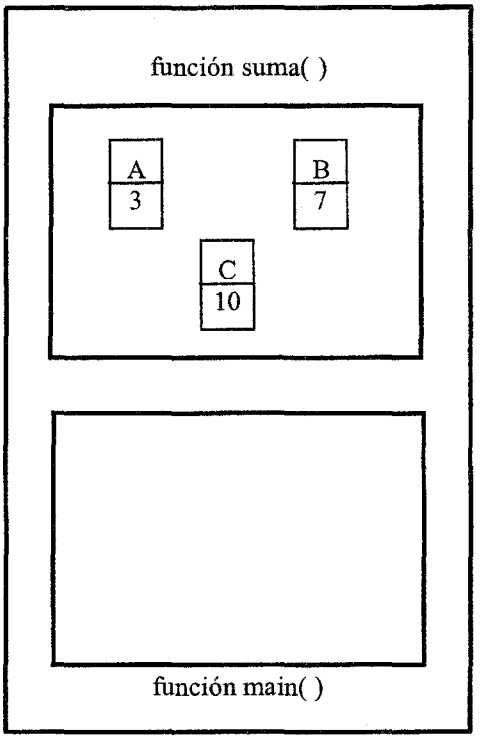


2o. Caso: La función *suma* recibe de la función main dos argumentos de tipo entero, pero al estar definida como una función de tipo void no regresa hacia main valor alguno.



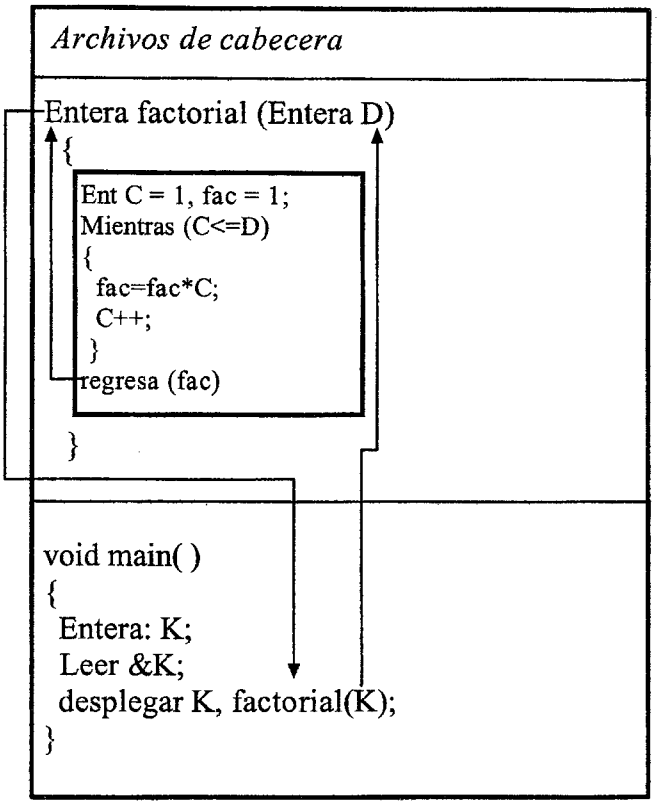
3er. Caso: La función *suma* no recibe argumentos de la función main y al estar definida como una función de tipo void tampoco regresa valores hacia la función principal.





RAM

Elaborar un algoritmo que contenga una función llamada *factorial* para calcular el factorial de un entero cualquiera

