Lenguaje de Diseño

Resolución de Problemas y Algoritmos

Ingeniería en Computación
Ingeniería en Informática

Universidad Nacional de San Luis
Departamento de Informática
Año 2017

Índice general

1. Lenguaje de Diseño de Algoritmos **2**
- 1.1. Introducción 2
- 1.2. Facilidades Provistas por un Lenguaje de Diseño 2
- 1.3. Formalización de Algoritmos 3
  - 1.3.1. Formalización del Ambiente de un Problema 3
  - 1.3.2. Prueba de Algoritmos 6
  - 1.3.3. Precisiones acerca de los Objetos 7
- 1.4. Expresiones 8
  - 1.4.1. Expresiones Aritméticas 8
  - 1.4.2. Expresión Relacional 13
  - 1.4.3. Expresiones Lógicas 16
  - 1.4.4. Asignación Carácter 17
- 1.5. Acciones Primitivas de Entrada - Salida de Datos 17
  - 1.5.1. Entrada de Datos 17
  - 1.5.2. Salida de Datos 17
- 1.6. Estructuras de Control 19
  - 1.6.1. Introducción 19
  - 1.6.2. La Estructura de Control Secuencial 19
  - 1.6.3. La Estructura de Control Condicional 19
  - 1.6.4. Estructura de Control de Repetición 26

2. Estructuración de Datos **31**
- 2.1. Introducción 31
- 2.2. Arreglo Lineal 31
  - 2.2.1. Operaciones con arreglos: Asignación y recuperación de valores 36

3. Subalgoritmos **40**
- 3.1. Introducción 40
- 3.2. Definición de Subalgoritmos 41
- 3.3. Invocación y Retorno de Subalgoritmos 42
- 3.4. Anexo 57

Capítulo 1

Lenguaje de Diseño de Algoritmos

1.1. Introducción

Un ***programa*** es un modelo de resolución de un problema escrito en un lenguaje de programación. De la definición anterior se desprende que escribir un programa implica:

1. Obtener una solución de un problema.
2. Expresar esta solución en un lenguaje de programación.

En general se puede decir que existe una distancia o ***diferencia*** entre lo que se podría denominar el ***lenguaje del problema*** y el ***lenguaje de programación***, en el sentido que el primero resulta menos rígido y con más posibilidades de expresión que el segundo.

El objetivo fundamental de un lenguaje de diseño es ser ***comprensible para las personas*** que van a interpretar los algoritmos escritos en él, mientras que el fin último de un lenguaje de programación es ser ***comprensible por la computadora*** que va a ejecutar el programa.

La finalidad de un lenguaje de diseño es brindar una herramienta que sirva de apoyo para el desarrollo de algoritmos. La idea es no sumar, a la complejidad del problema, las limitaciones impuestas por una notación estricta. Además, en muchas aplicaciones, es importante conseguir un algoritmo independiente del lenguaje de programación o lenguaje de implementación.

En general, cada programador, de acuerdo con su experiencia y habilidad, encontrará más expresiva una notación u otra. Imponer una notación específica, si bien, en parte implica contradecir los objetivos iniciales que justificaron el uso de los lenguajes de diseño, con la finalidad de comunicarnos, durante lo que resta del desarrollo del curso, necesitamos establecer algunas pautas para el lenguaje de diseño de algoritmos que usaremos.

1.2. Facilidades Provistas por un Lenguaje de Diseño

Los ***objetivos básicos*** de un lenguaje de diseño son:

1. Servir de apoyo durante el proceso de resolución de un problema.

2. Servir como etapa previa al proceso de codificación. La tarea de **traducción** del lenguaje de diseño a cualquier lenguaje de programación no debería ser muy complicada.

3. En los proyectos de desarrollo de software, en los que intervienen varias personas, el lenguaje de diseño debería permitir que cada una de ellas pueda tener una visión global del trabajo de los demás, difícil de conseguir analizando directamente los programas del resto del grupo.
4. Como los lenguajes de programación proveen diferentes conjuntos de primitivas y la traducción al lenguaje de programación es posterior al diseño, podemos elegir el lenguaje de programación apropiado según el conjunto de primitivas requerido.

Nuestra intención será, entonces, **proponer un lenguaje de diseño de algoritmos** que sirva de apoyo para la resolución de problemas y pueda ser traducido, en forma sistemática, a un programa.

1.3. Formalización de Algoritmos

Previo a la definición de nuestro lenguaje de diseño, necesitaremos precisar algunos conceptos. De ahora en adelante, el (**procesador**), como lo hemos definido en nuestro contexto, es equivalente a una **computadora**. La construcción del algoritmo es la etapa más dificultosa y, en éste y los próximos capítulos, daremos las herramientas básicas necesarias.

Las computadoras, como ya fue indicado, no pueden ejecutar directamente los algoritmos en forma literal como los venimos tratando. Es necesario codificarlos en un lenguaje de programación. En la mayor de los casos, la codificación no presenta grandes dificultades ya que, los lenguajes de programación tienden, cada vez más, a la formalización que se propone, cambiando esencialmente su sintaxis.

1.3.1. Formalización del Ambiente de un Problema

Lo primero a considerar en el proceso de resolución de problemas es la formalización de su ***ambiente***. Vamos a definir un conjunto de reglas que nos permitirán describir, con precisión y sin ambigüedad, los ***objetos del universo de un problema***.

Una característica que diferencia entre sí a los objetos, es que ***cada uno*** tiene un ***nombre*** que lo identifica unívocamente, o sea, si queremos citar diferentes objetos, damos una lista de sus nombres o identificadores.

Además, cada objeto tiene un ***uso*** específico que no se puede intercambiar. En el ejemplo de la calculadora, dado anteriormente, las teclas <dig> y <C>, tienen usos completamente diferentes; sin embargo las teclas <1>, <2>,...., <0> si bien su uso es diferente (no es lo mismo oprimir la tecla <1> que la tecla <2>), tienen una utilidad similar (sirven para oprimir dígitos).

Podemos decir que cada objeto tiene un ***tipo*** particular que indica características comunes a todos los estados posibles del objeto.

Los objetos más simples con los cuales nosotros trabajaremos durante el curso son los objetos numéricos: ***enteros*** y ***reales***; los ***lógicos*** y los ***caracteres***.

Otra característica importante de los objetos es su ***valor***.

En cada instante, todo objeto del ambiente tiene un valor, para algunos objetos, este valor puede cambiar luego de la ejecución de una acción.

Continuando con el ejemplo de la calculadora, en la versión 1 del algoritmo, en la acción "oprimir número", el objeto "número" toma el valor 124 la primera vez que la acción es ejecutada, el valor 59 la segunda vez y, la última vez, el valor 3. También existen objetos cuyos valores nunca cambian, por ejemplo, "oprimir <9>" el objeto 9 nunca cambia su valor.

Para resumir, podemos imaginarnos a los objetos de un ambiente como celdas rotuladas (por el nombre), donde además las celdas tienen un tamaño determinado (según el tipo) y contienen una información (un valor posible del conjunto de valores de un tipo dado)

128	a

NUMERO LETRA

Veamos un ejemplo para mostrar las características o atributos de los objetos del ambiente de un problema.

Enunciado: Se tiene un objeto de nombre NUMERO, de tipo numérico, tal que su valor puede ser un número entero positivo. Se quiere encontrar un algoritmo que determine el producto de los **n** primeros números enteros positivos (es decir, el factorial de $n = n!$).

Las ***acciones primitivas*** que puede ejecutar el procesador son:

1. Dar un valor a un objeto.
2. Calcular la suma de dos números.
3. Calcular el producto de dos números.

El procesador además interpreta la ***condición:*** "un número es menor o igual que otro" y un esquema repetitivo condicionado del tipo **MIENTRAS** <***condición***> **HACER** <***acciones_primitivas***> **FINMIENTRAS** la cual significa que el procesador ejecutará las acciones primitivas siempre que la condición se cumpla.

Una vez planteado el problema y conociendo las acciones primitivas que puede reconocer el procesador ***deberemos describir el ambiente*** sobre el cual trabajará.

El ***ambiente*** consiste del objeto ya descripto NUMERO. El valor inicial de este objeto ha sido determinado (en algún momento) y va a servir para la realización del cálculo (establece hasta que número deberé realizar los productos):

Si el valor inicial es 4, se calculará: 1 x 2 x 3 x 4 = 24 = 4!
Si el valor inicial es 5, se calculará: 1 x 2 x 3 x 4 x 5 = 120 = 5!
y así siguiendo.

Dentro del ambiente, será necesario guardar el valor final del resultado del cálculo, para ello se debe crear otro objeto al cual llamaremos FACTORIAL y será también de tipo entero.

En el producto intervienen números enteros positivos $(1, 2, \ldots, n)$ que deben estar representados en el ambiente. Para ello se crea un objeto al que llamamos I, de tipo entero. Su valor irá cambiando, de modo tal que, en él, queden representados los números enteros positivos entre 1 y el valor para el cual se desea calcular el factorial (valor de NUMERO).

Hemos dicho anteriormente que los objetos en todo instante tienen un valor. Es importante señalar, en este punto, en relación con el **cambio** del valor de un objeto, que en tanto no se guarde el valor precedente (para los objetos cuyos valores pueden cambiar) éste es imposible de recuperar. Se dice entonces que (**cada nuevo valor destruye al anterior**).

Los objetos FACTORIAL e I, al ser creados tienen un valor desconocido llamado ***indeterminado.***

Objeto	Descripción	Estado Inicial	Estado Final
NUMERO	Objeto de tipo entero positivo que representa al número del que se desea hallar el factorial	n	n
FACTORIAL	Objeto de tipo entero positivo en el cual se calcula el producto de los n primeros números enteros	Valor indeterminado	$n!$
I	Objeto de tipo entero que toma todos los valores enteros positivos desde 1 a n	Valor indeterminado	$n+1$

Una vez descripto el ambiente del problema, estamos en condiciones de elaborar un algoritmo, es decir, cuáles son las acciones que deberá ejecutar el procesador para transformar el ambiente, desde el estado inicial al estado final deseado. Si hacemos que el valor de I, tome sucesivamente los valores enteros positivos entre 1 y n, el primer valor que debe tomar I es 1.

Si a FACTORIAL no se le asigna un valor inicial, entonces su valor es indeterminado. Por lo tanto, si en el estado inicial, FACTORIAL posee un valor arbitrario f, en el estado final, su valor será $f \times n \times (n-1) \times \ldots \times 1$, lo cual es incorrecto porque no resuelve el problema, salvo que f sea 1 (elemento neutro de la multiplicación). Entonces, antes de comenzar a calcular el producto, se debe dar a FACTORIAL el valor inicial 1. Para calcular el valor de I se deben ejecutar las siguientes acciones:

1. Actualizar el valor de FACTORIAL multiplicando su valor corriente por el valor actual de I.

2. Construir en I el entero positivo siguiente para repetir la acción anterior hasta que se hayan efectuado todos los productos necesarios.

Ahora mostraremos las distintas etapas en que se pueden descomponer estas acciones no primitivas, aplicando la técnica de **refinamientos sucesivos**.

Versión 1:

T: Calcular el factorial de un número n dado.

Versión 2:

t_0. Declarar los nombres de los objetos a ser utilizados por el algoritmo (que constituyen su ambiente) y el tipo que se asocia con cada nombre.

t_1. Dar valores iniciales a los objetos NUMERO, FACTORIAL e I.

t_2. Actualizar el valor de FACTORIAL, multiplicando su valor actual por el valor actual de I.

t_3. Construir en I el entero positivo siguiente y repetir t_2 hasta que se hayan efectuado todos los productos requeridos.

Versión 3:

Ya podemos obtener un ***bosquejo*** de la versión final del algoritmo expresado en las acciones primitivas que anteriormente habíamos definido para el procesador:

ALGORITMO "Factorial de n"
$t_{0,1}$ FACTORIAL, NUMERO, I: entero
$t_{1,1}$ dar a NUMERO el valor n (que se desee)
$t_{1,2}$ dar a FACTORIAL el valor 1
$t_{1,3}$ dar a I el valor 1
MIENTRAS el valor de I sea menor o igual que el valor de NUMERO
HACER
$t_{2,1}$ multiplicar el valor de I por el valor de FACTORIAL
$t_{2,2}$ dar este nuevo resultado al objeto FACTORIAL
$t_{3,1}$ sumar 1 al valor de I
$t_{3,2}$ dar este nuevo resultado al objeto I
FINMIENTRAS

La declaración indicada en $t_{0,1}$, tiene por finalidad indicarle al procesador cuáles son los objetos del ambiente del algoritmo con los cuales debe trabajar.

Al ejecutarse las acciones $t_{1,1}$, $t_{1,2}$, $t_{1,3}$, $t_{2,2}$ y $t_{3,2}$ puede observarse una modificación del ambiente, por ejemplo, en el caso de $t_{1,1}$ el objeto FACTORIAL pasa de tener un valor indeterminado, antes de la ejecución de dicha acción, a tomar el valor 1 después de la ejecución de la acción.

1.3.2. Prueba de Algoritmos

Ahora deberíamos efectuar una verificación o prueba del algoritmo para determinar si hace lo que nosotros pensamos que hace. Supongamos que n = NUMERO = 3.

Acción ejecutada	Estado del ambiente después de ejecutada la acción		
	NUMERO	I	FACTORIAL
Inicialmente	indeterminado	indeterminado	indeterminado
t1.1	3	indeterminado	indeterminado
t1.2	3	indeterminado	1
t1.3	3	1	1
(1era repetición) 1 <= 3 verd.			
t2	3	1	1
t3	3	2	1
(2da repetición) 2 <= 3 verd.			
t2	3	2	2
t3	3	3	2
(3era repetición) 3 <= 3 verd.			
t2	3	3	6
t3	3	4	**6**
(4ta repetición) 4 <= 3 falso			
	fin de repetición y fin del algoritmo		

A partir de la elección de un valor, se puede probar el mal funcionamiento de un algoritmo: si el resultado obtenido, luego de una prueba, paso a paso, es incorrecto, se puede asegurar que el algoritmo no funciona. Sin embargo, la prueba que hemos realizado para el valor de NUMERO = 3, no asegura que nuestro algoritmo funciona correctamente para todo valor entero positivo puesto que no hemos demostrado, formalmente, que el mismo sea correcto independientemente de cual sea el valor inicial de NUMERO; sólo podemos afirmar que funciona correctamente para NUMERO = 3.

1.3.3. Precisiones acerca de los Objetos

En la sección anterior, en las acciones: "sumar 1 al valor de I y dar este nuevo resultado al objeto I" intervienen dos objetos, tales son I y 1. Durante la ejecución del algoritmo, el valor de I cambia, inicialmente está indeterminado, luego su valor es 1, después 2, etc.. En cambio el valor 1 permanece inalterable. En función de lo antedicho se puede clasificar a los objetos de la siguiente manera: ***variables*** y ***constantes***, según dicha clasificación, el objeto I, es una variable y el objeto 1 es una constante.

> Una ***variable*** es un objeto del ambiente cuyo valor puede cambiar y que posee además los siguientes atributos:
> un ***nombre*** que la identifica,
> un ***tipo*** que describe los valores que puede tomar la variable y las operaciones que con dicha variable pueden realizarse.

Cuando se define una variable, se debe precisar su nombre y su tipo. Definir una variable es crear un objeto para el procesador. En el momento de la creación de una variable, ésta tiene un valor desconocido.

> Una ***constante*** es un objeto cuyo valor no puede cambiar.

