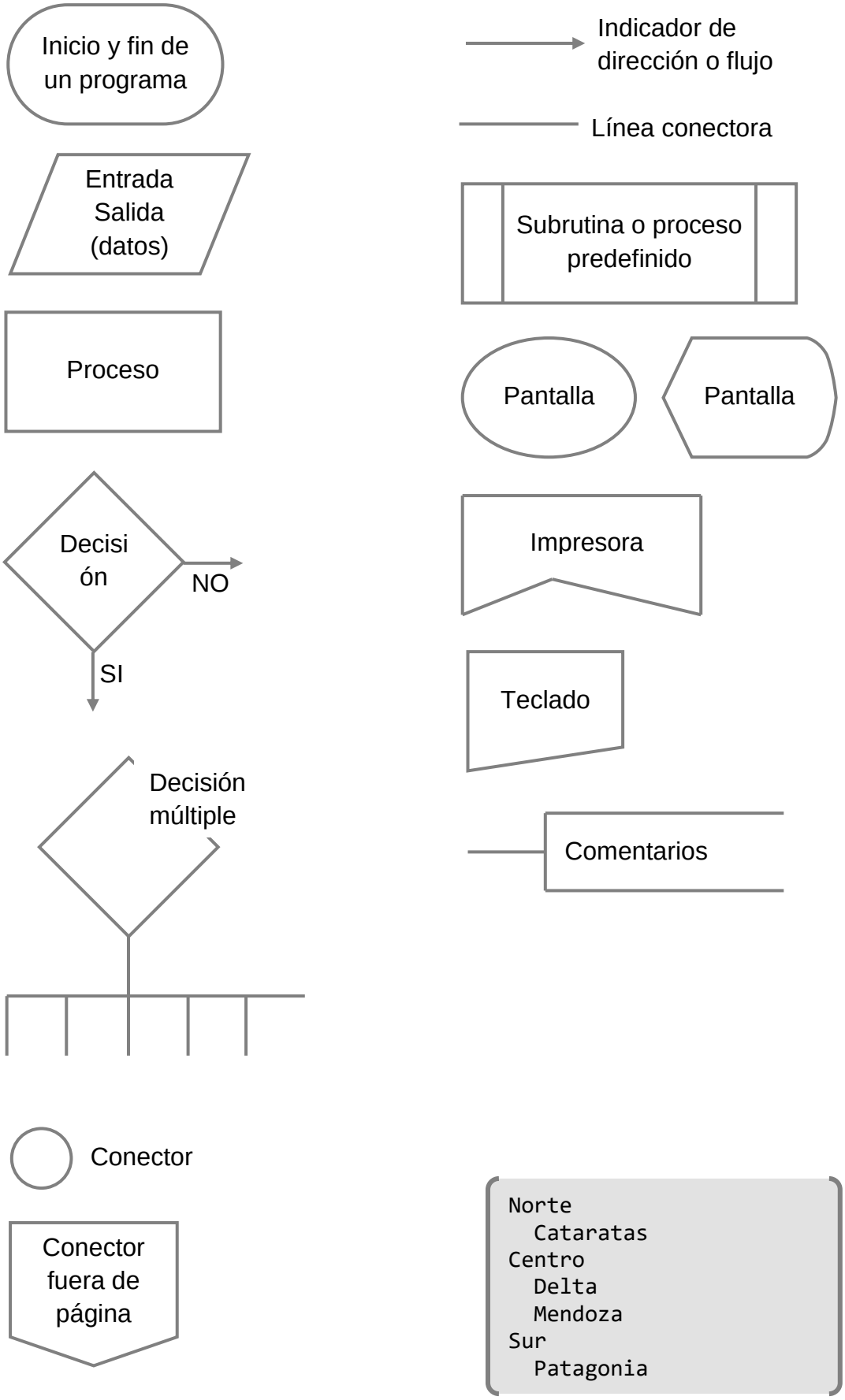
Tabla 2.1. Símbolos de diagrama de flujo. Página 102



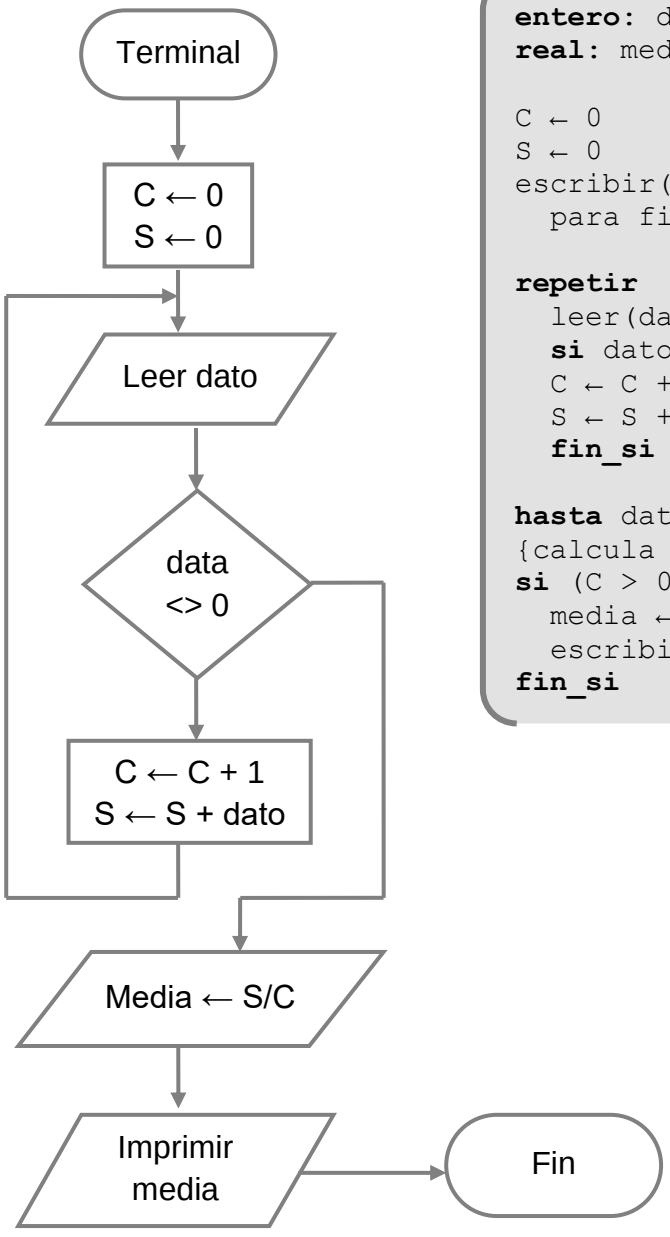

```
Norte
  Cataratas
Centro
  Delta
  Mendoza
Sur
  Patagonia
```

Ejemplo 2.07 página 104

Algoritmo

```
1. Inicializar contador de números C y variable suma S
2. Leer un número
3. Si el número leído es cero:
   _ calcular la media
   _ imprimir la media
   _ fin del proceso
   Si el número leído no es cero:
   _ calcular la suma
   _ incrementar en uno el contador de números
   _ ir al paso 2
4. Fin
```

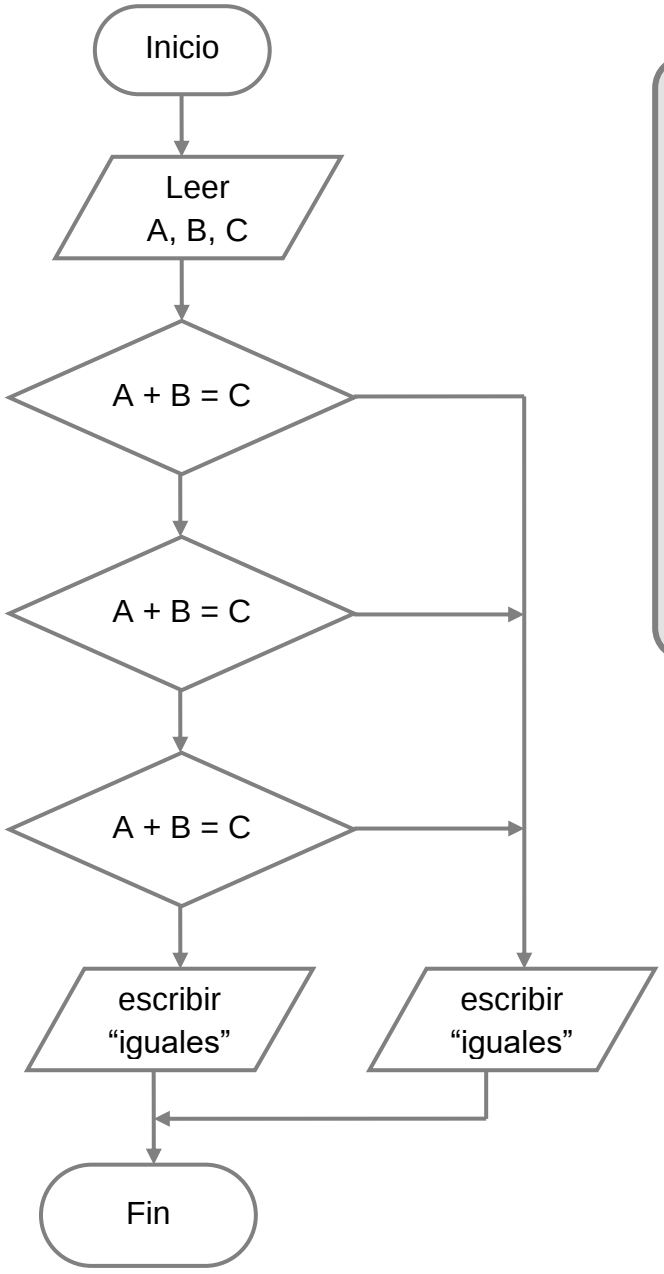
Diagrama de flujo




```
entero: dato, C
real: media, S

C ← 0
S ← 0
escribir('Datos numéricos;
  para finalizar se introduce 0')

repetir
  leer(dato)
  si dato <> = 0 entonces
  C ← C + 1
  S ← S + dato
  fin_si

hasta dato = 0
{calcula la media y la escribe}
si (C > 0) entonces
  media ← S/C
  escribir(media)
fin_si
```

Ejemplo 2.11 página 109

```
1. Leer los tres valores: A, B y C
2. Si A + B = C escribir “Iguales” y parar
3. Si A + C = B escribir “Iguales” y parar
4. Si B + C = A escribir “Iguales” y parar
5. Escribir “Distintas” y parar
```




```
entero: a, b, c
inicio
escribir('test con tres números:')
leer(a, b, c)

si (a + b = c) entonces
   escribir('Son iguales' a '+' b '=' c)
  sino si (a + c = b) entonces
   escribir('Son iguales' a '+' c '=' b)
  sino si (a + c = b) entonces
   escribir('Son iguales' b '+' c '=' a)
  sino
   escribir('Son distintas')
  fin_si
 fin_si
fin_si
fin
```

Ejemplo 2.12 página 111

```
1. Ingresar nombre
2. Ingresar horas
3. Establecer tarifa = 15
4. Establecer salario_bruto = 0
5. Si horas =< 35:
   _ salario_bruto = horas * tarifa
   Si horas > 35:
   _ salario_bruto = ((horas – 35) * tarifa * 1.5) +
      35 * tarifa
6. Si salario_bruto =< 1000:
   _ salario_neto = salario_bruto
   Si salario_bruto > 1000 && =< 1400:
   _ salario_neto = 1000 + ((salario_bruto - 1000) * 0.75)
   Si salario_bruto > 1400:
   _ salario_neto = 1000 + (400 * 0.75) +
      ((salario_bruto - 1400) * 0.45)
7. Escribir nombre, salario_bruto, tasas y salario_neto
8. Fin
```

```
tarifa: 15

inicio
escribir('Introducir nombre y horas')

//cáculo del salario bruto
si (horas =< 35) entonces
   salario_bruto = horas * tarifa
  sino si (horas > 35) entonces
   salario_bruto = 35 * tarifa + 1.5 * (horas – 35) * tarifa
  fin_si

//cálculo del salario neto
si (salario_bruto =< 1000) entonces
   salario_neto = salario_bruto
  sino si (salario_bruto > 1000) && (s_b =< 1400) entonces
   salario_neto = 1000 + (salario_bruto – 1000) * 0.75
  sino si (salario_bruto > 1400) entonces
   salario_neto = 1000 + 400 * 0.75 * (salario_bruto – 1400) * 0.45
  fin_si

escribir('Nombre, ' + salario_bruto + ', ' + tasas + ', ' +
  salario_neto)
fin
```

Ejercicio 2.1.a página 111

```
1. salir de casa
2. dirigirse a un teléfono público
3. cargarle monedas
4. marcar número
5. terminar llamada/colgar
6. si quedó vuelto:
   _ retirarlo
   si no:
   _ abandonar teléfono público
7. volver a casa
8. fin
```

Ejercicio 2.1.b página 111

```
1. reunir elementos:
  _ bol
  _ batidor
  _ sartén
  _ cuchara madera
  _ huevos
  _ queso cremoso
  _ sal fina
  _ aceite
2. colocar la sal fina y los hevos en el bol
3. batir con batidor hasta que estén espumosos
4. colocar sartén a fuego medio con una pizca de aceite
5. cuando sartén esté caliente, verter los huevos batidos
6. cuando huevos se comiencen a cocinar por debajo:
  _ colocarles encima unas rebanadas de queso cremoso
  _ envolver el queso con los bordes de huevo
7. esperar que se termine de cocinar
8. servir
9. fin
```

Ejercicio 2.1.c página 111

```
1. retirar rueda
2. desarmar uno de los bordes de la cubierta
3. retirar cámara
4. inflar cámara
5. si se dispone de pileta:
   _ sumergir cámara para detectar lugar de pinchazo
  si no:
   _ rociar cámara con agua con jabón
6. secar cámara
7. raspar en zona de pinchazo
8. colocar parche
9. esperar unos minutos
10. montar cámara
11. montar borde de cubierta
12. colocar rueda
```

Ejercicio 2.1.d página 111

```
1. reunir elementos:
  _ sartén
  _ espumadera
  _ huevos
  _ aceite
2. colocar unos centímetros de aceite en el sartén a fuego
   medio
3. cuando aceite esté caliente agregar los huevos fuera de
   la cáscara
4. con espumadera o cuchara, remover aceite caliente por
   encima de los huevos a medida que se cocinana
5. cuando burbujas en borde de huevos fritos estén doradas
   retirar con espumadera
6. servir
7. salar
8. fin
```

Ejercicio 2.3 página 111

```
1. leer números
2. número mayor: A
3. número menor: B
4. repetir
    _ dividir A por B
    _ dividir divisor por resto de división anterior
5. hasta que
    _ división sea exacta
6. leer e imprimir último divisor
```

Ejercicio 2.4 página 111

```
1. establecer SERIE como vacía
2. establecer TOTAL a cero
3. siguiente número es un cero?
4. si siguiente número es cero, imprimir SERIE y TOTAL y fin
5. si siguiente número no es cero, ejecutar pasos 6 y 7
6. leer número y agregar el dígito a la variable SERIE
7. incrementar TOTAL en 1
8. regresar a paso 3
```

Ejercicio 2.5 página 112

```
1. establecer SERIE como vacía
2. establecer ACTUAL a cero
3. establecer TOTAL a cero
4. siguiente número es un 99?
5. si siguiente número es 102, imprimir SERIE y TOTAL y fin
6. si siguiente número no es 102, ejecutar pasos 7 a 9
7. incrementar ACTUAL en 3
8. agregar el dígito ACTUAL a la variable SERIE
9. incrementar TOTAL en valor ACTUAL
10. regresar a paso 4
```

Ejercicio 2.6 página 112

```
1. leer los cuatro números
2. seleccionar y guardar mayor entre los primeros dos
3. seleccionar y guardar mayor entre el anterior y el tercero
4. seleccionar, guardar y mostrar el mayor entre el anterior
    y el cuarto
```

Ejercicio 2.7 página 112

```
1. leer los tres valores, A, B y C
2. si A + B = C escribir “Iguales” y parar
3. si A + C = B escribir “Iguales” y parar
4. si B + C = A escribir “Iguales” y parar
5. escribir “Distintos” y parar
```

Ejercicio 2.7 página 112

```
1. leer palabra
2. convertir palabra en minúscula
3. invertir palabra
4. si palabra invertida es igual a original, esctribir: “Es capicúa”
5. si no, escribir: “No es capicúa”
```

Ejemplo 3.1.1 página 127

```
n = 5
escribir n    5
escribir n++  5
escribir n    6
n = 5
escribir n    5
escribir ++n  6
escribir n    6
```

Ejemplo 3.3 página 128

```
-4 * 7 +2 ^ 3 / 4 - 5
-28 + 8 / 4 - 5
-28 + 2 -5
-28 - 5
-31
```

Ejemplo 3.10 página 133

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$x1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$x2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

```
x = (-b ± √(b ^ 2 - 4 * a * c)) / (2 * a)


x1 = (-b + √(b ^ 2     - 4 * a * c)) / (2 * a)
   = (-b + sqrt(sqr(b) - 4 * a * c)) / (2 * a)


x2 = (-b - √(b ^ 2 - 4 * a * c)) / (2 * a)
```

Ejemplo 3.11 página 136

```
A ← 3          A==3
B ← 4          B==4
C ← A + 2 * B  C==11
C ← C + B      C==15
B ← C - A      B==12
A ← B * C      A==180
```

Ejemplo 3.12 página 137

```
x ← 2                          x==2
x ← sqr(x + x)                 x==16
x ← sqrt(x + sqrt(x) + 5)      x==5
```

3.11 ESCRITURA DE ALGORITMOS/PROGRAMAS

```
algroritmo
cabecera del programa
sección de declaración
sección de acciones
```

Modelo propuesto de algoritmo, página 142

```
algoritmo raices
// resuelve una ecuación de 2° grado
var
  real : a, b, c
inicio
  leer(a, b, c)
  d ← b ^ 2 -4 * a * c
  si d < 0 entonces
    escribir('raices complejas)
  si_no
    si d = 0 entonces
      escribir(-b / (2 * a))
    si_no
      escribir((-b - raiz2(d)) / (2 * a))
      escribir((-b + raiz2(d)) / (2 * a))
    fin_si
  fin_si
fin si
```

Actividad 3.1 resuelta, página 143

```
algoritmo pinchazo
// resuelve un problema de pinchazo en ruta
inicio
  si gato del coche está averiado
    entonces llamar a la estación de servicio
    si_no levantar el coche con el gato
      repetir
        aflojar y sacar los tornillos de las ruedas
      hasta_que todos los tornillos estén flojos y quitados
      quitar la rueda
      poner la rueda de repuesto
      repetir
        poner los tornillos y apretarlos
      hasta_que estén puestos todos los tornillos
        bajar el gato
    fin_si
  fin_si
fin
```

Actividad 3.2 resuelta, página 143

```
(A) VALOR ← 4.0 * 5     VALOR=20.0
(B) X ← 3.0             X=3.0
    Y ← 2.0             Y=2.0
    VALOR ← X ^ Y - Y   VALOR=7.0
(C) VALOR ← 5           VALOR=5
    X ← 3               X=3
    VALOR ← VALOR * X   VALOR=15
```

Actividad 3.4 resuelta, página 143

```
X ← A + B + C     X=40
X ← A + B * C     X=255
X ← A + B / C     X=7.5
X ← A + B \ C     X=7
X ← A + B mod C   X=10
X ← (A + B) \ C   X=3
X ← A + (B / C)   X=7.5
Siendo A = 5  B = 25  C = 10
```

Actividad 3.5 resuelta, página 144

```
a. M / N + P
b. M + N /(P - Q)
c. (sen(x) + cos(x)) / tan(x)
d. (m + n) / (p - q)
e. (m + n / p) / ( q – r / s)
f. (-b + sqrt(b^2 – 4 * a * c)) / (2 * a)
```

Actividad 3.6 resuelta, página 144

a) 53
b) -8
c) 9
d) 12
e) 7
f) 90

Actividad 3.7 resuelta, página 145

	instrucc	**A**	**B**	**C**	**AUX**
(1)	A ← 5	5	--	--	--
(2)	B ← 10	--	10	--	--
(3)	C ← 15	--	--	15	--
	AUX ← A	5	10	15	5
	A ← C	15	10	15	5
	C ← B	15	10	10	5
	B ← AUX	15	5	10	5

Actividad 3.10 resuelta, página 146

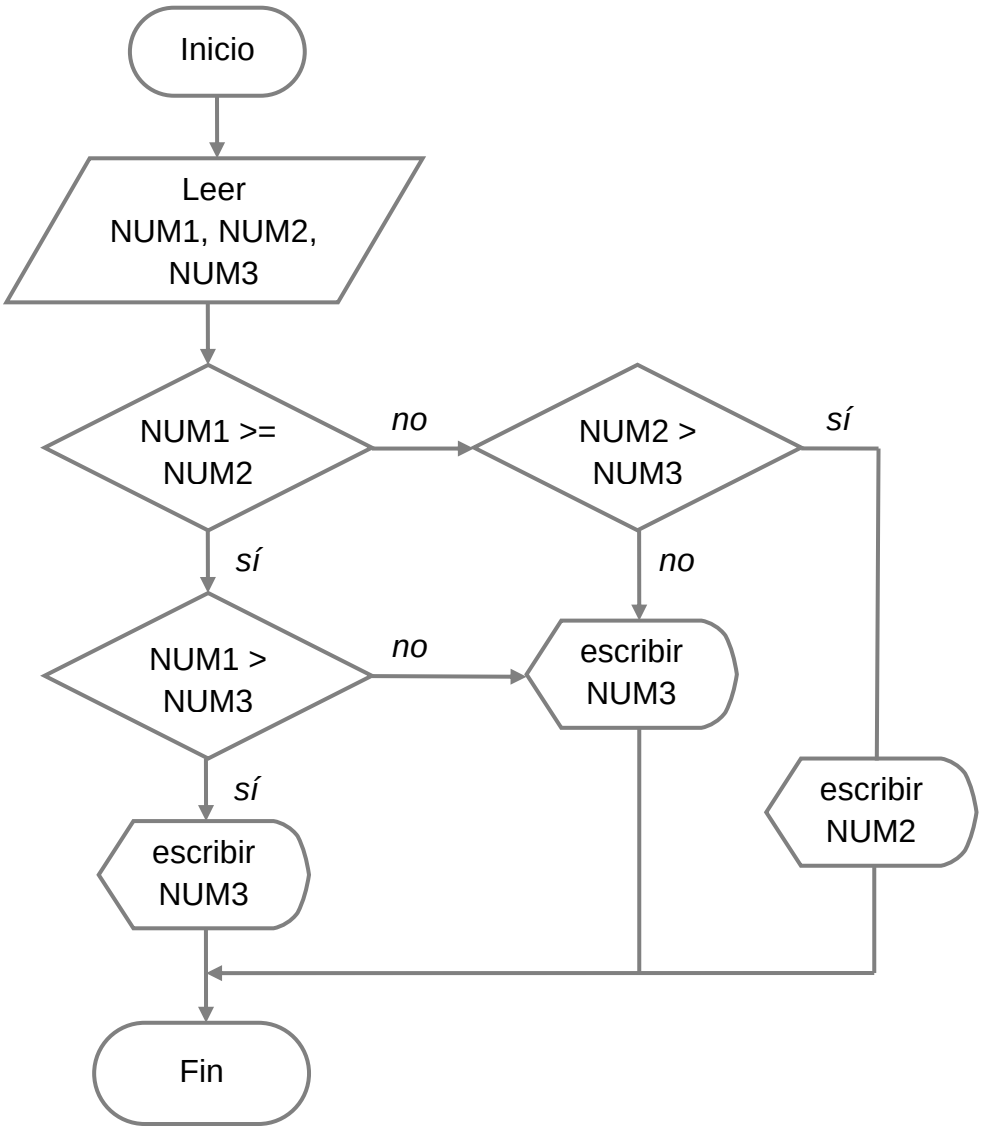
```
A ← 5          A=5
B ← A + 6      B=11
A ← A + 1      A=6
B ← A – 5      B=1
```