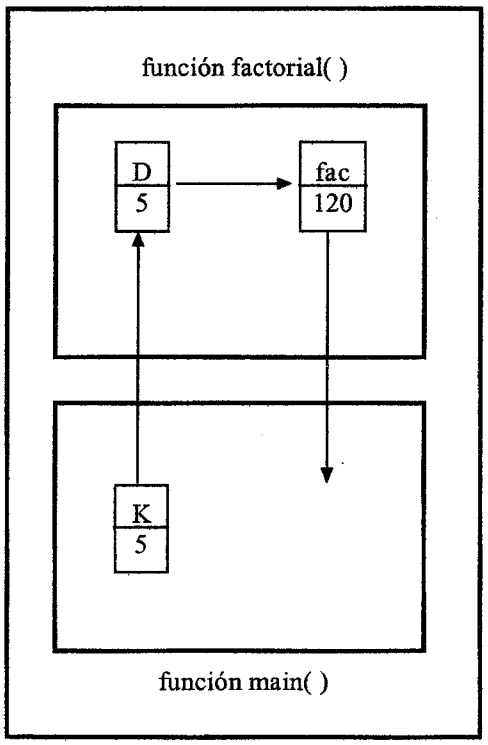


/* Se define a factorial como una función de tipo entero (esto significa que va a regresar como resultado un valor entero), y que recibe en D un argumento de tipo entero también, el cual proviene de la función principal, que en este caso corresponde al valor contenido en la variable K.

Entonces D, C y fac son variables locales de la función factorial, que se utilizan para calcular el factorial de D (el contenido de K,se copia en D) y el resultado se guarda en la variable fac, el cual es devuelto hacia la funcion main( ) para desplegar el factorial de K (o el factorial de D que finalmente son iguales)

La función main( ) únicamente utiliza una variable K de tipo entero.

Se lee y se guarda un valor en esa variable, se despliega y después se llama a la funcion factorial dándole a la variable K como argumento, como se explico con anterioridad */



RAM

Elaborar un algoritmo para calcular las combinaciones de N elementos tomados en grupos de R elementos, (${}_NC_R$ = N! / [(N-R)! * R!]), utilizando para ello la función *factorial*, para realizar el cálculo de los factoriales requeridos.

Archivos de cabecera

```
Entera factorial (Entera D)
{
  Ent C = 1, fac = 1;
  Mientras (C<=D)
  {
   fac=fac*C;
   C++;
  }
  regresa (fac)
}
```

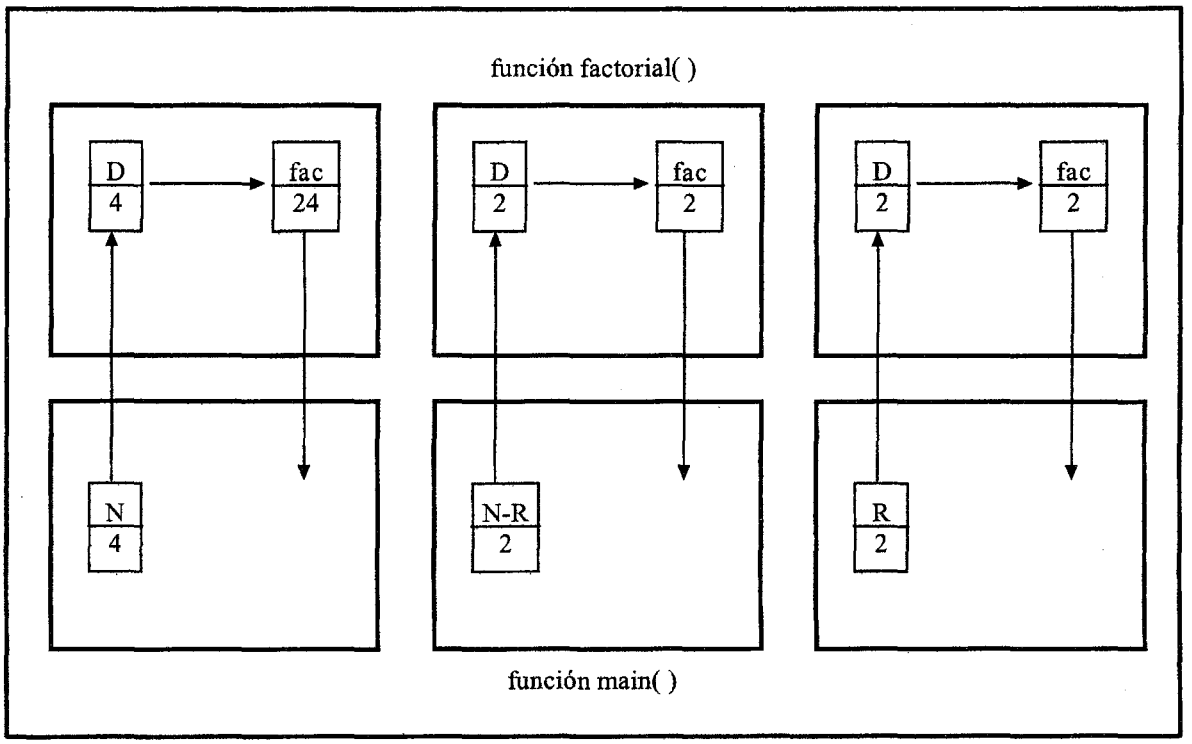
```
void main( )
{
Entera: N, R, Comb;
Desplegar "Dame los valores para N y R";
Leer &N, &R;
Comb = factorial(N) / [factorial(N-R)*factorial(R)];
Desplegar "NCR = ", Comb;
}
```