En la definición de variable hicimos referencia al concepto de ***tipo***. En realidad en la bibliografía puede encontrarse como ***tipo de datos. Dato*** es la expresión general que describe los objetos con los cuales opera un procesador. Existen diferentes tipos de datos, nosotros nos ocuparemos en este capítulo de los llamados ***tipos primitivos*** y, dentro de ellos, de los más simples: los numéricos (enteros y reales) los lógicos y los caracteres.

> El ***tipo entero***, consiste de un conjunto finito de valores enteros. La cardinalidad de este conjunto depende de las características del procesador.

> El ***tipo real***, consiste de un conjunto finito de valores reales. La cardinalidad de este conjunto también estará definida por las características del procesador.

Los números reales siempre tienen un punto decimal; las fracciones se guardan como números decimales.

> El ***tipo lógico***, también llamado ***tipo booleano***, es el conjunto de los valores de verdad: **VERDADERO y FALSO.**

El ***tipo carácter*** es el conjunto finito y ordenado de caracteres que el procesador puede reconocer.

En general, todos los procesadores reconocen el conjunto de caracteres que contiene, entre otros:

- las letras mayúsculas del abecedario .
- las letras minúsculas del abecedario.
- los dígitos decimales del 0. . . 9.
- el carácter de espacio blanco, caracteres especiales tales como: $*$, $+$, $-$, _, /, (,), ,, \$, $\wedge$, %, $<$, $>$, ″, .

Una constante de tipo carácter se escribe encerrada entre COMILLAS SIMPLES, por ejemplo 'A', 'a'.

1.4. Expresiones

Un procesador debe ser capaz de manipular los objetos del ambiente de un algoritmo. Es decir, debe ser capaz de calcular expresiones como: $2 + 3$, $a > b$, etc. Luego:

Una ***expresión*** describe un cálculo a efectuar cuyo resultado es un valor único.

Una ***expresión*** consta de ***operadores*** y ***operandos***. Según el tipo de los objetos que manipula, se clasifican en expresiones:

- aritméticas,
- relacionales,
- lógicas.

  El resultado de una expresión aritmética es de tipo numérico, el de una expresión relacional y el de una expresión lógica es de tipo lógico.

1.4.1. Expresiones Aritméticas

Un operando de una expresión aritmética puede ser, por ahora, una constante de tipo numérico, una variable de tipo numérico u otra expresión aritmética, encerrada entre paréntesis. Los operadores aritméticos que soporta **nuestro lenguaje de diseño** son:

Operador	**Significado**
+	suma
-	resta
$*$	producto
/	división
↑	potencia
//	resto de la /

Como regla general se considera que si dos operandos tienen el mismo tipo, el resultado también es del mismo tipo. Por ejemplo, la suma de dos números enteros da como resultado otro valor entero. A continuación se dan las reglas que nos permitirán determinar cómo se evaluará una expresión de dos o más operandos:

1. Todas las operaciones que están encerradas entre paréntesis se evalúan primero, cuando existen paréntesis anidados las expresiones más internas se evalúan primero.

2. Las operaciones aritméticas, dentro de una expresión, se ejecutan con el siguiente orden o precedencia:

Orden de precedencia	**Operadores**	**Significado**
1º	$\uparrow$	potenciación (se aplica de derecha a izquierda)
2º	$*, /, //$	multiplicación, división y resto (se aplican de izquierda a derecha)
3º	$+, -$	suma y resta (se aplican de izquierda a derecha)

La tabla anterior indica que en una expresión primero se evalúa la potenciación, luego el producto y/o la división, que tienen el mismo nivel de prioridad y, finalmente, la suma y/o resta. Tanto en el caso del producto y/o división como en el de la suma y/o resta cuando en la columna significado se dice "se aplican de izquierda a derecha" implica que si en una expresión aritmética hay seguidas tres operaciones, por ejemplo de producto, se comienza a calcular desde el que se encuentra más a la izquierda.

Ejemplo 1:

8 + 7 * 3 + 4 * 5 * 4
21 20
29 80
109

Figura 1.1:

Ejemplo 2:

8 + (7 + 3)) * 6 / 3
10 2
18
36

Figura 1.2:

En el último ejemplo ¿pensó Ud. que el resultado correcto era 1?, tal pensamiento es incorrecto porque los operandos de esta expresión son todos enteros, luego todos los resultados, incluyendo los intermedios, deben ser enteros. Siguiendo el orden de evaluación, dado en el cuadro anterior para operadores de igual prioridad (en este caso $*$ y /), de izquierda a derecha antes de multiplicar se debe dividir, entonces la primera operación es 1/10 cuyo resultado real es 0,1; pero al estar trabajando con tipo entero el resultado de esta operación es 0.

Ejemplo 3:

1 / 10 * 10
0
0

Figura 1.3:

Ejemplo:
Para la operación de potenciación se tiene:

$$3 \uparrow 4 = 81 \text{ (entero, entero} \longrightarrow \text{entero)}$$
$$3.0 \uparrow 4 = 81.0 \text{ (real, entero} \longrightarrow \text{real)}$$
$$2 \uparrow 3.5 = 11.31 \text{ (entero, real} \longrightarrow \text{real)}$$

donde la notación (tipo1, tipo2 $\longrightarrow$ tipo3) se interpreta que tipo1 es el tipo del primer operando, tipo2 es el tipo del segundo operando y tipo3 es el tipo del resultado.

Funciones primitivas o predefinidas

Además de las operaciones básicas como suma, resta, multiplicación, división y potencia, en general, existe otro conjunto de operadores especiales llamados ***funciones primitivas*** (en semejanza con las acciones primitivas) que el procesador puede ejecutar. Por ahora, nuestro **lenguaje de diseño** permite utilizar las siguientes ***funciones primitivas aritméticas*** que el procesador puede interpretar:

Nombre función	Tipo arg1	Tipo arg2	Tipo resultado	Significado
ABS	entero	-	entero	valor absoluto
ABS	real	-	real	valor absoluto
RC	entero	-	entero	raíz cuadrada
RC	real	-	real	raíz cuadrada

Al escribir el nombre de la función, a continuación, entre paréntesis, se escribe el objeto al cual se le aplicará la función, llamado ***argumento***, el procesador devolverá el resultado; por ejemplo, **RC**(9) = 3 y **RC**(0,25) = 0,5. Como las funciones primitivas devuelven valores, éstas pueden ser usadas como parte de una expresión aritmética.
Ejemplo:

10 * 5 + RC(64)
50 8
58

Figura 1.4:

Asignación Aritmética

En un ambiente dado, para dar valor a una variable, se utiliza el ***operador de asignación***, cuyo símbolo, en **nuestro lenguaje de diseño**, será $\longleftarrow$, tal operación se escribe como:

$$V \longleftarrow E$$

en la que:

1. V es el nombre de la variable a la cual el procesador va a asignarle (darle) el valor de E.
2. $\longleftarrow$, identifica al operador de asignación.
3. E representa el valor a asignar y puede ser una constante, otra variable, o el resultado de la evaluación de una expresión.

Según sea el tipo de V y E, la operación de asignación aritmética se clasifica como entera o real. En particular, diremos que $V \longleftarrow E$ es una asignación ***aritmética entera*** si:

1. V es una variable de tipo entero.
2. E es una constante entera, una variable entera, una expresión entera o una función primitiva que retorna un valor entero.

o $V \longleftarrow E$ es una ***asignación aritmética real*** si:

1. V es una variable de tipo real.
2. E es una constante real, una variable real, una expresión real o una función primitiva que retorna un valor real.

Ejemplos:

La acción $I \leftarrow$ **1** ***significa*** dar a la variable de nombre I, el valor 1.

Es ***importante remarcar*** que el número 1 ***reemplaza*** al valor que tuviere I antes de que se ejecute la acción de asignación. ***Siempre, cuando se asigna un nuevo valor a una variable, el valor anterior se pierde.***

La acción $A \leftarrow B$ ***significa*** dar a la variable A el valor de la variable B

Supongamos, por ejemplo, que el estado del ambiente **antes** de la ejecución de la acción de asignación era: A contenía el valor 6 y la variable B el valor 7. Gráficamente:

6	7
A	B

Luego de la ejecución de la acción primitiva de asignación $A \longleftarrow B$, el resultado es:

7	7
A	B

Note que mientras A pierde su viejo valor, B lo mantiene.

La acción $A \leftarrow E$, **con E siendo una expresión, *significa*** dar a la variable A el resultado de la evaluación de la expresión E.

Ejemplo:

Sea la siguiente acción de asignación

SUM ⟵ 3,5 + 4,0∗ (-7.2)

Primero se evalúa la expresión, con las reglas dadas para la evaluación de expresiones, luego el resultado - 25.3 se asigna a la variable SUM.

La acción $I \leftarrow I+x$ ***significa*** incrementar en x el valor de I y guardarlo en la variable I.

donde x puede ser una constante, una variable, el resultado de la evaluación de una expresión o el resultado de la evaluación de una función primitiva.

Ejemplo:

Sea la siguiente acción de asignación aritmética:

$$I \longleftarrow I + 1.0$$

Si antes de ejecutar la acción de asignación el estado del ambiente con respecto a I era que I contenía el valor 5.6, luego de la ejecución de la acción de asignación el estado del ambiente con respecto a I es que esta variable contendrá el valor 6.6. Esquemáticamente:

5.6

I

Luego:

6.6

I

Ejemplo:

$$A \longleftarrow 3 * 4 - B$$

Si el contenido de B fuere 14, luego el valor de la expresión es -2 y éste es el valor que contendrá la variable A. Es importante notar que cuando, dentro de una expresión aparece el nombre de una variable, para poder evaluar la expresión se consulta el valor de la variable, en el ejemplo la variable es B y antes de poder evaluar la expresión $3 * 4 - B$ se debe tomar el valor de B que es 14.

Por lo tanto, en una **variable** se puede, por un lado **almacenar** un valor como sucede en la variable A y, por otro, **consultar** el valor que ya tiene, como sucede en el ejemplo con la variable B.

Ejemplo:

Sean las variables A que contiene el valor 3 y B que contiene el valor -3, luego de las siguientes asignaciones:

$$C \longleftarrow \mathbf{ABS}(A)$$
$$C \longleftarrow \mathbf{ABS}(B)$$

¿Cuál será el valor de C en cada caso?

Para concluir esta sección mostraremos el algoritmo "Factorial de n", construido anteriormente, reescrito aplicando los conceptos de expresión y asignación.

ALGORITMO "Factorial de n"
t_0 FACTORIAL, I, NUMERO: entero
$t_{1,1}$ NUMERO $\longleftarrow$ 4
$t_{1,2}$ FACTORIAL $\longleftarrow$ 1
$t_{1,3}$ $I \longleftarrow 1$
MIENTRAS el valor de I sea menor o igual que el valor de NUMERO
HACER
$t_{2,1}$ FACTORIAL $\longleftarrow$ FACTORIAL $* I$
$t_{3,1}$ $I \longleftarrow I + 1$
FINMIENTRAS

1.4.2. Expresión Relacional

Concepto de Predicado

Dados los ***valores lógicos verdadero y falso***:

$$L = \{\textbf{VERDADERO, FALSO}\}$$

y un conjunto E cualquiera:

Un ***predicado*** es una aplicación de E en el conjunto L

Un predicado puede ser verdadero para ciertos elementos de E y falso para otros. Por ejemplo, sea E = *el conjunto de los números naturales* y sobre este conjunto definimos el siguiente predicado P: "x es par", el predicado P toma el valor verdadero para ciertos valores de x dentro del conjunto E, es decir, para el subconjunto $\{2, 4, 6, \ldots\}$ mientras que toma el valor falso para otros elementos de E como para los del subconjunto $\{1, 3, 5, \ldots\}$.

Otro ejemplo; si X e Y son dos variables de tipo entero, podemos definir el predicado binario P = *"X es mayor que Y"* denotado como "$X > Y$" y el conjunto E como el conjunto de pares de números enteros (X, Y).

Al diseñar un algoritmo, muchas veces la ejecución de un conjunto de acciones depende de la evaluación de un ***predicado o condición.*** Por ejemplo, en el algoritmo "Factorial de n", la condición de la repetición "el valor de NUMERO es menor o igual que el valor de I" es un predicado que se aplica sobre el conjunto de pares (NUMERO, I) en L.

Para expresar un predicado o condición a menudo se escribe una comparación entre dos valores del mismo tipo, por ejemplo ambos valores de tipo entero, real o carácter. Una comparación tal es llamada ***predicado elemental o expresión relacional.***

Operadores Relacionales

Para las ***comparaciones*** de valores ***numéricos*** en general o ***carácter***, los operadores relacionales con los que trabaja **nuestro lenguaje de diseño** son:

Operador Relacional	Significado
=	igual
<	menor
<=	menor o igual
>	mayor
>=	mayor o igual
<>	distinto

Para la ***comparación de valores lógicos*** se usan, sólo, los operadores $=$ y $<>$.

Si A es una variable entera cuyo valor es 3 y tenemos la constante entera 15, la tabla siguiente ejemplifica el cálculo de algunos posibles predicados:

Predicado	**Valor**
$A >= 15$	FALSO
$A = 15$	FALSO
$A < 15$	VERDADERO

Los predicados elementales o expresiones relacionales pueden combinarse, mediante los conectores u operadores lógicos para formar predicados compuestos: $\wedge$ (conjunción), $\vee$ (disjunción) y $\neg$ (la negación).

Para formalizar estos conceptos, sea una condición $\boldsymbol{p}$ y otra $\boldsymbol{q}$ entonces se pueden definir las tablas de verdad para cada operador lógico como sigue:

Conjunción (y lógico)

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

Como puede desprenderse de la tabla anterior, **la conjunción sólo devuelve un valor de verdad verdadero (V) cuando ambas proposiciones (p y q) son verdaderas, en cualquier otro caso devuelve falso (F)**.

Disjunción (o lógico inclusivo)

p	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

La disjunción devuelve un valor de verdad Verdadero cuando al menos una de las dos proposiciones o ambas son verdaderas.

Los resultados anteriores pueden generalizarse a condiciones compuestas que contengan más de dos condiciones simples.

Negación lógica

La negación se aplica a un predicado (que puede ser simple o compuesto) y su tabla de verdad es la siguiente:

p	$\neg\, p$
V	F
F	V

O sea, que simplemente **cambia el valor de verdad del predicado.**

Ejemplo:

Sean X**,** K y Z tres variables enteras cuyos valores son 1, 3 y 2, respectivamente, y sean los predicados elementales $p =$ **"**$X = 1$**", ** $q =$ **"**$K < 2$**" y r = "**$Z = 5$**".** Veamos, entonces la evaluación de la siguiente condición o predicado compuesto (de ahora en más nos referiremos, cuando hablemos de predicados, a las condiciones).

Sea el siguiente predicado compuesto:

$$(X = 1 \vee K < 2) \wedge Z = 5$$

Entonces para conocer su valor de verdad se puede construir la tabla de verdad correspondiente:

p	q	r	$p \vee q$	$(p \vee q) \wedge r$
V	V	V	V	V
V	V	F	V	F
V	F	V	V	V
F	V	V	V	V
F	F	V	F	F
F	V	F	V	F
V	**F**	**F**	**V**	**F**
F	F	F	F	F

Para el ejemplo que estamos tratando el resultado final es **falso** tal cual se señala, remarcado en negrita, en la tabla.