Actividad 3.11 resuelta, página 147

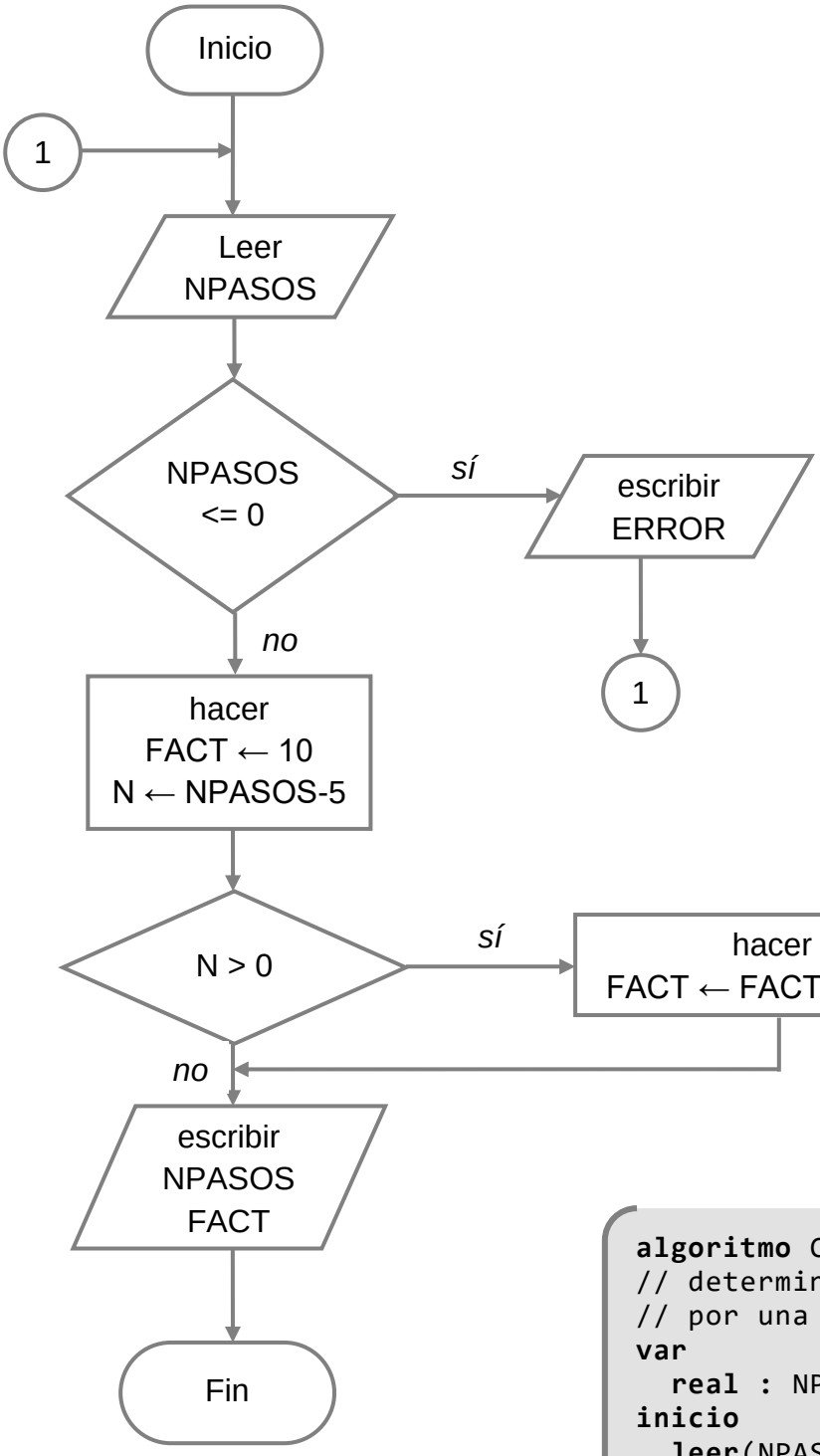
```
A ← 3          A=3
B ← 20         B=20
C ← A + B      C=23
B ← A + B      B=23
A ← B - C      A=0
```

Actividad 3.13 resuelta, página 148

```
algoritmo MAYOR
// determina el mayor de tres números reales
const
  real : NUM1, NUM2, NUM3
inicio
  leer(NUM1, NUM2)
  si NUM1 >= NUM2 entonces
    MAYOR ← NUM1
  si_no
    MAYOR ← NUM2
  fin_si
  leer(NUM3)
  si MAYOR < NUM3 entonces
    MAYOR ← NUM3
  fin_si
  escribir(MAYOR)
fin_si
```

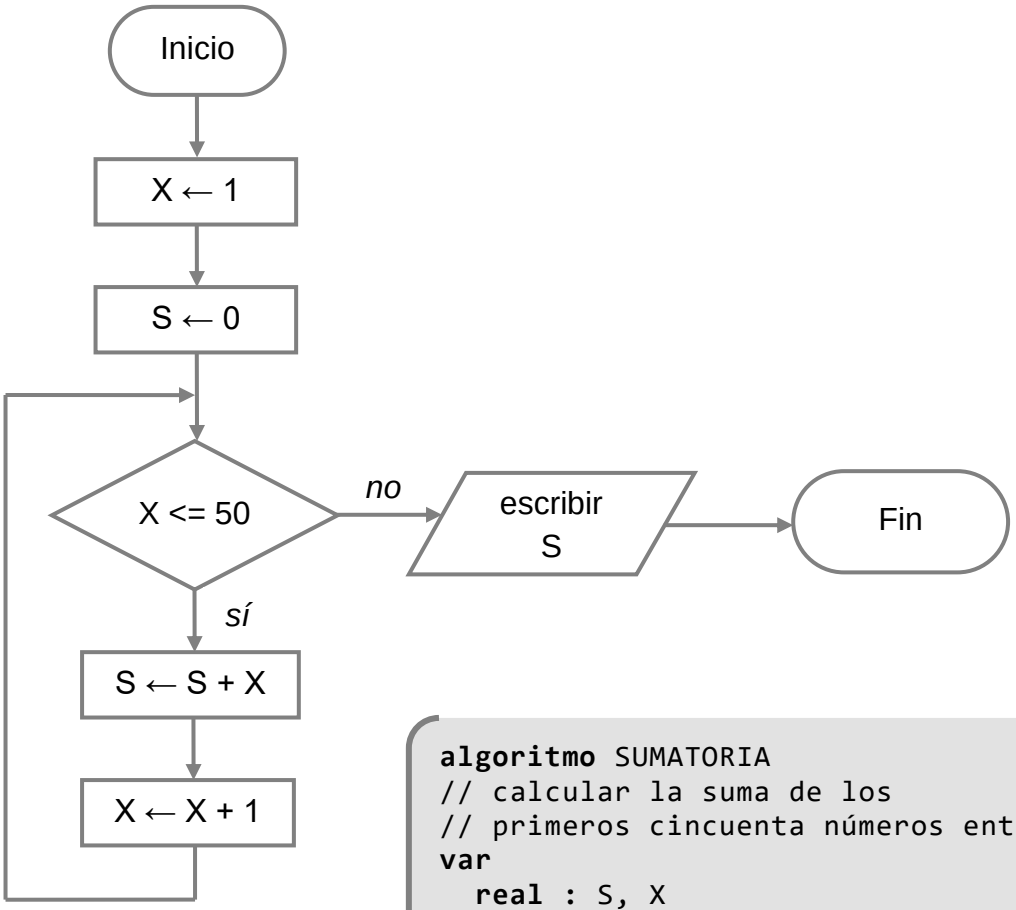


Actividad 3.14 resuelta, página 148



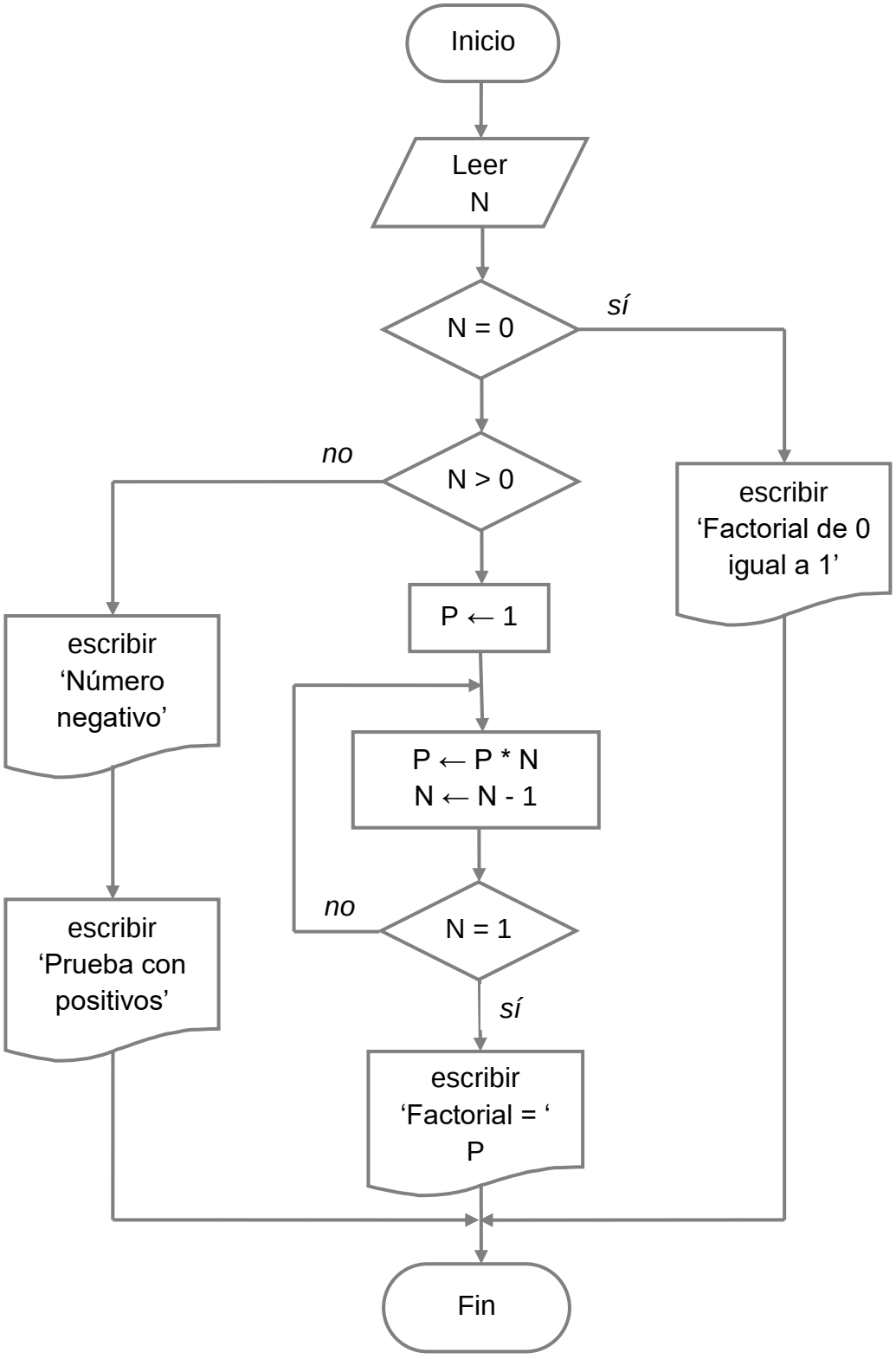

```
algoritmo COSTOLL
// determina la cantidad a pagar
// por una llamada telefónica
var
  real : NPASOS, N, FACT
inicio
  leer(NPASOS)
  leer(N)
  si NPASOS <= 0 entonces
    escribir(ERROR)
    volver(INICIO)
  si_no
    FACT ← 10
    N ← NPASOS - 5
  fin_si
  si N > 0 entonces
    FACT ← FACT + N * 5
  si_no
    escribir(NPASOS)
    escribir(FACT)
  fin_si
fin
```

Actividad 3.15 resuelta, página 149




```
algoritmo SUMATORIA
// calcular la suma de los
// primeros cincuenta números enteros
var
  real : S, X
inicio
      X ← 1
      S ← 0
  (1) si X <= 50 entonces
        S ← S + X
        X ← X + 1
        ir_a(1)
      si_no
        escribir(S)
      fin_si
fin
```

Actividad 3.16 resuelta, página 150

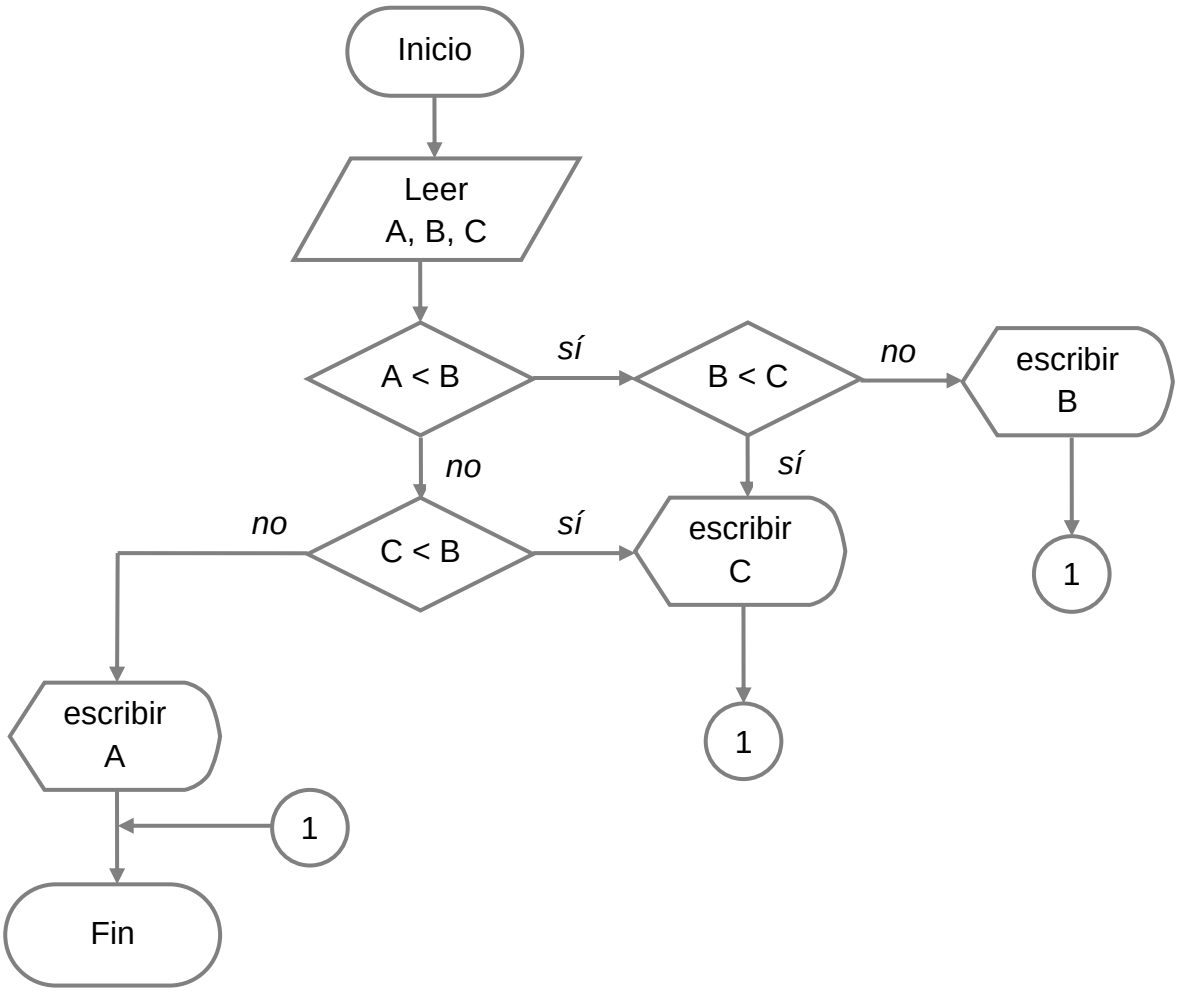


```
algoritmo FACTORIAL
// calcular el producto de los
// n primeros números naturales
var
  real : N, P
inicio
   leer(N)
   si N = 0 entonces
     escribir('Factorial de 0 = 1')
   si_no
     si N > 0 entonces
        P ← 1
  (1)   P ← P * N
        N ← N – 1
     si_no
       escribir('Número negativo')
       escribir('Prueba con positivos')
     fin_si
   fin_si
   si N = 1 entonces
     escribir('Factorial = ', P)
   si_no
     ir_a(1)
   fin_si
fin
```

Propuesta del apunte

```
algoritmo Factorial
var
  entero: N
  real : P
inicio
   leer(n)
   si N = 0 entonces
     escribir('Factorial de 0 = 1')
   si_no
     si N > 0 entonces
           P ← 1
       (1) P ← P * N
           N ← N - 1
       si N = 1 entonces
         escribir('Factorial = ', P)
       si_no
         ir_a(1)
       fin_si
     si_no
       escribir('Número negativo')
       escribir('Prueba con positivos')
     fin_si
   fin_si
fin
```

Actividad 3.18 resuelta, página 153
Alternativa de actividad 3.13, página 148



Ejercicio 3.2, página 155
No son identificadores válidos (e) y (h)

Ejercicio 3.2, página 155
No son constantes válidas (a), (d), (e), (f), (i) y (j)

Ejercicio 3.4, página 155
A = 2
B = 5
3 * A – 4 * B / A ^ 2 =
3 * 2 – 4 * 5 / 2 ^ 2 = –3.5

Ejercicio 3.5, página 155

4 / 2 * 3 / 6 + 6 / 2 / 1 / 5 ^ 2 / 4 * 2 =
(4 / 2 * 3 / 6) + (6 / 2 / 1 / (5 ^ 2) / 4 * 2) =
1 + (3 / 25 / 8) = 1.015

Ejercicio 3.6, página 155

a) sqrt(sqr(b)) – 4 * a * c
b) (sqr(x) + sqr(y)) / sqr(z)
c) (3 * x) + (2 * y) / (2 * z)
d) (a + b) / (c + d)
e) 4 * sqr(x) 2 * x 7
f) ((x + y) / x) – (3 * x / 5)
g) a / b * c
h) x * y * z
i) --
j) 2 * 3.14 * r
k) (4 / 3) * 3.14 * r ^ 3
l) --

Ejercicio 3.7, página 155

a) $b^2 + 4ac$
b) $3X^4 – 5X^3 + X12 – 17$
c) $\frac{b+d}{c+4}$
d) --

Ejercicio 3.8, página 156

A = 4
B = 5
C = 1
a) –13.75
b) 2.22
c) 324

Ejercicio 3.9, página 156

512

Ejercicio 3.10, página 156

a) 3
b) 1
c) 4
d) 0
e) 0
f) 0
g) 70 – 2 * 4 + 9 = 71
h) 17

Ejercicio 3.11, página 156

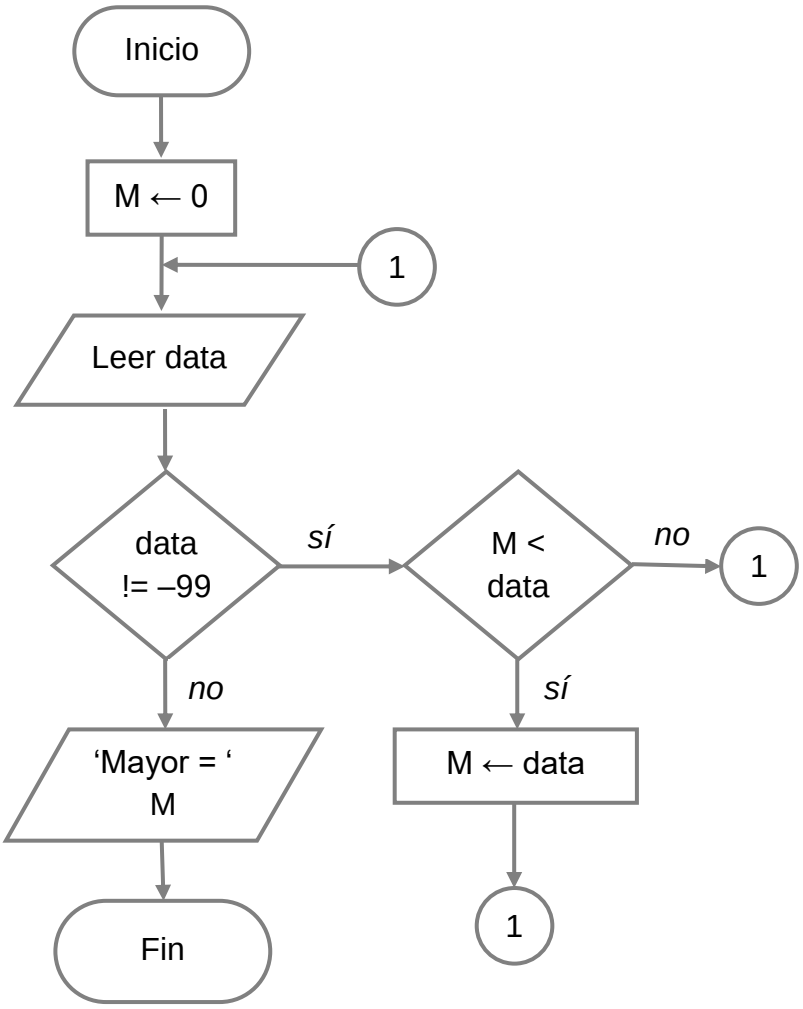
a) 2
b) 0
c) 0
d) 2
e) 1
f) 2
g) 2
h) 7 / 2 = 3.5
i) 18
j) 4
k) 4
l) --

Ejercicio 3.12, página 156

```
algoritmo SUMAPAIM
// calcular independiente la suma de
// los números pares e impares
// entre 1 y 200
var
  real : C, P, I
inicio
      C ← 0
      P ← 0
      I ← 1
  (1) si C mod 2 = 1 entonces
        si C <= 199 entonces
          I ← I + C
          C ← C + 1
          ir_a(1)
        si_no
          escribir('Suma de impares = ', I)
        fin_si
      si_no
        si C <= 200 entonces
          P ← P + C
          C ← C + 1
          ir_a(1)
        si_no
          escribir('Suma de pares = ', P)
        fin_si
      fin_si
fin
```

Ejercicio 3.13, página 156

```
algoritmo MAYORDESERIE
// lee una serie de números distintos
// de 0 y obtiene el mayor. El último
// es –99
var
  real : data, M
inicio
      M ← 0
  (1) leer(data)
      si data != –99 entonces
        si M < data entonces
          M ← data
          ir_a(1)
        si_no
          ir_a(1)
        fin_si
      si_no
        escribir('Número mayor = ', M)
      fin_si
fin
```



Ejercicio 3.14, página 156

```
algoritmo SUMAPAR
// calcular la suma y el producto de
// los números pares entre 20 y 400
var
  real : C, S, P
inicio
      C ← 20
      S ← 20
      P ← 20
  (1) si C < 399 entonces
        C ← C + 2
        S ← S + C
        P ← P * C
        ir_a(1)
      si_no
        escribir('Suma de pares = ', P)
      fin_si
fin
```