/* En este algoritmo se utiliza la función factorial explicada en el algoritmo anterior.

En este caso la función es llamada para calcular -- los factoriales de N, N-R y de R, respectivamente.

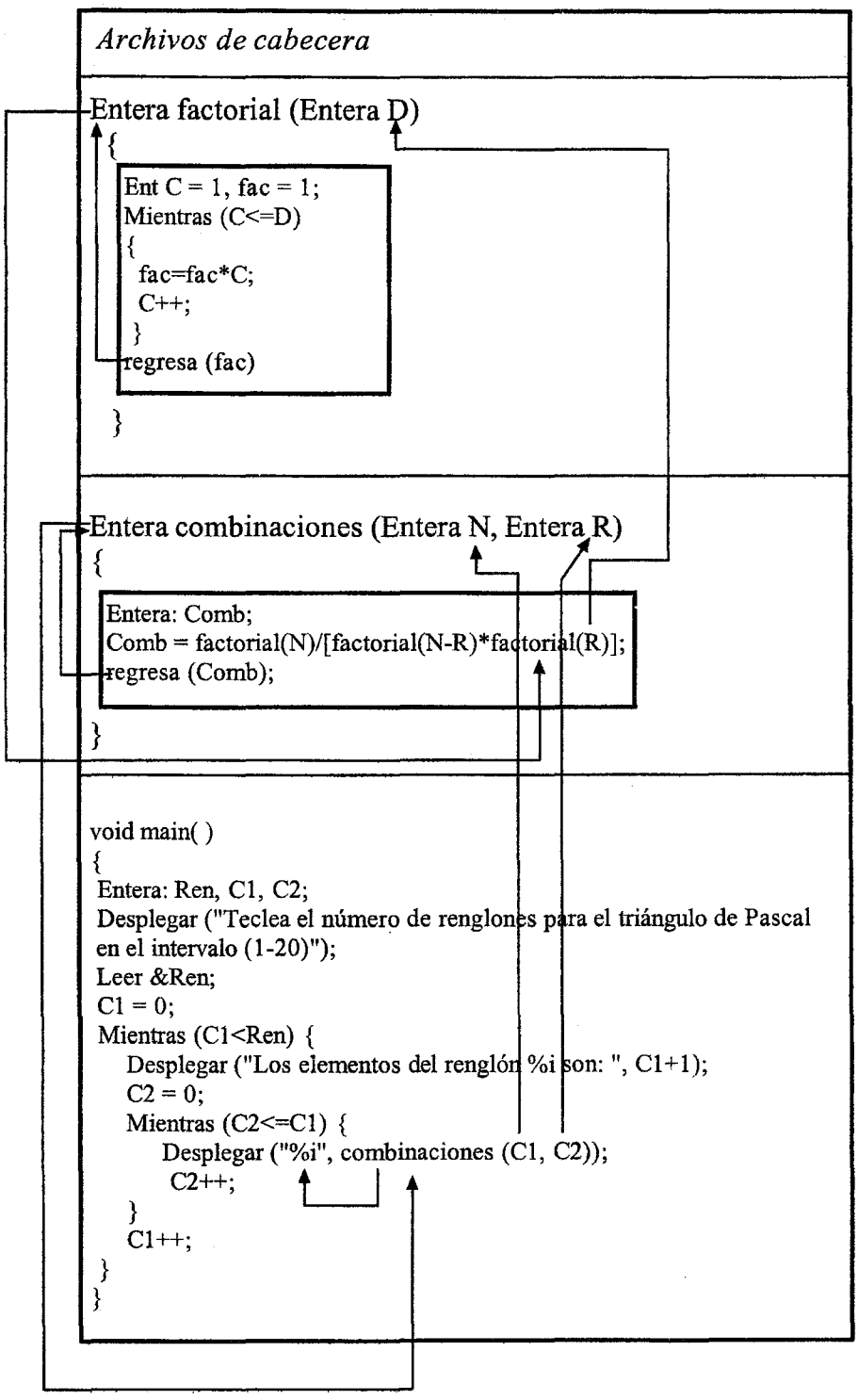
Primeramente se guarda el valor de N en D para - obtener N!, después el valor de N-R se copia a D para obtener (N-R)! y lo mismo sucede para R.