Visto de otra forma más gráfica:

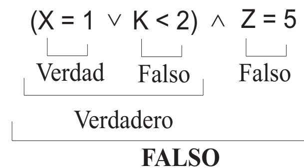


Figura 1.5:

Es importante notar que una condición que, matemáticamente, se escribe como:

$$A < B < C$$

el procesador no la reconocerá, sino que deberá reescribirse como:

$$(A < B) \wedge (B < C)$$

El orden de precedencia de los operadores es:

1º ¬ (no lógico)
2º ∧ (conjunción)
3º ∨ (disjunción)

Al igual que sucedía con las expresiones aritméticas, si dos operadores de igual prioridad aparecen seguidos la regla de precedencia indica que se evalúan de izquierda a derecha, por ejemplo:

$p \wedge q \wedge r$ se evaluará como $((p \wedge q) \wedge r)$

Ahora podemos dar nuestra versión final del algoritmo "Factorial de n":

ALGORITMO "Factorial de n"
  $t_{0,1}$ FACTORIAL, I, NUMERO: entero
  $t_{1,1}$ NUMERO $\longleftarrow$ 6
  $t_{1,2}$ FACTORIAL $\longleftarrow$ 1
  $t_{1,3}$ $I \longleftarrow 1$
  MIENTRAS I <= NUMERO **HACER**
    $t_{2,1}$ FACTORIAL $\longleftarrow$ FACTORIAL * I
    $t_{2,2}$ $I \longleftarrow I + 1$
  FINMIENTRAS

1.4.3. Expresiones Lógicas

Un operando de una expresión lógica puede ser una variable de tipo lógica o una expresión lógica. Los operadores lógicos que soporta **nuestro lenguaje de diseño** son los ya definidos en el punto anterior, es decir: $\wedge$, $\vee$, $\neg$.

Ejemplo:

Si A = **VERDADERO**, B = **FALSO** entonces la expresión $A \wedge B \vee (\neg B)$ da como resultado:

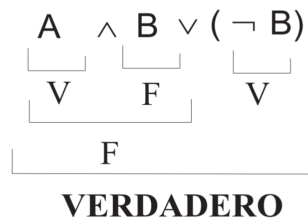


Figura 1.6:

Asignación Lógica

Diremos que:

$$V \longleftarrow E$$

es una ***asignación lógica*** si:

1. V, es una variable lógica.
2. E, es una constante lógica (**VERDADERO, FALSO**), una variable lógica, una expresión relacional o una expresión lógica.

Ejemplo:

Supongamos que H, T y Q son variables de tipo lógico, luego de ejecutar las siguientes acciones:

$$H \longleftarrow 2 < 5$$

$$T \longleftarrow H \vee (8 >= 9)$$

$$Q \longleftarrow \textbf{FALSO}$$

Los valores de H, T y Q son **VERDADERO, VERDADERO y FALSO** respectivamente.

1.4.4. Asignación Carácter

$$V \longleftarrow E$$

es una asignación de carácter si:

1. V es una variable de carácter.
2. E es una constante de carácter.

1.5. Acciones Primitivas de Entrada - Salida de Datos

1.5.1. Entrada de Datos

Un valor que no pertenece al ambiente puede introducirse al mismo, mediante una acción, que llamaremos ***lectura***.

> ***Lectura***, es toda acción que permite la entrada de uno o más valores del ambiente a través de un dispositivo. Una lectura es una asignación, en el sentido que toma valores del medio externo y lo asigna a las variables del ambiente. La lectura es una acción primitiva.

En **nuestro lenguaje de diseño** la lectura se denota de la siguiente manera:

LEER V

donde V es una variable del ambiente. O bien:

LEER $A, \ldots, X$

La acción **LEER** <lista de variables> asigna un nuevo valor a cada una de las variables que aparece en la lista

1.5.2. Salida de Datos

Un valor del ambiente puede comunicarse al mundo exterior, por ejemplo a través de la impresión sobre un papel.

> Llamaremos ***escritura*** a la acción primitiva que permite la salida de valores del ambiente a través de un dispositivo. Esta acción toma uno o más valores del ambiente y lo comunica al medio externo conservando dichas variables sus valores.

En **nuestro lenguaje de diseño** la notaremos de la siguiente manera:

ESCRIBIR V

donde V es la variable cuyo valor se desea comunicar al medio externo. O bien:

ESCRIBIR $A, \ldots, X$

Esta acción comunica los valores de las variables referenciadas uno al lado del otro. La acción **ESCRIBIR** $A, \ldots, X$ se puede expresar también de la siguiente manera:

ESCRIBIR A

......

......

ESCRIBIR X

En este caso los valores de las variables referencias se comunican en líneas individuales (una debajo de la otra).

Es importante destacar que, en el ambiente, cuando resulte conveniente, pueden nombrarse también los Datos Auxiliares, que aunque no son los requeridos por el problema, sirven como información de la/s etapa/s intermedia/s entre los Datos de Entrada y los Resultados o Datos de Salida. De la misma manera se puede pretender comunicar información no necesariamente almacenada en una variable, por ejemplo texto, o combinaciones de variables y texto.

ESCRIBIR "El resultado es"

O también:

ESCRIBIR "El resultado es", X

Con estas nuevas acciones podemos enriquecer el algoritmo "Factorial de n" de la siguiente manera:

ALGORITMO "Factorial de n"
COMENZAR
    $t_{0,1}$ FACTORIAL, I, NUMERO: entero
    $t_{1,1}$ **LEER** NUMERO
    $t_{1,2}$ FACTORIAL $\longleftarrow$ 1
    $t_{1,3}$ $I \longleftarrow 1$
    MIENTRAS I <= NUMERO **HACER**
        $t_{2,1}$ FACTORIAL $\longleftarrow$ FACTORIAL * I
        $t_{2,2}$ $I \longleftarrow I + 1$
    FINMIENTRAS
    ESCRIBIR FACTORIAL
FIN

Dato de Entrada: NUMERO

Dato de Salida: FACTORIAL

Dato Auxiliar: I

Notemos que la secuencia de acciones aparece ahora, delimitada por las palabras **COMENZAR y FIN**.

Esta forma de presentación de un algoritmo, es la que utilizaremos de ahora en adelante.

1.6. Estructuras de Control

1.6.1. Introducción

Estructurar el control de un conjunto de acciones, detalladas en un algoritmo, es brindar mecanismos que permitan indicar el **orden** en que las mismas van a ser llevadas a cabo.

Una de las mayores potencias de un procesador proviene de su capacidad de tomar decisiones y de determinar qué acción realizar al momento de ejecutar un algoritmo, sobre la base de los valores de algunos de los datos que se leen o bien, de los resultados de los cálculos que se realizan.

Vamos a explicar ahora tres conceptos muy importantes, tales son las *estructuras de control*:

Secuencial

Condicional

Repetición

que en general no modifican el ambiente sino el orden en que el procesador ejecuta las acciones primitivas.

1.6.2. La Estructura de Control Secuencial

Cuando no se indique lo contrario el flujo de control de ejecución de un algoritmo seguirá la **secuencia** implícita del mismo. Entendemos por secuencia implícita, que las acciones se ejecutan en el orden en que son escritas en el algoritmo, es decir desde la primera hacia la última ("desde arriba hacia abajo"). Al terminar de ejecutarse una acción se pasa a la inmediata siguiente que está perfectamente determinada y, así siguiendo, hasta alcanzar la última acción del algoritmo. Por ejemplo, en el caso del algoritmo factorial, las siguientes acciones primitivas representan una secuencia:

LEER NUMERO
FACTORIAL $\leftarrow$ 1
$I \leftarrow 1$

Lo mismo podría visualizarse gráficamente a través de lo que se conoce como un **diagrama de flujo** (que sirve para visualizar todos los posibles órdenes de ejecución de un algoritmo), en él las ***acciones primitivas*** que modifican el ambiente se escriben dentro de ***rectángulos***, entonces el ejemplo anterior quedaría expresado:

1.6.3. La Estructura de Control Condicional

Un algoritmo permite resolver problemas de una misma clase. Sin embargo, en la mayoría de las clases de problemas no es posible hallar **un conjunto de acciones secuenciales simples** que permitan resolver el mismo. Algunas acciones pueden tener que llevarse a cabo sólo para ciertas instancias del problema, o lo que es lo mismo, sólo si ciertas condiciones se satisfacen.

Cuanto más general intente ser nuestro algoritmo, es posible que deba considerar más alternativas. **Nuestro lenguaje de diseño** debe proveer algún mecanismo para expresar que un conjunto de acciones debe ejecutarse sólo bajo ciertas condiciones. El **condicional** permitirá que una o un conjunto de acciones primitivas se ejecuten sólo si cierta condición se cumple. Por ejemplo:

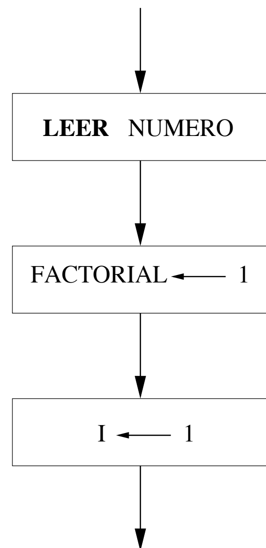


Figura 1.7:

Si $n > 0$ *entonces decrementar* n *en 1*

quiere decir que sólo restaremos 1 a n si n es un número positivo. Las condiciones prevén las distintas situaciones que pueden presentarse en la resolución del problema. El condicional permite que la ejecución de cierto conjunto de acciones quede sometida a una condición.

Supongamos tener dos variables enteras, X e Y, que tienen valores diferentes x e y; buscamos escribir el mayor valor. Esto es, si $X > Y$ debe escribirse el valor de X, en caso contrario el valor de Y. En este caso, es evidente que existen ***secuencias de acciones alternativas***, tales son "escribir el valor de X" o "escribir el valor de Y". La elección acerca de cuál de los dos valores escribir depende de la condición: el valor de X es o no mayor que el valor de Y.

El diagrama de flujo asociado con la acción condicional quedaría expresado:

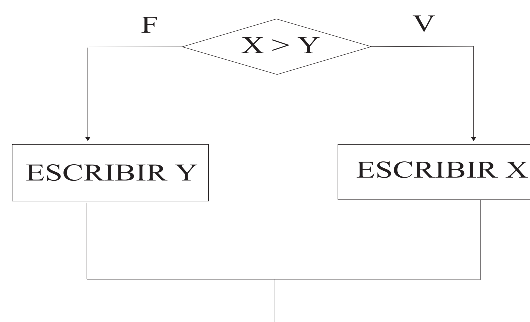


Figura 1.8:

donde $X > Y$ es un predicado que describe una condición que se debe evaluar. El resultado de esta acción completa es que el mayor de los dos valores, x o y, es el que se escribirá.

Ejemplo:

Supongamos que X tiene el valor 8 e Y el valor 3. Como 8 es mayor que 3, la condición $X > Y$ es Verdadera y, por lo tanto se escribirá el valor 8 como resultado de la acción primitiva **ESCRIBIR** X.

En este ejemplo sencillo hemos analizado la ***estructura de control condicional.*** El formato algorítmico de dicha estructura es:

SI <condición>
  ENTONCES
    <alternativa verdadera>
  SINO
    <alternativa falsa>
FINSI

El inicio de la estructura comienza con la palabra **SI** y el final con la palabra **FINSI.**

Se llaman ***delimitadores*** a las palabras **SI, ENTONCES, SINO y FINSI.**

Ahora podemos reescribir el ejemplo anterior de la siguiente manera

SI $X > Y$
  ENTONCES
    ESCRIBIR X
  SINO
    ESCRIBIR Y
FINSI

Tanto la condición como cada una de las alternativas de la estructura, pueden ser más complejas, por ejemplo, la alternativa verdadera y la alternativa falsa pueden consistir de una secuencia de acciones primitivas en lugar de una única acción (como en el ejemplo dado). La condición, a su vez, puede consistir de un predicado compuesto.

La estructura condicional en sí misma, se considera como un ente completo (una única acción), es decir al que se ingresa en el punto en el cual la condición se evalúa. Una vez evaluada la condición se toma por el camino indicado por ***una*** de las alternativas y, luego, el control pasa a la acción primitiva que sigue al delimitador **FINSI** (si dicha acción existe).

Es necesario asegurar que las condiciones sean mutuamente excluyentes, esto es, no puede ocurrir que dos o más condiciones sean satisfechas simultáneamente, pues caeríamos en un problema de ambigüedad: ¿Cuál de los bloques de acciones es ejecutado?

Otro ejemplo:

Enunciado: Calcular la raíz cuadrada de un número, si éste no es negativo; en caso contrario no calcular nada.

Ambiente del Algoritmo:

VARIABLE	DESCRIPCIÓN
A	Variable de entrada−salida de tipo entero. Como variable de entrada contendrá el número del cual se quiere calcular la raíz cuadrada; como variable de salida, contendrá la raíz cuadrada

Algoritmo:

Versión 1:
Declarar las variables a ser utilizadas por el algoritmo.
Leer el número del cual se desea calcular la raíz cuadrada, guardándolo en la variable A.
Si su valor es positivo o cero, entonces calcular la raíz cuadrada, usando la función primitiva correspondiente, devolver el valor en la variable A y escribir el valor de la raíz; sino no hacer nada.

Versión 2:
Declarar variables.
Leer el número del cual se desea calcular la raíz cuadrada, guardándolo en la variable A.
Si su valor es positivo o cero entonces
Calcular la raíz cuadrada usando la función primitiva correspondiente, guardando el valor de la raíz en la variable A
Escribir el valor de la raíz
Sino no hacer nada.

Versión 3: (final)

ALGORITMO "raíz cuadrada"
  COMENZAR
  A: entero
  {Entrada de datos}
  LEER A
  {Se determina si el valor de A es no negativo}
  SI $A >= 0$
    ENTONCES
      {Cálculo de la raíz cuadrada}
      A ← **RC**(A)
      {Salida del resultado}
      ESCRIBIR A
  FINSI
FIN

En el algoritmo anterior ***las frases u oraciones escritas entre llaves*** representan ***comentarios*** y sirven para documentar un algoritmo. Además el algoritmo anterior es, a la vez, un ejemplo donde un dato de entrada puede también ser utilizado como dato de salida. Puede suceder que, si la condición es falsa, no existan acciones a ejecutar (como en el ejemplo anterior). En estos casos, en la construcción algorítmica no aparece el delimitador **SINO** y el formato de la construcción condicional es:

SI <condición>
  ENTONCES
  <alternativa verdadera>
FINSI

Anidamiento de Estructuras de Decisión

Tanto la alternativa verdadera como la falsa, pueden contener a su vez, estructuras de decisión. Analicemos el siguiente esquema, donde: p y q, son predicados y a, b, c, y d, son acciones.

SI p
  ENTONCES
    a
  SINO
    b
    SI q
      ENTONCES
        c
      SINO
        d
    FINSI
FINSI

Veamos el resultado de todas las posibles ejecuciones de este algoritmo:

PREDICADOS		**ACCIONES A EJECUTAR**
p	q	
VERDADERO	**VERDADERO**	a
VERDADERO	**FALSO**	a
FALSO	**VERDADERO**	b, c
FALSO	**FALSO**	b, d

Lo cual se visualiza más fácilmente a través de un diagrama de flujo:

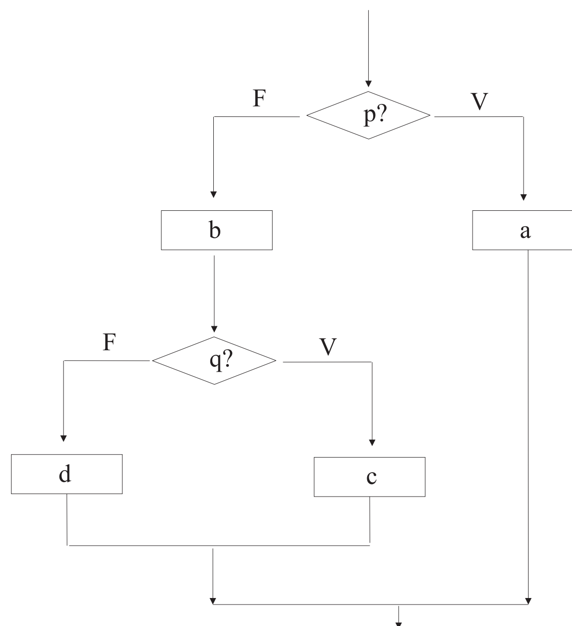


Figura 1.9:

A continuación se ejemplifica el uso de la estructura de control condicional anidada.