Ejercicio 3.15, página 156

```
algoritmo MAYORDE500
// lee 500 números enteros y obtiene
// cuántos son positivos
// N: número ingresado
// C: contador de ingresados
// P: contador de positivos
var
  real : N, C, P
inicio
      C ← 0
      P ← 0
  (1) leer(N)
      C ← C + 1
      si C <= 500 entonces
        si N >= 0 entonces
          P ← P + 1
          ir_a(1)
        si_no
          ir_a(1)
        fin_si
      si_no
        escribir('Cantidad de positivos = ', P)
      fin_si
fin
```

Ejercicio 3.16, página 156

```
algoritmo FACTURADOR
// emite la factura de una compra, de UNA
// unidad. El IVA es 15%. Si el bruto es
// mayor que 1000, descuento de 5%
var
  real : compra, B, IVA, D, F
inicio
  leer(compra)
    B ← compra * 1.15
    IVA compra * 0.15
    escribir('Compra = ', compra)
    escribir('IVA = ', IVA)
    si B < 1000 entonces
      escribir('Total = ', B)
    si_no
      D ← B * 0.05
      F ← B * 0.95
      escribir('Descuento = ', D)
      escribir('Final = ', F)
    fin_si
fin
```

```
algoritmo FACTURADOR
// emite la factura de una compra, de UNA
// O MAS unidades. El IVA es 15%. Si el
// bruto es mayor que 1000, descuento de 5%
var
  real : compra, Q, B, IVA, D, T
inicio
  leer(compra)
  leer(Q)
    B ← compra * Q * 1.15
    IVA ← compra * 0.15
    escribir('Compra = ', compra)
    escribir('Cantidad = ', Q)
    escribir('IVA = ', IVA)
    si B < 1000 entonces
      escribir('Total = ', B)
    si_no
      D ← B * 0.05
      T ← B * 0.95
      escribir('Descuento = ', D)
      escribir('Total = ', T)
    fin_si
fin
```

Ejercicio 3.17, página 156

```
algoritmo SUMACUAD
// calcular la suma de los cuadrados de
// los 100 primeros números naturales
var
  entero : S, X
inicio
      S ← 0
      X ← 1
  (1) si X <= 100 entonces
        S ← S + sqr(X)
        X ← X + 1
        ir_a(1)
      si_no
        escribir(S)
      fin_si
fin
```

Ejercicio 3.18, página 156

```
algoritmo SUMAPAR
// sumar los números pares del 2
// al 100 e imprimir su valor
var
  real : C, S
inicio
      C ← 0
      S ← 2
  (1) si C < 100 entonces
        C ← C + 2
        S ← S + C
        ir_a(1)
      si_no
        escribir('Suma de pares = ', S)
      fin_si
fin
```

Ejercicio 3.19, página 156

```
algoritmo SUMA10
// sumar 10 números introducidos
// por teclado
// N: número ingresado
// C: contador de ingresados
// S: suma total
var
  real : N, C, P
inicio
      C ← 0
      S ← 0
  (1) leer(N)
      C ← C + 1
      si C <= 10 entonces
        S ← S + N
        ir_a(1)
      si_no
        escribir('Suma total = ', P)
      fin_si
fin
```

Ejercicio 3.20, página 156

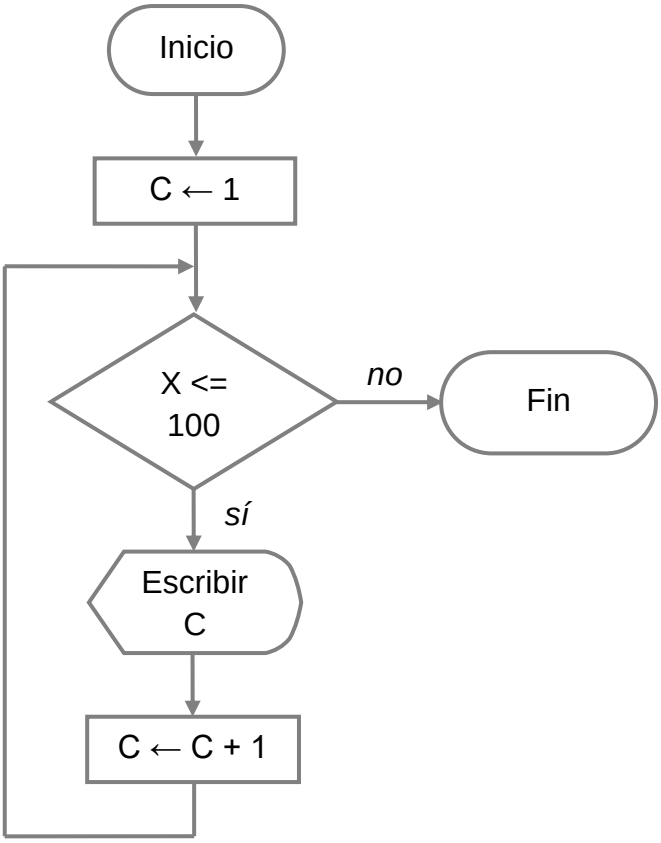
```
algoritmo MEDIA50
// calcular la media de 50 números
// e imprimir su resultado
// N: número del conjunto
// C: contador de ingresados
// S: suma total
// M: media
var
  real : N, C, P, M
inicio
      C ← 0
      S ← 0
  (1) leer(N)
      C ← C + 1
      si C <= 50 entonces
        S ← S + N
        ir_a(1)
      si_no
        M ← S / 50
        escribir('Media = ', M)
      fin_si
fin
```

Ejercicio 3.21, página 156

```
algoritmo LISTADE4S
// calcular los N (introducido por
// teclado) primeros múltipos de 4
var
  real : N, L, A
inicio
      C ← 0
      L ← 4
      A ← 4
      leer(N)
  (1) C ← C + 1
      si C <= N entonces
        A ← A + 4
        L ← 'L' + ', ' + A
        ir_a(1)
      si_no
        escribir('Lista = ', L)
      fin_si
fin
```

Ejercicio 3.22, página 156

```
algoritmo IMPRIMIR100
// diagrama que realice contador e
// imprima los 100 primeros números
// enteros
var
  entero : C
inicio
      C ← 1
  (1) si C <= 100 entonces
        escribir(C)
        C ← C + 1
        ir_a(1)
      fin_si
fin
```


Inicio
C ← 1
X <= 100
no
Fin
sí
Escribir C
C ← C + 1

Ejercicio 3.23, página 156

```
algoritmo SUMAMULTIPLE
// dados 10 números enteros
// imprimir suma de pares
// imprimir cúantos pares hay
// imprimir media de impares
// N: número dado
// C: contador de lecturas
// Cp: contador de pares
// Ci: contador de impares
// Sp: suma de pares
// Si: suma de impares
// Mi: media de los impares
var
  real : N, C, Cp, Ci, Sp, Si, Mi
inicio
      C ← 1
      Cp ← 0
      Ci ← 0
      Sp ← 0
      Si ← 0
  (1) si C <= 10 entonces
        leer(N)
        C ← C + 1

        si N mod 2 = 0 entonces
          Sp ← Sp + N
          Cp ← Cp + 1
        si_no
          Si ← Si + 1
          Ci ← Ci + 1
        fin_si

        ir_a(1)
      si_no
        Mi ← Si / Ci
        escribir('Suma de pares = ', Sp)
        escribir('Cantidad de pares = ', Cp)
        escribir('Media de los impares = ', Mi)
      fin_si
fin
```

Ejercicio 3.24, página 156

```
algoritmo CALCULARNOTAS
// calcular nota media de alumnos
// N: número de alumnos
// C: notas de cada alumno
// Sn: suma de notas
// Co: contador
// Nm: nota media
var
  entero : N, C, Sn, Co
  real : Nm
inicio
      Co ← 1
      C ← 0
      Sn ← 0
  (1) si Co <= N entonces
        leer(data)
        Co ← Co + 1
        Sn ← Sn + c
        ir_a(1)
      si_no
        Nm ← Sn / N
      fin_si
      escribir(Nm)
fin
```

Ejercicio 3.25, página 156

```
algoritmo SUMAPAR
// escribir la suma de los
// 10 primeros números pares
var
  real : C, S
inicio
      C ← 0
      S ← 2
  (1) si C < 20 entonces
        C ← C + 2
        S ← S + C
        ir_a(1)
      si_no
        escribir('Suma de pares = ', S)
      fin_si
fin
```

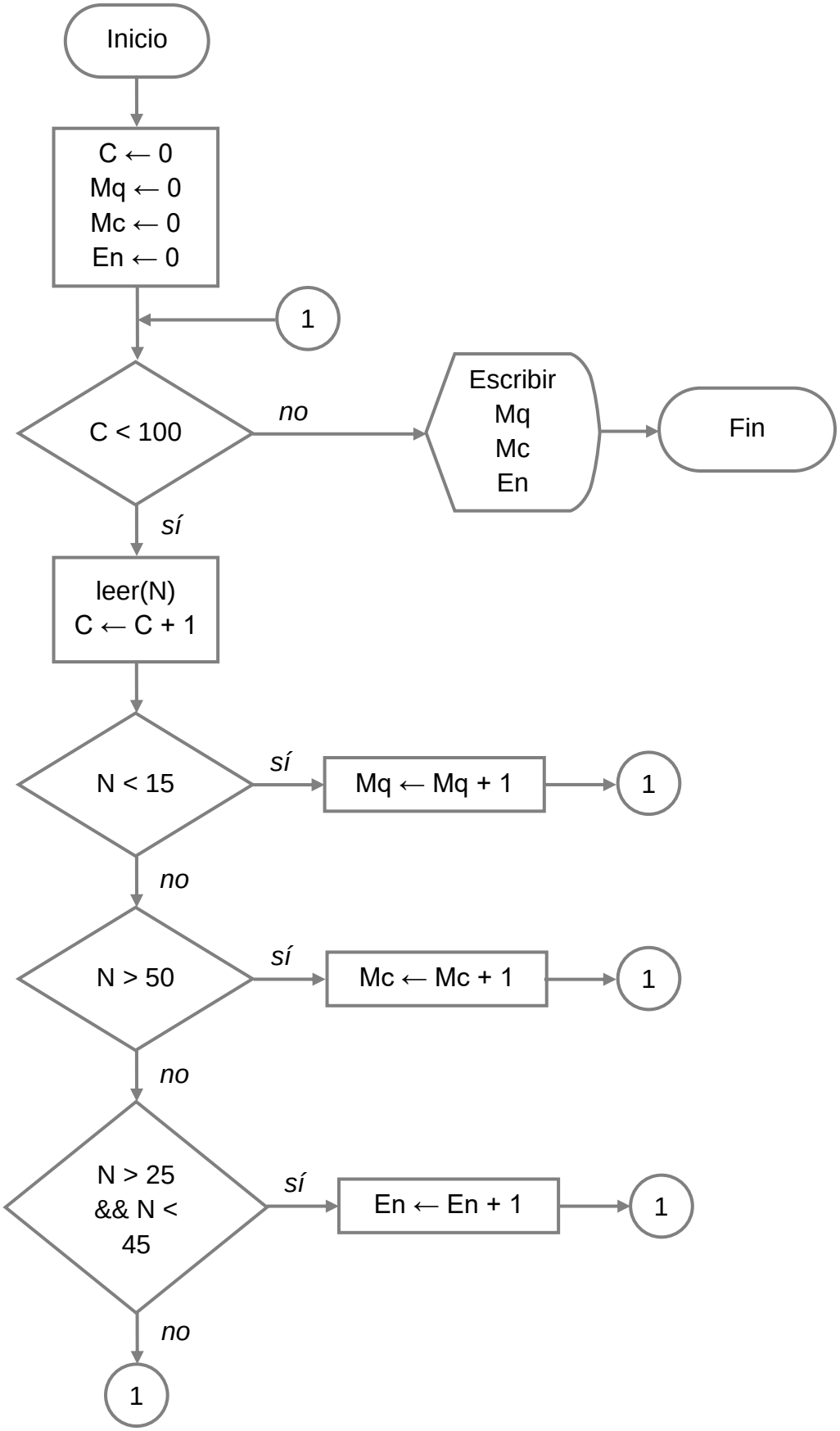
Ejercicio 3.26, página 156

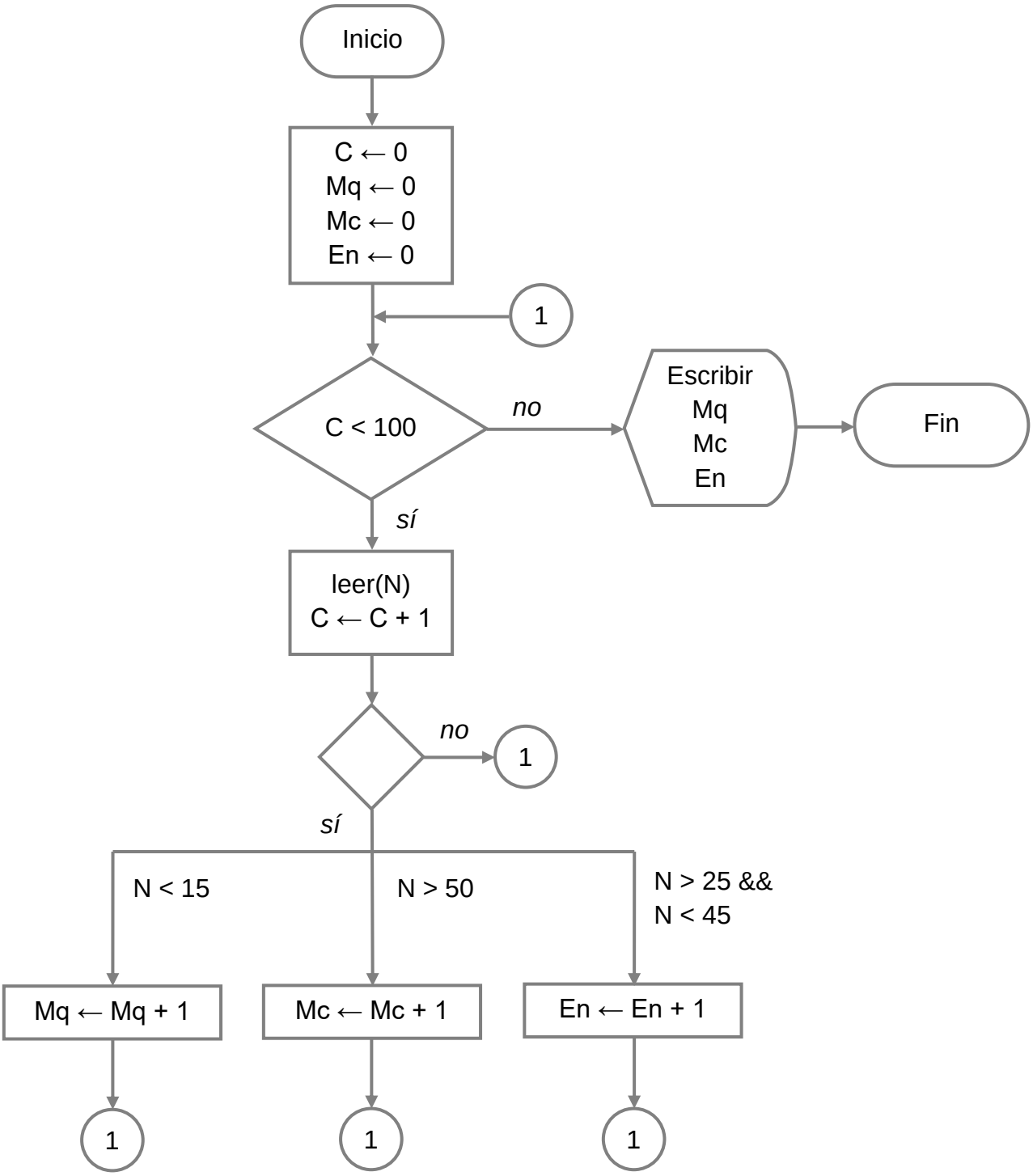
```
algoritmo POSARCHIVO
// suma positivos de un archivo
var
  real : data, S
inicio
      C ← 0
      S ← 0
  (1) si hay(data) entonces
        leer(data)
        si data >= 0 entonces
          S ← S + data
        si_no
          ir_a(1)
        fin_si
      si_no
        escribir('Suma de positivos = ', S)
      fin_si
fin
```

Ejercicio 3.27, página 156

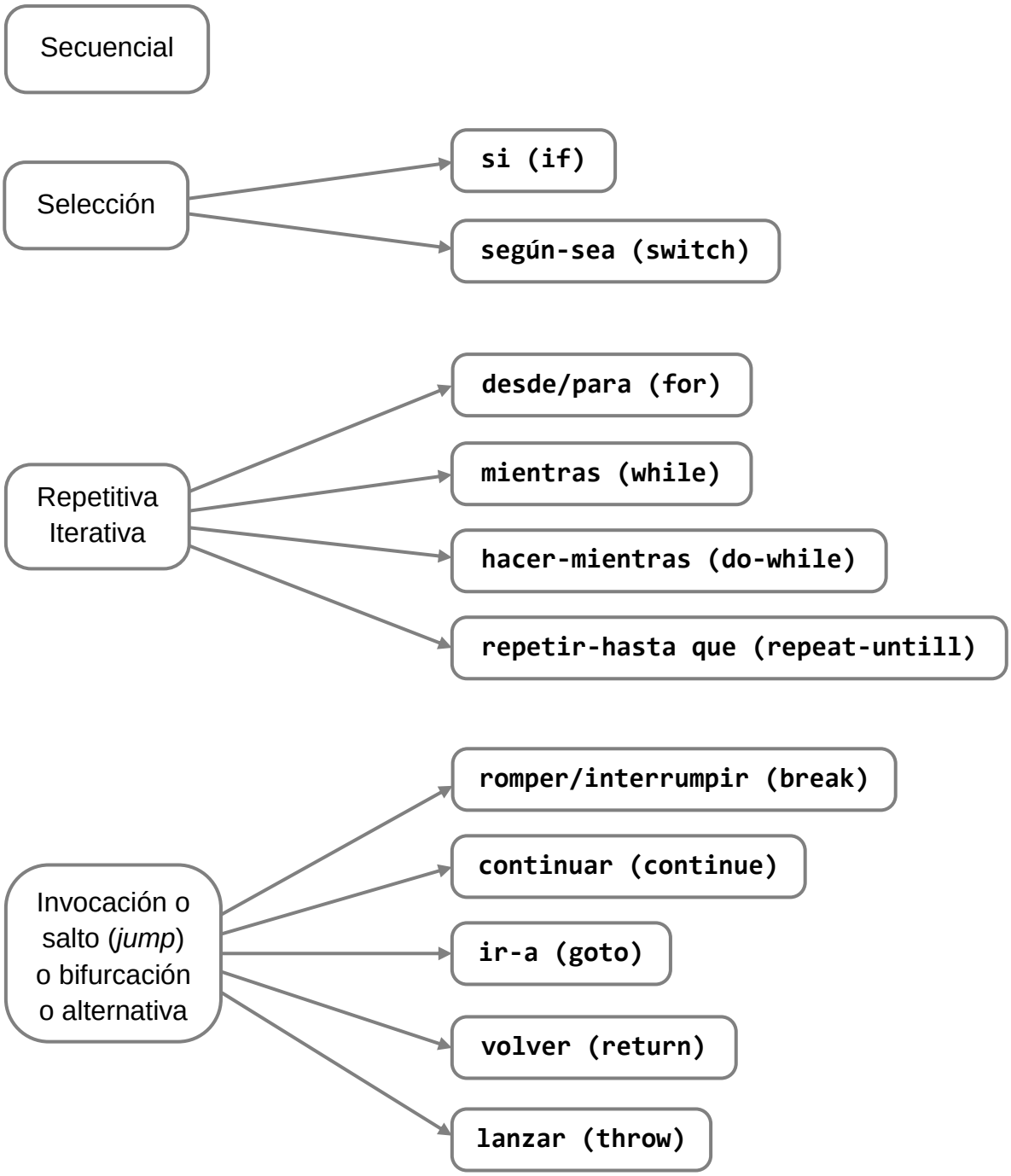
```
algoritmo RANGOS
// dados 100 números naturales
// cuántos son menores de 15
// cuántos son mayores de 50
// cuántos están entre 25 y 45
// N: número dado
// C: contador de lecturas
// Mq: menores de 15
// Mc: mayores de 50
// En: entre 25 y 45
var
  real : N, C, Mq, Mc, En
inicio
      C ← 0
      Mq ← 0
      Mc ← 0
      En ← 0
  (1) si C < 100 entonces
        leer(N)
        C ← C + 1
        si N < 15 entonces
          Mq ← Mq + 1
          ir_a(1)
        si_no
          si N > 50 entonces
            Mc ← Mc + 1
            ir_a(1)
          si_no
            si N > 25 && N < 45 entonces
              En ← En + 1
              ir_a(1)
            si_no
              ir_a(1)
            fin_si
          fin_si
        fin_si
      si_no
        escribir('Son menores de 15 = ', Mq)
        escribir('Son mayores de 50 = ', Mc)
        escribir('Están entre 25 y 45 = ', En)
      fin_si
fin
```

No me convence


Inicio
C ← 0
Mq ← 0
Mc ← 0
En ← 0
1
C < 100
no
Escribir
Mq
Mc
En
Fin
sí
leer(N)
C ← C + 1
N < 15
sí
Mq ← Mq + 1
1
no
N > 50
sí
Mc ← Mc + 1
1
no
N > 25
&& N <
45
sí
En ← En + 1
1
no
1


Inicio
C ← 0
Mq ← 0
Mc ← 0
En ← 0
1
C < 100
no
Escribir
Mq
Mc
En
Fin
sí
leer(N)
C ← C + 1
no
1
sí
N < 15
N > 50
N > 25 &&
N < 45
Mq ← Mq + 1
Mc ← Mc + 1
En ← En + 1
1
1
1