*/



RAM

Elaborar un algoritmo para generar los valores del triángulo de Pascal, hasta un número determinado de renglones, proporcionado por el usuario.



Archivos de cabecera

```
Entera factorial (Entera D)
{
  Ent C = 1, fac = 1;
  Mientras (C<=D)
  {
    fac=fac*C;
    C++;
  }
  regresa (fac)
}
```

```
Entera combinaciones (Entera N, Entera R)
{
  Entera: Comb;
  Comb = factorial(N)/[factorial(N-R)*factorial(R)];
  regresa (Comb);
}
```

```
void main( )
{
 Entera: Ren, C1, C2;
 Desplegar ("Teclea el número de renglones para el triángulo de Pascal
 en el intervalo (1-20)");
 Leer &Ren;
 C1 = 0;
 Mientras (C1<Ren) {
    Desplegar ("Los elementos del renglón %i son: ", C1+1);
    C2 = 0;
    Mientras (C2<=C1) {
        Desplegar ("%i", combinaciones (C1, C2));
        C2++;
    }
    C1++;
 }
}
```

/* La función factorial se vió ya en algoritmos anteriores, y en este caso su funcionamiento en el mismo */

/* La función combinaciones es una función que toma dos argumentos de tipo entero, en este caso llamados N y R, y que devuelve -- por resultado un valor entero obtenido de --- la fórmula para combinaciones, para lo cual se utiliza a su vez a la función factorial dentro de esta función */