Enunciado: Los operarios de una empresa trabajan en dos turnos; uno diurno, cuyo código es 1, y el otro nocturno cuyo código es 2. Se desea calcular el jornal para un operario sabiendo que, para el turno nocturno, el pago es de $5 la hora y, para el turno diurno es $3 la hora, pero en este último caso, si el día es Domingo se paga un adicional de $1 por hora.

Las fórmulas para calcular el jornal son:
fórmula 1 (para el turno nocturno) : jornal = 5 * horas trabajadas
fórmula 2a (turno diurno, no es domingo) : jornal = 3 * horas trabajadas
fórmula 2b (turno diurno, domingo) : jornal = (3 + 1) * horas trabajadas

Ambiente del Algoritmo:

VARIABLES	DESCRIPCIÓN
HORAS	Variable de entrada, de tipo entero, cuyo valor es la cantidad de horas trabajadas en un día, por un operario.
TURNO	Variable de entrada, de tipo entero, cuyo valor es el código del turno.
DIA	Variable de entrada, de tipo caracter, si su valor es '*d*', indica que es día domingo, sino tiene un valor '*n*'
JORNAL	Variable de salida, de tipo entero que contiene el valor de la paga que debe efectuarse.

Algoritmo:

Versión 1:
Declarar datos
Leer datos de entrada
Calcular el jornal
Informar resultados

Como las acciones de este algoritmo no son primitivas, aplicaremos la técnica de refinamiento sucesivo.

Versión 2:
Declarar datos
Leer los valores correspondientes a las horas trabajadas, el código del turno y el código del día.
Si el turno es nocturno
    entonces
        calcular el jornal usando la fórmula 1
    sino
        calcular el jornal usando la fórmula 2
finsi
Escribir el valor del jornal

Como la acción calcular el jornal usando la fórmula 2 no refleja todavía las restricciones del problema debemos realizar un nuevo refinamiento:

Versión 3:

```
Declarar datos
Leer el valor correspondiente a las horas trabajadas
Leer el valor del código del turno
Leer el valor del código del día.
Si el turno es nocturno
    entonces
        calcular el jornal usando la fórmula 1
    sino
        si el día no es domingo
            entonces
                calcular el jornal usando la fórmula 2a
            sino
                calcular el jornal usando la fórmula 2b
        finsi
finsi
Escribir el valor del jornal.
```

Versión 4 (final):

```
ALGORITMO "Jornales"
COMENZAR
HORAS, TURNO, JORNAL: entero
DIA: caracter
LEER HORAS, TURNO, DIA
SI TURNO = 2
    ENTONCES
        JORNAL ⟵ 5* HORAS
    SINO
        SI DIA <> 'd'
            ENTONCES
                JORNAL ⟵ 3* HORAS
            SINO
                JORNAL ⟵ (3 + 1)* HORAS
        FINSI
FINSI
ESCRIBIR JORNAL
FIN
```

La tabla siguiente describe tres ejecuciones distintas del algoritmo, a partir de distintos valores de entrada, los cuales son elegidos teniendo en cuenta que se ejecuten todas las posibles acciones del algoritmo.

Valores Iniciales	HORAS: 8 TURNO: 2 DIA: ‘*n*’	HORAS: 12 TURNO: 1 DIA: ‘*n*’	HORAS: 10 TURNO: 1 DIA: ‘*d*’
Acciones Ejecutadas	JORNAL ⟵ $5 * 8$	JORNAL ⟵ $3 * 12$	JORNAL ⟵ $4 * 10$
Valores Finales	JORNAL = 40	JORNAL = 36	JORNAL = 40

1.6.4. Estructura de Control de Repetición

En esta sección analizaremos dos estructuras de control repetitivas que permiten repetir una acción o una secuencia de acciones. Estas estructuras se diferencian, esencialmente, respecto de si el número de repeticiones es o no conocido "a priori".

Estructura de Repetición *"MIENTRAS <condicion> HACER ... FINMIENTRAS"*

El formato de esta estructura de control en **nuestro lenguaje de diseño de algoritmos** es:

MIENTRAS <condición> **HACER** <secuencia de acciones> **FINMIENTRAS**

En esta estructura de control la palabra **FINMIENTRAS** es un delimitador que se utiliza para indicar el fin de la secuencia de acciones a repetir. La cantidad de veces que se ejecutará la <secuencia de acciones> o el cuerpo de la repetición: $0, 1, 2, \ldots, n$ no es conocida de antemano sino que depende de la verdad o falsedad de la <condición> . La <condición> es evaluada antes de la ejecución de la secuencia. Si el resultado es verdadero, la <secuencia de acciones> se ejecuta; si la condición resulta falsa, finaliza la repetición, es decir, se ejecutará la primer acción primitiva que siga a **FINMIENTRAS** (si ésta existe). El correspondiente diagrama de flujo es:

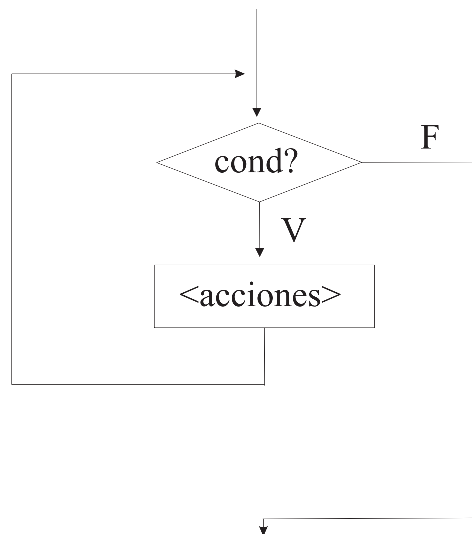


Figura 1.10:

Estructura de Repetición *"PARA... HACER ...FINPARA"*

El algoritmo "Factorial de n", desarrollado en secciones previas, incluye dos acciones que se repiten un número de veces conocido de antemano y que está dado por el valor de la variable NUMERO. Podemos entonces, describir con la siguiente frase, el tratamiento dado al algoritmo:

> "para cada uno de los valores de la variable I, desde 1 hasta el valor de la variable NUMERO, asignar a FACTORIAL, su valor multiplicado por el valor de I".

Para muchas situaciones, puede resultar útil disponer de una estructura de control repetitiva que permita que una secuencia de acciones se ejecute un número fijo, conocido de antemano, de veces. El formato de esta nueva estructura de control es el siguiente:

> **PARA** V **DESDE** Vi **HASTA** Vf **CON PASO** P **HACER**
> <secuencia de acciones>
> **FINPARA**

Donde:

1. V es una variable de tipo entero llamada ***variable de control*** de la repetición.
2. Vi**,** Vf **y** P pueden ser variables o constantes de tipo entero o expresiones aritméticas. Vi recibe el nombre de ***valor inicial***; Vf recibe el nombre de ***valor final y*** P ***es el paso***, o sea en cuanto se incrementa (o decrementa) V para llegar desde Vi a Vf y debe ser distinto de cero (pero puede ser positivo o negativo).

La forma en que el procesador interpreta esta estructura de control de repetición es la siguiente:

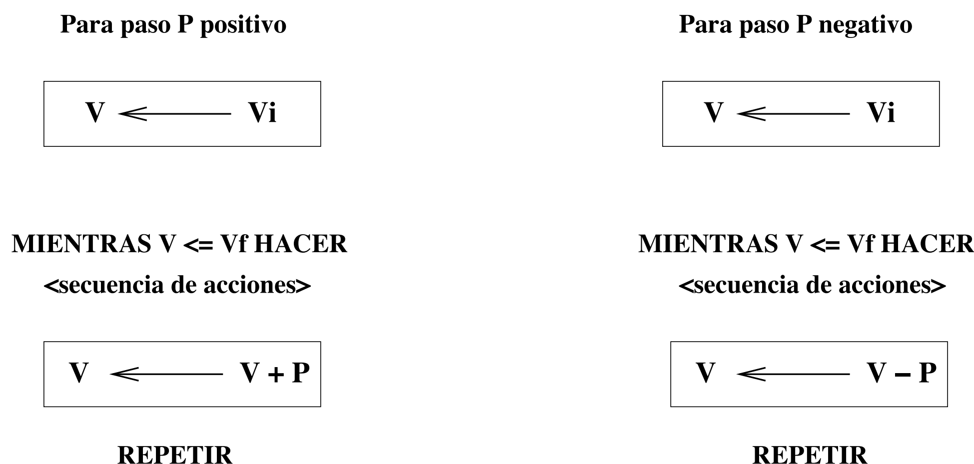


Figura 1.11:

El número de veces que se ejecuta repetitivamente la <secuencia de acciones> se determina al momento en que el procesador ingresa a la ejecución de esta estructura. Este número está dado por la parte entera de:

$$(Vf - Vi + P)/P$$

Si este número es cero o negativo, la repetición no se ejecuta. **No es aconsejable** incluir dentro de <secuencia de acciones> acciones que modifiquen ni a V (variable de control) ni a Vi, Vf y P, aunque puedan utilizarse para efectuar cálculos.

Para ejemplificar el uso de esta estructura, reescribamos el algoritmo “Factorial de n”:

ALGORITMO “Factorial de n”
COMENZAR
    NUMERO, FACTORIAL, I: entero
    LEER NUMERO
    FACTORIAL $\longleftarrow$ 1
    PARA I **DESDE** 1 **HASTA** NUMERO **CON PASO 1 HACER**
        FACTORIAL $\longleftarrow$ FACTORIAL * I
    FINPARA
    ESCRIBIR FACTORIAL
FIN

Ciclos Anidados

De la misma forma que es posible incluir, dentro de una estructura de decisión, otra estructura de decisión; también es posible insertar una estructura de repetición en el interior de otra estructura de repetición. Las reglas de anidamiento son similares en ambos casos: La estructura interna debe estar totalmente contenida dentro de la estructura externa, no permitiéndose el solapamiento.

Las figuras siguientes muestran casos válidos e inválidos de anidamiento:

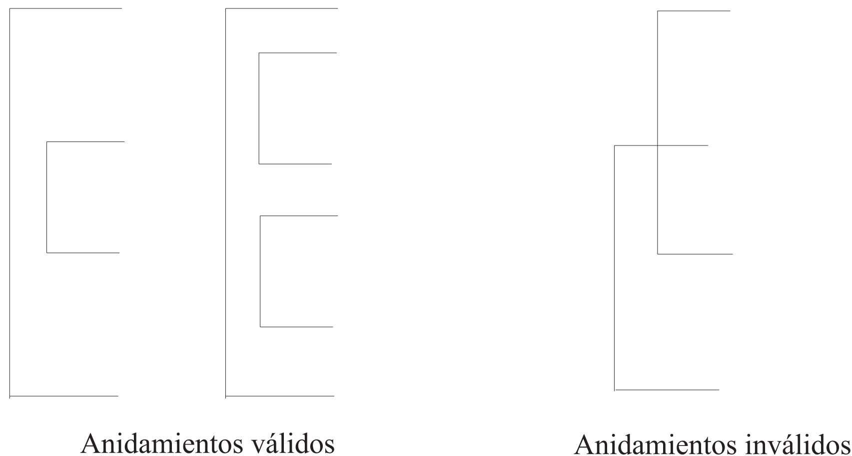


Figura 1.12:

Como un ejemplo de ciclos anidados, modifiquemos el algoritmo “Factorial de n” para calcular el factorial de varios números naturales (enteros positivos). El algoritmo “Factorial de n” utiliza una estructura de repetición para calcular el factorial de un número, el cual se incluye, totalmente, en otra para leer una serie de valores de entrada. El algoritmo modificado, lee un conjunto de N valores de entrada, calculando y escribiendo el valor del factorial para cada uno de ellos.

ALGORITMO "Factorial de varios números"
COMENZAR
  N, J, NUMERO, FACTORIAL, I: entero
  {leer cantidad de números}
  LEER N
  PARA J **DESDE** 1 **HASTA** N **CON PASO** 1 **HACER**
    LEER NUMERO
    FACTORIAL ⟵ 1
    PARA I **DESDE** 1 **HASTA** NUMERO **CON PASO** 1 **HACER**
      FACTORIAL ⟵FACTORIAL * I
    FINPARA
    ESCRIBIR FACTORIAL
  FINPARA
FIN

En la siguiente tabla, se resume el comportamiento del algoritmo para calcular el factorial de 3 y 5.

ACCIÓN	N	J	I	NUMERO	FACTORIAL	PANTALLA
LEER N	2					
1er iteración sobre J		1[2]				
LEER NUMERO				3		
FACTORIAL $\longleftarrow$ 1					1	
1er iteración sobre I			1[3]			
FACTORIAL $\longleftarrow$ FACTORIAL * I					1	
2da iteración sobre I			2			
FACTORIAL $\longleftarrow$ FACTORIAL * I					2	
3er iteración sobre I			3			
FACTORIAL $\longleftarrow$ FACTORIAL * I					6	
4ta iteración sobre I			4			
ESCRIBIR FACTORIAL						6
2da iteración sobre J		2				
LEER NUMERO				5		
FACTORIAL $\longleftarrow$ 1					1	
1er iteración sobre I			1[5]			
FACTORIAL $\longleftarrow$ FACTORIAL * I					1	
2da iteración sobre I			2			
FACTORIAL $\longleftarrow$ FACTORIAL * I					2	
3er iteración sobre I			3			
FACTORIAL $\longleftarrow$ FACTORIAL * I					6	
4ta iteración sobre I			4			
FACTORIAL $\longleftarrow$ FACTORIAL * I					24	
5ta iteración sobre I			5			
FACTORIAL $\longleftarrow$ FACTORIAL * I					120	
6ta iteración sobre I			6			
ESCRIBIR FACTORIAL						120
3er iteración sobre J		3				

Capítulo 2

Estructuración de Datos

2.1. Introducción

En el capítulo 1 se introdujo el concepto de objeto de dato que, como se recordará, podía ser un número entero o real, un carácter o un valor lógico. Ahora, extenderemos este concepto de objeto de dato a un conjunto o grupo de ellos.

Cuando nos encontramos resolviendo un problema particular, tenemos que tratar con la organización de sus datos en forma estructurada; por lo tanto, seleccionar los datos, adecuadamente, es un paso necesario y fundamental al definir y, posteriormente, resolver el problema. Todas las formas posibles en que los datos primitivos se relacionan lógicamente, definen estructuras de datos.

> Una **estructura de datos** es un conjunto de datos reunidos bajo un único nombre colectivo.

Las diferentes estructuras se diferencian por la forma en que sus componentes están relacionadas y por el tipo de los mismos. Todos los datos estructurados deben, en última instancia, ser construidos a partir de los datos primitivos.

Por ejemplo, una estructura conocida y muy simple es el número complejo, que toma la forma de un par ordenado de números reales, donde los números reales son de tipo primitivo. Otra estructura un poco más complicada, pero muy común, es el **arreglo lineal**.

2.2. Arreglo Lineal

Consideremos el siguiente problema:

Enunciado: Una empresa recibe mensualmente información sobre las ventas de cada una de sus tres sucursales y, desea obtener un listado de aquellas, cuyas ventas superan el promedio de las mismas.

Para dicho problema podemos encontrar distintos algoritmos que lo solucionen.