ESTRUCTURAS DE CONTROL



Ejemplo 4.4 página 163

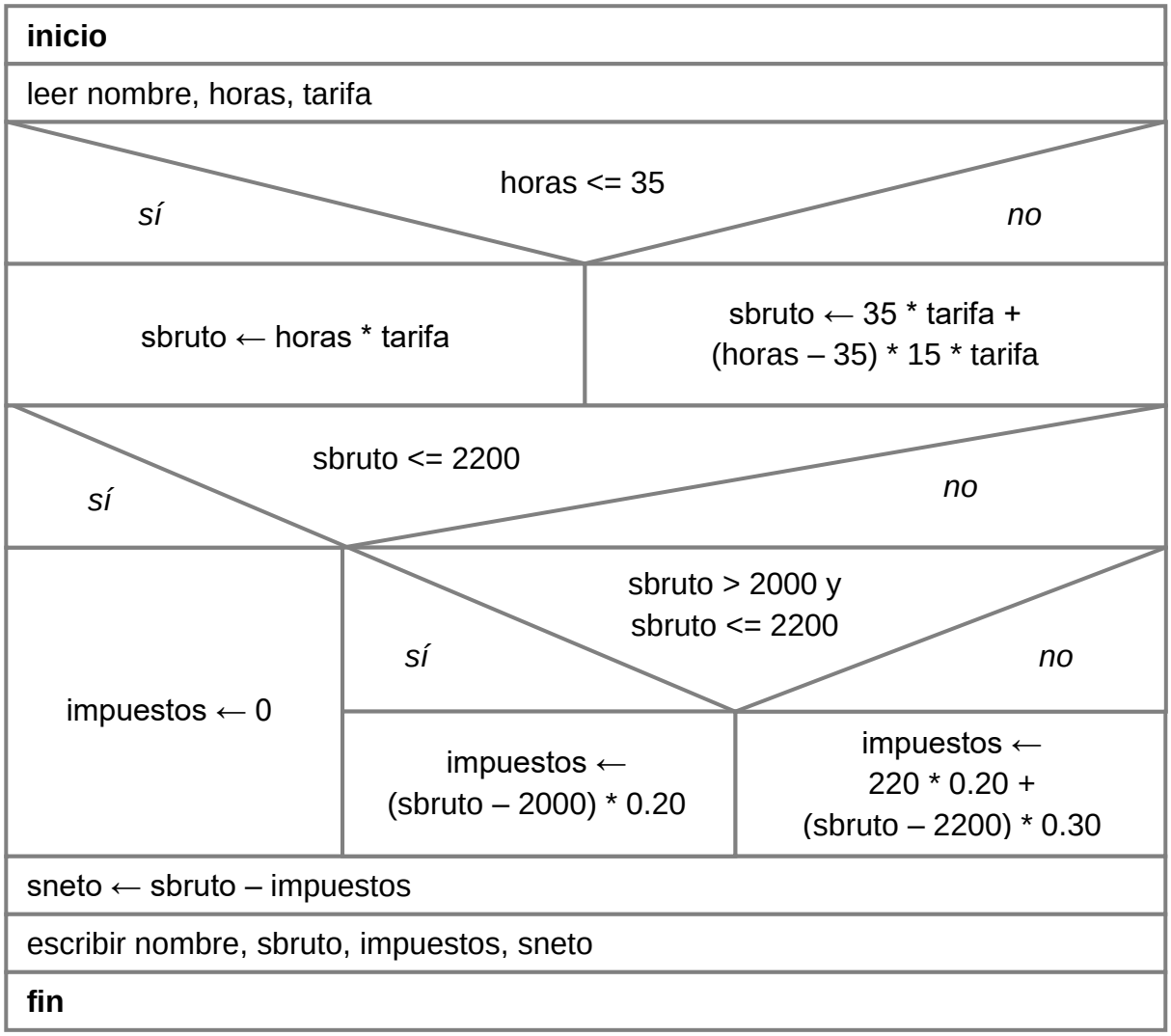
```
algoritmo media
// calcular la media de una
// serie de números positivos
//
// Tabla de variables
// real: s (suma)
// entera: n (contador de núm)
// real: m (media)
var
  real: s, m
  entera: n
inicio
  s ← 0
  n ← 0
datos:
  leer(x)
  si x < 0 entonces
    ir_a(media)
  si_no
    n ← n + 1
    s ← s + x
    ir_a(datos)
  fin_si
media:
  m ← s / n
  escribir(m)
fin
```

```
algoritmo media
// anterior simplificado
//
//
// Tabla de variables
// real: s (suma)
// entera: n (contador de núm)
// real: m (media)
var
  real: s, m
  entera: n
inicio
  s ← 0
  n ← 0
datos:
  leer(x)
  si x < 0 entonces
    media:
      m ← s / n
  si_no
    n ← n + 1
    s ← s + x
    ir_a(datos)
  fin_si
escribir(m)
fin
```

Ejemplo 4.5 página 164

```
1. inicio
2. leer nombre, horas trabajadas, tarifa horaria
3. verificar si horas trabajadas <= 35, en cuyo caso
     salario_bruto = horas * tarifa, en caso contrario,
     salario_bruto = 35 * tarifa + (horas – 35) * tarifa
4. cálculo de impuestos
     si salario_bruto <= 2000, entonces impuestos = 0
     si salario_bruto <= 2200, entonces
     impuestos = (salario_bruto – 2000) * 0.20
     si salario bruto > 2200, entonces
     impuestos = (salario_bruto – 2200) * 0.30 + (220 * 0.20)
5. cálculo del salario neto
     salario_neto = salario_bruto – impuestos
6. fin
```

```
algoritmo Nomina
var
  cadena : nombre
  real : horas, impuestos, sbruto, sneto
inicio
  leer(nombre, horas, tarifa)
  si horas <= 35 entonces
    sbruto ← horas * tarifa
  si_no
    sbruto ← 35 * tarifa + (horas – 35) * 1.5 * tarifa
  fin_si
  si sbruto <= 2000 entonces
    impuestos ← 0
  si_no
    si (sbruto > 2000) y (sbruto <= 2200) entonces
      impuestos ← (sbruto – 2000) * 0.20
    si_no
      impuestos ← (220 * 0.20) + (sbruto – 2200) * 0.30
    fin_si
  fin_si
  sneto ← sbruto – impuestos
  escribir(nombre, sbruto, impuestos, neto)
fin
```



Ejemplo 4.6 página 166

```
algoritmo Parte_fraccionaria
// detectar si un número tiene
// o no parte fraccionaria
var
  real: n
inicio
  escribir('Ingresar número ')
  leer(n)
  si n = trunc(n) entonces
    escribir('El número no tiene parte fraccionaria')
  si_no
    escribir('Número con parte fraccionaria')
  fin_si
fin
```

Ejemplo 4.7 página 166

```
algoritmo Bisiesto
// averiguar si un leído de teclado
// es o no bisiesto
var
  entero : año
inicio
  leer(año)
  si (año MOD 4 = 0) y (año MOD 100 <> 0) O (año MOD 400 = 0) entonces
    escribir('El año ', año, ' es bisiesto')
  si_no
    escribir('El año ', año, ' no es bisiesto')
  fin_si
fin
```

Ejemplo 4.8 página 167

```
algoritmo Area_triangulo
// calcular el área de un triángulo conociendo sus lados. La
// estructura selectiva es para el control de entrada de datos
// Area = √(p*(p-a)*(p-b)*(p-b))
// p = (a+b+c)/2
var
  real: a, b, c, p, area
inicio
  escribir('Ingresar los lados ')
  leer(a, b, c)
  p ← (a + b + c) / 2
  si (p > a) y (p > b) y (p > c) entonces
    area ← raiz2(p * (p - a) * (p - b) * (p - c))
    escribir('area')
  si_no
    escribir('No es un triángulo')
  fin_si
fin
```

Ejemplo 4.9 página 171

```
algoritmo DiasSemana
// escribir los nombres de los días de la semana
// en función de una variable introducida
var
  entero : DIA
inicio
  leer(DIA)
  segun_sea DIA hacer
    1. escribir('LUNES')
    2. escribir('MARTES')
    3. escribir('MIERCOLES')
    4. escribir('JUEVES')
    5. escribir('VIERNES')
    6. escribir('SABADO')
    7. escribir('DOMINGO')
  si_no
    escribir('ERROR')
  fin_segun
fin
```

Ejemplo 4.10 página 171

```
algoritmo Calificaciones
// convertir las calificaciones alfabéticas
// A, B, C, D, E y F a calificaciones numéricas
// 4, 5, 6, 7, 8 y 9
var
  caracter: LETRA
  entero: calificacion
inicio
  leer(LETRA)
  segun_sea LETRA hacer
    'A': calificacion ← 4
    'B': calificacion ← 5
    'C': calificacion ← 6
    'D': calificacion ← 7
    'E': calificacion ← 8
    'F': calificacion ← 9
  otros
    escribir('ERROR')
  fin_segun
fin
```

Ejemplo 4.11 página 172

```
algoritmo PAR_IMPAR
// determinar si número introducido
// por teclado es par o impar
var
  entero: numero
inicio
  leer(numero)
  si numero >= 1 y numero <= 10 entonces
    segun_sea numero hacer
      1, 3, 5, 7, 9: escribir('impar')
      2, 4, 6, 8, 10: escribir('par')
    fin_segun
  fin_si
fin
```

Ejemplo 4.12 página 172

```
algoritmo Dia_semana
// leída una fecha, decir el día de la semana,
// suponiendo que el día 1 fue lunes
var
  entero: dia
inicio
  escribir('Ingrese dia')
  leer(dia)
  segun_sea dia MOD 7 hacer
    1:
      escribir('Lunes')
    2:
      escribir('Martes')
    3:
      escribir('Miércoles')
    4:
      escribir('Jueves')
    5:
      escribir('Viernes')
    6:
      escribir('Sábado')
    0:
      escribir('Domingo')
  fin_segun
fin
```

Ejemplo 4.13 página 173

```
algoritmo Dia_semana_modificado
// según día de la semana el 1 del corriente,
// calcular el día de la semana que es hoy
var
  entero: día, d1
  caracter: dia1
inicio
  escribir('El día 1 fue (L,M,X,J,V,S,D) ')
  leer(dia1)
  segun_sea dia1 hacer
    'L':
      d1 ← 0
    'M':
      d1 ← 1
    'X':
      d1 ← 2
    'J':
      d1 ← 3
    'V':
      d1 ← 4
    'S':
      d1 ← 5
    'D':
      d1 ← 6
    si_no
      d1 ← -40
  fin_segun

  escribir('Ingrese día número')
  leer(dia)
  dia ← dia + d1

  segun_sea dia MOD 7 hacer
    1:
      escribir('Lunes')
    2:
      escribir('Martes')
    3:
      escribir('Miércoles')
    4:
      escribir('Jueves')
    5:
      escribir('Viernes')
    6:
      escribir('Sábado')
    0:
      escribir('Domingo')
  fin_segun
fin
```

Ejemplo 4.14 página 174

```
algoritmo Digitos
// indicar si un número entero leído de teclado
// tiene 1, 2, 3 o más dígitos. Considerar
// los negativos
var
  entero: n
inicio
  leer(n)
  segun_sea n hacer
    -9 .. 9:
      escribir('Tiene un dígito')
    -99 .. 99:
      escribir('Tiene dos dígitos')
    -999 .. 999:
      escribir('Tiene tres dígitos')
    si_no
      escribir('Tiene más de tres')
  fin_segun
fin
```

Ejemplo 4.15 página 176

```
algoritmo Mayor
// leer tres números A, B, y C
// y visualizar el valor del mayor
var
  real: A, B, C, mayor
inicio
  leer(A, B, C)
  si A > B entonces
    si A > C entonces
      mayor ← A
    si_no
      mayor ← C
    fin_si
  si_no
    si B > C entonces
      mayor ← B
    si_no
      mayor ← C
    fin_si
   fin_si
   escribir('Tiene más de tres')
fin
```

Ejemplo 4.18 página 178

```
algoritmo Hora_segundo_siguiente
// ingresada la hora HH, MM, SS
// calcula la hora dentro de un segundo
var
  entero : hh, mm, ss
inicio
  escribir('Ingresar hh,mm,ss')
  leer(hh, mm, ss)
  si (hh < 24) y (mm < 60) y (ss < 60) entonces
    ss ← ss + 1
    si ss = 60 entonces
      ss ← 0
      mm ← mm + 1
      si mm = 60 entonces
        mm ← 0
        hh ← hh + 1
        si hh = 24 entonces
          hh ← 0
        fin_si
      fin_si
    fin_si
      escribir(hh, ':', mm, ':', ss)
  fin_si
fin
```

Ejemplo 4.18 página 178

```
algoritmo Mayor
// leer tres números A, B, y C
// y visualizar el valor del mayor
var
  real: A, B, C, mayor
inicio
  leer(A, B, C)
  si A > B entonces
    si A > C entonces
      mayor ← A
    si_no
      mayor ← C
    fin_si
  si_no
    si B > C entonces
      mayor ← B
    si_no
      mayor ← C
    fin_si
   fin_si
   escribir('Tiene más de tres')
fin
```

Actividad resuelta 4.2 página 181

```
1. leer distancia, duración de la estancia y precio por kilómetro
2. comprobar si distancia > 800 km y duración > 7 días
3. cálculo del precio total del billete
     precio_total = distancia * 2.5
     si distancia > 800 km y duración > 7 días
     precio_total = (distancia * 2.5) – 30 / 100 * precio_total
```

```
algoritmo BILLETE
// determinar billete de ida y
// vuelta en avión
var
  entero : E
  real : D, PT
inicio
  leer(E)
  PT ← 2.5 * D
  si (D > 800) y (E > 7) entonces
    PT ← PT – PT * 30 / 100
  fin_si
  escribir('Precio del billete', PT)
fin
```

Actividad resuelta 4.3 página 181

```
1. leer nombre del turno, horas trabajadas (HT) y día de la semana
2. si el turno es nocturno, aplicar la fórmula JORNAL = 8 * HT
3. si el turno es diurno, aplicar la fórmula JORNAL = 5 * HT
4. si el día es domingo:
     turno_diurno     JORNAL = (5 + 2) * HT
     turno_nocturno   JORNAL = (8 + 3) * HT
```

```
algoritmo JORNAL
// determinar jornal diario
var
  cadena : Dia, Turno
  real : HT, Jornal
inicio
  leer(HT, Dia, Turno)
  si Dia < > 'Domingo' entonces
    si Turno = 'diurno' entonces
      JornaL ← 5 * HT
    si_no
      JornaL ← 8 * HT
    fin_si
  si_no
    si Turno = 'diurno' entonces
      JornaL ← 7 * HT
    si_no
      JornaL ← 11 * HT
    fin_si
  fin_si
  escribir('Jornal')
fin
```

Actividad resuelta 4.4 página 182

```
algoritmo DIAS_SEMANA1
// escribir los nombres de los días de la
// semana en función de la variable DIA
var
  entero : Dia
inicio
  leer(Dia)
  si Dia = 1 entonces
    escribir('LUNES')
  si_no
    si Dia = '2' entonces
      escribir('MARTES')
    si_no
      si Dia = '3' entonces
        escribir('MIERCOLES')
      si_no
        si Dia = '4' entonces
          escribir('JUEVES')
        si_no
          si Dia = '5' entonces
            escribir('VIERNES')
          si_no
            si Dia = '6' entonces
              escribir('SABADO')
            si_no
              si Dia = '7' entonces
                escribir('DOMINGO')
              si_no
                escribir('error')
                escribir('rango 1-7')
              fin_si
            fin_si
          fin_si
        fin_si
      fin_si
    fin_si
  fin_si
fin
```

```
algoritmo DIAS_SEMANA2
// alternativa del anterior
var
  entero : Dia
inicio
  leer(Dia)
  segun_sea Dia hacer
    1: escribir('LUNES')
    2: escribir('MARTES')
    3: escribir('MIERCOLES')
    4: escribir('JUEVES')
    5: escribir('VIERNES')
    6: escribir('SABADO')
    7: escribir('DOMINGO')
  en_otro_caso
    escribir('rango 1-7')
  fin_segun
fin
```

Ejercicio 4.1a página 185

```
algoritmo ESCRIBIR_SI
// escribir sentencia si
var
  real : angulo
inicio
  leer(angulo)
  si angulo = 90 entonces
    escribir('El ángulo es un ángulo recto')
  si_no
    escribir('El ángulo no es un ángulo recto')
  fin_si
fin
```

Ejercicio 4.1b página 185

```
algoritmo ESCRIBIR_SI
// escribir sentencia si
var
  real : temp
inicio
  leer(temp)
  si temp > 100 entonces
    escribir('Por encima del punto de ebullición')
  si_no
    escribir('Por encima del punto de ebullición')
  fin_si
fin
```

Ejercicio 4.1c página 185

```
algoritmo ESCRIBIR_SI
// escribir sentencia si
var
  real : num, pos, neg
inicio
  pos ← 0
  neg ← 0
  leer(num)
  si num > 0 entonces
    pos ← pos + num
  si_no
    neg ← neg + num
  fin_si
fin
```

Ejercicio 4.1d página 185

```
algoritmo ESCRIBIR_SI
// escribir sentencia si
var
  real : p
inicio
  si (x > y) y (z < 20) entonces
    leer(p)
  fin_si
fin
```

Ejercicio 4.1e página 185

```
algoritmo ESCRIBIR_SI
// escribir sentencia si
var
  real : tiempo
inicio
  si (distancia > 20) y (distancia < 35) entonces
    leer(tiempo)
  fin_si
fin
```

Ejercicio 4.2 página 185

```
algoritmo VALOR_PRIMERO
// introducir dos números e imprimir relación del primero
var
  real : num1, num2
inicio
  escribir('Introduzca un número')
  num1 ← leer(numero)
  escribir('Introduzca otro número')
  num2 ← leer(numero)
  si num1 > num2 entonces
    escribir('El primer número es el mayor')
  si_no
    si num1 < num2 entonces
      escribir('El primer número es el más pequeño')
    si_no
      escribir('Los números son iguales')
    fin_si
  fin_si
fin
```

Ejercicio 4.3 página 185

```
algoritmo VALOR_CENTRAL
// dados tres número deducir cuál es el central
var
  real : num1, num2, num3
inicio
  leer(num1, num2, num3)
  segun_sea num1 hacer
    num1 > num2 y num1 < num3
      escribir('El primer número es el central')
    num1 < num2 y num2 < num3
      escribir('El segundo número es el central')
    num1 < num2 y num2 > num3
      escribir('El tercer número es el central')
  fin_segun
fin
```

```
algoritmo VALOR_CENTRAL
// dados tres número deducir cuál es el central
// alternativa del anterior
var
  real : num1, num2, num3
inicio
  leer(num1, num2, num3)
  si num1 > num2 y num1 < num3 entonces
    escribir('El primer número es el central')
  si_no
    si num2 > num1 y num2 < num3 entonces
      escribir('El segundo número es el central')
    si_no
      escribir('El tercer número es el central')
    fin_si
  fin_si
fin
```

Ejercicio 4.4 página 185

```
algoritmo RAIZ_CUADRADA
// calcular la raíz cuadrada de un número
var
  real : num, rnum
inicio
  leer(num)
  si num > 0 entonces
    rnum ← raiz2(num)
    escribir('La raíz cuadrada es ', rnum)
  si_no
    escribir('El número es negativo')
  fin_si
fin
```

Ejercicio 4.5 página 185

```
algoritmo ES_PAR
// deducir si una variable numérica es par
var
  real : num
inicio
  leer(num)
  si num MOD 2 = 0 entonces
    escribir('El número ', num, ' es par')
  si_no
    escribir('El número ', num, ' es impar')
  fin_si
fin
```

Ejercicio 4.6 página 185

```
algoritmo FECHA_DIA_SIGUIENTE
// ingresada una fecha DIA, MES, AÑO
// calcula la fecha del día siguiente
var
  entero : DIA, MES, AÑO
inicio
  escribir('Ingresar DIA,MES,AÑO')
  leer(DIA, MES, AÑO)
  si (MES < 12) y (DIA < 30) entonces
    DIA ← DIA + 1
    si DIA = 31 entonces
      DIA ← 1
      MES ← MES + 1
      si MES = 13 entonces
        MES ← 1
        AÑO ← AÑO + 1
      fin_si
    fin_si
      escribir(hh, ':', mm, ':', ss)
  fin_si
fin
```

Ejercicio 4.7 página 185

```
algoritmo ESTADISTICA_PESOS
// estadística de pesos de alumnmos
var
  real : PESO, M39, M4049, M5059, M60
  entero : P39, P4049, P5059, P60
inicio
  P39 ← 0
  M39 ← 0
  P4049 ← 0
  M4049 ← 0
  P5059 ← 0
  M5059 ← 0
  P60 ← 0
  M60 ← 0
  datos:
    leer(PESO)
    segun_sea PESO hacer
      PESO = 0:
        ir_a(medias)
      PESO < 40:
        P39 ← P39 + 1
        M39 ← M39 + PESO
      PESO > 39 y PESO < 50:
        P4049 ← P4049 + 1
        M4049 ← M4049 + PESO
      PESO > 49 y PESO < 60:
        P5059 ← P5059 + 1
        M5059 ← M5059 + PESO
      PESO >= 60:
        P60 ← P60 + 1
        M60 ← M60 + PESO
    fin_segun
  ir_a(datos)
  medias:
    M39 ← M39 / P39
    M4049 ← M4049 / P4049
    M5059 ← M5059 / P5059
    M60 ← M60 / P60
  escribir(M39, M4049, M5059, M60)
fin
```

Ejercicio 4.8 página 185

```
algoritmo DIVISORES
// averiguar si introducidos dos números,
// uno es divisor del otro
var
  real : num1, num2
inicio
  escribir(‘Ingresar num1’)
  leer(num1)
  escribir(‘Ingresar num2’)
  leer(num2)
  si num1 MOD num2 = 0 entonces
    escribir(num2, ‘ es divisor de ’, num1)
  si_no
    si num2 MOD num1 = 0 entonces
      escribir(num1, ‘ es divisor de ’, num2)
    si_no
      escribir(‘No son divisores’)
    fin_si
  fin_si
fin
```

Ejercicio 4.9 página 185

```
algoritmo ANGULOS
// aceptar un ángulo en grados y visualizar el tipo de
// ángulo correspondiente
var
  entero : ANG
inicio
  escribir(‘Introduzca ANG’)
  leer(ANG)
  segun_sea ANG hacer
    ANG < 90: escribir(‘Angulo agudo’)
    ANG > 90: escribir(‘Angulo obtuso’)
    ANG = 90: escribir(‘Angulo recto’)
  en_otro_caso
    escribir(‘Introduzca un ángulo válido’)
  fin_segun
fin
```

Ejercicio 4.10 página 185

```
algoritmo CALIFICACIONES
// convertir las calificaciones numéricas
// a calificaciones alfabéticas
var
  caracter: LETRA
  entero: calificacion
inicio
  escribir('Introduzca calificación')
  leer(calificacion)
  segun_sea calificacion hacer
    calificacion >= 90:
      LETRA ← A
    calificacion < 90 y calificacion >= 80:
      LETRA ← B
    calificacion < 80 y calificacion >= 70:
      LETRA ← C
    calificacion < 70 y calificacion = 69:
      LETRA ← D
    calificacion < 69:
      LETRA ← F
  en_otro_caso
    escribir('ERROR')
  fin_segun
  escribir('La calificación es ', calificacion)
fin
```

Ejercicio 4.11 página 186

```
algoritmo OP_VARIABLE
// seleccionar operación aritmética a ejecutar dependiendo
// del valor de una variable 'seleccionOp'
var
  caracter: seleccionOp
  real: num1, num2, num3
inicio
  escribir('Introduzca operación')
  leer(seleccionOp)
  segun_sea seleccionOp hacer
    SUMA:
      num3 ← num1 + num2
    RESTA:
      num3 ← num1 – num2
    MULTIPLICACION:
      num3 ← num1 * num2
    DIVISION:
      num3 ← num1 / num2
  fin_segun
fin
```

Ejercicio 4.12 página 186

```
algoritmo CODIGO_DE_OPERACION
// complicación del anterior
var
  entero: COD
  real: num1, num2, num3
inicio
  escribir(‘Introduzca dos números’)
  escribir(‘Introduzca código’)
  leer(COD)
  segun_sea COD hacer
    1:num3 ← num1 + num2
    2:num3 ← num1 – num2
    3:num3 ← num1 * num2
    4:num3 ← num1 / num2
  en_otro_caso
    escribir(‘Algo salió mal’)
  fin_segun
  escribir(‘num3’)
fin
```

Ejercicio 4.13 página 186

```
algoritmo CHECKFORM
// chequeo de introducción de datos en formulario
var
  real : MES, DIA
inicio
  escribir('Introduzca un mes (1 para Enero, 2 para Febrero, ...)')
  escribir('Introduzca un día del mes')
  leer(MES)
  leer(DIA)

  si MES < 1 y MES > 12 entonces
    ir_a(1)
  si_no
    si MES != trunc(MES) entonces
      ir_a(1)
    fin_si
  fin_si

  si DIA < 1 y DIA > 31 entonces
    ir_a(2)
  si_no
    si DIA != trunc(DIA) entonces
      ir_a(2)
    fin_si
  fin_si

  si MES = 2 entonces
    si DIA < 1 y DIA > 29 entonces
      ir_a(2)
    si_no
      si DIA != trunc(DIA) entonces
        ir_a(2)
    fin_si
  fin_si

  (1) escribir('El mes introducido no es válido')
  (2) escribir('El día introducido no es válido')
fin
```

Ejercicio 4.14 página 186

```
algoritmo ASCENSOR
// chequeo de introducción de datos en formulario
// PISO = piso actual
var
  entero : TECLA, SUBIR, BAJAR, PISO, RECORR
inicio
  SUBIR ← 25
  BAJAR ← 00
  leer(TECLA)
  leer(PISO)

  si TECLA > PISO entonces
    RECORR ← TECLA – PISO
    subirCoche(RECORR)
  si_no
    RECORR ← PISO – TECLA
    bajarCoche(RECORR)
  fin_si

fin
```