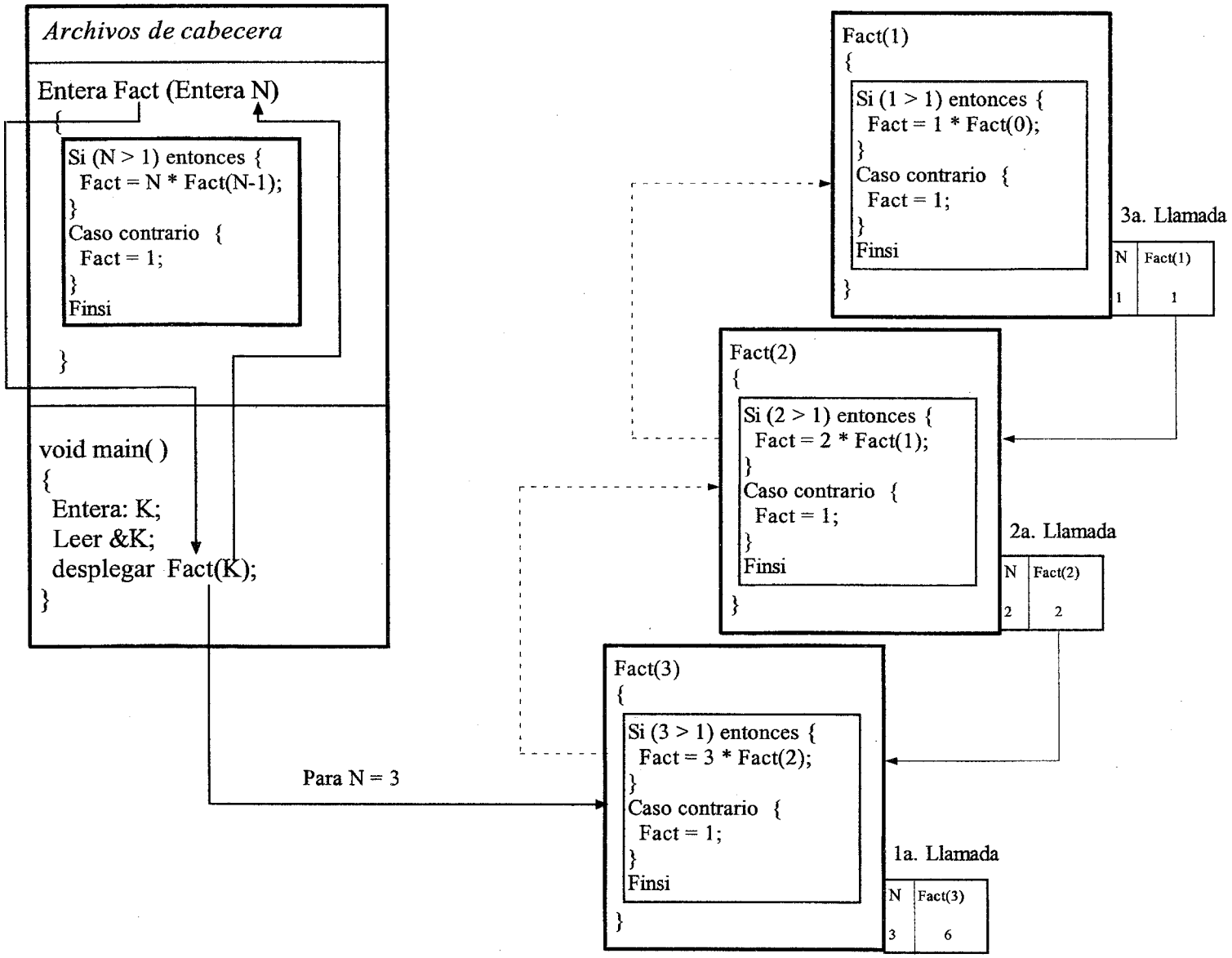
/* Función principal:

Se declaran las variables Ren (número de --- renglones para el triángulo de Pascal), C1 y C2 (contadores) todas de tipo entero.
Se solicita al usuario teclear el número de --- renglones en un rango entre 1 y 20. Dicho -- valor se lee y se guarda en la variable Ren.