Solución 1:

Método: Para realizar esta tarea con las herramientas provistas hasta el momento, por nuestro lenguaje de diseño, pueden leerse dos veces los datos de las tres sucursales: en la primera lectura, se determinará el promedio de las ventas y, una vez que este promedio sea conocido, puede realizarse la segunda lectura de las ventas de las sucursales, para determinar, cuáles tienen ventas superior al

promedio. Finalmente, se informará cuales son las sucursales, con sus correspondientes ventas, que cumplen con esta condición.

Ambiente del algoritmo:

VARIABLES	**DESCRIPCION**
VENTAS	Variable de entrada, de tipo real, en la cual se leen las ventas de las distintas sucursales y de salida, donde se muestra la venta de la sucursal si corresponde.
PROMEDIO	Variable auxiliar, de tipo real, en la cual se calcula el promedio de ventas.
I	Variable de salida, de tipo entero, cuyo valor indica el número de la sucursal.
3	constante entera que indica el número de sucursales de la empresa.

Algoritmo:

Versión 1:
T_1 Declarar objetos variables.
T_2 Ingresar datos de entrada.
T_3 Calcular el promedio de ventas.
T_4 Comprobar cuál o cuáles de las sucursales tiene venta superior al promedio e informar.

Versión 2: (Aplicando la técnica de refinamientos sucesivos obtenemos)
T_1 Declarar objetos variables.
$T_{2,1}$ Para cada sucursal ingresar ventas y acumular sus valores.
T_3 Calcular el promedio de ventas.
$T_{4,1}$ Para cada sucursal ingresar venta y comprobar si su venta es mayor o igual al promedio e informar.

Versión 3 (final):

```
ALGORITMO "Ventas"
  COMENZAR
    VENTAS, PROMEDIO: real
    I: entero
    PROMEDIO ⟵ 0
    PARA I DESDE 1 HASTA 3 CON PASO 1 HACER
      LEER VENTAS
      PROMEDIO ⟵ PROMEDIO + VENTAS
    FINPARA
    PROMEDIO ⟵ PROMEDIO / 3
    PARA I DESDE 1 HASTA 3 CON PASO 1 HACER
      LEER VENTAS
      SI VENTAS >= PROMEDIO
        ENTONCES
          ESCRIBIR I, VENTAS
      FINSI
    FINPARA
  FIN
```

Si bien este algoritmo funciona es ineficiente por el hecho de leer, dos veces, el mismo conjunto de datos de entrada.

Solución 2:

Una forma alternativa de obtener los mismos resultados es definiendo tres variables reales, cuyos valores correspondan a las ventas de cada una de las sucursales.

Ambiente del algoritmo:

VARIABLES	**DESCRIPCION**
VENT1	Variable de entrada, real, cuyo valor representa las ventas de la sucursal 1 y de salida en caso que corresponda.
VENT2	Variable de entrada, real, cuyo valor representa las ventas de la sucursal 2 y de salida en caso que corresponda.
VENT3	Variable de entrada, real, cuyo valor representa las ventas de la sucursal 3 y de salida en caso que corresponda.
PROMEDIO	Variable auxiliar, real, en la cual se calcula el promedio de las ventas.
3	constante entera que indica la cantidad de sucursales de la empresa.

Algoritmo:

```
ALGORITMO "Ventas"
  COMENZAR
      VENT1, VENT2, VENT3, PROMEDIO: real
      LEER VENT1, VENT2, VENT3
      PROMEDIO ⟵ (VENT1 + VENT2 + VENT3) / 3
      SI VENT1 >= PROMEDIO
         ENTONCES
            ESCRIBIR "1-", VENT1
      FINSI
      SI VENT2 >= PROMEDIO
         ENTONCES
            ESCRIBIR "2-", VENT2
      FINSI
      SI VENT3 >= PROMEDIO
         ENTONCES
            ESCRIBIR "3-", VENT3
      FINSI
   FIN
```

Notemos que este algoritmo es válido, sólo, para tres sucursales. ¿Cómo plantearíamos el algoritmo, por ejemplo, para 100 sucursales?.

Si utilizáramos el mismo enfoque deberíamos definir 100 variables, una para cada sucursal: VENT1, VENT2, . . . , VENT100 y el algoritmo anterior quedaría:

```
ALGORITMO "Ventas"
   COMENZAR
      VENT1, VENT2, ..., VENT100, PROMEDIO: real
      LEER VENT1, VENT2, ..., VENT100
      PROMEDIO ⟵ (VENT1 + VENT2 + ... + VENT100) / 100
      SI VENT1 >= PROMEDIO
         ENTONCES
            ESCRIBIR "1–", VENT1
      FINSI
      SI VENT2 >= PROMEDIO
         ENTONCES
            ESCRIBIR "2–", VENT2
      FINSI
               .
               .
               .
      SI VENT100 >= PROMEDIO
         ENTONCES
            ESCRIBIR "100–", VENT100
      FINSI
   FIN
```

Como podemos observar hemos utilizado los símbolos:
... y
.
.
.

Al primero, lo hemos usado para abreviar la escritura de la declaración, lectura y la suma de las 100 variables y, al segundo, para indicar la comprobación de las ventas desde la sucursal 3 hasta la 99.

Estos dos símbolos no son comprendidos por ningún procesador y la solución sería escribir, explícitamente, las 100 declaraciones, lecturas, suma más los100 condicionales.

Solución 3:

Una forma más eficiente de resolver este problema consiste en reunir las ventas de las 100 sucursales bajo un único nombre VENTA.

VENTA podría estar representada, esquemáticamente, por la siguiente tabla:

120	578	625		1230
1	2	3		100

Luego, para designar uno de los 100 valores utilizamos su ubicación en la tabla: VENTA[3] representa el tercer valor (625) de la variable VENTA e índice 3 indica la tercera sucursal.

De esta manera definimos un **arreglo lineal** o **vector**, de nombre VENTA, cuyos 100 elementos son VENTA [1], VENTA[2], VENTA[3], . . ., VENTA[100].

Se define entonces a una variable con estructura de arreglo como:

Un conjunto de variables del mismo tipo.

Todo arreglo debe ser declarado previo a su utilización. La declaración del arreglo se realiza de la siguiente manera:

VENTA: arreglo[1...100] de real

Donde:

- Se indica el **nombre** de la variable arreglo; VENTA
- El **tipo de la variable**; que es un arreglo.
- El **tipo de las componentes de la variable**, en nuestro caso; reales.
- La **dimensión**, que indica la cantidad de elementos que tiene el arreglo, indicando el **límite inferior**, **(siempre 1)**, y el **límite superior (siempre mayor al limite inferior)** que en el ejemplo es 100.

Para designar un elemento utilizamos el nombre del arreglo, seguido de una expresión encerrada entre corchetes. El valor de la expresión, da la ubicación del elemento, dentro del arreglo.
Ejemplos:

- VENTA[15] donde 15 es una constante, designa al décimo quinto elemento del arreglo.
- VENTA[I] donde I es una variable y su valor i designa al i-ésimo elemento del arreglo y puede tomar valores entre 1 y 100.
- VENTA[$K + 5$], si el valor de la variable K es k, entonces la expresión denota al $(k + 5)$-ésimo elemento del arreglo.

Cada elemento de un arreglo puede ser de cualquiera de los tipos primitivos: entero, real, caracter o lógico, **pero todos del mismo tipo**. Mientras que la expresión que indica la posición de cada elemento es un número natural, denominado **subíndice**. Por consiguiente, si el subíndice es real, 0 o negativo, la evaluación de la expresión da error.

Cada componente de un arreglo se denota explícitamente, y es accedida directamente, mencionando el nombre del arreglo seguido de una expresión encerrada entre corchetes, a la que se llama **subíndice** del arreglo.

Ahora podemos reescribir el algoritmo "Ventas" usando este nuevo concepto.

Ambiente del algoritmo:

VARIABLES	DESCRIPCION
VENTA	Arreglo, de entrada y de salida, de 100 elementos, cada elemento de tipo real y representando las ventas de cada sucursal.
I	Variable de tipo entero, que se usará como índice del arreglo y que tomará los valores 1,..., 100
PROMEDIO	Variable auxiliar, de tipo real donde se calcula el promedio de las ventas.
100	constante entera que indica el número de sucursales de la empresa.

Algoritmo:

```
ALGORITMO "Ventas"
  COMENZAR
    VENTA: arreglo[1...100] de real
    PROMEDIO: real
    I: entero
    PROMEDIO ⟵ 0
    PARA I DESDE 1 HASTA 100 CON PASO 1 HACER
      LEER VENTA[I]
      PROMEDIO ⟵ PROMEDIO + VENTA[I]
    FINPARA
    PROMEDIO ⟵ PROMEDIO / 100
    PARA I DESDE 1 HASTA 100 CON PASO 1 HACER
      SI VENTA[I] >= PROMEDIO
        ENTONCES
          ESCRIBIR I, VENTA[I]
      FINSI
    FINPARA
  FIN
```

Luego de la declaración de las variables, se asigna a la variable PROMEDIO el valor inicial 0 para luego continuar con una iteración que repite 100 veces la acción LEER que permite ingresar las ventas de cada sucursal e ir acumulándola en la variable PROMEDIO. En cada ejecución de la iteración se ingresa un dato; por ejemplo, cuando I vale 1 en VENTA[1] se ingresa el valor de las ventas de la sucursal 1 y así siguiendo. Luego, en secuencia, sigue el cálculo del promedio y, por último, una repetición que comprueba, para cada sucursal, si las ventas de la misma, superan o no, el valor del promedio calculado y, en caso afirmativo, imprime el número de la sucursal y el valor de la venta de la misma.

2.2.1. Operaciones con arreglos: Asignación y recuperación de valores

Al igual que sucede con cualquier variable simple es posible **asignar un valor** a un elemento de un arreglo y **recuperar su valor**.

Por ejemplo, sea V declarado como V: arreglo [1..50] de enteros, entonces la operación de asignación:

$$V[10] \longleftarrow 15$$

indica que al décimo elemento de V se le asigna el valor 15, mientras que en:

$$J \longleftarrow 13 * J + V[I]$$

Primero, se obtiene el valor de la variable J que se multiplica por la constante 13 y, al resultado, se le suma el valor del i-ésimo elemento del arreglo V y todo el resultado de la expresión se almacena en la variable J.

Otro ejemplo:

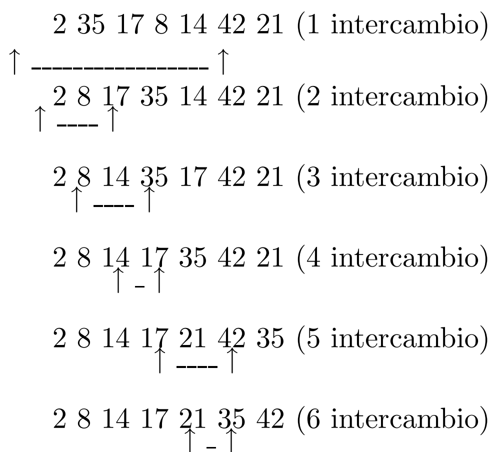
Enunciado: Escribir un algoritmo que ordene de menor a mayor los elementos de un arreglo de 7 elementos enteros.

Método: El método para efectuar el ordenamiento que utilizaremos consiste en:

1. Encontrar el menor elemento, entre los n, del arreglo.
2. Intercambiar el elemento encontrado con el primero del arreglo.
3. Repetir estas operaciones con los $n-1$ elementos restantes, obteniendo, el segundo menor elemento del arreglo; proseguir con los $n-2$ elementos restantes, hasta que quede solamente el mayor valor.

En el ejemplo siguiente, mostramos cómo se intercambian en cada caso los valores:

estado inicial: 21 35 17 8 14 42 2



Ambiente del algoritmo "Ordenar":

VARIABLES	DESCRIPCION
V	Variable de entrada-salida que es el arreglo de enteros a ordenar y donde queda el arreglo ordenado.
7	constante de tipo entero que contiene la dimensión o cantidad de elementos del arreglo.
I, J	Variables auxiliares enteras que se usan como índices del arreglo
MIN	Variable auxiliar de tipo entero que representa al valor del índice donde se encuentra el elemento mínimo de V
VAL_MI	Variable auxiliar, entera, usada para el intercambio de dos valores en el arreglo.

Versión 1:
T_1 Declarar objetos variables.
T_2 Ingresar los elementos del arreglo a ordenar
T_3 Hasta que el vector no esté ordenado hacer
T_4 Asignar a MIN el valor del índice donde se encuentra el menor elemento del arreglo
T_5 Intercambiar en el lugar que corresponda en el ordenamiento el valor V[MIN].
T_6 Cuando el arreglo esté ordenado imprimir el arreglo

Versión 2:
T_1 Declarar objetos variables.
$T_{2,1}$ Para el índice del arreglo desde 1 hasta el número de elementos hacer
$T_{2,2}$ Leer un valor y guardarlo en la posición que indica el índice.
T_3 Hasta que el vector no esté ordenado hacer
$T_{4,1}$ Recorrer el arreglo buscando el elemento con menor valor.
$T_{4,2}$ Asignar a MIN el valor del índice donde se encuentra el menor elemento del arreglo
T_5 Intercambiar en el lugar que corresponda en el ordenamiento el valor V[MIN].
$T_{6,1}$ Cuando el arreglo esté ordenado recorrer el arreglo imprimiendo cada elemento.

Versión 3 (final):

```
ALGORITMO "Ordenar"
    COMENZAR
        V: arreglo [1...7] de entero
        I, J, MIN,VAL_MI: entero
        PARA I DESDE 1 HASTA 7 CON PASO 1 HACER
            LEER V[I]
        FINPARA
        PARA I DESDE 1 HASTA 7 - 1 CON PASO 1 HACER
            MIN ⟵ I
            PARA J DESDE I + 1 HASTA 7 CON PASO 1 HACER
                SI V[J] < V[MIN]
                    ENTONCES
                        MIN ⟵ J
                FINSI
            FINPARA
            VAL_MI ⟵ V[MIN]
            V[MIN] ⟵ V[I]
            V[I] ⟵ VAL_MI
        FINPARA
        PARA I DESDE 1 HASTA 7 CON PASO 1 HACER
            ESCRIBIR V[I]
        FINPARA
    FIN
```

Una ejecución parcial del algoritmo usando como valores de entrada los dados como ejemplo para explicar el funcionamiento del método de ordenamiento a usar es:

después de la lectura	$V[1]$	$V[2]$	$V[3]$	$V[4]$	$V[5]$	$V[6]$	$V[7]$	I	MIN	J	VAL_MI
	21	**35**	**17**	**8**	**14**	**42**	**2**				
ingreso al **para** externo								1^6			
1er iteración sobre I								**1**	**1**		
ingreso al **para** interno										2^7	
1er iteración sobre J										**2**	
2da iteración sobre J									**3**	**3**	
3er iteración sobre J									**4**	**4**	
4ta iteración sobre J										**5**	
5ta iteración sobre J										**6**	
6ta iteración sobre J									**7**	**7**	
salida del **para** interno	**2**	35	17	8	14	42	**21**				**2**
2da iteración sobre I								**2**	**2**		
ingreso al **para** interno										3^7	
1er iteración sobre J									**3**	**3**	
2da iteración sobre J									**4**	**4**	
3er iteración sobre J										**5**	
4ta iteración sobre J										**6**	
5ta iteración sobre J										**7**	
salida del **para** interno	2	**8**	17	**35**	14	42	21				**8**

Queda como ejercicio para el alumno completar la tabla de ejecución del algoritmo.

En la tabla anterior:

- Los valores en negrita representan los valores de las variables que se modifican durante la ejecución que se está analizando.
- Para las variables de control de la repetición, por ejemplo I, al ingresar al PARA externo, usar la notación 1^6 se interpreta: el 1 como valor inicial de dicha variable y el superíndice 6 como el valor final de la misma.