Ejemplo 5.01 página 188
Ejemplo 5.1 página 190

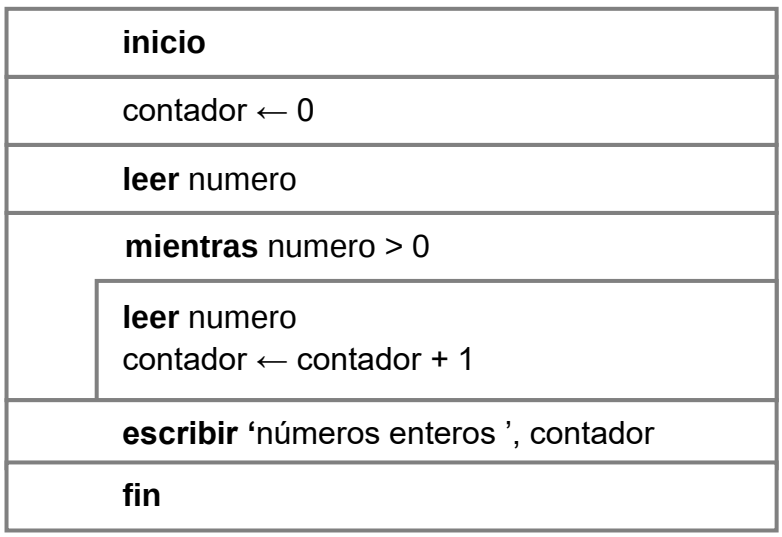
```
algoritmo SUMA
// sumar una lista de números introducidos por teclado
var
  entero : N, TOTAL
  real : NUMERO,  SUMA
inicio
  leer(N)
  TOTAL ← N
  SUMA ← 0
  mientras TOTAL > 0 hacer
    leer(NUMERO)
    SUMA ← SUMA + NUMERO
    TOTAL ← TOTAL – 1
  fin_mientras
    escribir(‘La suma de los ’, N, ‘ números es ’, SUMA)
fin
```

Ejemplo 5.02 página 189

```
algoritmo SUMA
// variante del anterior
var
  entero : N, TOTAL
  real : NUMERO,  SUMA
inicio
  leer(N)
  TOTAL ← N
  SUMA ← 0
  repetir
    leer(NUMERO)
    SUMA ← SUMA + NUMERO
    TOTAL ← TOTAL – 1
  hasta_que TOTAL = 0
    escribir(‘La suma de los ’, N, ‘ números es ’, SUMA)
fin
```

Ejemplo 5.2 página 191

```
algoritmo CUENTA_ENTEROS
// contar los enteros positivos introducidos por teclado
var
  entero : numero, contador
inicio
  contador ← 0
  leer(num)
  mientras numero > 0 hacer
    leer(numero)
    contador ← contador + 1
  fin_mientras
    escribir(‘El número de enteros positivos es ’, contador)
fin
```



Ejemplo 5.2.2 página 193

```
algoritmo INTERES_CAPITAL
// visualizar el interés producido por un capital
var
  real : capital, tasa, interes
inicio
  leer(capital)
  tasa ← 10
  mientras tasa <> 20 hacer
    interes ← tasa * 0.01 * capital
    escribir('El interés producido es ', interes)
    tasa ← tasa + 2
  fin_mientras
  escribir('Continuación')
fin
```

```
algoritmo INTERES_CAPITAL
// visualizar el interés producido por un capital
// bucle infinito
var
  real : capital, tasa, interes
inicio
  leer(capital)
  tasa ← 10
  mientras tasa <> 20 hacer
    interes ← tasa * 0.01 * capital
    escribir('El interés producido es ', interes)
    tasa ← tasa + 3
  fin_mientras
  escribir('Continuación')
fin
```

Ejemplo 5.2.3 página 194

```
algoritmo SUMA_ENTRADAS
var
  real : N, Suma
  caracter : Resp
inicio
  escribir('Existen más números en la lista s/n')
  leer(Resp)
  mientras (Resp = 'S') o (Resp = 's') hacer
    escribir('numero')
    leer(N)
    Suma ← Suma + N
    escribir('Existen más números (s/n)')
    leer(Resp)
  fin_mientras
fin
```

Ejemplo 5.4 página 194

```
algoritmo ITERACIONES
// en el primer caso produce 0, 1, 2, 3, 4 y 5
// en el segundo caso produce 1, 2, 3, 4, 5 y 6
var
  real : N, Suma
  caracter : Resp
inicio
  i ← 0
  mientras i < 6 hacer
    escribir(i)
    i ← i + 1
  fin_mientras

  i ← 0
  mientras i < 6 hacer
    escribir(i)
    i ← i + 1
  fin_mientras
fin
```

Ejemplo 5.5 página 195

```
algoritmo MEDIA_NOTAS
// calcular la media de un conjunto de notas, con un valor
// centinela de –99 que detecte el fin del bucle
var
  real : media
  entero : total, n, nota
inicio
  total ← 0
  n ← 0  // número de alumnos
  leer(nota)
  mientras nota <> –99 hacer
    total ← total + nota
    n ← n + 1
    leer(nota)
  fin_mientras
  media ← total / n
  escribir('La media es ', media)
fin
```

Ejemplo 5.6 página 196

```
algoritmo INVERTIR_NUMERO
// leer un número y obtener el número inverso
var
  entero : num, digitoSig
inicio
  num ← 198
  escribir(‘Número: ← ’, num)
  escribir(‘Número en orden inverso: ’)
  hacer
    digitoSig = num MOD 10
    escribir(digitoSig)
    num ← num DIV 10
  mientras num > 0
fin
```

Ejemplo 5.7 página 198

```
algoritmo POTENCIAS
// visualizar las potencias de dos
// cuerpos cuyos valores estén en
// el rango 1 a 1000
// ejercicio con while
potencia = 1;
while(potencia < 1000){
  cout << potencia << endl;
  potencia *= 2;
} // fin de while
```

```
algoritmo POTENCIAS
//
//
//
// ejercicio con do-while
potencia = 1;
do{
  cout << potencia << endl;
  potencia *= 2;
} while(potencia < 1000)
```

Ejemplo 5.5.1 página 198

```
algoritmo REPETIR
var
  real : numero
  entero : contador
inicio
  contador ← 1
  repetir(‘Número: ← ’, num)
    leer(numero)
    contador ← contador + 1
  hasta_que contador > 30
    escribir(‘Números leídos 30 ’)
fin
```

Ejemplo 5.8 página 199

```
algoritmo FACTORIAL
// calcular el factorial de un número N
// N! = N*(N-1)*( N-2),…,3*2*1
var
  entero : I, N
  real : Factorial
inicio
  leer(N)
  Factorial ← 1
  I ← 1
  repetir
    Factorial ← Factorial * I
    I ← I + 1
  hasta_que I = N + 1
  escribir('El factorial del número ', N, ' es ', Factorial)
fin
```

Ejemplo 5.9 página 200

```
algoritmo MAS_PEQUEÑO
// encontrar el natural más pequeño (num) para el cual la suma
// 1+2+3+…+num es menor o igual que límite
var
  entero : num, limite, suma
inicio
  leer(limite)
  num ← 0
  suma ← 0
  repetir
    num ← num + 1
    suma ← suma + num
  hasta_que suma > limite
  escribir(num, suma)
fin
```

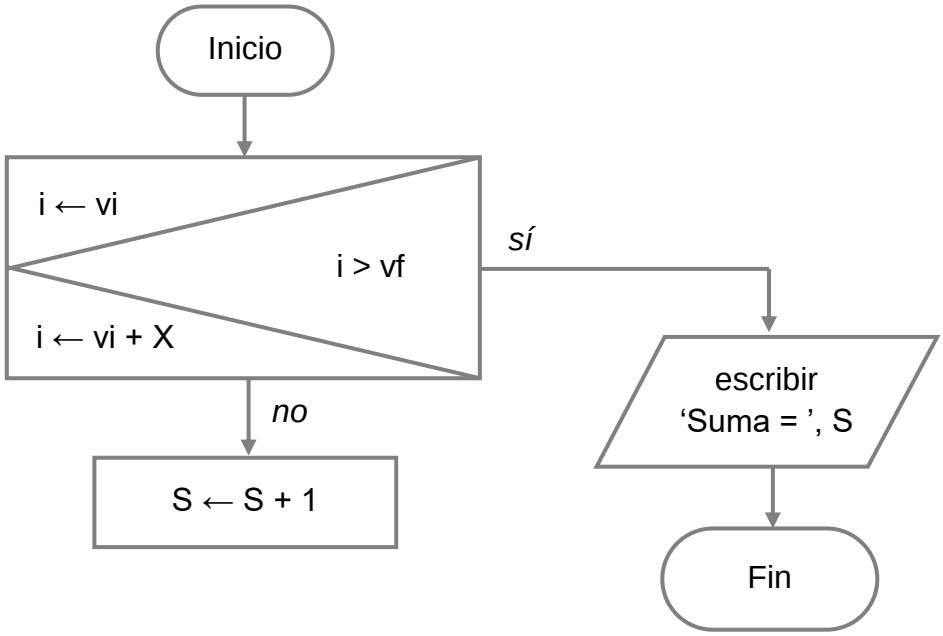
Ejemplo 5.10 página 200

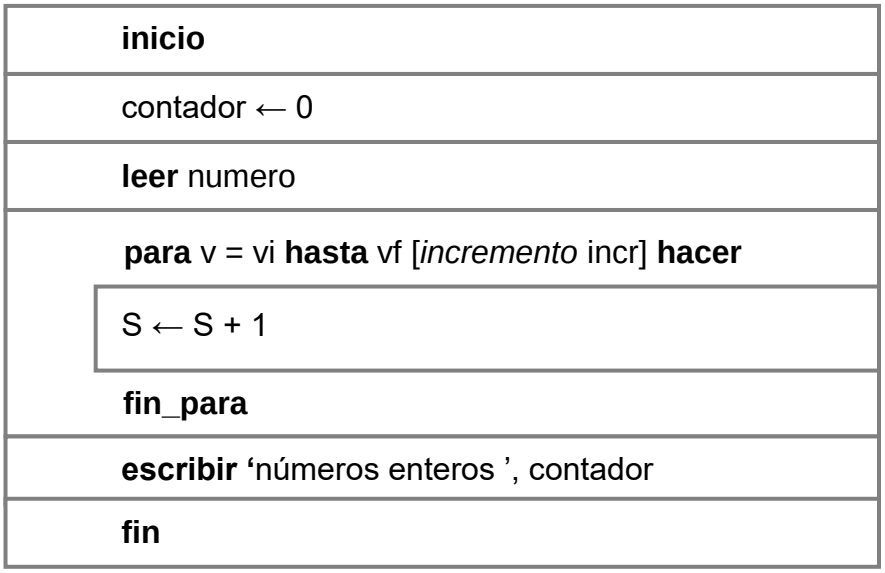
```
algoritmo UNO_CIEN
// escribir los números del 1 al 100
var
  entero : num
inicio
  num ← 1
  repetir
    escribir(num, suma)
    num ← num + 1
  hasta_que num > 100
fin
```

Ejemplo 5.11 página 201

```
algoritmo VALIDAR_MES
// detectar entradas entre 1 y 12, rechazando las restantes
var
  entero : mes
inicio
  escribir('Introducir número de mes')
  repetir
    leer(limite)
    si (mes < 1) o (mes > 12) entonces
      escribir('Valor entre 1 y 12')
    fin_si
  hasta_que (mes >= 1) y (mes <= 12)
fin
```

Ejemplo siguiente página 203






```
algoritmo SUMA
// calcular la suma de los N primeros enteros
// este algoritmo equivale a los dos diagramas anteriores
var
  entero : I, N, S
inicio
  S ← 0
  leer(N)
  desde I ← 1 hasta N hacer
    S ← S + 1
  fin_desde
  escribir(‘Suma = ’, S)
fin
```

Ejemplo 5.12 página 205

```
algoritmo 1
leer(informacion)
repetir
  procesar(informacion)
  leer(informacion)
hasta_que fin_de_lectura
```

```
algoritmo 2
leer(informacion)
mientras_no fin_de_lectura
  procesar(informacion)
  leer(informacion)
fin_mientras
```

```
algoritmo 3
iterar
  leer(informacion)
  si fin_de_lectura entonces
    salir_bucle
  fin_si
  procesar(informacion)
fin_mientras
```

Ejemplo 5.13 página 206

```
algoritmo PEJ
hacer
  escribir(‘Introduzca un número de identificación’)
  leer(numId)
  si (numId < 1000 o numId > 1999) entonces
    escribir(‘Número no válido’)
    escribir(‘Por favor, introduzca otro número’)
  si_no
    interrumpir
  fin_si
mientras (expresión cuyo valor sea siempre verdadero)
```

Ejemplo 5.14 página 207

```
algoritmo PEJ
var entero : t
desde t ← 0 hasta t < 100 incremento 1 hacer
  escribir(t)
  si (t = 1d) entonces
    interrumpir
  fin_si
fin_desde
```

Ejemplo 5.15 página 207

```
algoritmo PEJ
// imprime 1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15, 17, 18, 19
i = 0
desde i = 0 hasta 20 inc 1 hacer
  si (i mod 4 = 0) entonces
    continuar
  fin_si
  escribir(i, ‘, ’)
fin_desde
```

Ejemplo 5.16 página 211

```
algoritmo CIUDADMAYOR
// deducir la ciudad más grande de una provincia
// ocho veces
var
  entero : i      // contador de provincias
  entero : j      // contador de ciudades
  entero : MAYOR  // ciudad de mayor población
  entero : CIUDAD // población de la ciudad
inicio
  i ← 0
  mientras i <= 8 hacer
    MAYOR ← 0
    i ← 1
    mientras j <= 25 hacer
      leer(CIUDAD)
      si CIUDAD > MAYOR entonces
        MAYOR ← CIUDAD
      fin_si
      j ← j + 1
    fin_mientras
    escribir('La ciudad mayor es ', MAYOR)
    i ← i + 1
  fin_mientras
fin
```

Ejemplo 5.17 página 212

```
algoritmo FACTORIAL2
// calcular el factorial de n números leídos del terminal
var
  entero : i, NUM, n
  real : FACT
inicio
  {lectura de la cantidad de números}
  leer(n)
  desde i ← 1 hasta n hacer
    leer(n)
    FACT ← 1
    desde j ← 1 hasta n hacer
      FACT ← FACT * j
    fin_desde
    escribir( 'El factorial del número ', NUM, 'es ', FACT)
  fin_desde
fin
```

Ejemplo 5.18 página 213

```
algoritmo PRIMOS
// imprimir todos los números primos entre 2 y 100 inclusive
var
  entero : i, divisor
  logico : primo
inicio
  desde i ← 2 hasta 100 hacer
    primo ← verdad
    divisor ← 2
    mientras (divisor <= raiz2(i)) y primo hacer
      si i MOD divisor = 0 entonces
        primo ← falso
      si_no
        divisor ← divisor + 1
      fin_si
    fin_mientras
      si primo entonces
        escribir(i,   ' ')
      fin_si
  fin_desde
fin
```

Actividad resuelta 5.1 página 216

```
algoritmo FACTORIAL
// calcular el factorial de un número N utilizando la estructura
// desde
var
  entero : I, N
  real : FACTORIAL
inicio
  leer(N)
  FACTORIAL ← 1
  desde I ← 1 hasta N hacer
    FACTORIAL ← FACTORIAL * 1
  fin_desde
  escribir('El factorial de ', N, 'es ', FACTORIAL)
fin
```

Actividad resuelta 5.2 página 216

```
algoritmo POTENCIAS
// imprimir las 30 primeras potencias de 4
var
  entero : N
inicio
  desde N ← 1 hasta 30 hacer
    escribir(4 ^ N)
  fin_desde
fin
```

Actividad resuelta 5.3 página 216

```
algoritmo SUMANENTEROS
// calcular la suma de los N primeros enteros utilizando desde
var
  entero : I, N
  real : SUMA
inicio
  leer(N)
  SUMA ← 0
  desde I ← 1 hasta N hacer
    SUMA ← SUMA + 1
  fin_desde
  escribir(SUMA)
fin
```

Actividad resuelta 5.4 página 217

```
algoritmo SUMAIMPARES
// imprimir la suma de los números impares menores o iguales que n
var
  entero : I, N
  real : SUMA
inicio
  SUMA ← 0
  leer(N)
  desde I ← 1 hasta N inc 2 hacer
    SUMA ← SUMA + 1
  fin_desde
  escribir(SUMA)
fin
```

Actividad resuelta 5.5 página 217

```
algoritmo COCIENTE
// dados dos enteros, calcular su cociente y su resto
var
  entero : M, N, Q, R
inicio
  leer(M, N)  // M dividendo, N divisor
  R ← M
  Q ← 0
  repetir
    R ← R – N
    Q ← Q + 1
  hasta_que R < N
  escribir('dividendo', M, 'divisor', N, 'cociente', Q, 'resto', R)
fin
```

Actividad resuelta 5.6 página 217

```
algoritmo SUMAPARES
// obtener la suma de los números pares hasta 1000 inclusive
// método 1
var
  real : N, S
inicio
  S ← 2
  N ← 4
  mientras N <= 1000 hacer
    S ← S + N
    N ← N + 2
  fin_mientras
fin
```

```
algoritmo SUMAPARES
// obtener la suma de los números pares hasta 1000 inclusive
// método 2. Idéntica cabecera y declaraciones
var
  real : N, S
inicio
  S ← 2
  N ← 4
  repetir
    S ← S + N
    N ← N + 2
  hasta_que N > 1000
fin
```

Actividad resuelta 5.7 página 218

```
algoritmo BUSCARVOCAL
// buscar y escribir la primera vocal leída del teclado
var
  caracter : P
inicio
  repetir
    leer(P)
  hasta_que P = ‘a’ o P = ‘e’ o P = ‘i’ o P = ‘o’ o P = ‘u’
  escribir(‘Primera vocal: ’, P)
fin
```

Actividad resuelta 5.8 página 218

```
algoritmo MENOR100
// leer una serie de números hasta obtener uno inferior a 100
var
  real : N
inicio
  repetir
    escribir(‘Introduzca un número’)
    leer(P)
  hasta_que N < 100
  escribir(‘El número es: ’, N)
fin
```

Actividad resuelta 5.9 página 218

```
algoritmo SN
// escribir en una pantalla una frase
var
  caracter : RESP
inicio
  repetir
    escribir('Desea continuar S/N')
    leer(RESP)
  hasta_que (RESP = 'S') o (RESP = 'N')
fin
```

Actividad resuelta 5.10 página 219

```
algoritmo NUMERO1_5
// leer sucesivamente números del teclado hasta que aparezca un
// número comprendido entre 1 y 5
var
  entero : N
inicio
  repetir
    escribir('Introduzca un número comprendido entre 1 y 5')
    leer(N)
  hasta_que (N >= 1) o (N <= 5)
  escribir('El número encontrtado es: ', N)
fin
```

Actividad resuelta 5.11 página 219

```
algoritmo FACTORIAL
// calcular el factorial de un número N
var
  entero : I, N
  real : F
inicio
  F ← 1
  I ← 1
  leer(N)
  repetir
    F ← F * I
    I ← I + 1
  hasta_que I = N + 1
  escribir('El factorial d: ', N, 'es ', F)
fin
```

Actividad resuelta 5.12 página 220

```
algoritmo MAXIMO
// calcular el máximo de una serie de 100 números
var
  entero : I, N, MAX
inicio
  leer(N)
  I ← 1
  MAX ← N
  repetir
    I ← I + 1
    leer(N)
    si N > MAX entonces
      MAX ← N
    fin_si
  hasta_que I = 100
  escribir('El mayor es ', MAX)
fin
```

```
algoritmo MAXIMO
// calcular el máximo de una serie de 100 números
var
  entero : I, N, MAX
inicio
  leer(N)
  MAX ← N
  para I = 2 hasta I = 100 hacer
    leer(N)
    si N > MAX entonces
      MAX ← N
    fin_si
  fin_para
  escribir('El mayor es ', MAX)
fin
```

```
algoritmo MAXIMOPOSITIVOS
// calcular el máximo de una serie de 100 números
var
  entero : I, N, MAX
inicio
  leer(N)
  I ← 0
  MAX ← 0
  repetir
    leer(N)
    I ← I + 1
    si N > MAX entonces
      MAX ← N
    fin_si
  hasta_que I = 100
  escribir('El mayor es ', MAX)
fin
```

Actividad resuelta 5.13 página 223

```
algoritmo TABLAS
// obtener las tablas de multiplicar
var
  entero : I, J, PROD
inicio
  para I ← 1 hasta 9 hacer
  escribir('Tabla del ', I)
    para J ← 1 hasta 10 hacer
      PROD ← I * J
      escribir(I, 'por ', J, '= ', PROD)
    fin_para
  fin_para
fin
```

Actividad resuelta 5.14 página 223

```
algoritmo MAYOR
// calcular el número mayor de una lista de N números
var
  entero : I, N
  real : NUM, MAX
inicio
  leer(N)
  leer(NUM)
  MAX ← NUM
  para I ← 2 hasta N hacer
    leer(NUM)
    si NUM > MAX entonces
      MAX ← NUM
    fin_si
  fin_para
fin
```

Actividad resuelta 5.15 página 224

```
algoritmo MAXMIN
// determinar los valores máximo y mínimo de una lista de 100 números
var
  entero : I, N
  real : MAX, MIN, NUM
inicio
  leer(NUM)
  MAX ← NUM
  MIN ← NUM
  para I ← 2 hasta 100 hacer
    leer(NUM)
    si NUM > MAX entonces
      MAX ← NUM
    si_no
        si NUM < MIN entonces
          MIN ← NUM
        fin_si
    fin_si
  fin_para
  escribir('Máximo ', MAX, 'Mínimo ', MIN)
fin
```

Actividad resuelta 5.16 página 224

```
algoritmo MEDIAPAREJAS
// determinar el valor máximo de las medias correspondientes a
// parejas de valores sucesivos. El último valor es 999
var
  entero : N1, N2
  real : M, MAX
inicio
  leer(N1, N2)
  MAX ← (N1 + N2) / 2
  mientras (N1 != 999) o (N2 != 999) hacer
    leer(N1, N2)
    M ← (N1 + N2) / 2
    si M > MAX entonces
      MAX ← M
    fin_si
  fin_para
  escribir('La media máxima es ', MAX)
fin
```

Actividad resuelta 5.17 página 225

```
algoritmo CHECK_INT
// detectar entradas numéricas –enteros– erróneas
var
  entero : SW
  real : N
inicio
  SW ← 0
  leer(NUM)
  MIN ← NUM
  mientras SW ← 0 hacer
    leer(N)
    si N != ent(N) entonces
      escribir('Dato no válido')
      escribir('Ejecute nuevamente')
      SW ← 1
    si_no
      escribir('Correcto, ', N, ' es entero')
    fin_si
  fin_mientras
fin
```

Actividad resuelta 5.18 página 225

```
algoritmo FACTORIAL_REPEAT
// calcular el factorial de un número con la estructura repetir
var
  entero : I, N
  real : FACT
inicio
  repetir
    leer(N)
  hasta_que N > 0
  FACT ← 1
  repetir
    FACT ← FACT * I
    I ← I + 1
    hasta_que I = N + 1
    escribir('El factorial de ', N, ' es ', FACT)
fin
```