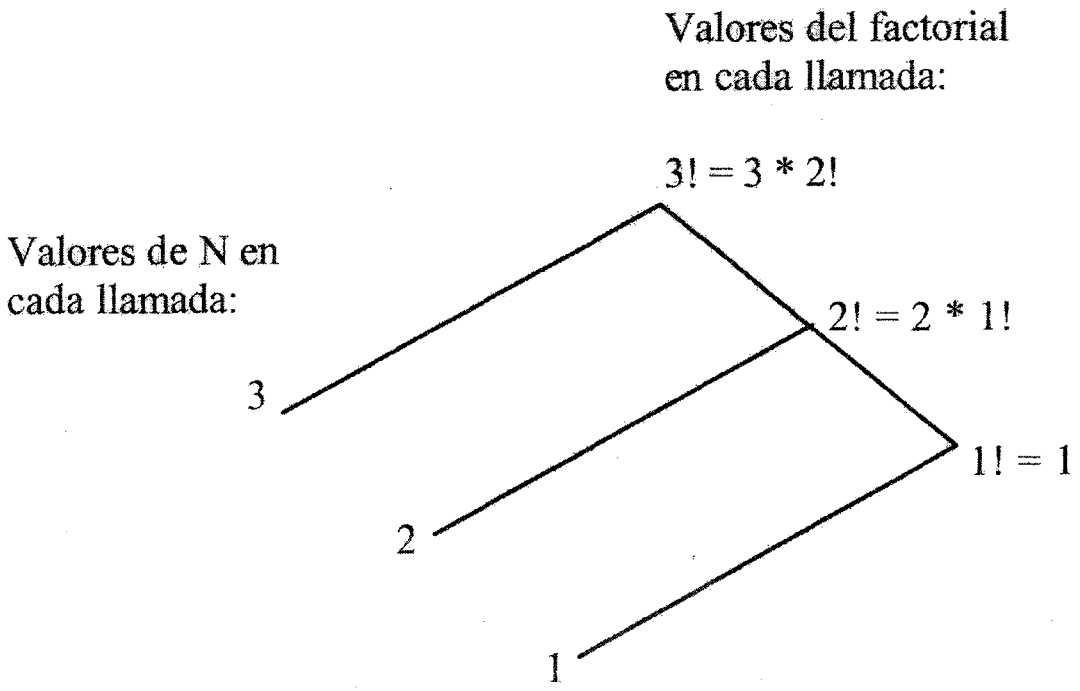
Se despliegan los elementos del triángulo de Pascal por renglones, para lo cual se llama a la funcion combinaciones, para obtener los elementos de cada renglón del triángulo de -- Pascal.

*/

Elaborar un algoritmo para calcular el factorial de un entero N utilizando una función recursiva (función que puede llamarse a si misma).



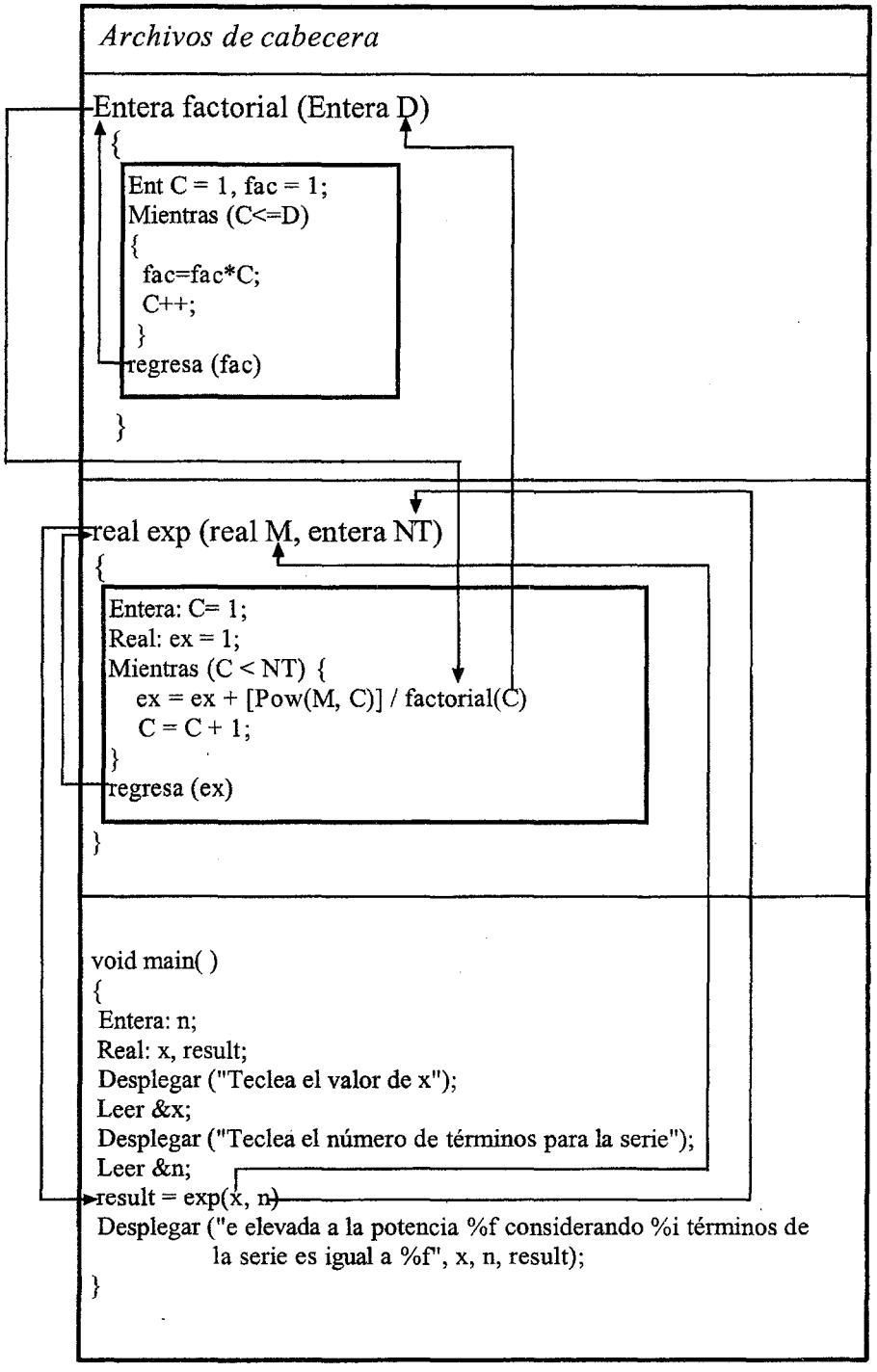
El proceso recursivo del cálculo del factorial para N = 3 se puede representar esquemáticamente como:



Elaborar un algoritmo para calcular

$$e^x = \exp(x) \approx \sum_{k=0}^{N} x^k / k! = x^0/0! + x^1/1! + x^2/2! + \ldots + x^N/N!$$

donde el usuario proporcionará el valor de x y el número de términos a considerar para la serie.



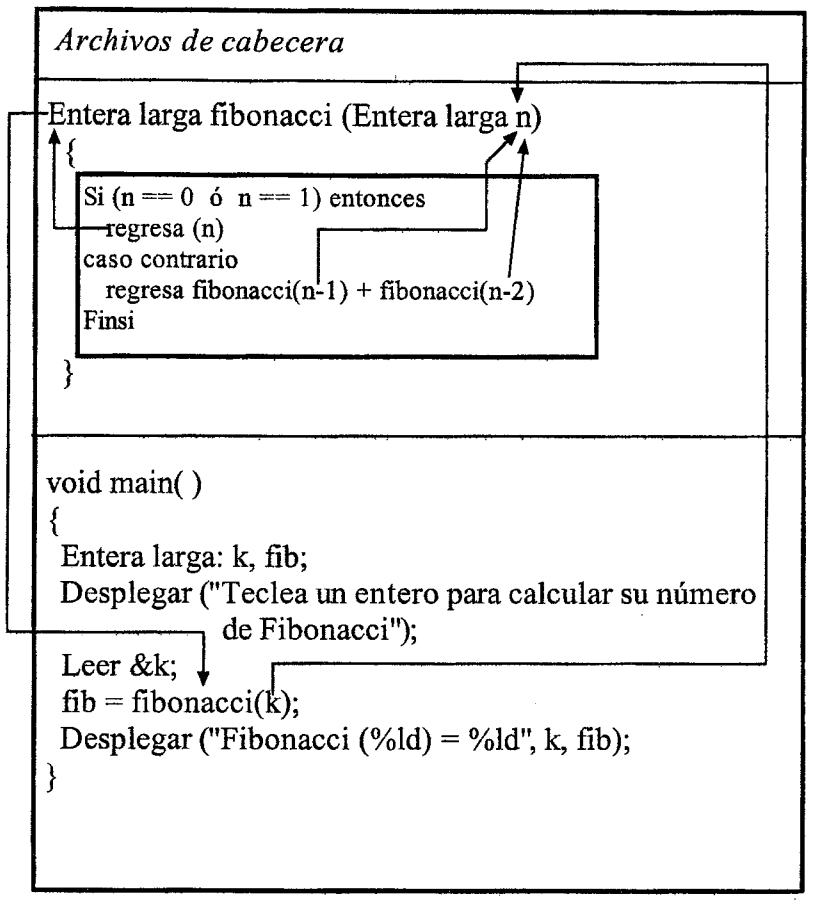
```
Archivos de cabecera

Entera factorial (Entera D)
{
  Ent C = 1, fac = 1;
  Mientras (C<=D)
  {
    fac=fac*C;
    C++;
  }
  regresa (fac)
}

real exp (real M, entera NT)
{
  Entera: C= 1;
  Real: ex = 1;
  Mientras (C < NT) {
     ex = ex + [Pow(M, C)] / factorial(C)
     C = C + 1;
  }
  regresa (ex)
}

void main( )
{
 Entera: n;
 Real: x, result;
 Desplegar ("Teclea el valor de x");
 Leer &x;
 Desplegar ("Teclea el número de términos para la serie");
 Leer &n;
 result = exp(x, n)
 Desplegar ("e elevada a la potencia %f considerando %i términos de
             la serie es igual a %f", x, n, result);
}
```