Capítulo 3

Subalgoritmos

3.1. Introducción

Dado un problema relativamente complejo, la manera más conveniente de resolverlo es dividirlo en tareas o módulos de menor complejidad. Ha estos módulos, en lenguaje de diseo se los denomina **subalgoritmos**.

Un subalgoritmo tiene que ser **lógicamente coherente y autocontenido** y formar parte de un algoritmo de mayor envergadura. Por **lógicamente coherente** queremos significar que contiene un conjunto determinado de acciones que conducen a la solución de una tarea específica, relativamente simple; mientras que **autocontenido** significa que es independiente de cualquier otro problema y de sus soluciones. Los subalgoritmos son una facilidad del lenguaje de diseño (y de los lenguajes de programación) que permiten que determinados conjuntos de acciones puedan ser provistos en una forma modular. La **característica** de este conjunto de acciones es que realizan tareas comunes que pueden utilizarse para resolver distintos tipos de problemas o bien, dentro de un mismo problema, con distintos datos.

Estos subalgoritmos se escriben una vez y, luego, son usados por todos aquellos que requieran de ellos. Por ejemplo: ordenar datos en orden ascendente/descendente, como así también las funciones primitivas, definidas en el capítulo anterior (raíz cuadrada, módulo, etc.).

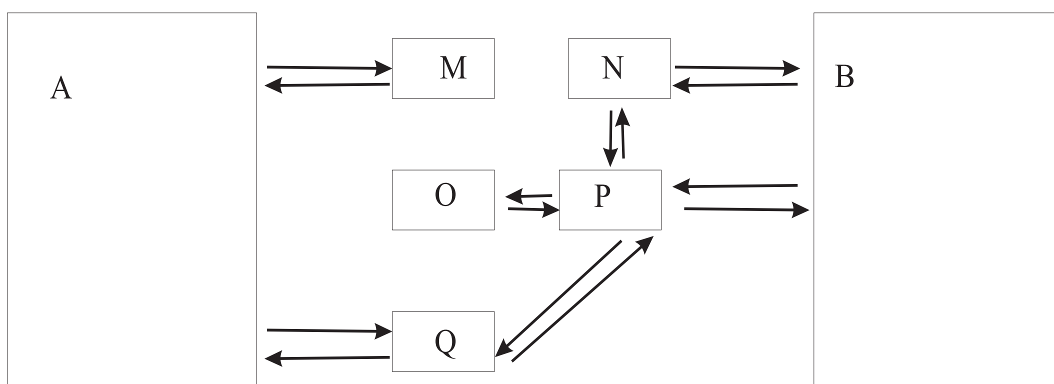


Figura 3.1:

Lo expresado puede verse gráficamente en la Figura 3.1:

En esta figura los subalgoritmos M, N, P, O y Q han sido escritos una vez y son usados indistintamente por los algoritmos A y B, o entre ellos.

3.2. Definición de Subalgoritmos

Los subalgoritmos se definen colocando la palabra **SUBALGORITMO**, seguida por la especificación de su nombre, los argumentos o parámetros, las declaraciones de sus propias variables globales y la secuencia de acciones a realizar dentro del cuerpo del mismo.

SUBALGORITMO "Nombre" (<**Clase del parámentro**> <Nombre del parámetro>: <Tipo del parámetro>, ...)
  COMENZAR
    Secuencia de acciones
  FIN

El nombre de un subalgoritmo se construye siguiendo las mismas reglas que para la contrucción de una variable.

Los parámetros que aparecen en la definición de un subalgoritmo se conocen con el nombre de **parámetros formales**. Los parámetros formales **se utilizan sólo** en las **acciones** que conforman el **cuerpo** del subalgoritmo, al igual que las variables locales que hayan definido. Estos parámetros permiten ingresar datos, egresar datos o ambas cosas al subalgotirmo y de acuerdo a esto, pueden definirse de tres diferentes clases:

- **in:** son sólo **parámetros de entrada**, esto implica que se utilizan para proveer los datos que necesita el subalgoritmo.
- **out:** son sólo **parámetros de salida**, donde el subalgoritmo devuelve los resultados de la ejecución de sus acciones.
- **in out:** son parámetros que, por un lado, proveen los datos de entrada al subalgoritmo y, por otro, en ellos el subalgoritmo devuelve los resultados de su ejecución.

Para ejemplificar, utilizaremos el problema del cálculo del factorial de un número entero positivo visto en el capítulo 1.

SUBALGORITMO "Factorial" (**in** N: entero, **out** FAC: entero)
  COMENZAR
    I: entero
      FAC $\longleftarrow$ 1
      PARA I **DESDE** 1 **HASTA** N **CON PASO** 1 **HACER**
      FAC $\longleftarrow$ FAC$*I$
      FINPARA
  FIN

En el ejemplo del subalgoritmo FACTORIAL, el parámetro N es un parámetro de entrada de tipo entero, que le provee al subalgoritmo el valor del número para el cual debe calcular el factorial, mientras que el parámetro FAC es un parámetro de salida de tipo entero y es donde el algoritmo devuelve el resultado del cálculo del factorial.

Las variables que se declaran dentro de un subalgoritmo sólo pertencen al ambiente del subalgoritmo donde están definidas y se conocen con el nombre de **variables locales**. Tienen el mismo ámbito de uso que los parámetros formales del subalgoritmo. Tal es el caso de la variable I. Las sentecias **COMENZAR** Y **FIN** definen el ambiente del subalgoritmo y la secuencia de acciones comprendidas entre dichas sentencias forman parte del cuerpo del subalgoritmo.

3.3. Invocación y Retorno de Subalgoritmos

Para que las acciones descriptas en el subalgoritmo sean ejecutadas, es necesario que el mismo sea invocado desde un algoritmo principal o desde otro subalgoritmo a fin de proporcionarle los argumentos o parámetros de entrada para poder, con ellos, ejecutar dichas acciones. Es importante aclarar que si el subalgoritmo fue definido con una determinada cantidad de parámetros formales, debe ser invocado con esa misma cantidad de parámetros actuales. Además, deben coincidir los tipos utilizados en la invocación con los tipos declarados en la definición del subalgoritmo.

Cada vez que un subalgoritmo es invocado desde un algoritmo (o subalgoritmo) se establece, automáticamente, una **correspondencia** entre los parámetros formales y los actuales. Esta correspondencia está definida **por la posición que los parámetros ocupan dentro de la lista de parámetros**, es decir, el primer parámetro actual se corresponde con el primer parámetro formal; el segundo parámetro actual con el segundo parámetro formal y, así siguiendo, hasta completar todos los parámetros.

Para los **parámetros formales** que fueron definidos como:

- **in:** los **parámetros actuales** pueden ser constantes, variables (definidas en el ambiente del algoritmo invocante), expresiones o valores de funciones.
- **out** o **in out:** los **parámetros actuales** deben ser variables definidas en el ambiente del algoritmo invocante, pues allí el subalgoritmo devuelve sus resultados.

A continuación, se da un ejemplo de cómo un algoritmo principal, usa el subalgortimo FACTORIAL.

Enunciado: Dado un número **n** entero positivo, ingresado por el usuario, se desea calcular e imprimir el factorial de **n**, de $\mathbf{n}^2$ y de $\mathbf{n}^3$, usando el subalgoritmo FACTORIAL definido previamente.

Ambiente del Algoritmo:

VARIABLES	**DESCRIPCIÓN**
N	Variable de entrada, entera, contendrá el número ingresado
Fact	Variable de salida, entera, que contendrá los distintos valores de factorial.

ALGORITMO "Cálculo de factoriales"
COMENZAR
FACT, N: entero
LEER N
FACTORIAL(N, FACT)
ESCRIBIR FACT
FACTORIAL(Nˆ 2, FACT)
ESCRIBIR FACT
FACTORIAL(Nˆ 3, FACT)
ESCRIBIR FACT
FIN

Si hubiésemos tenido que resolver el mismo problema, sin contar con la facilidad del subalgoritmo, ¿cómo sería el algoritmo que lo resuelve?

El algoritmo llama o **invoca** al subalgoritmo, escribiendo su nombre seguido de la lista de argumentos, los cuales deben coincidir en cantidad, tipo y orden con los que el mismo fue definido. En el ejemplo anterior siempre que se invoca al subalgoritmo FACTORIAL, se le pasan dos argumentos de tipo entero.

Los objetos que sustituyen a los parámetros formales cuando el subalgoritmo se invoca, se denominan **parámetros actuales**.

En este ejemplo,el subalgoritmo FACTORIAL, devuelve el resultado en su segundo parámetro (FACT).

Entonces, cuando desde un algoritmo se invoca a un subalgoritmo **la ejecución del algoritmo se suspende y el control de la ejecución se transfiere a la primera acción del cuerpo del subalgoritmo**.

Luego de la ejecución de las acciones que forman el cuerpo del subalgoritmo, el **control de la ejecución** retorna a la acción que sigue a la invocación del subalgoritmo en el algoritmo o subalgoritmo que lo invocó.

Suponiendo que el valor leído en NUMERO es 4, ¿Cuáles serían los valores de salida escritos por el algoritmo "Cálculo de factoriales"?.

En la Introducción se mencionó que los subalgoritmos son una facilidad provista por el Lenguaje de Diseño (y los lenguajes de programación) que permite:

1. Dentro de un mismo problema, realizar la misma tarea sobre conjuntos distintos de datos.

2. Modularizar el problema, de esta manera cada subtarea puede ser realizada por un subalgoritmo.

El algoritmo anterior, "Cálculo de Factoriales", es un ejemplo del punto 1.

El algoritmo, que desarrollaremos a continuación, es un ejemplo que ilustra el punto 2.

Otro ejemplo:

Enunciado: El dueño de una granja lechera ha decidido utilizar una parte del campo que dedica a pastura para construir una granja de vegetales y desea cercar un lote rectangular cercano a su casa. También desea arar el lote y enriquecer el suelo con los nutrientes necesarios para los vegetales que quiere plantar. Antes de seguir adelante con su proyecto desea estimar cuanto le costará. Por lo tanto, le solicita a un empleado que le diseñe un algoritmo que calcule los costos del proyecto informando

la cantidad de metros de cerco necesarios, el costo del cerco, la cantidad de fertilizante requerida y el costo total del fertilizante.

El granjero conoce el precio por metro del cerco requerido, el costo por gramo de los nutrientes químicos y cuántos m^2 de suelo fertiliza cada gramo de nutriente.

Además el granjero desea experimentar con diferentes tamaños de lotes y comparar sus costos.

Método: Para resolver este problema vamos a aplicar un enfoque modular. Qué significa esto? Tratar de ver en qué subtareas podemos dividir el problema:

T_1 Dados el largo y ancho del lote encontrar su perímetro (para conocer cuánto debemos cercar) y el área del mismo (para conocer la cantidad de fertilizante a utilizar).

T_2 Dado el costo por metro del cerco y la longitud a cercar (perímetro del lote) calcular el costo total del cerco.

T_3 Dada el área del lote y conociendo cuanto terreno permite fertilizar 1 gramo de nutriente, calcular la cantidad de fertilizante necesaria.

T_4 Dada la cantidad de fertilizante requerida y su precio por gramo calcular el costo total del fertilizante.

Cada una de estas tareas puede ser visualizada como un módulo. Cada uno de estos módulos es lógicamente coherente (porque cada uno de ellos realiza una tarea bien delimitada) y también es autocontenido (porque cada uno de ellos es independiente, es decir de otros problemas y sus soluciones).

Gráficamente podemos visualizarlos en la Figura 3.2, donde Largo y Ancho son los valores de entrada que requiere el módulo para poder realizar su tarea y Perímetro y Área son los resultados que devuelve, y que se pueden visualizar en la Figura 3.3.

Figura 3.2:

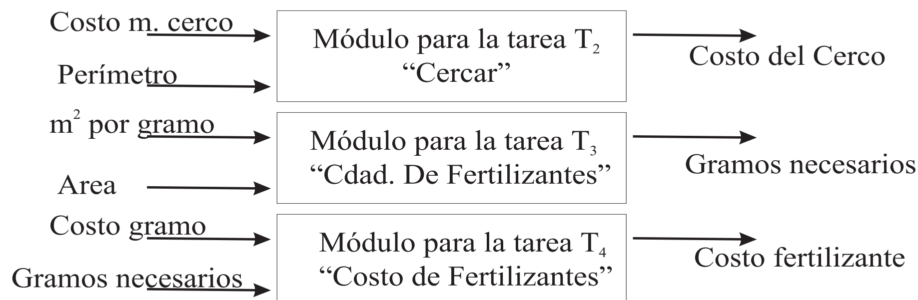


Figura 3.3:

Cada uno de estos módulos o tareas puede implementarse usando subalgoritmos. Esto es lo que vamos a hacer a continuación.

1er Paso: Construir el subalgoritmo Cálculo_Geométrico

Descripción del Ambiente:

VARIABLES	**DESCRIPCION**
Largo	Parámetro formal de entrada (**in**) de tipo real, contiene el alto del lote.
Ancho	Parámetro formal de entrada (**in**) de tipo real, contiene el ancho del lote.
Perimetro	Parámetro formal de salida (**out**) de tipo real, devuelve el perímetro del lote.
Area	Parámetro formal de salida (**out**) de tipo real, devuelve el área del lote.

Construcción del Subalgoritmo

Versión 1:
$T_{1,1}$. Calcular el perímetro del lote (rectángulo)
$T_{1,2}$. Calcular el área del lote

Versión 2:
$T_{1,1,1}$. Guardar en el parámetro de salida Perimetro el resultado del cálculo del mismo.
$T_{2,1,1}$. Guardar en el parámetro de salida Area el resultado del cálculo de la misma

Para calcular el perímetro del rectángulo conocemos la fórmula que es:

2 *(Ancho + Largo)

y también conocemos la fórmula para el área:

Ancho * Largo

entonces estamos ya en condiciones de escribir, usando el lenguaje de diseño, la versión final del subalgoritmo.

Lenguaje de Diseño

```
SUBALGORITMO "Cálculo_Geométrico" (in Largo, Ancho: real, out Perímetro, Area: real)
    COMENZAR
        Perimetro ⟵ 2 * (Ancho + Largo)
        Área ⟵ Ancho * Largo
    FIN
```

2do. Paso: Construir el subalgoritmo "Cercar"

Descripción del Ambiente:

VARIABLES	**DESCRIPCION**
Costo_m_cerco	Parámetro formal de entrada (**in**) de tipo real, contiene el costo del cerco por metro.
Perimetro	Parámetro formal de entrada (**in**) de tipo real, contiene el perímetro del lote.
Costo_del_cerco	Parámetro formal de salida (**out**) de tipo real, devuelve el costo del cercado del lote.

Construcción del Subalgoritmo

Versión 1:
T_1 Calcular el costo del cercado multiplicando el perímetro por el costo del metro de cercado y devolver el resultado en el parámetro de salida.

Lenguaje de Diseño

SUBALGORITMO "Cercar" (**in** Costo_m_cerco, Perimetro: real, **out** Costo_del_cerco: real)
COMENZAR
Costo_del_cerco ⟵ Perimetro * Costo_m_cerco
FIN

3er. Paso: Construir el subalgoritmo "Cdad_de_fertilizante"

Descripción del Ambiente:

VARIABLES	DESCRIPCION
m2_por_gramo	Parámetro formal de entrada (**in**) de tipo real, contiene la cantidad de m^2 que cubre 1 gr. de fertilizante.
Area	Parámetro formal de entrada (**in**) de tipo real, contiene la superficie del lote.
Gramos_necesarios	Parámetro formal de salida (**out**) de tipo real, devuelve la cantidad de fertilizante requerido.