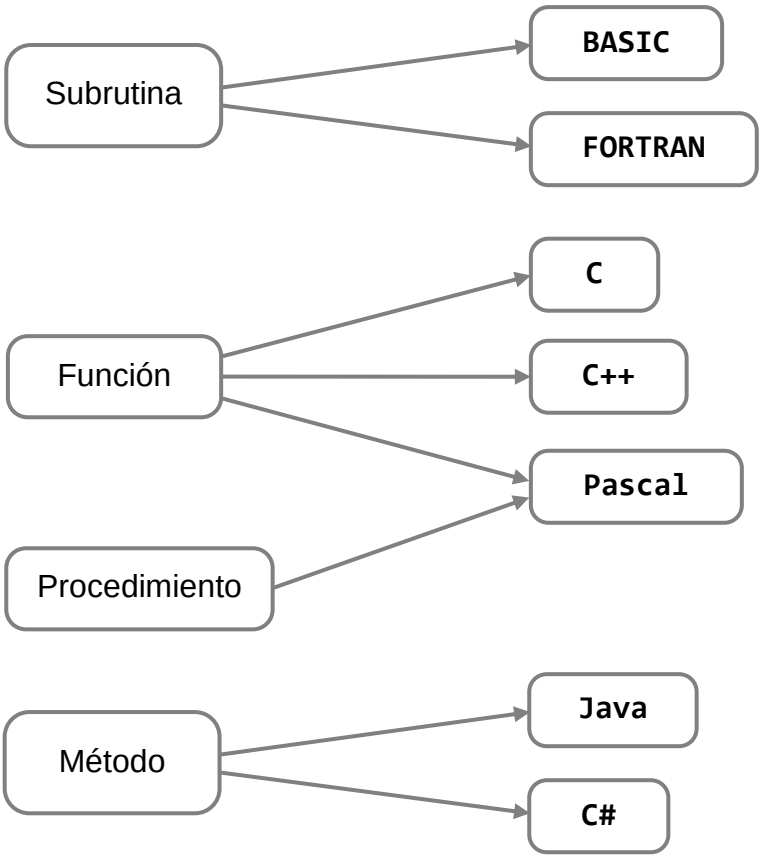
```
algoritmo FACTORIAL_FOR
// calcular el factorial de un número con la estructura para
var
  entero : I, N
  real : FACT
inicio
  leer(N)
  si N < 0 entonces
    escribir('El número debe ser positivo ')
  si_no
    FACT ← 1
    si N > 1 entonces
      desde I ← 2 hasta N hacer
        FACT ← FACT * I
      fin_desde
    fin_si
    escribir('El factorial de ', N, ' es ', FACT)
  fin_si
fin
```

Actividad resuelta 5.19 página 227

```
algoritmo MEDIA_NOTAS
// calcular la media de las notas de tres materias de una serie
// de alumnos. La marca final es '***'
var
  cadena : NOMBRE
  real : MEDIA, MAT1, MAT2, MAT3
inicio
  // entrada de datos de alumnos
  leer(NOMBRE)
  mientras NOMBRE != '***' hacer
    leer(MAT1, MAT2, MAT3)
    MEDIA ← (MAT1 + MAT2 + MAT3) / 3
    escribir(NOMBRE, MEDIA)
    leer(NOMBRE)
  fin_mientras
fin
```

FALTAN LOS EJERCICIOS

SUBPROGRAMAS



Ejemplo 6.1 página 236

```
// definición de la función: y=x^n (potencia n de x)
real : funcion potencia(E real:x; E entero:n)
var
  entero : i, y
inicio
  y ← 1
  para i ← 1 hasta abs(n) hacer
    y ← y * x
  fin_desde
  si n < 0 entonces
    y ← 1 / y
  fin_si
  devolver(y)
fin_funcion
```

Ejemplo 6.2 página 236

```
// función potencia de N (positivo, posible fraccionario) elevado
// a A (real)
algoritmo POTENCIA
var
  real : a, n
inicio
  escribir('Ingresar un número positivo')
  leer(n)
  escribir('Ingresar un exponente')
  leer(a)
fin

real : funcion potencia(E real: n, a)
inicio
  devolver(EXP(a * LN(n)))
fin_funcion
```

Ejemplo 6.4 página 237

```
// realizar el diseño de la función y=x^3 (cálculo del cubo
// de un número)
algoritmo PRUEBA
var
  entero : n
inicio // programa principal
  N ← cubo(2)
  escribir('2 al cubo es ', N)
  escribir('3 al cubo es ', cubo(3))
fin

real : funcion cubo(E entero: x)
inicio
  devolver(x * x * x)
fin_funcion
```

Ejemplo suelto página 241

```
PROCEDIMIENTO
procedimiento division (E entero: Dividendo, Divisor; S entero:
  Cociente, Resto)
inicio
  Cociente ← Dividendo DIV Divisor
  Resto ← Dividendo – Cociente * Divisor
fin_procedimiento

ALGORITMO PRINCIPAL
algoritmo ARITMETICA
var
  entero : M, N, P, Q, S, T
inicio
  leer(M, N)
  llamar_a division(M, N, P, Q)
  escribir(P, Q)
  llamar_a division(M * N – 4, N + 1, S, T)
  escribir(S, T)
fin
```

ACTIVIDADES Y EJERCICIOS
EN EL ARCHIVO "fun-de-progr.cpp"

EJERCICIOS CAPITULO 6 FALTAN

Ejemplo 7.1 página 283

```
// lectura de veinte valores enteros de un vector F
algoritmo LEER_VECTOR
tipo
  array[1..20] de entero : FINAL
var
  FINAL : F
inicio
  desde i ← 1 hasta 20 hacer
    leer(F[i])
  fin_desde
fin
```

```
// variante del anterior
algoritmo LEER_VECTOR
tipo
  array[1..20] de entero : FINAL
var
  FINAL : F
inicio
  i ← 1
  mientras i <= 20 hacer
    leer(F[i])
    i ← i + 1
  fin_mientras
fin
```

```
// otra variante
algoritmo LEER_VECTOR
tipo
  array[1..20] de entero : FINAL
var
  FINAL : F
inicio
  i ← 1
  repetir
    leer(F[i])
    i ← i + 1
  hasta_que i > 20
fin
```

Ejemplo 7.2 página 284

```
// procesa un array PUNTOS haciendo: a) lectura del array
// b) cálculo de la suma de los valores del array c) cálculo
// de la media de los valores
algoritmo MEDIA_PUNTOS
const
  LIMITE = 40
tipo
  array[1..LIMITE] de real : PUNTUACION
var
  PUNTUACION : PUNTOS
  real : suma, media
  entero : i
inicio
  suma ← 0
  escribir('Datos del array')
  desde i ← 1 hasta LIMITE hacer
    leer(PUNTOS[i])
    suma ← suma + PUNTOS[i]
  fin_desde
  media ← suma / LIMITE
  escribir('La media es ', media)
fin

// ampliación para visulizar los elementos cuyo valor es superior
// a la media
escribir('Elementos del array superiores a la media')
desde i ← 1 hasta LIMITE hacer
  si PUNTOS[i] > media entonces
    escribir(PUNTOS[i])
  fin_si
fin desde
```

Actividad resuelta 8.1 página 330

```
// eliminar los blancos de una frase terminada en un punto
// se supone que es posible leer los caracteres de uno en uno
F array de caracteres de la frase dada
G array de caracteres de la nueva frase
I contador del array F
J contador del array G
algoritmo BLANCO
inicio
  I ← 1
  J ← 0
  F[I] ← leercar()
  repetir
    si F[I] <> ‘ ’ entonces
      J ← J + 1
      G[I] ← F[I]
    fin_si
    I ← I + 1
    F[I] ← leercar()
  hasta_que F[I] = ‘.’
  // escritura de la nueva frase G
  desde I ← 1 hasta J hacer
    escribir(G[I])  // no avanzar línea
  fin_desde
fin
```

Algoritmo página 359

```
// creación de un archivo secuencial
algoritmo CREA_SEC
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ..........................
  fin_registro
  archivo_s de datos_personales: arch
var
  arch              : f
  datos_personales  : persona
inicio
  crear (f, <nombre_en_disco>)
  abrir (f, e, <nombre_en_disco>)
  leer_reg (persona)
  // se utiliza un procedimiento para no tener que detallar
  // la lectura
  mientras no ultimo_dato(persona) hacer
    escribir_f_reg (f, persona)
    // la escritura se realizará campo a campo
    leer_reg (persona)
  fin_mientras
  cerrar(f)
fin
```

Algoritmo/s página 360

```
// consulta de un archivo secuencial
algoritmo CONSULTA_SEC
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ..........................
  fin_registro
  archivo_s de datos_personales: arch
var
  arch              : f
  datos_personales  : persona


inicio
  abrir (f, l, <nombre_en_disco>)
  leer_reg (persona)
  mientras no fda(f) hacer
    leer_f_reg (f, persona)
  fin_mientras
  cerrar(f)
fin


inicio
  abrir (f, l, <nombre_en_disco>)
  leer_reg (persona)
  mientras no fda(f) hacer
    escribir_reg (persona)
    leer_f_reg (f, persona)
  fin_mientras
  cerrar(f)
fin
```

Algoritmo página 360

```
// búsqueda de un determinado registro, con un campo clave ‘x’
// si el archivo no está indexado
algoritmo CONSULTA_SEC
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ..........................
  fin_registro
  archivo_s de datos_personales: arch
var
  arch              : f
  datos_personales  : persona
  <tipo_dato1>      : clavebus
  logico            : encontrado
inicio
  abrir (f, l, <nombre_en_disco>)
  encontrado ← falso
  leer (clavebus)
  mientras no encontrado y no fda(f) hacer
    leer_f_reg (f, persona)
    si igual (clavebus, persona) entonces
      encontrado ← verdadero
    fin_si
  fin_mientras
  si no encontrado entonces
    escribir (“No existe”)
  si_no
    escribir_reg(persona)
  fin_si
  cerrar(f)
fin
```

Algoritmo página 361

```
// búsqueda de un determinado registro, con un campo clave ‘x’
// si el archivo está indexado en orden creciente por el campo
// por el cual realizamos la búsqueda
algoritmo CONSULTA2_SEC
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ..........................
  fin_registro
  archivo_s de datos_personales: arch
var
  arch              : f
  datos_personales  : persona
  <tipo_dato1>      : clavebus
  logico            : encontrado, pasado

inicio
  abrir (f, l, <nombre_en_disco>)
  encontrado ← falso
  pasado ← falso
  leer (clavebus)
  mientras no encontrado y no pasado y no fda(f) hacer
    leer_f_reg (f, persona)
    si igual (clavebus, persona) entonces
      encontrado ← verdadero
    si_no
      si menor (clavebus, persona) entonces
        pasado ← verdadero
      fin_si
    fin_si
  fin_mientras
  si no encontrado entonces
    escribir (“No existe”)
  si_no
    escribir_reg(persona)
  fin_si
  cerrar(f)
fin
```

Algoritmo página 362

```
// dar de alta un determinado registro
// similar a añadir datos a un archivo
algoritmo AÑADE_SEC
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ..........................
  fin_registro
  archivo_s de datos_personales: arch
var
  arch              : f
  datos_personales  : persona

inicio
  abrir (f, e, <nombre_en_disco>)
  leer_reg (persona)
  mientras no ultimo_dato(persona) hacer
    escribir_f_reg (f, persona)
    leer_reg (persona)
  fin_mientras
  cerrar(f)
fin
```

Algoritmo página 363

```
// dar de baja un registro utilizando un archivo transitorio
algoritmo BAJA_SEC
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ..........................
  fin_registro
  archivo_s de datos_personales: arch
var
  arch              : f, faux
  datos_personales  : persona, personaaux
  logico            : encontrado

inicio
  abrir (f, l, “antiguo”)
  crear (faux, “nuevo”)
  leer (personaaux.nombre_campo1)
  encontrado ← falso
  mientras no fda(f) hacer
    leer_f_reg (f, persona)
    si personaaux.nombre_campo1 = persona.nombre_campo1 entonces
      encontrado ← verdad
    si_no
      escribir_f_reg (faux, persona)
    fin_si
  fin_mientras
  si no encontrado entonces
    escribir (“no esta”)
  fin_si
  cerrar(f, faux)
  borrar(“antiguo”)
  renombrar(“nuevo”,  “antiguo”)
fin
```

Algoritmo página 364

```
// modificar un registro utilizando un vector con un indicador
// o bandera
algoritmo MODIFICACION_SEC
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ..........................
  fin_registro
  archivo_s de datos_personales: arch
var
  arch              : f, faux
  datos_personales  : persona, personaaux
  logico            : encontrado

inicio
  abrir (f, l, “antiguo”)
  crear (faux, “nuevo”)
  leer (personaaux.nombre_campo1)
  encontrado ← falso
  mientras no fda(f) hacer
    leer_f_reg (f, persona)
    si personaaux.nombre_campo1 = persona.nombre_campo1 entonces
      encontrado ← verdad
      modificar (persona)
    fin_si
    escribir_f_reg (faux, persona)
  fin_mientras
  si no encontrado entonces
    escribir (“no esta”)
  fin_si
  cerrar(f, faux)
  borrar(“antiguo”)
  renombrar(“nuevo”,  “antiguo”)
fin
```

```
// subprograma de modificación de registro
procedimiento modificar (E/S datos_personales: persona)
var caracter : opcion
  entero     : n

inicio
  escribir (“R.- registro completo”)
  escribir (“C.- campos individuales”)
  escribir (“elija opcion:”)
  leer (opcion)
  segun sea opcion hacer
    ‘R’
      visualizar (persona)
      leer_f_reg (persona)
    ‘C’
      presentar (persona)
      solicitar_campo (n)
      introducir_campo (n, persona)
  fin_segun
fin_procedimiento
```

Algoritmo página 365

```
// creacion de un campo ocupado para distinguir un registro dado de
// baja o modificado de un alta o que nunca contuvo información
algoritmo CREA_DIR
const
  max = <valor>
tipo
  registro: datos_personales
    <tipo_dato1>: cod
    <tipo_dato2>: ocupado
    .......... : ............
    <tipo_daton>: nombre_campon
    .......... : ............
  fin_registro
var
  arch              : f
  datos_personales  : persona

inicio
  crear f, <nombre_en_disco>)
  abrir (f, l/e, <nombre_en_disco>)
  desde i ← 1 hasta max hacer
    persona.ocupado ← ‘ ’
    escribir (f, i, persona)
  fin_desde
  cerrar(f)
fin
```

Algoritmo página 366

```
// alta mediante tratamiento por transformación de clave
algoritmo ALTAS_DIR_TRCL
const
  findatos = <valor1>
  max      = <valor2>
tipo
  registro: datos_personales
    <tipo_dato1>: cod
    <tipo_dato2>: ocupado
    .......... : ............
    <tipo_daton>: nombre_campon
    .......... : ............
  fin_registro
  archivo_d de datos_personales: arch
var
  arch              : f
  datos_personales  : persona, personaaux
  logico            : encontradohueco
  entero            : posi

inicio
  abrir (f, l/e, <nombre_en_disco>)
  leer (personaaux.cod)
  posi ← HASH (personaaux.cod)
  leer (f, posi, persona)
  si persona.ocupado = '*' entonces
    encontradohueco ← falso
    posi ← findatos
    mientras posi < max y no encontradohueco hacer
      posi ← posi + 1
      leer (f, posi, persona)
      si persona.ocupado <> '*' entonces
        encontradohueco ← verdad
      fin_si
    fin_mientras
  si_no
    encontradohueco ← verdad
  fin_si
  si encontradohueco entonces
    leer_otros_campos(personaaux)
    persona ← personaaux
    persona.ocupado ← '*'
    escribir (f, posi, persona)
  si_no
    escribir ("no está")
  fin_si
  cerrar(f)
fin
```

Actividad resuelta 9.1 página 376

```
// creación e introducción de los primeros datos en un archivo
// secuencial, de nombre PERSONAL
algoritmo EJERCICIO_9_1
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ..... : .............
  fin_registro
  archivo_d de datos_personales: arch
var
  arch              : f
  datos_personales  : persona

inicio
  crear (f, 'PERSONAL')
  abrir (f, e, 'PERSONAL')
  llamar_a leer_reg (persona)
  // procedimiento para la lectura de un
  // registro campo a campo
  mientras no ultimo_dato(persona) hacer
    llamar_a escribir_f_reg (f, persona)
    // procedimiento auxiliar para la escritura
    // en el archivo del registro campo a campo
    llamar_a leer_reg (persona)
  fin_mientras
  cerrar(f)
fin
```

Actividad resuelta 9.2 página 376

```
// añadir nueva información al archivo PERSONAL anterior
algoritmo EJERCICIO_9_2
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ..... : .....
  fin_registro
  archivo_d de datos_personales: arch
var
  arch              : f
  datos_personales  : persona

inicio
  abrir (f, e, 'PERSONAL')
  llamar_a leer_reg (persona)
  mientras no ultimo_dato (persona) hacer
    llamar_a escribir_f_reg (f, persona)
    llamar_a leer_reg (persona)
  fin_mientras
  cerrar(f)
fin
```

Actividad resuelta 9.3 página 377

```
// mostrar por pantalla el contenido de todos los registros
// del archivo PERSONAL
algoritmo EJERCICIO_9_3
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ..... : .....
  fin_registro
  archivo_d de datos_personales: arch
var
  arch              : f
  datos_personales  : persona

inicio
  abrir (f, l, 'PERSONAL')
  mientras no fda (f) hacer
    llamar_a leer_reg (f, persona)
    llamar_a escribir_f_reg (persona)
  fin_mientras
  cerrar(f)
fin
```

```
// si existe un registro especial que marca el fin del archivo,
// la función FDA se activaría al leer este registro

inicio
  abrir (f, l, 'PERSONAL')
  llamar_a leer_reg (f, persona)
  mientras no fda (f) hacer
    llamar_a escribir_f_reg (persona)
    llamar_a leer_reg (f, persona)
  fin_mientras
  cerrar(f)
fin
```

Ejemplo 10.2.1 página 388

```
// algoritmo de clasificación de intercambio o de la burbuja

inicio
  desde I ← 1 hasta 7 hacer
    si elemento[I] > elemento[I + 1] entonces
      intercambiar (elemento[I], elemento[I + 1])
    fin_si
  fin_desde
fin
```

Ejemplo 10.2.1 página 388

```
// describir los pasos para clasificar en orden ascendente el vector
// 72 64 50 23 84 18 37 99 45 08

72 64 50 23 84 18 37 99 45 08
64 50 23 72 18 37 84 45 08 99
50 23 64 18 37 72 45 08 84 99
23 50 18 37 64 45 08 72 84 99
23 18 37 50 45 08 64 72 84 99
18 23 37 08 45 50 64 72 84 99
18 23 08 37 45 50 64 72 84 99
18 08 23 37 45 50 64 72 84 99
08 18 23 37 45 50 64 72 84 99
```

```
// método 1
algoritmo BURBUJA1
  // declaraciones
inicio
  desde I ← 1 hasta N hacer  // lectura del vector
    leer(X[I])
  fin_desde
  desde I ← 1 hasta N-1 hacer  // clasificación del vector
    desde J ← 1 hasta J ← N-1 hacer
      si X[J] > X[J+1] entonces  // intercambiar elementos
        AUX ← X[J]
        X[J] ← X[J+1]
        X[J+1] ← AUX
      fin_si
    fin_desde
  fin_desde
  desde J ← 1 hasta N hacer  // imprimir lista clasificada
    escribir(X[J])
  fin_desde
fin
```

```
// método 2
algoritmo BURBUJA2
  // declaraciones
inicio
  // ...
  desde I ← 1 hasta N-1 hacer
    desde J ← 1 hasta N-I hacer
      si X[J] > X[J+1] entonces
        AUX ← X[J]
        X[J] ← X[J+1]
        X[J+1] ← AUX
      fin_si
    fin_desde
  fin_desde
fin
```

```
// método 3
algoritmo BURBUJA3
  // declaraciones
inicio
  // lectura del vector
  BANDERA ← 'F'  // F, falso; V, verdadero
  I ← 1
  mientras (BANDERA == 'F') Y (I < N) hacer
    BANDERA ← 'V'
    desde K ← 1 hasta N-I hacer
      si X[K] > X[K+1] entonces
        intercambiar(X[K], X[K+1])
        BANDERA ← 'F'
      fin_si
    fin_desde
    I ← I+1
  fin_mientras
fin
```

Ejemplo 10.2.2 página 393

```
// ordenación por inserción
algoritmo CLAS_INSERCION_1
  // declaraciones
inicio
  // lectura del vector
  desde I ← 2 hasta N hacer
    AUX ← X[I]
    K ← I-1
    SW ← falso
    mientras no (SW) y (K >= 1) hacer
      si AUX < X[K] entonces
        X[K+1] ← X[K]
        K ← K-1
      si_no
        SW ← verdadero
      fin_si
    fin_mientras
    X[K+1] ← AUX
  fin_desde
fin
```

```
// ordenación por inserción mejorada
algoritmo CLAS_INSERCION_BINARIA
  // declaraciones
inicio
  // ...
  desde I ← 2 hasta N hacer
    AUX ← X[I]
    P ← 1  // primero
    U ← I-1  // último
    mientras P <= U hacer
      C ← (P+U) div 2
      si AUX < X[C] entonces
        U ← C-1
      si_no
        P ← C+1
      fin_si
    fin_mientras
    desde K ← I-1 hasta P decremento 1 hacer
      X[K+1] ← X[K]
    fin_desde
      X[P] ← AUX
  fin_desde
fin
```

Ejemplo 10.2.3 página 396

```
// ordenación por selección
algoritmo CLAS_SELECCION_NIVEL_1
var
  entero: I, J  // índices del vector V
  arreglo: X
  entero: AUX  // variable auxiliar para intercambio
  entero: N  // número de elementos del vector V
inicio
  // lectura del vector
  desde I ← 1 hasta N-1 hacer
    Buscar elemento menor de X[I], X[I+1], ..., X[N] e intercambiar
        con X[I]
  fin_desde
fin

algoritmo CLAS_SELECCION_NIVEL_2
inicio
  I ← 1
  repetir
    Buscar elemento menor de X[I], X[I+1], ..., X[N] e intercambiar
        con X[I]
    I ← I + 1
  hasta_que I = N
fin

algoritmo CLAS_SELECCION_NIVEL_3
inicio
  I ← 1
  repetir
    Buscar elemento menor de X[I], X[I+1], ..., X[N]
    // suponiendo que es X[K]
    Intercambiar X[K] y X[I]
    I ← I + 1
  hasta_que I = N
fin
```

```
// las instrucciones “buscar” e “intercambiar” se refinan
algoritmo CLAS_SELECCION_NIVEL_4a
inicio
  I ← 1
  repetir
    AUX ← X[I]  // AUX es el valor más pequeño
    K ← I       // K es la posición
    J ← I
    repetir
      J ← J + 1
      si X[J] < AUX entonces
        AUX ← X[J]  // actualizar AUX
        K ← J       // K, posición
      fin_si
    hasta_que J = N  // AUX = X[K] es ahora el más pequeño
    X[K] ← X[I]
    X[I] ← AUX
    I ← I + 1
  hasta_que I = N
fin


// el mismo algoritmo con la estructura “mientras”
algoritmo CLAS_SELECCION_NIVEL_4b
inicio
  I ← 1
  mientras I < N hacer
    AUX ← X[I]
    K ← I
    J ← I
    mientras J < N hacer
      J ← J + 1
      si X[J] < AUX entonces
        AUX ← X[J]
        K ← J
      fin_si
    fin_mientras
    X[K] ← X[I]
    X[I] ← AUX
    I ← I + 1
  fin_mientras
fin

// el algoritmo de ordenación con la estructura “desde”
algoritmo CLAS_SELECCION_NIVEL_4c
inicio
  I ← 1
  desde I ← N hasta N-1 hacer
    AUX ← X[I]
    K ← I
    desde J ← I+1 hasta N hacer
      si X[J] < AUX entonces
        AUX ← X[J]
        K ← J
      fin_si
    fin_desde
    X[K] ← X[I]
    X[I] ← AUX
  fin_desde
fin
```

Ejemplo 10.3 página 399

```
// ordenación por método de Shell
algoritmo SHELL
const
  n = 50
tipo
  array[1..n] de entero: lista
var
  lista  : L
  entero : k, i, j, salto

inicio
  llamar_a llenar(L)
  salto ← N DIV 2
  mientras salto > 0 hacer
    desde i ← (salto+1) hasta N hacer
      j ← i – salto
      mientras j > 0 hacer
        k ← j + salto
        si L[j] <= L[k] entonces
          j ← 0
        si_no
          llamar_a intercambio L[j], L[k]
        fin_si
        j ← j – salto
      fin_mientras
    fin_desde
    salto ← ent ((1 + salto) / 2)
  fin_mientras
fin
```

Ejemplo 10.4 página 402

```
// ordenación por método de quicksort
algoritmo QUICKSORT_1
inicio
  establecer x al valor de un elemento arbitrario de la lista
  mientras division no esté terminada hacer
    recorrer de izquierda a derecha para un valor >= x
    recorrer de derecha a izquierda para un valor <= x
    si los valores localizados no están ordenados entonces
      intercambiar los valores
    fin_si
  fin_mientras
fin
```

```
// ordenación por método de quicksort
algoritmo QUICKSORT_2
  llenar (A)
  i ← L
  j ← R
  x ← A ((L + R) div 2)
inicio
  mientras i <= j hacer
    mientras A[i] < x hacer
      i ← i + 1
    fin_mientras
    mientras A[i] > x hacer
      j ← j – 1
    fin_mientras
    si i <= j entonces
      llamar_a intercambiar(A[i], A[j])
      i ← i + 1
      j ← j – 1
    fin_si
  fin_mientras
fin
```

```
// ordenación por método de quicksort
algoritmo QUICKSORT_3
  // lista a evaluar de 10 elementos
  // IZQUIERDO, índice de búsqueda (recorrido) desde la izquierda
  // DERECHO, índice de búsqueda desde la derecha

inicio
  llenar (X)
  // inicializar índice para recorridos desde la izquierda y derecha
  IZQUIERDO ← ALTO // ALTO parámetro q indica principio de la sublis
  DERECHO ← BAJO   // BAJO parámetro q indica final de la sublista
  A ← X[1]
  // realizar los recorridos
  mientras IZQUIERDO <= DERECHO hacer
  // búsqueda o recorrido desde la izquierda
    mientras (X[IZQIUERDO] < A) Y (IZQUIERDO < BAJO) hacer
      IZQUIERDO ← IZQUIERDO + 1
    fin_mientras
    mientras (X[DERECHO] > A) Y (DERECHO > ALTO) hacer
      DERECHO ← DERECHO – 1
    fin_mientras
    // intercambiar elemento
    si IZQUIERDO <= DERECHO entonces
      AUX ← X[IZQUIERDO]
      X[IZQUIERDO] ← X[DERECHO]
      X[DERECHO] ← AUX
      IZQUIERDO ← IZQUIERDO + 1
      DERECHO ← DERECHO – 1
    fin_si
  fin_mientras
  // fin búsqueda; situar elemento seleccionado en su posición
  si IZQUIERDO < BAJO + 1 entonces
    AUX ← X[DERECHO]
    X[DERECHO] ← X[1]
    X[1] ← AUX
  si_no
    AUX ← X[BAJO]
    X[BAJO] ← X[1]
    X[1] ← AUX
  fin_si
fin
```

Ejemplo 10.5 página 405

```
// búsqueda secuencial de un elemento en un vector
algoritmo BUSQUEDA_SECUENCIAL_1
  // declaraciones
inicio
  llenar (A, n)
  leer (t)
  // recorrido del vector
  desde i ← 1 hasta n hacer
    si A[i] = t entonces
      escribir ('Elemento encontrado')
      escribir ('en posición', i)
    fin_si
  fin_desde
fin
```

```
// reescritura
algoritmo BUSQUEDA_SECUENCIAL_2
  // ...
inicio
  llenar (A, n)
  leer (t)
  i ← 1
  mientras (A[i] <> t) Y (i <= n) hacer
    i ← i + 1
    // este bucle se detiene bien con A[i] = t o bien con i > n
  fin_mientras
  si A[i] = t entonces  // condición de parada
    escribir (‘El elemento se ha encontrado en la posición’, i)
  si_no
    escribir (‘El número no se encuentra en el vector’)
  fin_si
fin
```

```
// reescritura
algoritmo BUSQUEDA_SECUENCIAL_3
  // ...
inicio
  llenar (A, n)
  leer (t)
  i ← 1
  mientras (A[i] <> t) Y (i < n) hacer
    i ← i + 1
    // este bucle se detiene bien con A[i] = t o bien con i >= n
  fin_mientras
  si A[i] = t entonces
    escribir (‘El elemento se ha encontrado en la posición’, i)
  si_no
    escribir (t, ‘no existe en el vector’)
  fin_si
fin
```

```
// reescritura
algoritmo BUSQUEDA_SECUENCIAL_4
  // ...
inicio
  llenar (A, n)
  leer (t)
  i ← 1
  mientras i <= n hacer
    si A[i] = t entonces
      escribir (‘El elemento se ha encontrado en la posición’, i)
      i ← n + 1
    si_no
      i ← i + 1
    fin_si
  fin_mientras
fin
```

```
// reescritura. Búsqueda secuencial con centinela
algoritmo BUSQUEDA_SECUENCIAL_5
  // ...
inicio
  llenar (A, n)
  leer (t)
  i ← 1
  A[n + 1] ← t
  mientras i <= n hacer
    i ← n + 1
  fin_mientras
  si i = n + 1 entonces
    escribir ('No se ha encontrado el elemento')
  si_no
    escribir ('Se ha encontrado el elemento')
  fin_si
fin
```

```
// reescritura. Búsqueda secuencial con interruptor/switch
algoritmo BUSQUEDA_SECUENCIAL_6
  // ...
inicio
  llenar (A, n)
  leer (t)
  i ← 1
  ENCONTRAO ← falso
  mientras (no ENCONTRADO) Y (i <= n) hacer
    si A[i] = t entonces
      ENCONTRAO ← verdadero
    fin_si
    i ← n + 1
  fin_mientras
  si ENCONTRADO entonces
    escribir ('El número ocupa el lugar', i - 1)
  si_no
    escribir ('El número no se encuentra en el vector')
  fin_si
fin
```

```
// reescritura. Búsqueda secuencial con interruptor/switch
algoritmo BUSQUEDA_SECUENCIAL_7
  // ...
inicio
  llenar (A, n)
  leer (t)
  i ← 1
  ENCONTRAO ← falso
  mientras i <= n hacer
    si A[i] = t entonces
      ENCONTRAO ← verdadero
      escribir ('El número ocupa el lugar', i)
    fin_si
    i ← n + 1
  fin_mientras
  si_no ENCONTRADO entonces
    escribir ('El número no se encuentra en el vector')
  fin_si
fin
```

```
// reescritura. Búsqueda secuencial con interruptor/switch
algoritmo BUSQUEDA_SECUENCIAL_8
  // ...
inicio
  llenar (A, n)
  leer (t)
  i ← 0
  ENCONTRAO ← falso
  repetir
    i ← n + 1
    si A[i] = t entonces
      ENCONTRAO ← verdadero
    fin_si
  hasta_que ENCONTRADO O (i = n)
  // ...
fin
```

```
// reescritura. Búsqueda secuencial con interruptor/switch
algoritmo BUSQUEDA_SECUENCIAL_9
  // ...
inicio
  llenar (A, n)
  leer (t)
  ENCONTRAO ← falso
  desde i ← 1 hasta i = n hacer
    si A[i] = t entonces
      ENCONTRAO ← verdadero
    fin_si
  fin_desde
  si ENCONTRADO entonces
    escribir ('Elemento encontrado')
  si_no
    escribir ('Elemento no encontrado')
  fin_si
fin
```

Ejemplo 10.6 página 410

```
// algoritmo de búsqueda binaria. Elemento buscado: K
algoritmo BUSQUEDA_BINARIA
  // ...
inicio
  llenar (X, N)
  ordenar (X, N)
  leer (K)
  BAJO ← 1
  ALTO ← N
  CENTRAL ← ent ((BAJO + ALTO) / 2)
  mientras (BAJO <= ALTO) Y (X[CENTRAL] <> K) hacer
    si K < X[CENTRAL] entonces
      ALTO ← CENTRAL – 1
    si_no
      BAJO ← CENTRAL + 1
    fin_si
    CENTRAL ← ent ((BAJO + ALTO) / 2)
  fin_mientras
  si K = X[CENTRAL] entonces
    escribir (‘Valor encontrado en’, CENTRAL)
  si_no
    escribir (‘Valor no encontrado’)
  fin_si
fin
```

Ejemplo 10.7 página 411

```
// algoritmo de búsqueda binaria de un NOMBRE introducido
algoritmo BUSQUEDA_NOMBRE
  // ...
  // NOMBRE      array de carácteres
  // N           número de nombres del array NOMBRE
  // ALTO        puntero al extremo superior del intervalo
  // BAJO        puntero al extremo inferior del intervalo
  // CENTRAL     puntero al punto central del intervalo
  // X           nombre introducido por el usuario
  // ENCONTRADO  bandera o centinela
inicio
  llenar (NOMBRE, N)
  leer (X)
  BAJO ← 1
  ALTO ← N
  ENCONTRADO ← falso
  mientras (no ENCONTRADO) Y (BAJO <= ALTO) hacer
    CENTRAL ← ent ((BAJO + ALTO) / 2)
    // verificar nombre central en este intervalo
    si NOMBRE[CENTRAL] = X entonces
      ENCONTRADO ← verdadero
    si_no
      si NOMBRE[CENTRAL] > X entonces
        ALTO ← CENTRAL – 1
      si_no
        BAJO ← CENTRAL + 1
      fin_si
    fin_si
  fin_mientras
  si ENCONTRADO entonces
    escribir (‘Nombre encontrado’)
  si_no
    escribir (‘Nombre no encontrado’)
  fin_si
fin
```

Ejemplo suelto página 419

```
// intercalación de elementos de dos vectores ‘A’ y ‘B’
// pasos generales
si elemento i de A es menor que elemento j de B entonces
  transferir elemento i de A a C
  avanzar i (incrementar en 1)
si_no
  transferir elemento j de B a C
  avanzar j
fin si
```

```
// refinamiento del anterior
// estado inicial de los algoritmos (índices¿?)
i ← 1
j ← 1
k ← 0
mientras (i <= M) Y (j <= N) hacer
  // seleccionar siguiente elemento de A ó B y añadir a C
  // incrementar k
  k ← k + 1
  si A[i] < B[j] entonces
    C[k] ← A[i]
    i ← i + 1
  si_no
    C[k] ← B[j]
    j ← j + 1
  fin_si
fin_mientras
```

```
// si los vectores tienen cantidades diferentes de elementos, el
// algoritmo de copia de los elementos restantes es
si i <= M entonces
  desde r ← i hasta M hacer
    k ← k + 1
    C[k] ← A[r]
  fin_desde
si_no
  desde r ← j hasta N hacer
    k ← k + 1
    C[k] ← B[r]
  fin_desde
fin_si
```

```
// algoritmo total resultante de la intercalación de dos vectores A
// y B ordenados en uno C
algoritmo INTERCALACION
  leer (A, B)  // A, B vectores de M y N elementos
  i ← 1
  j ← 1
  k ← 0

inicio
  mientras (i <= M) Y (j <= N) hacer
    // seleccionar siguiente elemento de A ó B y añadir a C
    k ← k + 1
    si A[i] < B[j] entonces
      C[k] ← A[i]
      i ← i + 1
    si_no
      C[k] ← B[j]
      j ← j + 1
    fin_si
  fin_mientras
  // copiar vector restante
  si i <= M entonces
    desde r ← i hasta M hacer
      k ← k + 1
      C[k] ← A[r]
    fin_desde
  si_no
    desde r ← j hasta N hacer
      k ← k + 1
      C[k] ← B[r]
    fin_desde
  fin_si
  escribir (C)  // vector clasificado
fin
```

Actividad 10.1 resuelta, página 421

```
algoritmo CLASIFICACION
  // declaraciones
inicio
  // lectura del vector
  BANDERA ← 'F'  // F, falso; V, verdadero
  I ← 1
  mientras (BANDERA == 'F') Y (I < N) hacer
    BANDERA ← 'V'
    desde K ← 1 hasta N-I hacer
      si X[K] > X[K+1] entonces
        intercambiar(X[K], X[K+1])
        BANDERA ← 'F'
      fin_si
    fin_desde
    I ← I+1
  fin_mientras
fin
```

Actividad 10.2 resuelta, página 422

```
// clasificar los números A y B
algoritmo CLASIFICAR_1
inicio
  leer (A, B)
  si A > B entonces
    intercambiar(A, B)
  fin_si
  escribir ('Más grande', A)
  escribir ('Más pequeño', B)
  fin_mientras
fin
```

```
algoritmo CLASIFICAR_2
inicio
  leer (A)
  MAX ← A
  leer (B)
  MIN ← B
  si B > A entonces
    MAX ← B
    MIN ← A
  fin_si
  escribir ('Máximo =', MAX)
  escribir ('Mínimo =', MIN)
  fin_mientras
fin
```

Actividad 10.3 resuelta, página 422

```
// busqueda lineal
algoritmo BUSQUEDA_1
  // N      número de elementos de la lista: entero
  // J      posición del elemento en la lista: entero
  // K      contador del bucle de búsqueda: entero
  // X      numero dado: entero
  // LISTA  conjunto de números enteros
var
  entero: I, K, X, N
  array[1..50] de entero: lista

inicio
  leer (N)
  desde I ← 1 hasta N hacer
    leer (LISTA[I])
  fin_desde
  repetir
    I ← I + 1
  hasta_que (LISTA[I] >= X) O (I = 50)
  si LISTA[I] = X entonces
    escribir ('El número dado no está en la lista')
    // insertar el elemento X en la lista
    si N < 50 entonces
      desde K ← N hasta I decremento 1 hacer
        LISTA[K + 1] ← LISTA[K]
      fin_desde
      LISTA[I] ← X
      N ← N + 1
      escribir ('Insertado en ', I)
    fin_si
  fin_si
  // escritura del vector LISTA
  desde I ← 1 hasta N hacer
    escribir (LISTA[I])
  fin_desde
fin
```

```
// busqueda dicotómica (binaria)
algoritmo BUSQUEDA_2
var
  entero: I, N, X, K, INF, SUP, CENTRAL, POSICION
  logico: SW
  array[1..50] de entero: LISTA

inicio
  leer (N)
  desde I ← 1 hasta N hacer
    leer (LISTA[I])  // la lista tiene que estar ordenada
  fin_desde
  ordenar (LISTA, N)
  leer (X)
  SW ← falso
  INF ← 1
  SUP ← N
  repetir
    CENTRAL ← (SUP – INF) DIV 2 + INF
    si LISTA[CENTRAL] = X entonces
      escribir (‘Número encontrado en la lista’)
      POSICION ← CENTRAL
      escribir (POSICION)
      SW ← verdad
    si_no
      si X < LISTA[CENTRAL] entonces
        SUP ← CENTRAL
      si_no
        INFERIOR ← CENTRAL + 1
      fin_si
      si (INF = SUP) y (LISTA[INF] = X) entonces
        escribir (‘El número está en la lista’)
        POSICION ← INF
        escribir (POSICION)
      fin_si
    fin_si
  hasta_que (INF = SUP) O SW
  si no(SW) entonces
    escribir (‘El número no existe en la lista’)
    si X < lista(INF) entonces
      POSICION ← INF
    si_no
      POSICION ← INF + 1
    fin_si
    escribir (POSICION)
    desde K ← N hasta POSICION decremento 1 hacer
      LISTA[K + 1] ← LISTA[K]
    fin_desde
    LISTA[POSICION] ← X
    N ← N + 1
  fin_si
  // escritura de la lista
  desde I ← 1 hasta N hacer
    escribir (LISTA[I])
  fin_desde
fin
```

Actividad 10.4 resuelta, página 425

```
// ordenar de mayor a menor un vector de N elementos (N <= 40), que
// son registros con los campos día, mes y año, de tipo entero
algoritmo ORD_FECHAS
tipo registro: fechas
  inicio
    entero: dia
    entero: mes
    entero: año
  fin_registro
  array[1..40] de fechas: arr
var
  arr: f
  entero: n

inicio
  pedirfechas (f, n)
  ordenarfechas (f, n)
  presentarfechas (f, n)
fin

logico funcion esmenor(E fechas: fecha1, fecha2)
inicio
  si (fecha1.año < fecha2.año) O
     (fecha1.año = fecha2.año) Y (fecha1.mes < fecha2.mes) O
     (fecha1.año = fecha2.año) Y (fecha1.mes < fecha2.mes) Y
     (fecha1.dia < fecha2.dia) entonces
       devolver (verdad)
  si_no
       devolver (falso)
  fin_si
fin_funcion

procedimiento pedirfechas(S arr:f; S entero:n)
  var
    entero: i, dia
  inicio
    i ← 1
    escribir (‘Deme la ’, i, ‘a fecha’)
    escribir (‘Día: ’)
    leer (dia)
    mientras (dia <> 0) Y (i <= 40) hacer
      f[i].dia ← dia
      escribir (‘Mes: ’)
      leer (f[i].mes)
      escribir (‘Año: ’)
      leer (f[i].año)
      n ← i
      i ← i + 1
      si i <= 40 entonces
        escribir (‘Deme la ’, i, ‘a fecha’)
        escribir (‘Día: ’)
        leer (dia)
      fin_si
    fin_mientras
fin_procedimiento
```

```
// continuacion
procedimiento ordenarfechas(E/S arr:f; E entero:n)
  var
    entero: salto, j
    logico: ordenada
    fechas: AUX
  inicio
    salto ← 1
    mientras salto > 1 hacer
      salto ← salto DIV 2
      repetir
        ordenada verdad
        desde j ← 1 hasta n – salto hacer
          si esmenor (f[j], f[j + salto]) entonces
            AUX ← f[j]
            f[j] ← f[j + salto]
            f[j + salto] ← AUX
            ordenada ← falso
          fin_si
        fin_desde
       hasta ordenada
    fin_mientras
fin_procedimiento

procedimiento presentarfechas(E/S arr:f; E entero:n)
  var
    entero: i
  inicio
    desde i ← 1 hasta n hacer
      escribir f[i].dia, f[i].mes, f[i].año)
    fin_mientras
fin_procedimiento
```

Actividad 10.5 resuelta, página 427

```
// dada la lista de fechas ordenada decreciente de antes:
// buscar/informar si una fecha está o no; y dónde correspondería
// insertarla, o la posición en dónde/a partir de dónde está
algoritmo BUSCAR_INSERTAR_FECHAS
tipo registro: fechas
  inicio
    entero: dia
    entero: mes
    entero: año
  fin_registro
  array[1..40] de fechas: vector

var
  arr: f
  entero: n, posic, cont
  fechas: fecha
  logico: esta

inicio
  pedirfechas (f, n)
  ordenarfechas (f, n)
  presentarfechas (f, n)
  escribir (‘Deme fecha a buscar (dd mm aa))
  leer (fecha.dia, fecha.mes, fecha.año)
  buscar (f, n, fecha, esta, posic, cont)
  si esta entonces
    si cont > 1 entonces
      escribir (‘Aparece a partir de la posición: ’, posic, ‘ ’,
        cont, ‘ veces’)
    si_no
      escribir (‘Está en la posición: ’, posic)
    fin_si
  si_no
    si n = 40 entonces
      escribir (‘No está. Array lleno’)
    si_no
      insertar (f, n, fecha, posic)
      presentarfechas (f, n)
    fin_si
  fin_si
fin
--------
logico funcion esmenor(E fechas: fecha1, fecha2)
  inicio
    ...................
  fin_funcion
--------
logico funcion esigual(E fechas: fecha1, fecha2)
  inicio
    si (fecha1.año = fecha2.año) Y
       (fecha1.mes = fecha2.mes) Y
       (fecha1.dia = fecha2.die) entonces
         devolver (verdad)
    si_no
         devolver (falso)
    fin_si
  fin_funcion
```

```
// continuación
procedimiento pedirfechas(S arr:f; S entero:n)
  var
    entero: i, dia
  inicio
    ......................
fin_procedimiento
--------
procedimiento ordenarfechas(E/S arr:f; E entero:n)
  var
    entero: salto, j
    logico: ordenada
    fechas: AUX
  inicio
    ........................
fin_procedimiento
```

```
// continuación
procedimiento buscar(E arr: f; E entero: n; E fechas: fecha;
                     S logico: esta; S entero: posic, cont)
  var
    entero: primero, ultimo, central, i
    logico: encontrado

  inicio
    primero ← 1
    ultimo ← n
    esta ← falso
    mientras (primero <= ultimo) Y (no esta) hacer
      central ← (primero + ultimo) DIV 2
      si esigual(f[central], fecha) entonces
        esta ← verdad
      si_no
        si esmenor(f[central], fecha) entonces
          ultimo ← central - 1
        si_no
          primero ← central + 1
        fin_si
      fin_si
    fin_mientras
      cont ← 0
    si esta entonces
      i ← central - 1
      encontrado ← verdad
      mientras (i >= 1) Y (encontrado) hacer
        si esigual(f[i], f[central]) entonces
          i ← i - 1
        si_no
          encontrado ← falso
        fin_si
      fin_mientras
      i ← i + 1
      encontrado ← verdad
      posic ← i
      mientras (i <= 40) Y encontrado hacer
        si esigual(f[i], f[central]) entonces
          cont ← cont + 1
          i ← i + 1
        si_no
          encontrado ← falso
        fin_si
      fin_mientras
    si_no
      posic ← primero
    fin_si
fin_procedimiento
```

```
// continuación
procedimiento insertar(E/S arr: f; E/S entero: n;
                       E fechas: fecha; E entero: posic)
  var
    entero: i

  inicio
    desde i ← n hasta posic decremento 1 hacer
      f[i + 1] ← f[i]
    fin_desde
      f[posic] ← fecha
      n ← n + 1
fin_procedimiento
--------
procedimiento presentarfechas(E arr: f; E entero: n)
  var
    entero: i
  inicio
    ..........................
fin_procedimiento
```

Actividad 10.6 resuelta, página 429

```
// procedimiento de búsqueda binaria de forma recursiva
algoritmo BUSQUEDA_BINARIA
  tipo
    array[1..10] de entero: arr
  var
    arr: a
    entero: num, posic, i

inicio
  desde i ← 1 hasta 10 hacer
    leer (a[i])
  fin_desde
  ordenar(a)
  escribir('Indique el número a buscar en el arreglo: ')
  leer(num)
  busqueda (a, posic, 1, 10, num)
  si posic > 0 entonces
    escribir ('Existe el número en la posición: ', posic)
  si_no
    escribir ('No existe el número en el arreglo')
  fin_si
fin
--------
procedimiento ordenar(E/S arr: a)
  ......................
  inicio
    ...................
fin_procedimiento
--------
procedimiento busqueda(E arr: a; S entero: posic;
                       E entero: primero, ultimo, num)
  // devuelve 0 si no existe el elemento en el arreglo, y
  // si existe devuelve su posición
  var
    entero: central
  inicio
    si primero > ultimo entonces
      posic ← 0
    si_no
      central ← (primero + ultimo) DIV 2
      si a[centra] = num entonces
        posic ← central
      si_no
        si num > a[central] entonces
          primero ← central + 1
        si_no
          ultimo ← central – 1
        fin_si
        busqueda (a, posic, primero, ultimo, num)  // escribir ??
      fin_si
    fin_si
  fin_procedimiento
```

Actividad 10.7 resuelta, página 430

```
// quicksort
algoritmo QUICKSORT
  tipo
    array[1..10] de entero: arr
  var
    arr: a
    entero: k

inicio
  desde k ← 1 hasta 10 hacer
    leer (a[k])
  fin_desde
  rapido (a, 10)
  desde k ← 1 hasta 10 hacer
    escribir (a[k])
  fin_desde
fin
--------
procedimiento intercambiar (E/S entero: m, n)
  var
    entero: AUX
  inicio
    AUX ← m
    m ← n
    n ← AUX
fin_procedimiento
```

```
// continuacion
procedimiento partir (E/S arr: a; E entero: primero, ultimo)
  var
    entero: i, j, central
  inicio
    i ← primero
    j ← ultimo
    // encontrar elemento pivote, central, y almacenar
    central ← a[(primero + ultimo) DIV 2]
    repetir
      mientras a[j] < central hacer
        i ← i + 1
      fin_mientras
      mientras a[j] > central hacer
        j ← j - 1
      fin_mientras
      si i <= j entonces
        intercambiar (a[i], a[j])
        i ← i + 1
        j ← j - 1
      fin_si
    hasta_que i > j

    si primero < j entonces
      partir (a, primero, j)
    fin_si
    si i > ultimo entonces
      partir (a, i, ultimo)
    fin_si
  fin_procedimiento
--------
procedimiento rapido (E/S arr: a; E entero: n)
  inicio
      partir (a, 1, n)
  fin_procedimiento
```

Ejemplo suelto página 437

```
inicio
  // fusión de dos archivos
  1. poner archivo 1 en cinta 1, archivo 2 en cinta 2
  2. seleccionar de los dos primeros registros de archivo 1 y
      archivo 2 el registro de clave más pequeña y almacenarlo
      en un nuevo archivo 3
  3. mientras (archivo 1 no vacío) Y (archivo 2 no vacío) hacer
  4. eleccionar el registro siguiente con clave más pequeña y
      almacenarlo en el archivo 3
      fin_mientras
      // uno de los archivos no está aún vacío
  5. almacenar resto archivo en archivo 3 registro a registro
fin
```

Ejemplo 11.1 página 437

```
// algoritmo de fusión de archivos
algoritmo FUSION_ARCHIVO
  var
    entero: ventana1, ventana2, ventanaS
    archivo_s de entero: F1, F2, FR
    // ventana2, ventanaS  claves de los archivos F1, F2
    // ventanaS            clave del archivo FR

inicio
  abrir (F1, l, 'nombre1')
  abrir (F2, l, 'nombre2')
  crear (FR, 'nombre3')
  abrir (FR, e, 'nombre3')
  leer (F1, ventana1)
  leer (F2, ventana2)
  mientras no FDA(F1) Y no FDA(F2) hacer
    si ventana1 <= ventana2 entonces
      ventanaS ← ventana1
      escribir (FR, ventanaS)
      leer (F1, ventana1)
    si_no
      ventanaS ← ventana2
      escribir (FR, ventanaS)
      leer (F2, ventana2)
    fin_si
  fin_mientras
  // lectura terminada de F1 o F2
  mientras no FDA(F1) hacer
    ventanaS ← ventana1
    escribir (FR, ventanaS)
    leer (F1, ventana1)
  fin_mientras
  mientras no FDA(F2) hacer
    ventanaS ← ventana2
    escribir (FR, ventanaS)
    leer (F2, ventana2)
  fin_mientras
  cerrar (F1, F2, FR)
fin
```

Ejemplo 11.4.2 página 440

```
algoritmo PARTICION_X_CONTENIDO
....
inicio
  abrir (f, l, ‘nombre’)
  crear (F1, ‘nombre1’)
  abrir (F1, e, ‘nombre1’)
  crear (F2, ‘nombre2’)
  abrir (F2, e, ‘nombre2’)
  leer (v)
  mientras no FDA(F) hacer
    leer_reg (f, r)
    si v = r.c entonces
      escribir_reg (f1, r)
    si_no
      escribir_reg (f2, r)
    fin_si
  fin_mientras
  cerrar (F1, F2, FR)
fin
```

Ejemplo 11.4.3 página 441

```
// selección por sustitución
algoritmo PARTICION_X_SUSTITUCION
const n = <valor>
tipo
  registro: datos_personales
    <tipo_dato>: c
    ...
  fin_registro

  registro: datos
    datos_personales: dp
    logico:  congela
  fin_registro

  array[1..n] de datos: arr
  archivo_s de datos_personales: arch

var
  datos_personales: r
  arr             : a
  arch            : f1, f2, f
  logico          : sw
  entero          : numcongelados, y, posicionmenor
```

```
// continuación
algoritmo PARTICION_X_SUSTITUCION
inicio
  abrir (f, l, ‘nombre’)
  crear (f1, e, ‘nombre1’)
  abrir (f1, ‘nombre1’)
  crear (f2, ‘nombre2’)
  abrir (f2, e, ‘nombre32’)
  numcongelados ← 0
  desde i ← 1 hasta n hacer
    si no fda(f) entonces
      leer_reg (f, r)
      a[i].dp ← r
      a[i].congela ← falso
    si_no
      a[i].congela ← verdadero
      numcongelados ← numcongelados + 1
    fin_si
  fin_desde
  sw ← verdad
  mientras no fda(f) hacer
    mientras (numcongelados < n) Y no fda(f) hacer
      buscar_no_congelado_menor(a, posicionmenor)
      si sw entonces
        escribir_reg (f1, a[posicionmenor].dp)
      si_no
        escribir_reg (f2, a[posicionmenor].dp)
      fin_si
      leer_reg (f, r)
      si r.c > a[posicionmenor].dp.c entonces
        a[posicionmenor].dp ← r
      si_no
        a[posicionmenor].dp ← r
        a[posicionmenor].congela ← verdad
        numcongelados ← numcongelados + 1
      fin_si
    fin_mientras
    sw ← no sw
    descongelar(a)
    numcongelados ← 0
  fin_mientras
  mientras numcongelados < n hacer
    buscar_no_congelado_menor (a, posicionmenor)
    si  entonces
      escribir_reg (f1, a[posicionmenor].dp)
    si_no
      escribir_reg (f2, a[posicionmenor].dp)
    fin_si
    a[posicionmenor].congela verdad
    numcongelados ← numcongelados + 1
  fin_mientras
  cerrar (f, f1, f2)
fin
```

Ejemplo 11.4.4 página 443

```
// partición por secuencias
algoritmo PARTICION_1
tipo
  registro: datos_personales
    <tipo_dato>: c
    ...
  fin_registro
  archivo_s de datos_personales: arch
var
  datos_personales: r
  arr             : a
  arch            : f, f1, f2
  logico          : sw
  entero          : i, n

inicio
  abrir (f, l, 'nombre')
  crear (f1, 'nombre1')
  abrir (f1, e, 'nombre1')
  crear (f2, 'nombre2')
  abrir (f2, e, 'nombre2')
  i ← 0
  leer (n)
  sw ← verdad
  mientras no fda(f) hacer
    leer_reg (f, r)
    si sw entonces
      escribir_reg (f1, r)
    si_no
      escribir_reg (f2, r)
    fin_si
      i ← i + 1
    si i = n entonces
      sw ← no sw
      i ← 0
    fin_si
  fin_mientras
  cerrar (f, f1, f2)
fin
```

```
// partición por secuencias
algoritmo PARTICION_2
tipo
  registro: datos_personales
    <tipo_dato>: c
    ...
  fin_registro
  archivo_s de datos_personales: arch
var
  datos_personales: r
  arch            : f, f1, f2
  logico          : sw

inicio
  abrir (f, l, 'nombre')
  crear (f1, 'nombre1')
  abrir (f1, e, 'nombre1')
  crear (f2, 'nombre2')
  abrir (f2, e, 'nombre2')
  sw ← verdad
  mientras no fda(f) hacer
    leer_reg (f, r)
    si sw entonces
      escribir_reg (f1, r)
    si_no
      escribir_reg (f2, r)
    fin_si
    sw ← no sw
  fin_mientras
  cerrar (f, f1, f2)
fin
```

Ejemplo 12.1 página 462

```
// leer el elemento j-ésimo de una lista P
1. conocer longitud de la lista L
2. si L = 0 visualizar “error lista vacía”
   si_no comprobar si el elemento j-ésimo está dentro del rango
   permitido de elementos 1<=j<=L; en este caso, asignar el valor del
   elemento P(j) a una variable B; si el elemento j-ésimo no está
   dentro del rango, visualizar un mensaje de error “elemento
   solicitado no existe en la lista”
3. fin
```

```
// pseudocódigo
procedimiento acceso (E lista: P; S elementolista: B; entero: L, J)
inicio
  si L = 0 entonces
    escribir (‘Lista vacía’)
  si_no
    si (j >= 1) Y (j <= L) entonces
      B ← P(j)
    si_no
      escribir (‘ERROR: elemento no existente’)
    fin_si
  fin_si
fin
```

Ejemplo 12.1 página 462

```
// borrar un elemento j de la lista P
inicio
  si L = 0 entonces
    escribir (‘Lista vacía’)
  si_no
    leer (j)
    si (j >= 1) Y (j <= L) entonces
      desde i ← j hasta L – 1 hacer
        P[i] ← P[i + 1]
      fin_desde
      L ← L + 1
    si_no
      escribir (‘ERROR: elemento no existente’)
    fin_si
  fin_si
fin
```

Ejemplo 12.4.1 página 468

```
// creación de una lista enlazada
tipo
  puntero_a nodo: punt
  registro: tipo_elemento
       ...:...
  fin_registro
  registro: nodo
    tipo_elemento: elemento
    punt         : sig
  fin_registro

var
  punt         : inic, posic, anterior
  tipo_elemento: elemento
  logico       : encontrado

inicio
  inicializar (inic)
  ...
fin

procedimiento inicializar (S punt: inic)
  inicio
    inic ← nulo
  fin_procedimiento
```

Ejemplo 12.4.1.b página 468

```
// inserción de un elemento. Pasos:
// 1º situación de partida
// 2º reservar(aux)
// 3º introducir la nueva información en aux→.elemento
// 4º hacer que aux→sig apunte a donde lo hace anterior→.sig
// 5º conseguir que anterior→.sig apunte a donde lo hace aux

procedimiento insertar (E/S punt: inic, anterior;
                        E tipo_elemento: elemento)
  var punt: aux
  inicio
    reservar(aux)
    aux→.elemento ← elemento
    si anterior = nulo entonces
      aux→.sig ← inic
      inic ← aux
    si_no
      aux→.sig ← anterior→.sig
      anterior→.sig ← aux
    fin_si
      anterior ← aux  // opcional
  fin procedimiento
```

Ejemplo 12.4.1.c página 469

```
// eliminación de un elemento de una lista enlazada. Primero se
// comprueba que no esté vacía
logico funcion vacia (E punt: inic)
inicio
  devolver (inic = nulo)
fin_funcion

// pasos siguientes:
// 1º situación de partida
// 2º anterior→.sig apunta a donde posic→.sig
// 3º liberar(posic)

procedimiento suprimir (E/S punt: inic, anterior, posic)
  inicio
    si anterior = nulo entonces
      inic ← posic→.sig
    si_no
      anterior→.sig ← posic→.sig
      anterior→.sig ← aux
    fin_si
      liberar (posic)
      anterior ← nulo  // opcional
      posic ← inic     // opcional
  fin_procedimiento
```

Ejemplo 12.4.1.d página 471

```
// recorrido de una lista enlazada
// se utiliza una variable de tipo puntero auxiliar
procedimiento recorrer (E punt: inic)
  var punt: posic
  inicio
    posic ← inic
    mientras posic <> nulo hacer
      proc_escribir (posic→.elemento)
      posic ← posic→.sig
    fin_mientras
  fin_procedimiento
```

Ejemplo 12.4 página 471

```
// cálculo del número de elementos de una lista enlazada
procedimiento contar (E punt: primero; S entero: n)
  var punt: p
  inicio
    n ← 0        // contador de elementos
    p ← primero
    mientras p <> nulo hacer
      n ← n + 1
      p ← p→.sig
    fin_mientras
  fin_procedimiento
```

```
// acceso a un elemento de una lista enlazada
procedimiento consultar (E punt: inic; S punt: posic, anterior;
                         E tipo_elemento: elemento;
                         S logico: encontrado)
  inicio
    encontrado ← falso
    anterior ← nulo
    posic ← inicial
    mientras no igual(posic→.elemento, elemento) Y
             (posic <> nulo) hacer
    // igual es una función qeu compara los elementos que se le
    // pasan como parámetros. Se recurre a ella porque si se
    // tratara de registros, se compararía únicamente la
    // información alamacenada en un determinado campo
      si  // no se entiende
    fin_mientras
      si igual(posic→.elemento, elemento) entonces
        encontrado ← verdad
      si_no
        encontrado ← falso
      fin_si
  fin_procedimiento
```

Ejemplo 12.5 página 472

```
// encontrar el nodo de una lista que contiene la información de
// valor t, suponiendo que la lista almacena datos de tipo entero
procedimiento encontrar (E punt: primero; E entero: t)
  var punt  : p
      entero: n
  inicio
    n ← 0
    p ← primero
    mientras (p→.info <> t) Y (p <> nulo) hacer
      n ← n + 1
      p ← p→.sig
    fin_mientras
    si p→.info = t entonces
      escribir ('Se encuentra en el nodo ', n, ' de la lista')
    si_no
      escribir ('No encontrado')
    fin_si
  fin_procedimiento
```

```
// mejoramiento del anterior
procedimiento encontrar (E punt: primero; E entero: t)
  var punt  : p
      entero: n
  inicio
    n ← 0
    p ← primero
    mientras (p→.info < t) Y (p <> nulo) hacer
      n ← n + 1
      p ← p→.sig
    fin_mientras
    si p→.info = t entonces
      escribir ('Se encuentra en el nodo ', n, ' de la lista')
    si_no
      escribir ('No encontrado')
    fin_si
  fin_procedimiento
```

Ejemplo 12.4.2.a página 474

```
// creación de una lista enlazada con arrays
const
  max = <expresión>
tipo
  registro: tipo_elemento
       ...:...
  fin_registro
  registro: tipo_nodo
    tipo_elemento: elemento
    ent          : sig  // actúa como puntero, almacenando la
      // posición donde se encuentra el siguiente elemento
      // de una lista
  fin_registro
  array[1..max] de tipo_nodo: arr

var
  entero: inic, posic, anterior, vacio
  arr: m  // representa la memoria de la computadora
  tipo_elemento: elemento
  logico: encontrado

inicio
  iniciar (m, vacio)
  inicializar (inic)
  ...
fin

procedimiento inicializar (S entero: inic)  // lista de elementos
  inicio
    inic ← 0
  fin_procedimiento

procedimiento iniciar (S arr: m; S entero: vacio) // lista de vacíos
  var
    entero: i
  inicio
    vacio ← 1
    desde i ← 1 hasta max – 1 hacer
      m[i].sig ← i + 1
    fin_desde
    m[max].sig ← 0
    // como ya no hay más posiciones libres a las que apuntar,
    // recibe un 0
  fin_procedimiento
```

Ejemplo 12.4.2.b página 475

```
// inserción de un elemento
// procedimiento que proporciona a través de aux la primera
// posición vacía para almacenar en ella el nuevo elemento
procedimiento reservar (S entero: aux; E arr: m; E/S entero: vacio)
inicio
  si vacio = 0 entonces
    // memoria agotada
    aux ← 0
  si_no
    aux ← vacio
    vacio ← m[vacio].sig
  fin_si
fin_procedimiento

// colocar un nuevo elemento a continuación del anterior
procedimiento insertar (E/S entero: inic, anterior, vacio;
                        E tipo_elemento: elementos;
                        E/S arr: m)
var
  entero: aux
inicio
  reservar(aux, m, vacio)
  si aux = 0 entonces OVERFLOW
  m[aux].elemento ← elemento
  si anterior = 0 entonces
    m[aux].sig ← inic
    inic ← aux
  si_no
    m[aux].sig ← m[anterior].sig
    m[anterior].sig ← aux
  fin_si
  anterior ← aux  // opcional
  // prepara anterior para que, si no especificamos otra cosa, la
  // siguiente inserción se realic a continuación de la actual
fin_procedimiento
```

Ejemplo 12.4.2.c página 477

```
// eliminación de un elemento
// insertamos el elemento eliminado en la lista de vacíos
procedimiento liberar (E entero: posic; E/S arr: m;
                       E/S entero: vacio)
inicio
  m[posic].sig ← vacio
  vacio ← posic
fin_procedimiento

// eliminamos un elemento de la lista
procedimiento insertar (E/S entero: inic, anterior, posic, vacio;
                        E/S arr: m)
inicio
  si anterior = 0 entonces
    inic ← m[aux].sig
  si_no
    m[anterior].sig ← m[posic].sig
  fin_si
  liberar (posic, m, vacio)
  anterior ← 0  // opcional
  posic ← inic  // opcional
  // las dos últimas instrucciones preparan los punteros para que,
  // si no especificamos otra cosa, la siguiente eliminación se
  // realice por el principio de la lista
fin_procedimiento
```

Ejemplo 12.4.2.d página 479

```
// recorrido de una lista
// al recorrer la lista va mosstrando por pantalla los diferentes
// elementos que la componen
procedimiento recorrer (E/S entero: inic)
var
  entero: posic
inicio
  posic ← inicio
  mientras posic != 0 hacer
    // se recurre a un procedimiento, proc_escribir(...), para
    // presentar por pantalla los campos del registro pasado
    // como parámetro
    proc_escribir(m[posic].elemento)
    posic ← m[posic].sig
  fin_mientras
fin_procedimiento
```

Ejemplo 12.4.2.e página 479

```
// búsqueda de un determinado elemento en una lista
procedimiento consultar (E entero: inic; S entero: posic, anterior;
                         E tipo_elemento: elemento;
                         S logico: encontrado; E arr: m)
  inicio
    anterior ← 0
    posic ← inicial
    // las funciones menor(...) e igual(...) comparan los
    // registros por un determinado campo
    mientras menor(m[posic].elemento, elemento) Y (posic <> 0) hacer
      anterior ← posic
      posic ← m[posic].sig
    fin_mientras
    si (posic = 0) entonces
      encontrado ← falso
    si igual(m[posic].elemento, elemento)
      encontrado ← verdad
  fin_procedimiento
```

Ejemplo 12.7 página 484

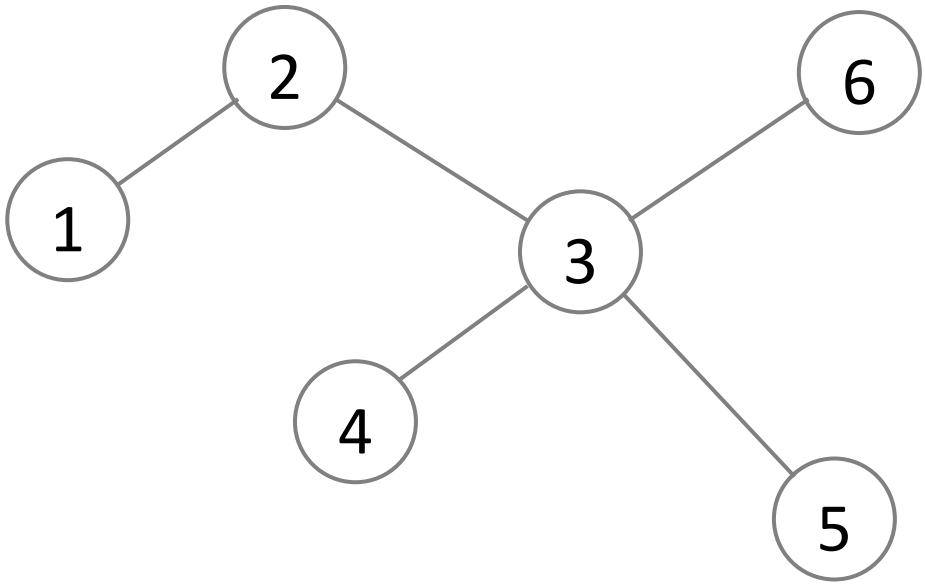
```
// funciones de implementación de pilas con punteros
algoritmo pilas_con_punteros
  tipo
    puntero_a nodo: punt
    registro: tipo_elemento
      ......:......
    fin_registro
    registro: nodo
      tipo_elemento: elemento
      punt: cima
    fin_registro
  var
    punt: cima
    elemento: tipo_elemento

  inicio
    inicializar(cima)
  fin

  procedimiento inicializar(S punt: cima)
    inicio
      cima ← nulo
    fin_procedimiento

  logico funcion vacia(E punt: cima)
    inicio
      devolver(cima = nulo)
    fin_funcion

  procedimiento consultarCima(E punt: cima;
                              S tipo_elemento: elemento)
    inicio
      si no vacia(cima) entonces
        elemento ← cima→.elemento
      fin_si
    fin_procedimiento

  // 1º cima apunta al último elemento de la pila
  // 2º reservar(aux)
  // 3º se introduce la información en aux→.elemento
  // 4º se hace que aux→.cima apunte a donde cima
  // 5º se cambia cima para que apunte a donde aux
  // 6º la pila tiene un elemento más
  procedimiento meter (E/S punt: cima; E tipo_elemento: elemento)
    var
      punt: aux
    inicio
      reservar(aux)
      aux→.elemento ← elemento
      aux→.cima ← cima
      cima ← aux
    fin_procedimiento
```

```
// continuación
// 1° cima apunta al último elemento de la pila
// 2° se hace que aux apunte a donde apuntaba cima
// 3° y que cima pase a apuntar a donde cima→.cima
// 4° liberar(aux)
// 5° la pila tiene un elemento menos
procedimiento sacar (E/S punt: cima; E tipo_elemento: elemento)
  var
    punt: aux
  inicio
    si no vacia(cima) entonces
      aux ← cima
      elemento ← cima→.elemento
      cima ← cima→.cima
      liberar (aux)
      // liberar es un procedimiento para la eliminación de
      // variables dinámicas
    fin_si
  fin_procedimiento
```

Ejemplo 12.6 página 489

```
// leer un texto y separar los carácteres letras, dígitos y
// restantes carácteres para se utilizado posteriormente
algoritmo LECTURA_CARACTER
  const max = <valor>
tipo
  array [1...max] de caracter: pila
var
  entero: cima1, cima2, cima3
  pila: pilaletras, piladigitos, pilaotroscar

inicio
  crear (cima1)
  crear (cima2)
  crear (cima3)
  elemento ← leercar
  mientras (cicodigo (elemento) <> 26) Y no llena(cima1)
            Y no llena(cima2) Y no llena(cima3) hacer
    // se sale del bucle en cuanto se llena alguna de las
    // pilas o se pulse ^Z
    si (elemento >= 'A') Y (elemento <= 'Z') O
       (elemento >= 'a') Y (elemento <= 'z') entones
      meter (cima1, pilaletras, elemento)
    si_no
      si (elemento >= '0') Y (elemento <= '9') entonces
        meter (cima2, piladigitos, elemento)
      si_no
        meter (cima3, pilaotroscar, elemento)
      fin_si
    fin_si
    elemento ← leercar
  fin_mientras
fin_procedimiento

procedimiento crear(S entero: cima)
  inicio
    cima ← 0
  fin_procedimiento

logico funcion llena(E entero: cima)
  inicio
    devolver(cima = max)
  fin_funcion

procedimiento meter(E/S entero: cima; E/S tipo_elemento: elemento)
  inicio
    cima ← cima + 1
    p[cima] ← elemento
  fin_procedimiento
```

Ejemplo 12.5.2.1 página 539



Ejemplo 12.5.2.1.b página 539