/* La función factorial se vió ya en algoritmos anteriores, y en este caso su funcionamiento en el mismo */

/* Se define una funcion real o de punto flotante llamada exp la cual recibirá dos argu -- mentos provenientes de la función principal: uno de tipo real que proviene de la variable x y que es recibido en M (valor de x) y otro de tipo entero proveniente de n, recibido en NT (número de términos que se considerarán en la serie). Se definen además dos variables internas para la función que son: C (contador) y ex (para el valor aproximado de la exponencial), esta última de tipo real. Como recordará Pow es una función de librería de C y factorial es una función definida antes de exp */

/* En la función principal se definen las variables n (número de términos de la serie) como entera y x (valor de x) y result (para evaluar - la exponencial de x) ambas como punto flo--- tante.
Se leen y se guardan los valores para x y n -- respectivamente y se manda llamar la funcion exp donde sus parámetros serán precisamente x y n, guardándose el resultado generado - en la variable result.
Finalmente se despliega un mensaje indicando el valor obtenido para la exponencial con ese- valor de x y para el número de términos es-- pecificado */

Elaborar un algoritmo para calcular los números de la serie de Fibonacci por medio de una función recursiva.



/* Se define la función fibonacci como entera larga que recibe un argumento n también del tipo entero largo (esto debido a que los números de fibonacci crecen rápidamente), como una función recursiva, en la que si el entero es 0 ó 1 se regresa ese valor, y en caso contrario se regresa la suma de los dos números de fibonacci anteriores (aquí es donde se genera la recursión) */

/* La función principal declara dos variables del tipo entero largo: k (entero del que se obtendrá su número fibonacci) y fib (para guardar el resultado generado por la función).
Se solicita al usuario teclear un entero para el cual se obtendrá su fibonacci; este valor se lee y se guarda en k para ser pasado al argumento n de la función. El resultado de la función se almacena en fib.
Finalmente se despliega un comentario indicando el número del que se obtuvo el fibonacci así como el correspondiente valor */

Esta obra se terminó de imprimir
en diciembre de 2002
en el taller de imprenta del
Departamento de Publicaciones
de la Facultad de Ingeniería
Ciudad Universitaria, México, D.F.
C.P. 04510

Secretaría de Servicios Académicos

El tiraje consta de 250 ejemplares
más sobrantes de reposición