Construcción del Subalgoritmo

Versión 1:
T_1 Calcular la cantidad de fertilizante y devolver el resultado en el parámetro de salida.

Versión 2:
$T_{1,1}$ Dividir el área del terreno por la cantidad de m2 que cubre 1 gramo de fertilizante y devolver el resultado en el párametro de salida.

Lenguaje de Diseño

SUBALGORITMO "Cdad_de_fertilizante" (**in** m2_por_gramo, Area: real, **out** Gramos_necesarios: real)
COMENZAR
Gramos_necesarios ⟵ Area / m2_por_gramo
FIN

4to. Paso: Construir el subalgoritmo "Costo_del_fertilizante"

Descripción del Ambiente:

VARIABLES	DESCRIPCION
Costo_gramo	Parámetro formal de entrada (**in**) de tipo real, contiene el costo del gramo del fertilizante.
Gramos_necesarios	Parámetro formal de entrada (**in**) de tipo real, contiene la cantidad de gramos de fertilizantes.
Costo_fertilizante	Parámetro formal de salida (**out**) de tipo real, devuelve el costo del fertilizante requerido.

Construcción del Subalgoritmo

Versión 1:
T_1 Calcular el costo total del fertilizante necesario para todo el lote y guardarlo en el parámetro de salida

Versión 2:
$T_{1,1}$ Multiplicar los gramos necesarios por el costo por gramo devolver el resultado en el párametro de salida.

Lenguaje de Diseño

SUBALGORITMO "Costo_del_fertilizante" (**in** Costo_gramo, Gramos_necesarios: real **out** Costo_fertilizante: real)
COMENZAR
Costo_fertilizante ⟵ Costo_gramo * Gramos_necesarios
FIN

5to. Paso: Construir el algoritmo "Costo_Proyectos", que usará los subalgoritmos elaborados en los pasos 1 a 4

Descripción del ambiente:
Todos los parámetros de entrada (in) son variables que deben estar declaradas en el algoritmo invocante, lo mismo sucede con los parámetros de salida (out), donde los subalgoritmos devuelven los resultados. Entonces el ambiente del algoritmo es:

VARIABLES	DESCRIPCION
L_lote	Variable de entrada, real, en la que se ingresará el largo del lote
A_lote	variable de entrada, real, en la que se ingresará el ancho del lote
P_lote	Variable de salida, real, que mostrará la cantidad de metros de cerco necesarios.
Area_lote	Variable auxiliar, real, para guardar superficie del lote.
Costo_m_c	Variable de entrada, de tipo real, que contendrá el costo por metro del cerco.
Costo_cerco	Variable de salida, tipo real, que mostrará el costo del cerco.
m2_gramo_fert	Variable de entrada de tipo real, contendrá la cantidad de m^2 que cubre 1 gr. de fertilizante.
Gramos_req	Variable de salida, de tipo real, devuelve la cantidad de fertilizante requerido.
Costo_gr_fert	Variable de entrada, de tipo real, que contendrá el precio del fertilizante.
Costo_t_f	Variable de salida, de tipo real, que mostrará el costo del fertilizante necesario

Construcción del Algoritmo

Versión 1:
T_1 Declarar objetos
T_2 Ingresar datos
T_3 Invocar al subalgoritmo que calcula perímetro y área.
T_4 Invocar al subalgoritmo que calcula el costo del cerco del lote.
T_5 Mostrar los metros requeridos de cerco y el costo.
T_6 Invocar al subalgoritmo que calcula los gramos necesarios de fertilizante.
T_7 Invocar al subalgoritmo que calcula el costo total del fertilizante.
T_8 Mostrar la cantidad necesaria de fertilizante y su costo total.

Lenguaje de Diseño

ALGORITMO "Costo_Proyectos"
COMENZAR
{declaración de variables}
L_lote, A_lote, P_lote, Ar_Lote Costo_m_c, Costo_cerco, m2_gramo_fert, Gramos_req, Costo_gr_fert, Costo_t_f: real
{Ingreso de datos de entrada}
LEER L_lote, A_lote, Costo_m_c, m2_gramo_fert, Costo_gr_fert
{Cálculo e impresión del costo del cerco}
Calculo_Geometrico (L_lote, A_lote, P_lote, Ar_lote)
Cercar (Costo_m_c, P_lote, Costo_cerco)
Escribir P_lote, Costo_cerco
{Cálculo de cantidad de fertilizante}
Cdad_de_fertilizante (m2_gramo_fert, Ar_lote, Gramos_req)
{Cálculo e impresión del costo del fertilizante}
Costo_del_fertilizante (Costo_gr_fert, Gramos_req, Costo_t_f)
Escribir Gramos_req, Costo_t_f
FIN

Ejecución del Algoritmo: Vista de manera esquemática la ejecución del algoritmo se puede visulaizar en la Figura 3.4.

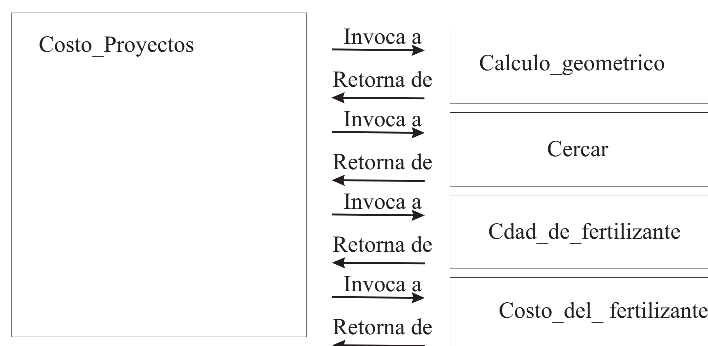


Figura 3.4:

1) después de la acción de lectura										
					Variables del algoritmo					
L_lote	A_lote	P_lote	Ar_lote	Costo_m_c	Costo_cerco	m2_gramo_fert		Gramos_req	Costo_gr_fer	Costo_t_f
46	125			2,48		6			6,98	
2) después de la invocación al subalgoritmo Cálculo_Geométrico										
					Variables del algoritmo					
L_lote	A_lote	P_lote	Ar_lote	Costo_m_c	Costo_cerco	m2_gramo_fert		Gramos_req	Costo_gr_fer	Costo_t_f
46	125			2,48		6			6,98	
			Variables del subalgoritmo Calculo_Geometrico							
Largo	Ancho									
46	125									
3) después del retorno del subalgoritmo Cálculo_Geométrico										
					Variables del algoritmo					
L_lote	A_lote	P_lote	Ar_lote	Costo_m_c	Costo_cerco	m2_gramo_fert		Gramos_req	Costo_gr_fer	Costo_t_f
46	125	**342**	**5550**	2,48		6			6,98	
4) después de la invocación al subalgoritmo Cercar										
					Variables del algoritmo					
L_lote	A_lote	P_lote	Ar_lote	Costo_m_c	Costo_cerco	m2_gramo_fert		Gramos_req	Costo_gr_fer	Costo_t_f
46	125	342	5550	2,48		6			6,98	
			Variables del subalgoritmo Cercar							
Costo_m_cerco		Perimetro								
2,48		342								
5) después del retorno del subalgoritmo Cercar										
					Variables del algoritmo					
L_lote	A_lote	P_lote	Ar_lote	Costo_m_c	Costo_cerco	m2_gramo_fert		Gramos_req	Costo_gr_fer	Costo_t_f
46	125	342	5550	2,48	**848,16**	6			6,98	
6) después de la invocación al subalgoritmo Cdad_de_fertilizante										
					Variables del algoritmo					
L_lote	A_lote	P_lote	Ar_lote	Costo_m_c	Costo_cerco	m2_gramo_fert		Gramos_req	Costo_gr_fer	Costo_t_f
46	125	342	5550	2,48	848,16	6			6,98	
			Variables del subalgoritmo Cdad_de_fertilizantes							
m2_por_gramo		Area								
6		125								
7) después del retorno del subalgoritmo Cdad_de_fertilizante										
					Variables del algoritmo					
L_lote	A_lote	P_lote	Ar_lote	Costo_m_c	Costo_cerco	m2_gramo_fert		Gramos_req	Costo_gr_fer	Costo_t_f
46	125	342	5550	2,48	848,16	6		**22,2**	6,98	
8) después de la invocación al subalgoritmo Costo_del_fertilizante										
					Variables del algoritmo					
L_lote	A_lote	P_lote	Ar_lote	Costo_m_c	Costo_cerco	m2_gramo_fert		Gramos_req	Costo_gr_fer	Costo_t_f
46	125	342	5550	2,48	848,16	6		22,2	6,98	
			Variables del subalgoritmo Costo_de_ fertilizante							
Costo_gramo	Gramos_necesarios									
6,98	22,2									
9) después del retorno del subalgoritmo Costo_del_ fertilizante										
					Variables del algoritmo					
L_lote	A_lote	P_lote	Ar_lote	Costo_m_c	Costo_cerco	m2_gramo_fert		Gramos_req	Costo_gr_fer	Costo_t_f
46	125	342	5550	2,48	848,16	6		22,2	6,98	**154,96**

Figura 3.5:

La tabla de ejecución manual del algoritmo puede verse en la Figura 3.5.

Como puede observarse en la tabla de ejecución del algoritmo, al ejecutarse la acción:

LEER L_lote, A_lote, Costo_m_c, m2_gramo_fert, Costo_gr_fert

Se inicializan las variables de entrada del algoritmo.

Luego de ejecutar la invocación al subalgoritmo:

Calculo_Geometrico (L_lote, A_lote, P_lote, Ar_lote)

los parámetros formales de entrada Largo y Ancho, del subalgoritmo, reciben los valores 46 y 125 respectivamente, que son los valores de los parámetros actuales (L_lote, A_lote del algoritmo principal). En consecuencia, la relación que existe entre los parámetros formales de entrada y los actuales es que estos últimos se utilizan para **dar valores iniciales** a los parámetros formales del tipo **in**.

Durante la ejecución del subalgoritmo se modifican los valores de los parámetros actuales que se asocian con los parámetros formales de tipo **out** (salida) del subalgoritmo: Perimetro y Area, quedando en consecuencia P_lote con un valor de 342 y Ar_lote con 5550. Un razonamiento análogo se utiliza para los restantes pasos de la ejecución del algoritmo.

Construir la tabla de ejecución del algoritmo suponiendo que los valores leídos fueron:

a) 30, 50, 3.05, 5, 6.5 para Llote, A_lote, Costo_m_c, m2_gramo_fert y Costo_gr_fert respectivamente.

b) 55, 110, 2.9, 4, 7.1 para Llote, A_lote, Costo_m_c, m2_gramo_fert y Costo_gr_fert respectivamente.

c) 60, 60, 2.3, 6.5, 7.18 para Llote, A_lote, Costo_m_c, m2_gramo_fert y Costo_gr_fert respectivamente.

Mostrando en cada caso las correspondientes salidas.

Si el parámetro actual de entrada es una expresión, ésta es evaluada al momento de producirse la invocación y, su resultado, es asignado al parámetro formal correspondiente. Así, por ejemplo, si en el algoritmo existiese la siguiente acción de invocación para el subalgoritmo Calculo_Geometrico:

Calculo_Geometrico (L_lote + 5, A_lote, P_lote, Ar_Lote)

Si el valor leído para L_lote fuese 24 entonces con el parámetro formal Largo, en la definición de Calculo_Geometrico se asociará el valor 29.

La misma situación se crea cuando el parámetro actual de entrada es el valor de una función. Por ejemplo, sea el valor leído para A_lote, - 64 si se efectúa la siguiente invocación:

Calculo_Geometrico (L_lote, **ABS**(A_lote), P_lote, Ar_lote)

El valor que se asocie con el parámetro formal Ancho será 64.

Otro ejemplo:

Supongamos que deseamos intercambiar el contenido de dos variables enteras, entonces como es una acción frecuente, decidimos diseñar un subalgoritmo que llamaremos "Intercambio", que tenga dos parámetros del tipo **in out** y unavariable auxiliar. En consecuencia el **ambiente del subalgoritmo** será:

VARIABLES	**DESCRIPCION**
A	parámetro de entrada/salida (in-out) que contiene uno de los valores a intercambiar, de tipo entero
B	parámetro de entrada/salida (in-out) que contiene uno de los valores a intercambiar, de tipo entero.
AUX	vble auxiliar del subalgoritmo, necesaria para poder hacer el intercambio, de tipo entero.

Versión 1:

T_1 Guardar el contenido de una de las variables, por ejemplo A en AUX.

T_2 Almacenar el contenido de la otra variable, B, en A.

T_3 Guardar el contenido de AUX en A.

En consecuencia, la versión definitiva del subalgoritmo es:

Lenguaje de Diseño

SUBALGORITMO "Intercambio" (**in out** A, B: entero)
COMENZAR
AUX: entero
AUX $\longleftarrow$ A
A $\longleftarrow$ B
B $\longleftarrow$ AUX
FIN

Y, supongamos ahora, que deseamos usar este subalgoritmo en un algoritmo que ordene tres valores numéricos en forma ascendente, los valores que se desea ordenar son leídos en las variables X, Y y Z. La salida del algoritmo debe ser X con el menor valor, Y con el que sigue y Z con el mayor.

Ambiente del Algoritmo "Ordenar tres números"

VARIABLES	DESCRIPCION
X	variable de E/S, que contiene el 1er número a ordenar, de tipo entero
Y	variable de E/S, que contiene el 2do número a ordenar, de tipo entero
Z	variable de E/S, que contiene el 3er número a ordenar, de tipo entero

Versión 1:
T_1 Declarar variables X, Y, Z como enteros.
T_2 Leer valores para X, Y, Z.
T_3 Comparar los valores de las 3 variables y ordenarlas de manera tal que en X quede el menor valor, en Y el que le sigue y en Z el mayor valor.
T_4 Mostrar X, Y, Z ordenadas.

Versión 2:

$T_{3,1}$ Comparar X mayor Y, si lo es
dejar el valor de X en Y y el de Y en X
$T_{3,2}$ Comparar Y mayor Z, si lo es
dejar el valor de Y en Z y el de Z en Y
Comparar X mayor Y, si lo es
dejar el valor de X en Y y el de Y en Z

Lenguaje de Diseño

ALGORITMO “Ordenar tres números”
  COMENZAR
    X, Y, Z: entero
    LEER X, Y, Z
    SI X > Y
      ENTONCES
        Intercambio (X, Y)
    FINSI
    SI Y > Z
      ENTONCES
        Intercambio(Y,Z)
        SI X > Y
          ENTONCES
            Intercambio(X,Y)
        FINSI
    FINSI
    ESCRIBIR X, Y, Z
  FIN

El algoritmo lee tres números enteros, por ejemplo, 127, 44 y 12 y, llamando al subalgoritmo tres veces con distintos parámetros actuales obtiene la secuencia ordenada de los tres valores de entrada.

Ejecución del Algoritmo:

VARIABLES DEL ALGORITMO			**VARIABLES DEL SUBALGORITMO**		
1 invocación					
X	Y	Z	A	B	AUX
127	44	12	**127**	**44**	
					127
			44		
				127	
44	**127**				
2 invocación					
44	127	12	**127**	**12**	
					127
			12		
				127	
	12	**127**			
3 invocación					
44	12	127	44	12	
					44
			12		
				44	
12	**44**				

Y al escribir X, Y, Z se obtiene como salida 12, 44 y 127.

Ejecutar el algoritmo con distintos juegos de datos de entrada de manera tal de verificar todas las posibilidades.

4.4 Arreglos como parámetros formales de un subalgoritmo

Cuando se usan arreglos como parámetros formales de un subalgoritmo, éstos deben indicarse sólo por su nombre. Tanto el parámetro formal, como el correspondiente parámetro actual, deben ser arreglos del mismo tipo, debiéndose agregar a la lista de parámetros dos datos más, que corresponden al límite inferior y superior del índice del arreglo.

Formalizando:

SUBALGORITMO *Nombre Subalgoritmo* (**in out**NOM-ARREGLO: arreglo de **Tipo de datos**, **in** LIM−INF, LIM−SUP: entero)
COMENZAR
Cuerpo del subalgortimo
FIN

Ejemplo:

Enunciado: Escribir un subalgoritmo que ordene de menor a mayor los elementos de un arreglo de enteros.

Método: El método para efectuar el ordenamiento que utilizaremos consiste en:
Encontrar el menor elemento, entre los n, del arreglo.
Intercambiar el elemento encontrado con el primero del arreglo.
Repetir estas operaciones con los $n-1$ elementos restantes, obteniendo, el segundo menor elemento del arreglo; proseguir con los $n-2$ elementos restantes, hasta que quede solamente el mayor valor.
En el ejemplo siguiente, mostramos como se intercambian en cada caso los valores:

estado inicial: 21 35 17 8 14 42 2

2 35 17 8 14 42 21 (1 intercambio)

2 8 17 35 14 42 21 (2 intercambio)

2 8 14 35 17 42 21 (3 intercambio)

2 8 14 17 35 42 21 (4 intercambio)

2 8 14 17 21 42 35 (5 intercambio)

2 8 14 17 21 35 42 (6 intercambio)

Para realizar el intercambio de los dos valores podemos utilizar el subalgoritmo definido en un ejemplo anterior:

SUBALGORITMO "Intercambio" (**in out** A, B: entero)
COMENZAR
AUX: entero
AUX ⟵ A
A ⟵ B
B ⟵ AUX
FIN

Ambiente del subalgoritmo "Ordenar":

VARIABLES	DESCRIPCION
VECTOR	Parámetro formal de entrada-salida que es el arreglo de enteros a ordenar.
N	Parámetro formal de entrada de tipo entero que contiene el límite superior.
M	Parámetro formal de entrada de tipo entero que contiene el límite inferior.
I, J	Variables enteras que se usan como índices del arreglo
MINIMO	Variable de tipo entero que representa al valor del índice donde se encuentra el elemento mínimo de VECTOR

Versión 1:

para I desde M hasta N - 1 hacer
asignar a MINIMO el valor del índice correspondiente al menor elemento entre VECTOR[I],...., VECTOR[N]
intercambiar VECTOR[I] CON VECTOR[MINIMO]
finpara

Lenguaje de diseño

SUBALGORITMO "Ordenar" (**in out** VECTOR: arreglo de entero, **in** M,N: entero)
COMENZAR
I, J, MINIMO: entero
PARA I **DESDE** M **HASTA** N - 1 **CON PASO** 1 **HACER**
MINIMO ⟵ I
PARA J **DESDE** I + 1 **HASTA** N **CON PASO** 1 **HACER**
SI VECTOR[J] < VECTOR[MINIMO]
ENTONCES
MINIMO ⟵ J
FINSI
FINPARA
Intercambio (VECTOR[I], VECTOR[MINIMO])
FINPARA
FIN

Ahora podemos definir un algoritmo que ordene distintos arreglos haciendo uso del subalgoritmo "Ordenar".

Ambiente del algoritmo:

VARIABLES	DESCRIPCION
VEC1	Variable de entrada, arreglo de enteros de dimensión 30
VEC2	Variable de entrada, arreglo de enteros de dimensión 50
VEC3	Variable de entrada, arreglo de enteros de dimensión 100
I	Variable auxiliar, entera, que denota al índice de un arreglo

Versión 1:
Declaración de variables
Leer VEC1, VEC2 Y VEC3
Ordenarlos
Imprimirlos

Lenguaje de Diseño

```
ALGORITMO "Ordenar_vectores"
  COMENZAR
    VEC1: arreglo [1..30] de entero
    VEC2: arreglo [1..50] de entero
    VEC3: arreglo [1..100] de entero
    I: entero
    PARA I DESDE 1 HASTA 30 CON PASO 1 HACER
      LEER VEC1[I]
    FINPARA
    PARA I DESDE 1 HASTA 50 CON PASO 1 HACER
      LEER VEC2[I]
    FINPARA
    PARA I DESDE 1 HASTA 100 CON PASO 1 HACER
      LEER VEC3[I]
    FINPARA
    Ordenar (VEC1, 1, 30)
    Ordenar (VEC2, 1, 50)
    Ordenar (VEC3, 1,100)
    PARA I DESDE 1 HASTA 30 CON PASO 1 HACER
      ESCRIBIR VEC1[I]
    FINPARA
    PARA I DESDE 1 HASTA 50 CON PASO 1 HACER
      ESCRIBIR VEC2[I]
    FINPARA
    PARA I DESDE 1 HASTA 100 CON PASO 1 HACER
      ESCRIBIR VEC3[I]
    FINPARA
  FIN
```

Analizando el algoritmo podemos determinar que el mismo trozo de código de lectura/escritura se usa en diferentes momentos con distintos arreglos, entonces podríamos pensar en mejorar dicho algoritmo, escribiendo separadamente dos subalgoritmos: uno, para el ingreso de datos en una variable de tipo arreglo y, otro, para la impresión de los datos de una variable estructurada arreglo.

La definición de dichos subalgoritmos sería:

```
SUBALGORITMO "Lectura arreglos" (in out VEC arreglo de entero, in M, N:entero)
  COMENZAR
    I: entero
    PARA I DESDE M HASTA N CON PASO 1 HACER
      LEER VEC[I]
    FINPARA
  FIN
```

```
SUBALGORITMO "Salida arreglos"(in VEC: arreglo de entero, in M, N: entero)
  COMENZAR
    I: entero
    PARA I DESDE M HASTA N CON PASO 1 HACER
      ESCRIBIR VEC[I]
    FINPARA
  FIN
```

Y la nueva versión de nuestro algoritmo "Ordenar vectores" será:

```
ALGORITMO "Ordenar_vectores"
  COMENZAR
    VEC1: arreglo [1..30]de entero
    VEC2: arreglo [1..50] de entero
    VEC3: arreglo [1..100] de entero
    Lectura Arreglos (VEC1, 1, 30)
    Lectura Arreglos (VEC2, 1, 50)
    Lectura Arreglos (VEC3, 1, 100)
    Ordenar (VEC1, 1, 30)
    Ordenar (VEC2, 1, 50)
    Ordenar (VEC3, 1, 100)
    Salida Arreglos (VEC1, 1, 30)
    Salida Arreglos (VEC2, 1, 50)
    Salida Arreglos (VEC3, 1, 100)
  FIN
```

Donde las instrucciones para leer/escribir los valores de un arreglo fueron escritas una sola vez en los correspondientes subalgoritmos y, en consecuencia, en los algoritmos sólo se invoca a dichos subalgoritmos con los parámetros actuales que correspondan en cada caso.

¿Por qué en la definición del subalgoritmo "Lectura Arreglos" el parámetro formal VEC fue declarado como **in out** mientras que en el subalgoritmo "Salida Arreglos" fue declarado sólo como **in**?

3.4. Anexo

Ejemplo completo de Subalgoritmos con arreglos como parámetros

Problema:

Ingresar valores enteros en un arreglo de 30 posiciones. Posteriormente, imprimir las posiciones pares y a continuación las posiciones impares. Las tareas de ingreso e impresión deben ser realizadas por subalgoritmos.

Algoritmo Principal

Versión 1:

$\mathbf{T}_1$ Definir e inicializar objetos

$\mathbf{T}_2$ Invocar al subalgoritmo Ingreso con los parámetros adecuados

$\mathbf{T}_3$ Invocar al subalgoritmo Imprimir con los parámetros adecuados para que imprima las posiciones pares

$\mathbf{T}_4$ Invocar al subalgoritmo Imprimir con los parámetros adecuados para que imprima las posiciones impares

Versión 2:

$\mathbf{T}_1$

$\mathbf{T}_{1,1}$ Declarar un arreglo ARR de enteros de 30 posiciones

$\mathbf{T}_2$

$\mathbf{T}_{2,1}$ Invocar al subalgoritmo Ingreso con los parámetros (ARR,1,30)

$\mathbf{T}_3$

$\mathbf{T}_{3,1}$ Invocar al subalgoritmo Imprimir con los parámetros (ARR,2,30)

$\mathbf{T}_4$

$\mathbf{T}_{4,1}$ Invocar al subalgoritmo Imprimir con los parámetros (ARR,1,29)

Lenguaje de Diseño:

```
ALGORITMO "PRINCIPAL"
COMENZAR
        ARR: arreglo[1..30] de entero
        INGRESO (ARR, 1, 30)
        IMPRIMIR (ARR, 2, 30)
        IMPRIMIR (ARR, 1, 29)
FIN
```

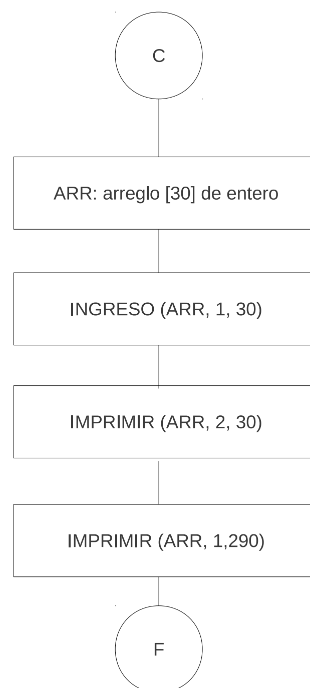


Figura 3.6:

Subalgoritmo Ingreso:

Versión 1:
$\mathbf{T}_1$ Definir e inicializar objetos
$\mathbf{T}_2$ Colocar 30 números enteros ingresados por el usuario en un arreglo

Versión 2:
$\mathbf{T}_1$
  $\mathbf{T}_{1,1}$ Declarar un objeto ARR que sea un arreglo de enteros como parámetro formal
  $\mathbf{T}_{1,2}$ Declarar un objeto LI de tipo entero como parámetro formal
  $\mathbf{T}_{1,3}$ Declarar un objeto LS de tipo entero como parámetro formal
$\mathbf{T}_2$
  Repetir mientras LI sea menor o igual a LS
  $\mathbf{T}_{2,1}$ Pedir al usuario que ingrese un número entero
  $\mathbf{T}_{2,2}$ Leer el número ingresado y guardarlo en la posición LI del arreglo ARR
  $\mathbf{T}_{2,3}$ Incrementar en 1 el objeto LI

Lenguaje de Diseño:

SUBALGORITMO “INGRESO” (**in out** A: arreglo de entero, **in** LI, LS:entero)
 COMENZAR
  I: entero
  I ⟵ LI
  MIENTRAS I <= LS **HACER**
   ESCRIBIR Ïngrese un nú mero entero”
   LEER A[I]
   I ⟵ I + 1
  FINMIENTRAS
 FIN

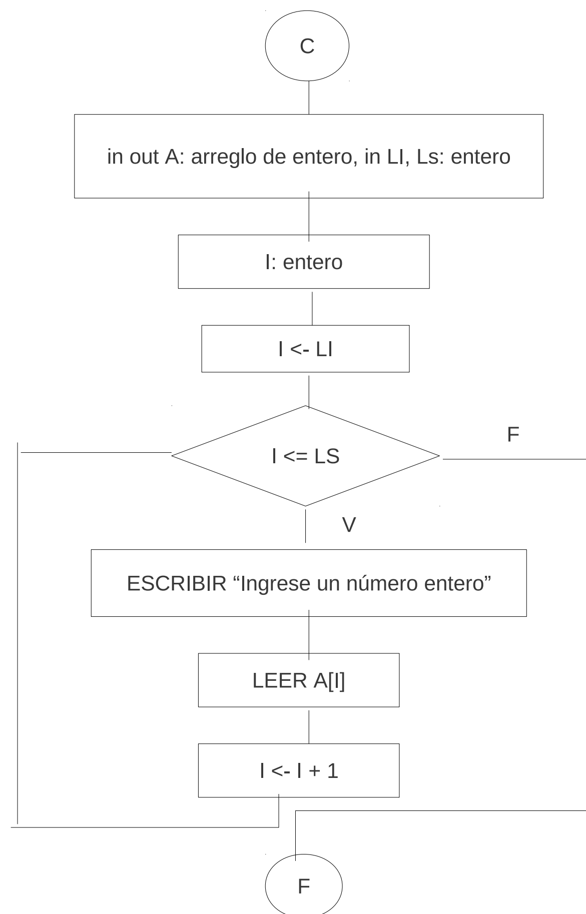


Figura 3.7:

Subalgoritmo Imprimir:

Versión 1:
$\mathbf{T}_1$ Definir e inicializar objetos
$\mathbf{T}_2$ Recorrer el arreglo teniendo en cuenta las posiciones indicadas por los parámetros e imprimirlas

Versión 2:
$\mathbf{T}_1$
  $\mathbf{T}_{1,1}$ Declarar un objeto A que sea un arreglo de enteros como parámetro formal
$\mathbf{T}_2$
  $\mathbf{T}_{2,1}$ Declarar un objeto LI de tipo entero como parámetro formal
$\mathbf{T}_3$
  $\mathbf{T}_{3,1}$ Declarar un objeto LS de tipo entero como parámetro formal
$\mathbf{T}_4$
  Repetir mientras LI sea menor o igual a 30
  $\mathbf{T}_{4,1}$ Mostrar el número de la posición LI del arreglo A
  $\mathbf{T}_{4,2}$ Incrementar en 2 el objeto LI

Lenguaje de Diseño:

```
SUBALGORITMO "IMPRIMIR" (in out A: arreglo de entero, in LI, LS:entero)
   COMENZAR
      I: entero
      PARA I DESDE li HASTA ls CON PASO 2 HACER
         ESCRIBIR "número:", A[I]
      FINPARA
   FIN
```

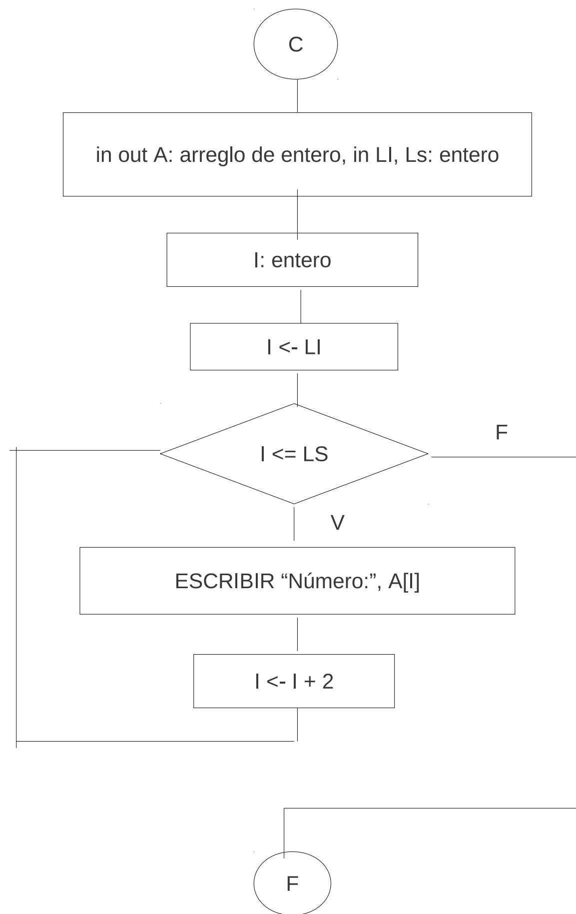


Figura 3.8: