Algoritmos y programación en pseudocódigo

Diego Fernando Duque
Yana Saint-Priest Velásquez
Patricia Segovia
Diego Fernando Loaiza

Editorial Tecnológica de Costa Rica

VIGILADA MINEDUCACIÓN
USC UNIVERSIDAD SANTIAGO DE CALI
EDITORIAL

ALGORITMOS Y PROGRAMACIÓN EN PSEUDOCÓDIGO

VIGILADA MINEDUCACIÓN

USC UNIVERSIDAD SANTIAGO DE CALI

EDITORIAL

ALGORITMOS Y PROGRAMACIÓN EN PSEUDOCÓDIGO

Autores

Diego Fernando Duque

Yana Saint-Priest Velásquez

Patricia Segovia

Diego Fernando Loaiza

Editorial Tecnológica de Costa Rica

Algoritmos y programación en pseudocódigo / Diego Fernando
Duque y otros. -- Bogotá : Universidad Santiago de Cali, 2017
312 páginas : tablas, gráficos ; 24 cm.
Incluye índice temático
1. Ingeniería de sistemas 2. Programación estructurada
3. Algoritmos (Computadores) I. Duque, Diego Fernando, autor.
005.133 cd 21 ed.
A1580914

CEP-Banco de la República-Biblioteca Luis Ángel Arango

Algoritmos y programación en pseudocódigo

1a. Edición 100 ejemplares
Cali, Colombia - 2017
ISBN: 978-958-8920-54-2

© Editorial Tecnológica de Costa Rica
Instituto Tecnológico de Costa Rica
Correo electrónico: editorial@itcr.ac.cr
www.editorialtecnologica.tec.ac.cr
Apdo. 159-7050, Cartago
Tel: (+506) 2550-2297 / 2550-2336
Fax: (+506) 2552-5354

Director de la Editorial
Dagoberto Arias Aguilar
Correo: darias@itcr.ac.cr

Comité Editorial
Maribel Jiménez Montero
Dagoberto Arias Aguilar
Tania Moreira Mora
Kattia Calderón Mora
Gustavo Rojas Moya
Luko Hilje Quirós
Esteban González Arguedas
Eddie Gómez S.

Coordinación Editorial
Edward Javier Ordóñez

Diagramación e impresión
Artes Gráficas del Valle S.A.S.
Tel. 333 2742

Distribución y Comercialización
Universidad Santiago de Cali
Publicaciones
Calle 5 No. 62 - 00
Tel: 518 3000, Ext. 405 - 489

Sugerencias y Comentarios al autor:
diego.duque01@usc.edu.co;
yana.saint-priest00@usc.edu.co;
pasego@usc.edu.co; diego.loaiza02@usc.edu.co

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE ILUSTRACIONES

PRÓLOGO

En el pasado reciente, la programación de computadores estaba supeditada a un conjunto de expertos, formados o no en la academia, que respondían por igual, al título de ingenieros y más específicamente de sistemas; las grandes compañías no discriminaban títulos, a un empírico le bastaba con mostrar que tenía conocimiento en el área y era aceptado, de esta forma se crearon grandes sistemas de información que impactaron e impactan aún sectores como el Bancario y Gobierno.

Con el desarrollo de la tecnología y el advenimiento de nuevos dispositivos "caseros" de fácil manipulación para el usuario y la presencia del computador en prácticamente todas las actividades, se creó un nuevo orden, en el cual, es imperante dominar estas herramientas y aún más, saberlas programar, indiferente de cual sea la profesión; de hecho, representa un plus para quien programe, conociendo su actividad o proceso de negocio, porque logra un mayor aprovechamiento de los recursos computacionales y enriquece su labor.

La programación ha sido considerada una actividad compleja y se han escrito e inventado múltiples metodologías para su enseñanza, involucrándola desde el trabajo escolar, e incluso preescolar. Lo cierto y comprobado, es que esta actividad se aprende, haciendo, practicando, involucrándola en el día a día, hasta volverla un hábito. No es fácil establecer cuál es el mejor método para aprender a programar, porque todos no aprenden de la misma forma, sin embargo, existen prácticas reconocidas por quienes han estado en la pedagogía que bien manejadas apoyan en gran forma al desarrollo de la lógica y la creatividad en la resolución de problemas.

En el presente libro, se condensa la experiencia adquirida en años de arduo trabajo, por los profesores Yana Elida Saint-Priest Velásquez y Diego Fernando Duque Betancourt, quienes han tenido a cargo esa compleja tarea de formar programadores Tecnólogos e Ingenieros. Ellos, conocedores de las necesidades de un estudiante en formación, presentan un libro de fundamentación básica en programación, en lenguaje sencillo, que permite al estudiante desarrollar la lógica requerida para adentrarse en este campo, ayudándolo a fortalecer sus propias habilidades, a partir de técnicas y estrategias que le permiten indagar y proponer soluciones a situaciones específicas del mundo real, con una metodología basada en problemas y en casos de estudio.

El libro consta de cuatro capítulos, que en adecuada secuencia le permiten al estudiante partir de lo básico a situaciones con mayor complejidad, cumpliendo con aprendizajes esperados en cada capítulo, pensado para estudiantes que inician sus estudios profesionales.

Los autores, profesores de la Facultad de Ingeniería de la Universidad Santiago de Cali, demuestran con esta obra, lo esfuerzos que desde el interior de la Universidad y sus programas realizan sus docentes para el mejoramiento de sus prácticas pedagógicas.

Para el Director del Programa de Ingeniería de Sistemas, es un honor haber sido invitado a participar con el prólogo de esta importante obra, más aún, considerando que los estudiantes del programa serán los usuarios directos de ella y que los docentes involucrados, son parte fundamental del mismo.

Se invita a lectores, técnicos o no, a dejarse tentar por la programación y acompañarse con obras como la presente, que hacen fácil este arte.

JORGE ENRIQUE LEON ARDILA
Director del Programa de Ingeniería en Sistemas
Director del Programa de Tecnología en Sistemas
Universidad Santiago de Cali.

INTRODUCCIÓN

¿Para qué aprender a programar? El mundo es cada vez más digital, en todas las actividades de las personas se encuentra presente un dispositivo digital, sea éste un computador, un teléfono celular, una Tablet, la televisión, en los carros, de igual forma se encuentra presente en las áreas tales como medicina, educación, gobierno, economía, mercado, arte, deporte, entre otros. Aprender a programar es adquirir una herramienta de pensamiento que permite desarrollar la imaginación, la creatividad, la capacidad analítica, ayuda a enfrentarte a grandes problemas y dividirlos en problemas más simples. Programar es entender cómo funciona una máquina, hablarle en su propio idioma e indicarle como dar la respuesta que se desea obtener. Hoy en día es prioritario aprender a programar, cualquiera que sea la profesión economista, ingeniero, músico, administrador, deportista, médico y afines, entre otros, en cualquier momento van a necesitar programar, este es el momento de empezar, para esto se presenta el libro Algoritmos y Programación, cuyo objetivo es precisamente acompañar al lector en el camino de aprender a programar.

La Universidad Santiago de Cali dentro de sus propósitos de formación tiene una apuesta hacia la formación integral de sus estudiantes en el campo tecnológico, para lo cual propone como uno de sus cursos transversales a Algoritmos y Programación, el cual encamina al futuro profesional de cualquier disciplina para alcanzar capacidades de orden superior tales como: la habilidad para analizar problemas, o situaciones problemáticas; la habilidad para proponer soluciones a dichas situaciones y la habilidad para aplicar creatividad en las soluciones propuestas.

Algoritmos y Programación en Pseudocódigo es una propuesta, por parte del equipo de profesores del área de Desarrollo de Software del Departamento de las Tecnologías de las Información y las Comunicaciones –TIC-, como un material de apoyo a las clases, y como acompañamiento al trabajo independiente de sus estudiantes, propiciando actividades de autoformación. Está dirigido a estudiantes en los cursos de inicio de carrera y se ha diseñado bajo la metodología basada en problemas y en casos de estudio, lo que permite al estudiante enfrentarse a situaciones reales, que conducen a alcanzar la competencia relacionada con el análisis de las situaciones problemáticas y la proposición de soluciones mediante el uso del computador, logrando de ésta forma un alto grado de motivación en el aprendizaje.

Algoritmos y Programación en Pseudocódigo presenta una serie de temas que tienen una secuencia en complejidad y profundidad de tal manera, que la competencia del curso que acompaña, se va adquiriendo a través de aprendizajes esperados en cada capítulo, los que se van complementando e incrementando a medida que se avanza en el contenido.

El libro está construido con una fundamentación básica de programación, lo que permite que el estudiante se apropie de un lenguaje técnico adecuado y de técnicas o estrategias que lo conducen a centrarse en el análisis de situaciones problemáticas o casos reales, frente a los cuales se realizan la búsqueda y proposición de soluciones que satisfagan acertadamente cada situación.

La estructura de cada capítulo consiste en la presentación de conceptos básicos; ejemplos con diferentes grados de complejidad; demostraciones que permiten visualizar en forma muy detallada los pasos a seguir en el análisis de las situaciones problemáticas y el planteamiento de posibles soluciones; ejercicios para que el estudiante practique, respuestas a algunos ejercicios planteados con el fin de que el estudiante tenga una referencia guía de su respuesta o solución propuesta y ejercicios sin respuesta que servirán para el trabajo en clase.

Algoritmos y Programación en Pseudocódigo consta de 4 capítulos: Tipos de Datos y Expresiones, en este capítulo se encuentran conceptos de datos, su representación y la forma de utilizarlos en el desarrollo de expresiones algorítmicas que sirven para la solución de problemas; Algoritmos y Pseudocódigo, se presenta este capítulo como una estrategia para solucionar problemas, que consiste en pasos precisos, finitos y definidos y, su escritura en pseudo-

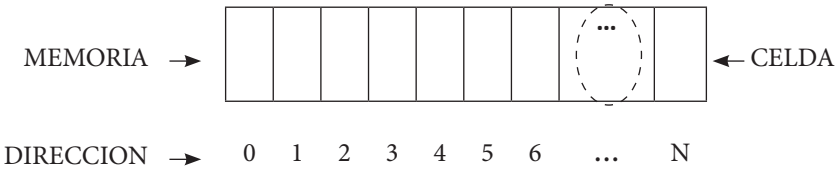
código como un lenguaje intermedio entre el lenguaje natural y el lenguaje de máquina; Instrucciones Condicionales, este capítulo presenta al lector la forma de construir soluciones alternas, a un mismo problema, cuya ejecución se encuentra sujeta al cumplimiento de una o más condiciones; y por último Instrucciones de Repetición, cuya necesidad se evidenciada en este capítulo cuando se ejemplifican problemas cuyas soluciones requieren realizar un conjunto de operaciones iguales más de una vez.

Capítulo 1

1. TIPOS DE DATOS Y EXPRESIONES

1.1 TIPOS DE DATOS

Todo dato que se utilice en un programa de computador debe ser almacenado en memoria. La memoria del computador está dividida en "pedazos" del mismo tamaño dentro de los cuales se puede guardar información. Cada "pedazo" es una celda y cada celda tiene asociada una dirección única en memoria que permite conocer su ubicación y acceder la información contenida en ella para consultarla, modificarla o borrarla.



Los datos se pueden clasificar en: **Simples** o **Estructurados**, según la cantidad de celdas que se utilicen para almacenarlos. Los tipos **Simples** utilizan una sola celda, los **Estructurados**, más de una dependiendo de la cantidad de datos a almacenar.

Los tipos de datos **simples** que existen en el pseudocódigo son:

1.1.1 Numéricos

Que a su vez se clasifican en:

a. Enteros
Ejemplo: 3, -3, 1234, 0, 6 -45

b. No Enteros o Número con Punto Decimal

Ejemplo: 3.5, -2.02, 4.3

Para abreviar de ahora en adelante serán llamados decimales.

Por tratarse de datos simples requieren de una sola celda para ser almacenados. Por ejemplo:

-3	-2.02	1234

1.1.2 Lógicos o Booleanos

Representan sólo dos valores: falso o verdadero, o en inglés false o true, que se abrevian con F y V.

Verdadero	Falso
True	False
V	F

1.1.3 Caracteres

Pueden ser una letra del alfabeto, un dígito o un símbolo especial (incluido el espacio en blanco), a todos estos se les conoce como símbolos alfanuméricos.

Ejemplo de letras:	**a, b, c; A, B, C**
Ejemplo de dígitos:	**0, 1, 2, 3, 4, 5, 6, 7, 8, 9**
Ejemplo de símbolos:	**+, /, *, ?, %, $, #, !, , ä, .**

Este tipo de datos se escribe entre comillas simples, por ejemplo:

‘a’
‘0’
‘ ’
‘W’

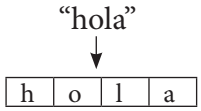
← **Todo dato alfanumérico usa comillas simples**

En este caso los dígitos entre comillas son totalmente diferentes a los datos tipo Numérico, por tratarse de caracteres no pueden ser empleados en operaciones aritméticas.

Entre comillas puede haber más de un carácter, a esto se le conoce como **cadena de caracteres** o simplemente **cadena**. Las cadenas se escriben entre comillas dobles en lugar de comillas simples, y son un tipo de dato **estructurado** porque requieren más de una celda. Por ejemplo:

"hola" "0987hn" "casa" "pedro"

La cantidad de celdas depende de la cantidad de caracteres que se encuentran entre comillas, por ejemplo:



Cuatro caracteres

Cuatro celdas

1.2 IDENTIFICADORES, CONSTANTES Y VARIABLES

En un programa de computador, las celdas no se reconocen por su dirección sino por un nombre o **identificador**. El identificador de la celda debe iniciar con una letra posterior a ella pueden existir más letras o números; el único símbolo permitido en un identificador es el guion bajo '_'.

Los siguientes son ejemplos correctos de identificadores:

Los siguientes son identificadores incorrectos:

En pseudocódigo los **identificadores** son sensibles a mayúsculas y minúsculas, lo que significa que Suma, suma y SUMA no son el mismo identificador, por lo tanto corresponden a tres celdas diferentes.

Según el comportamiento de la información almacenada en las celdas, éstas se pueden clasificar en dos tipos:

- **Constantes**: celdas cuyo valor nunca cambia durante la ejecución del programa.

- **Variables**: celdas cuyo valor cambia durante la ejecución del programa. Cuando esto ocurre, el valor viejo es reemplazado por el nuevo sin modificar el tamaño de la celda.

1.3 EXPRESIONES ARITMÉTICAS, RELACIONALES Y LÓGICAS

Toda expresión que se construya debe cumplir con:

*Operando*1 *Operador* *Operando*2

Donde *Operando* representa un dato y el *Operador,* el símbolo que representa la operación que se va a realizar.

1.3.1 Operadores Aritméticos

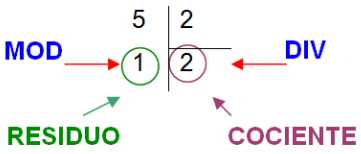
Son utilizados para construir expresiones aritméticas, los operandos son datos de tipo numéricos y el resultado obtenido también es numérico. En pseudocódigo existen los siguientes operadores aritméticos, ver Tabla 1:

Tabla 1. Operadores Aritméticos

Operador	Operación	Ejemplo	Resultado
**	Potencia	3**3	27
*	Multiplicación	3*3	9
/	División	3/2	1.5
MOD	Módulo	5 **MOD** 2	1
DIV	División entera	5 **DIV** 2	2
+	Suma	5 + 2	7
-	Resta	5 - 2	3

Fuente: Autores

Los operadores **MOD** y **DIV** son utilizados para obtener el residuo y el cociente respectivamente en una operación de división pero, a diferencia del operador **/**, cuando se usa **MOD** o **DIV**, la división es entera, es decir, nunca el resultado será un número decimal. Por ejemplo:



Los operadores aritméticos tienen una **precedencia** que indica cuál de ellos debe resolverse primero. Ejemplos:

```
3 + 5 * 2      primero se resuelve la multiplicación, luego la suma.
7 / 2 ** 4     primero se resuelve la potencia, luego la división.
```

Adicionalmente los operadores pueden ser **asociativos** por la derecha o por la izquierda, lo que significa que si en una expresión aritmética todos los operadores son el mismo, se resolverán de izquierda a derecha en orden o viceversa, dependiendo del operador. Ejemplos:

```
3 + 4 + 5      primero se suma 3 + 4 al resultado se le suma el 5.
2 ** 3 ** 6    el 2 se eleva al resultado obtenido de elevar 3 a  la 6 primero.
```

Una expresión aritmética puede contener paréntesis, lo cual modifica la precedencia y/o asociación de los operadores. Los paréntesis deben resolverse primero. Ejemplos:

```
(3 + 5) * 2    primero se resuelve la suma, luego la multiplicación.
2 ** (6 / 3)   primero se resuelve la división, luego la potencia.
```

La Tabla 2 resume los operadores aritméticos, precedencias y asociaciones:

Tabla 2. Operador, Precedencia y Asociación

OPERADOR	PRECEDENCIA	ASOCIACIÓN
( )	Mayor	
**	↓	Por la derecha
* , /, MOD, DIV		Por la izquierda
+, -	Menor	Por la izquierda

Fuente: Autores.

Existen ecuaciones matemáticas que incluyen otros operadores que pueden ser convertidas en expresiones algorítmicas usando los operadores aritméticos mencionados. Por ejemplo: $\sqrt[n]{A}$ que es equivalente a $A^{1/n}$ algorítmicamente queda:

$$\sqrt[n]{A} \quad \rightarrow \quad A**\left(1/n\right)$$

$$\sqrt{A} \quad \rightarrow \quad A**\left(1/2\right)$$

Aritmética *Algorítmica*

Los paréntesis son necesarios para que A quede elevado a 1 sobre n, sino primero se elevaría A a La 1 y luego el resultado se dividiría entre n.

Y la expresión:

$$A^{-n} = \frac{1}{A^n}$$

Puede escribirse algorítmicamente así:

A^{-n}	$\rightarrow$	$1/A**n$
Aritmética		*Algorítmica*

En el pseudocódigo existen algunas operaciones que también pueden ser resueltas directamente, sin necesidad de conversión, a este tipo de operaciones se les conoce como funciones.

La función raíz cuadrada en el pseudocódigo tiene el siguiente formato:

Raíz ( )

Dentro de los paréntesis se coloca el valor al cuál se le desea sacar la raíz cuadrada, puede ser una expresión o un número. Por ejemplo:

$\sqrt{A}$	$\rightarrow$	$Raiz(A)$
$\sqrt{A}$	$\rightarrow$	$Raiz(A)$
$\sqrt{25}$	$\rightarrow$	$Raiz(25)$
$\sqrt{(b^2 - 4ac)}$	$\rightarrow$	$Raiz(b**2 - 4*a*c)$
Aritmética		*Algorítmica*

Entonces una misma expresión aritmética se puede convertir a algorítmica de dos maneras:

$$\sqrt{(b^2 - 4a\ )} \quad \rightarrow \quad (b**2 - 4*a*c)**\left(1/2\right)$$

$$\sqrt{(b^2 - 4a\ )} \quad \rightarrow \quad Raiz(b**2 - 4*a*c)$$

Aritmética *Algorítmica*

Recuerde que la función **Raíz** se puede usar únicamente para obtener la raíz cuadrada y que las letras que no tienen comillas simples son nombres de variables.

Las funciones trigonométricas como seno, coseno y tangente tienen su representación en el pseudocódigo así:

Sen ( ) **Cos ( )** **Tan ( )**

Y dentro de los paréntesis se coloca el número respectivo o la expresión.

Entonces el Seno de Θ se puede obtener usando Sen(Θ) o calculando:

$$Sen\Theta = \frac{CatetoOpuesto}{Hipotenusa}$$

1.3.2 Operadores Relacionales

Son utilizados para construir expresiones relacionales o de comparación, estas expresiones incluyen el manejo de todos los tipos de datos y el resultado obtenido es Verdadero o Falso. En el Pseudocódigo existen los siguientes operadores relacionales, ver Tabla 3:

Tabla 3. Operadores Relacionales

Operador	Operación	Tipo de Datos
=	Igual que	Numéricos, Lógicos y Caracteres, Cadena
<>	Diferente	
>=	Mayor o igual que	Numéricos, Caracteres
<=	Menor o igual que	
>	Mayor que	
<	Menor que	

Fuente: Autores

Ejemplos:

Expresión	Resultado	Explicación
"hola" = "Hola"	F	Es F porque los caracteres mayúsculas y minúsculas son diferente
F <> V	V	Es V porque falso y verdadero son diferentes
3 <= 2	F	Es F porque 3 es mayor que 2
5 < 12	V	Es V porque 5 es menor que 12
'a' >= 'a'	V	Es V porque aunque 'a' no es mayor que 'a' son iguales

1.3.3 Operadores Lógicos

Son utilizados para construir expresiones lógicas, estas expresiones incluyen el manejo de sólo los tipos de datos lógicos o booleanos y el resultado obtenido es Verdadero o Falso. Dentro de estos operadores existe uno solo que no necesita de dos operandos, el **NOT,** lo que lo convierte en un operador unario. En el Pseudocódigo existen los siguientes operadores lógicos, ver Tabla 4:

Tabla 4. Operadores Lógicos

Operador	Operación	Ejemplo	Resultado
AND	Conjunción	V **AND** F	F
OR	Disyunción	F **OR** F	F
NOT	Negación	**NOT** F	V

Fuente: Autores

A continuación se presenta el resultado de la aplicación de los tres operadores lógicos, por medio de sus tablas de verdad, ver Tabla 5:

Tabla 5. Operadores Lógicos

A	NOT A
F	V
V	F

A	B	A AND B	A OR B
F	F	F	F
F	V	F	V
V	F	F	V
V	V	V	V

Fuente: Autores.

Los operadores lógicos, como los aritméticos, tienen precedencia de operadores y son asociativos por la izquierda.

OPERADOR	PRECEDENCIA	ASOCIACIÓN
( )	**Mayor**	
NOT	↓	Por la derecha
AND	↓	Por la izquierda
OR	**Menor**	Por la izquierda

Por ejemplo:

NOT ((V OR F) AND V)
Primero se resuelve el paréntesis interno, quedando NOT (V AND V).
Luego se resuelve el siguiente paréntesis, quedando NOT V.
Por último se aplica la negación, dando como resultado final F.

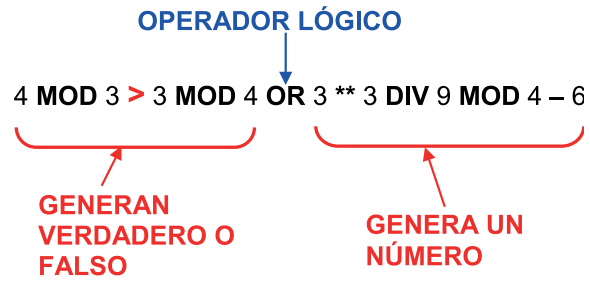
Una expresión puede contener al mismo tiempo operadores de cualquiera de los tres tipos, en la Tabla 6 se resume la precedencia de todos los operadores:

Tabla 6. Precedencia de los Operadores

Operador	Jerarquía
( )	**Mayor**
******	↓
***, /, MOD, DIV**	↓
+, -	↓
>, <, >=, <=, =, <>	↓
NOT	↓
AND	↓
OR	**Menor**

Fuente: Autores.

Recuerde que los operandos con los que trabaja todo operador lógico son los tipo lógico, es decir, Verdadero o Falso, por tanto, si en una expresión se tienen operadores lógicos para resolverla observe los operandos – o el conjunto de operaciones – que acompañan a estos operadores y determine si éstos son valores – o generan valores – lógicos si no es así, la expresión no se puede resolver, por ejemplo:



La anterior expresión no se puede resolver ya que el operador lógico **OR**, no tiene a su lado derecho, un operando lógico.

1.4 EJERCICIOS

1.4.1 Ejercicios con Respuesta

1. Resuelva las siguientes expresiones aritméticas:
   - **a.** 2 * 6 ** 3 / 6
   - **b.** 8.0 / (6.5 + 13.2) ** (48.2 – 69.7 + 23.5)
   - **c.** *Con explicación* → ((15 * 8 **MOD** 6 + 24 **DIV** 2 ** 3) ** 3 / 4) * (5 ** 1 / 2 + 1 / 4 + 2 ** 3 – 4)

2. Resuelva las siguientes expresiones relacionales:
   - **a.** 90485 >= 39493
   - **b.** 'y' <> 'Y'
   - **c.** "María" == "María"

3. Resuelva las siguientes expresiones lógicas:
   - **a.** V OR F AND V
   - **b.** V AND F OR V OR NOT V
   - **c.** NOT (V OR F) AND (F AND V)

4. Resuelva las siguientes expresiones que incluye todos los operadores:

   a. 64 + 25 > 65 * 60 OR 30 / 15 < 6 **3
   b. *Con explicación* → **(**2 ** 3 **DIV** 2 **MOD** 5 **>** 1 **MOD** 4 ** 2**)** **OR NOT (((**15 * 8 **MOD** 6 **-** **(**24 **DIV** 2 ** 3**)** **3 - 7**)** ** 3 + 4**)** <>** 5**)**

5. *Con explicación* → Convierta la siguiente expresión aritmética a algorítmica:

$$Y = \frac{\left(\sqrt{C^{-3}P^{4}/\left(2^{J}Q^{-2}\right)}\right)^{n}}{G^{3}P^{2}}$$

1.4.2 Ejercicios sin Respuesta

6. Resuelva las siguientes expresiones aritméticas:

   a. 29.7 + 5.0 ** 2.0
   b. ((2 - 3) ** 4 * 5 / (4 + 3 * 9))
   c. 49.38 + 127.73 - 15.02 * 6.83 / 3.22

7. Resuelva las siguientes expresiones relacionales:

   a. 'A' >= 'Z'
   b. 429 = 429.0
   c. "pseudocódigo" <> "pseudocódigo"

8. Resuelva las siguientes expresiones lógicas:

   a. V OR V AND F OR V
   b. NOT ((F OR F) AND (V OR V) OR V)
   c. NOT F AND (F OR V) AND (NOT V AND F)

9. Resuelva las siguientes expresiones que incluye todos los operadores:

   a. 3 * 5 > 230 / 12 AND F
   b. "Pedro" <> "PedrO" OR (3 + 56 * 2 / 5) > 10 AND NOT V

1. Convierta la siguiente expresión aritmética a algorítmica:

$$Y = \sqrt[3]{\frac{A}{4}} \times \frac{\sqrt{C}}{A**B}$$

1.4.3 Respuesta a los Ejercicios

1. Resuelva las siguientes expresiones aritméticas:

a. 2 * 6 ** 3 / 6
Respuesta:
2 * 6 ** 3 / 6
= 2 * 216 / 6
= 432 / 6
= 72

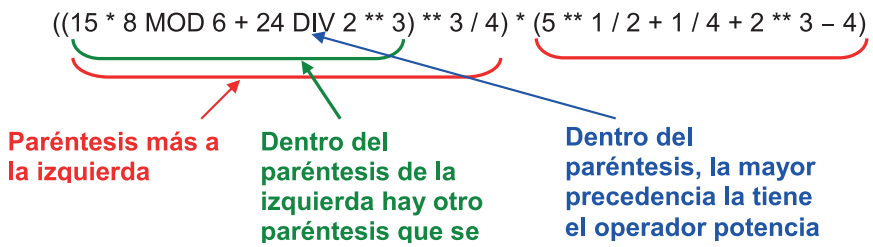
b. 8.0 / (6.5 + 13.2) ** (48.2 – 69.7 + 23.5)
Respuesta:
8.0 / (6.5 + 13.2) ** (48.2 – 69.7 + 23.5)
= 8.0 / 9.7 ** (48.2 – 69.7 + 23.5)
= 8.0 / 9.7 ** (-21.5 + 23.5)
= 8,0 / 9,7 ** 2
= 8,0 / 94,09
= 0.09

c. Con explicación → ((15 * 8 **MOD** 6 + 24 **DIV** 2 ** 3) ** 3 / 4) * (5 ** 1 / 2 + 1 / 4 + 2 ** 3 – 4)
Respuesta:
((15 * 8 MOD 6 + 24 DIV 2 ** 3) ** 3 / 4) * (5 ** 1 / 2 + 1 / 4 + 2 ** 3 – 4)

Para resolver cualquier expresión aritmética primero comience identificando los operadores que existen en la expresión, entonces

((15 * 8 MOD 6 + 24 DIV 2 ** 3) **3 / 4) * (5 ** 1 / 2 + 1 / 4 + 2 ** 3 – 4)

Una vez identificados los operadores se aplica la precedencia para resolver la expresión. Como el ejemplo tiene varios paréntesis, se comienza con el que se encuentre más a la izquierda.



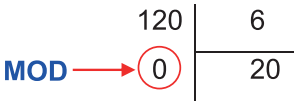
Al resolver la potencia queda:

((15 * 8 MOD 6 + 24 DIV 8) ** 3 / 4) * (5 ** 1 / 2 + 1 / 4 + 2 ** 3 – 4)

Dentro del paréntesis quedan tres operadores con el mismo nivel de precedencia, aplicando la regla, se comienza con el que se encuentre más a la izquierda – la multiplicación –, por lo que se resuelve primero la multiplicación, entonces:

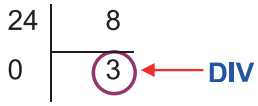
((120 MOD 6 + 24 DIV 8) ** 3 / 4) * (5 ** 1 / 2 + 1 / 4 + 2 ** 3 – 4)

Se resuelve el **MOD**



((0 + 24 DIV 8) ** 3 / 4) * (5 ** 1 / 2 + 1 / 4 + 2 ** 3 – 4)

Luego el **DIV**



((0 + 3) ** 3 / 4) * (5 ** 1 / 2 + 1 / 4 + 2 ** 3 – 4)

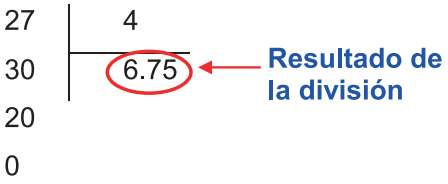
Por último se resuelve el operador suma, y se destruye el paréntesis más interno, quedando:

(3 ** 3 / 4) * (5 ** 1 / 2 + 1 / 4 + 2 ** 3 – 4)

Quedan dos paréntesis, nuevamente se selecciona el de la izquierda, dentro de éste se resuelve primero el operador potencia, quedando:

(27 / 4) * (5 ** 1 / 2 + 1 / 4 + 2 ** 3 – 4)

Se resuelve la división y así se destruye el paréntesis



6.75 * (5 ** 1 / 2 + 1 / 4 + 2 ** 3 – 4)

En el paréntesis existen dos operadores potencia por lo que se comienza resolviendo el de la izquierda, entonces:

6.75 * (5 / 2 + 1 / 4 + 2 ** 3 – 4)

Luego se resuelve la siguiente potencia

6.75 * (5 / 2 + 1 / 4 + 8 – 4)

En la expresión aritmética hay dos operadores /, se resuelve primero el de la izquierda, entonces

6.75 * (2.5 + 1 / 4 + 8 – 4)

Luego la siguiente división

$$6.75 * (2.5 + 0.25 + 8 - 4)$$

Quedan tres operadores con la misma precedencia, se resuelve primero la suma que se encuentra más a la izquierda

$$6.75 * (2.75 + 8 - 4)$$

La siguiente suma

$$6.75 * (10.75 - 4)$$

Después la resta y al resolver ésta se destruye el paréntesis quedando

$$6.75 * 6.75$$

Por último la multiplicación y el resultado final de la expresión aritmética es:

45.5625

2. Resuelva las siguientes expresiones relacionales:

a. 90485 >= 39493
Respuesta:
TRUE

b. 'y' <> 'Y'
Respuesta:
TRUE

c. "María" == "Maria"
Respuesta:
FALSE

3. Resuelva las siguientes expresiones lógicas:

a. V OR F AND V
Respuesta:
V OR F AND V
= V OR F
= V

b. V AND F OR V OR NOT V
Respuesta:
V AND F OR V OR NOT V
= V AND F OR V OR F
= F OR V OR F
= V OR F
= V

c. NOT (V OR F) AND (F AND V)
Respuesta:
NOT (V OR F) AND (F AND V)
= NOT V AND (F AND V)
= F AND F
= F

4. Resuelva las siguientes expresiones que incluye todos los operadores:

a. 64 + 25 > 65 * 60 OR 30 / 15 < 6 **3
Respuesta:
64 + 25 > 65 * 60 OR 30 / 15 < 6 **3
= 89 > 65 * 60 OR 30 / 15 < 6 **3
= 89 > 3900 OR 30 / 15 < 6 **3
= 89 > 3900 OR 2 < 6 **3
= 89 > 3900 OR 2 < 216
= F OR 2 < 216
= F OR V
= V

b. Con explicación → **(**2 ** 3 **DIV** 2 **MOD** 5 **>** 1 **MOD** 4 ** 2**)** **OR NOT (((**15 * 8 **MOD** 6 **-** **(**24 **DIV** 2 ** 3**)** **3 **-** 7**)** ** 3 + 4**)** **<>** 5**)**
Respuesta:

(2 ** 3 **DIV** 2 **MOD** 5 **>** 1 **MOD** 4 ** 2) **OR NOT** (((15 * 8 **MOD** 6 **-** (24 **DIV** 2 ** 3) **3 **-** 7) ** 3 **+** 4) **<>** 5)

Primero la potencia

(2 ** 3 **DIV** 2 **MOD** 5 **>** 1 **MOD** 4 ** 2) **OR NOT** (((15 * 8 **MOD** 6 **-** (24 **DIV** 2 ** 3) **3 **-** 7) ** 3 **+** 4) **<>** 5)

(8 **DIV** 2 **MOD** 5 **>** 1 **MOD** 4 ** 2) **OR NOT** (((15 * 8 **MOD** 6 **-** (24 **DIV** 2 ** 3) **3 **-** 7) ** 3 **+** 4) **<>** 5)

Luego la otra potencia

Va el DIV

(8 **DIV** 2 **MOD** 5 > 1 **MOD** 16) **OR NOT** (((15 * 8 **MOD** 6 **-** (24 **DIV** 2 ** 3) **3 **-** 7) ** 3 **+** 4) **<>** 5)

Sigue el MOD

(4 **MOD** 5 **>** 1 **MOD** 16) **OR NOT** (((15 * 8 **MOD** 6 **-** (24 **DIV** 2 ** 3) **3 **-** 7) ** 3 **+** 4) **<>** 5)

Sigue el MOD

(4 **>** 1 **MOD** 1) **OR NOT** (((15 * 8 **MOD** 6 **-** (24 **DIV** 2 ** 3) **3 **-** 7) ** 3 **+** 4) **<>** 5)

Sigue el >, y se destruye el paréntesis

(4 **>** 1) **OR NOT** (((15 * 8 **MOD** 6 **-** (24 **DIV** 2 ** 3) **3 **-** 7) ** 3 **+** 4) **<>** 5)

V OR NOT (((15 * 8 **MOD** 6 **-** (24 **DIV** 2 ** 3) **3 **-** 7) ** 3 **+** 4) **<>** 5)

Primero paréntesis interno

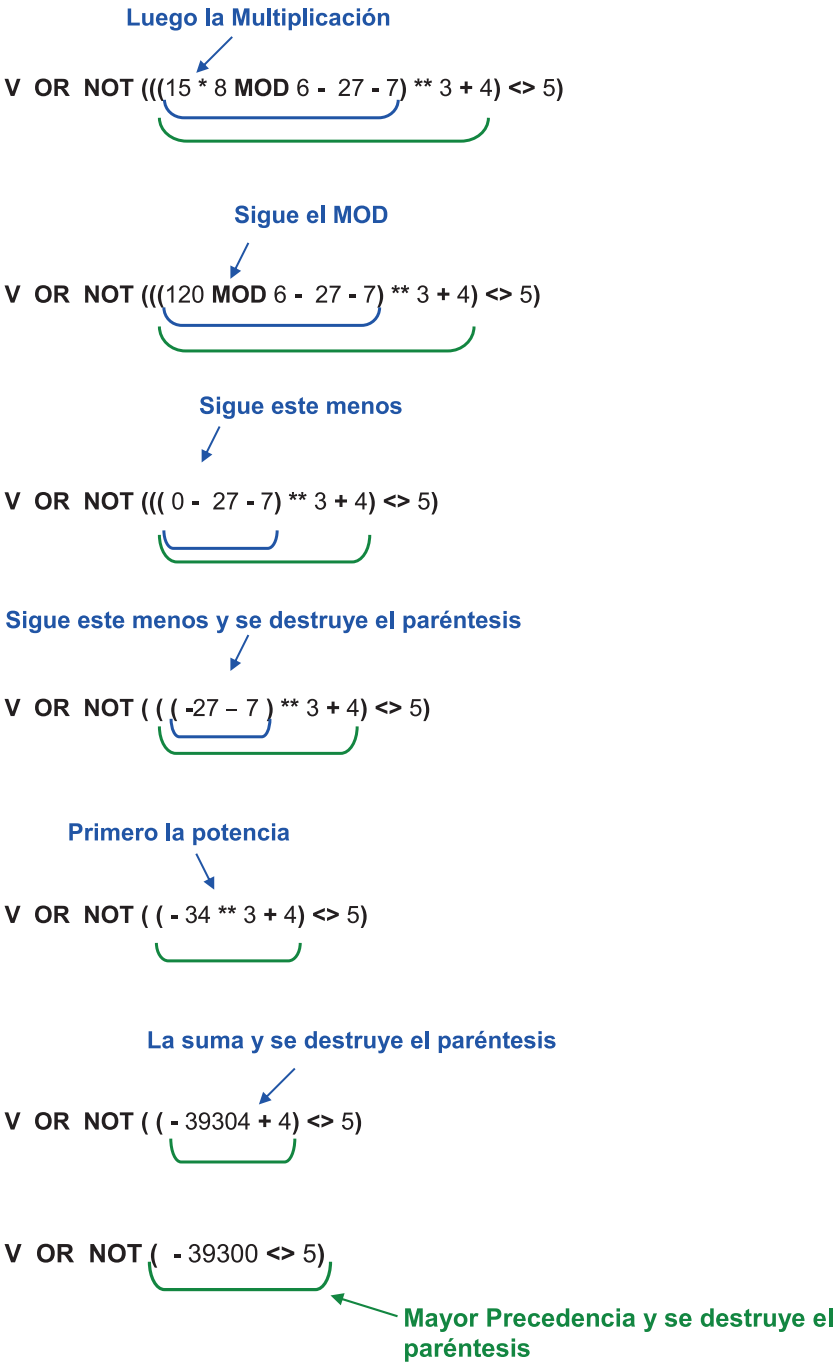
Paréntesis más interno, primero la potencia

Va el DIV y se destruye el paréntesis

V OR NOT (((15 * 8 **MOD** 6 **-** (24 **DIV** 8) **3 **-** 7) ** 3 **+** 4) **<>** 5)

Primero la potencia

V OR NOT (((15 * 8 **MOD** 6 **-** 3 ** 3 **-** 7) ** 3 **+** 4) **<>** 5)


Luego la Multiplicación
V OR NOT (((15 * 8 MOD 6 - 27 - 7) ** 3 + 4) <> 5)
Sigue el MOD
V OR NOT (((120 MOD 6 - 27 - 7) ** 3 + 4) <> 5)
Sigue este menos
V OR NOT (((0 - 27 - 7) ** 3 + 4) <> 5)
Sigue este menos y se destruye el paréntesis
V OR NOT (((-27 – 7) ** 3 + 4) <> 5)
Primero la potencia
V OR NOT ((- 34 ** 3 + 4) <> 5)
La suma y se destruye el paréntesis
V OR NOT ((- 39304 + 4) <> 5)
V OR NOT (- 39300 <> 5)
Mayor Precedencia y se destruye el paréntesis

V OR NOT V

V OR F

V RESULTADO

5. Con explicación → Convierta la siguiente expresión aritmética a algorítmica:

$$Y = \frac{\left(\sqrt{C^{-3}P^{4} \Big/ \left(2^{J}Q^{-2}\right)}\right)^{n}}{G^{3}P^{2}}$$

Respuesta:

Para resolverlo se pude tomar cada elemento que integra la expresión y convertirlo a su respectiva representación en algorítmica, entonces:

$$C^{-3} \rightarrow 1 \Big/ C**3 \qquad P^{4} \rightarrow P**4$$

$$2^{J} \rightarrow 2**J \qquad Q^{-2} \rightarrow 1 \Big/ Q**2$$

$$G^{3} \rightarrow G**3 \qquad P^{2} \rightarrow P**2$$

La expresión algorítmica queda:

$$Y \leftarrow \left(\left(\left(1 \Big/ C**3 \;*\; P**4\right) \Big/ \left(2**J \;*\; 1 \Big/ Q**2\right)\right)**\left(1 \Big/ 2\right)\right)**n \Big/ \left(G**3 \;*\; P**2\right)$$

Capítulo 2

2. ALGORITMOS Y PSEUDOCODIGOS

2.1 CARACTERÍSTICAS DE UN ALGORITMO

Un Algoritmo es una secuencia de pasos para resolver un problema y debe contar con las siguientes características:

Preciso: cada paso debe ser claro y exacto en su construcción para que así determine puntualmente lo que se desea hacer.

Definido: toda vez que se ejecute el algoritmo con los mismos datos de entrada, éste debe generar el mismo resultado.

Finito: todo algoritmo debe tener un fin.

El algoritmo se construye usando palabras del idioma y debe poder ser entendido por cualquier persona. Cuando se utilizan líneas de código se crea un programa y éstos se escriben usando lenguajes de programación. Si se emplea cuasi código en la construcción del programa, se tiene un Pseudocódigo.

Un algoritmo se construye en forma general usando los siguientes pasos:

1. Inicio.
2. Capturar, conocer o ingresar todos los datos que permitan resolver el problema.
3. Resolver, calcular o llevar a cabo los procesos con los datos capturados. Se sugiere en este punto colocar las ecuaciones necesarias.
4. Mostrar, visualizar o imprimir todos los resultados que el problema exija.
5. Fin.

El siguiente ejemplo sencillo muestra un error clásico en la elaboración de un algoritmo:

> Construya un algoritmo que encuentre y muestre el valor del área de un rectángulo.

PASOS	
1	Conocer los valores de altura y base
2	Calcular el área
3	Mostrar el valor del área hallado

El algoritmo no es preciso en sus pasos porque no indica a que figura geométrica hay que calcularle el área, por lo tanto no se puede identificar la fórmula a aplicar. Además, para mostrar que un algoritmo termina es conveniente incluir como último paso la palabra fin.

El algoritmo para resolver el ejemplo anterior sería entonces:

PASOS	
1	Inicio
2	Conocer los valores de altura y base del rectángulo
3	Calcular el área del rectángulo
4	Mostrar el valor del área del rectángulo
5	Fin

2.2 TIPOS DE INSTRUCCIONES

A partir de los algoritmos se pueden elaborar programas y para ello simplemente basta con "convertir" los pasos del algoritmo a sus instrucciones respectivas según el lenguaje de programación que se esté utilizando pero, como existen diferentes lenguajes, inicialmente se construirán los programas usando una de las herramientas de programación: el Pseudocódigo.

El Pseudocódigo se construye con instrucciones que no son específicas de un lenguaje de programación pero que reflejan un comportamiento similar; al tener un programa en pseudocódigo se puede convertir éste con mayor facilidad a un lenguaje de programación.

Las siguientes son algunas de las instrucciones que hacen parte del pseudocódigo.

2.2.1 Instrucción Inicio / Fin

Se utilizan para dar comienzo y terminación al pseudocódigo, por cada programa escrito debe existir un solo **Inicio** y **Fin**. A continuación se muestran como se escriben las instrucciones en el pseudocódigo y a su vez, su representación en la segunda herramienta de programación: el diagrama de flujo, ver Tabla 7.

Tabla 7. Representación de las instrucciones Inicio y Fin en el diagrama de flujo.

PSEUDOCÓDIGO Instrucción	DIAGRAMA DE FLUJO
Inicio	
Fin	

Fuente: Autores

Inicio y **Fin** son Palabras Reservadas, es decir, palabras que pueden ser usadas únicamente para cumplir una función específica, por tanto, éstas no pueden ser usadas como identificadores.

2.2.2 Instrucción de Asignación

Esta instrucción permite representar las operaciones aritméticas en el mundo del pseudocódigo.

Siempre que se desee realizar un procedimiento, operación o cálculo – con operadores aritméticos, relacionales y/o lógicos –, se debe utilizar una instrucción de Asignación.

En el mundo de las matemáticas se puede encontrar una expresión como la siguiente:

$$\mathbf{Y = X + 2}$$

En donde para saber el valor de Y primero se debe conocer a X y luego llevar a cabo la operación de suma.

En el mundo del pseudocódigo la anterior expresión se convierte a:

El símbolo Flecha representa la instrucción de Asignación.

Para conocer el valor de Y primero se debe resolver el lado derecho de la instrucción. En forma general se interpreta toda instrucción de Asignación de la siguiente manera:

"El resultado del lado derecho de la flecha se almacena, guarda o asigna en el lado izquierdo".

1. Las letras al lado **izquierdo** de la asignación, como en el ejemplo anterior la **Y**, representan un identificador.
2. Las letras al lado **derecho** de la asignación puede representar dos cosas:

   a. Si no llevan comillas, como en el ejemplo anterior la **X**, son un identificador.
   b. Si llevan comillas (simples o dobles), son un dato tipo carácter. En el ejemplo: **Y ← 'X',** la letra **X** se almacena en **Y**.

Para poder emplear identificador dentro de una expresión, éste debe:

1. Estar declarado: lo que significa que el computador sabe que existe un espacio de memoria con ese nombre y conoce el tipo de dato que se almacenará en él.
2. Estar inicializado: es necesario que los identificadores del lado derecho tengan un valor inicial para emplearlo en la expresión. Por lo tanto la secuencia de instrucciones correcta sería:

X ← 3
Y ← X + 2

3. Corresponder en tipo de dato: tanto los identificadores de la derecha como los de la izquierda deben almacenar datos del mismo tipo.

Ejemplos:

INSTRUCCIÓN	REPRESENTACIÓN DE LA EJECUCIÓN Ejecución de las instrucciones	Id
	X y **Y** son de tipo entero	
X ← 3	Al ejecutarse se almacena en **X** el valor de 3, así:	X 3
Y ← X + 2	El contenido de la celda **X** se suma con el número 2 y se almacena en **Y**, entonces en **Y** se guarda el resultado, el número 5, así:	Y 5

INSTRUCCIÓN	REPRESENTACIÓN DE LA EJECUCIÓN Ejecución de las instrucciones	Id
	X y **Y** son de tipo decimal	
X ← 3	Al ejecutarse se almacena en **X** el valor de 3, así:	X 3
Y ← X + 2	El contenido de la celda **X** se suma con el número 2 y se almacena en **Y**, entonces en **Y** se guarda el resultado, el número 5, así:	Y 5
Y ← 2.89	Al ejecutarse se almacena en **Y** el valor de 2.89 esto ocasiona que el anterior valor de **Y** (5) se pierda y sea reemplazado por uno nuevo (2.89)	Y 2.89
	Note en este ejemplo que se manejan datos enteros y decimales en la misma variable. En la ejecución de las instrucciones no se genera error ya que los números decimales contienen a los números enteros.	

INSTRUCCIÓN	REPRESENTACIÓN DE LA EJECUCIÓN Ejecución de las instrucciones	Id
	X es de tipo entero	
X ← 3	Al ejecutarse se almacena en **X** el valor de 3, así:	X 3
X ← "2"	Al ejecutarse se genera un error, ya que el tipo de dato ahora es carácter y la variable sólo recibe números enteros	

INSTRUCCIÓN	REPRESENTACIÓN DE LA EJECUCIÓN Ejecución de las instrucciones	Id
	T es de tipo cadena	
T ← "hola"	Al ejecutarse se almacena en **T** el dato "hola". Note como el identificador **T** está conformada por 4 celdas para poder almacenar la palabra	T h o l a
T ← "ola"	El contenido de la celda **T** se destruye y se guarda la palabra "ola"	T o l a
T ← "holas"	No se puede ejecutar ya que el dato "holas" está conformado por 5 letras y **T** representa sólo 4 celdas	

INSTRUCCIÓN	REPRESENTACIÓN DE LA EJECUCIÓN Ejecución de las instrucciones	Id
X ← 7	**X** es de tipo entero. Al ejecutarse se almacena en **X** el valor de 7, así:	X 7
Y ← '8'	**Y** es de tipo carácter. Al ejecutarse se almacena en **Y** el valor de '8', así:	Y 8
Z ← X + Y	Esta instrucción no se puede ejecutar en el mundo del pseudocódigo, porque no se pueden sumar enteros y caracteres	

La instrucción de Asignación utiliza la figura del rectángulo para su representación en el diagrama de flujo, en la Tabla 7 se muestran algunos ejemplos:

Tabla 7. Representación de la instrucción de Asignación en el diagrama de flujo.

PSEUDOCÓDIGO Instrucción	DIAGRAMA DE FLUJO
T ← "hola"	T ← "hola"
X1 ← (- b + Raíz (b**2 - 4*a*c)) / (2*a)	X1 ← (- b + Raíz (b**2 - 4*a*c)) / (2*a)
Suma ← Suma + nota	Suma ← Suma + nota

Fuente: Autores

2.2.3 Instrucción de Lectura

La gran mayoría de problemas necesitan para ser resueltos, datos ingresados por el usuario y éstos deben ser capturados por el programa, en el mundo del pseudocódigo existe una instrucción que permite hacer esto, la instrucción **Leer**. Entonces **Leer** es una palabra reservada.

La instrucción **Leer** tiene el siguiente formato general en el pseudocódigo

Dentro del paréntesis se debe colocar el identificador en donde se guardará la información a capturar. Para poder ejecutar una instrucción **Leer** el usuario debe ingresar el dato, el computador lo captura y automáticamente lo almacena en la celda correspondiente.

La instrucción **Leer** puede construirse de dos maneras:

1. **Leer** (A)

   Cuando se ejecuta esta instrucción, en **A** se almacena el dato ingresado por el usuario.

2. Leer (a, b, c)

Si se desea capturar más de un dato a la vez, se debe colocar dentro de los paréntesis, separados por comas, una cantidad de identificadores igual a la cantidad de datos a capturar.

En el ejemplo, **a**, **b** y **c** son tres identificadores, por tanto, se necesitan ingresar tres datos.

Toda instrucción **Leer** debe cumplir con lo siguiente:

1. Lo(s) identificador(es) que se encuentre(n) dentro de los paréntesis, debe(n) estar declaradas.
2. Debe existir coincidencia en el tipo de dato que se va a capturar con el tipo de dato que puede almacenar la celda que se ha colocado dentro de los paréntesis.

3. Cuando se tiene más de una celda dentro de los paréntesis de la instrucción **Leer**, se debe tener mucho cuidado en cómo se le sugiere al usuario el ingreso de los datos para que cuando los ingrese, éstos coincidan con la cantidad de celdas y el tipo de dato que puede guardar cada una de ellas.
   Por ejemplo: si se quiere capturar los datos de peso, edad y altura en cm de una persona se construye la instrucción **Leer** de la siguiente manera – para el ejemplo se asume que todas las celdas son de tipo entero –:

   Leer (peso, edad, altura)

   Si una persona tiene: Peso 60 kilos
   Edad 26 años
   Altura 150 cm

   Pero los digita de la siguiente manera:

   60 150 26

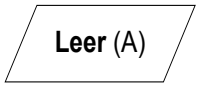
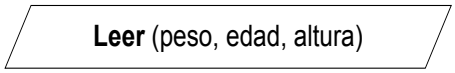
   Al ejecutarse la instrucción **Leer**, automáticamente se guarda 60 en peso, 150 en edad y, 26 en altura. Al computador no le preocupa que la persona tenga 60 kilos de peso, 150 años y 26 cm. de altura, esos fueron los datos digitados y con ellos el programa trabajará. El computador no generará error alguno sin embargo los resultados obtenidos cuando se ejecute el programa serán incorrectos.

Para ser más claro, una alternativa a la instrucción **Leer** (peso, edad, altura) sería:

Leer (peso)
Leer (edad)
Leer (altura)

La instrucción **Leer** se representa con un romboide en el mundo del diagrama de flujo, a continuación, ver la Tabla 8 con algunos ejemplos:

Tabla 8. Representación de la instrucción Leer en el diagrama de flujo.

PSEUDOCÓDIGO Instrucción	DIAGRAMA DE FLUJO
Leer (A)	
Leer (peso, edad, altura)	

Fuente: Autores

La instrucción **Leer** se necesita en casi todos los problemas, a continuación se muestran algunos ejemplos de enunciados típicos y sus características:

- Aquellos en los cuales aparece la palabra conocidos, dados, ingresar o capturar:

  Dados tres números construya un pseudocódigo que muestre el mayor de ellos.
  Para poder resolver el problema es necesario conocer los valores de los tres números y éstos deben ser suministrados por el usuario.

  Escriba un Programa que lea un número entero N y calcule el resultado de la siguiente serie: $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\ldots\pm\frac{1}{N}$

  En este segundo ejemplo, se conocen como deben ser los términos a generar sin embargo, la cantidad de los mismos está determinada por el valor que tome **N,** el cual debe ser ingresado por el usuario. Si el usuario ingresa el número 3, el programa debe mostrar el resultado de sumar $1-\frac{1}{2}+\frac{1}{3}$ es decir, 0.8333333.

- Problemas que resuelven una ecuación cuya solución se obtiene por reemplazar en ella los identificadores por números.

  Construya un pseudocódigo que muestre el resultado de las raíces reales de la siguiente expresión $aX^2 + bX + c = 0$
  Para encontrar la solución de las raíces reales se necesita conocer los valores de a, b y c.

1. Aquellos problemas cuya solución sólo se puede obtener por el proceso que se le haga a los datos suministrados por el usuario. Por ejemplo:

   Diseñe un algoritmo que determine si un número es o no es, par positivo.
   Para resolver el problema, es necesario que el usuario ingrese el número a evaluar y así determinar si éste es o no un par positivo.

Normalmente la gran mayoría de los enunciados muestran la necesidad de solicitar datos al usuario.

2.2.4 Instrucción de Escritura

Durante la solución de un problema a través de pseudocódigo, generalmente se necesita mostrar resultados parciales o totales, esto se logra con la instrucción **Escribir**. En forma general siempre que se desee mostrar, visualizar o imprimir los resultados, en el mundo del pseudocódigo se usa la instrucción **Escribir**.

La instrucción **Escribir** tiene el siguiente formato:

Dentro de los paréntesis se pueden colocar cadenas de texto, identificadores o combinación de ambos. **Escribir** es una palabra reservada. Toda instrucción **Escribir** al ser ejecutada visualizará en pantalla lo que se ha colocado dentro de los paréntesis.

La instrucción **Escribir** tiene los siguientes formatos:

1. **Escribir** (A)

   Cuando se ejecuta esta instrucción el computador muestra el contenido del identificador siempre y cuando esté declarado y contenga un dato.

2. **Escribir** (" texto ")

   Siempre que se desee mostrar algún comentario, información o texto en general el uso de este tipo de formato es el adecuado. Este tipo de formato no requiere cumplir con condición alguna.

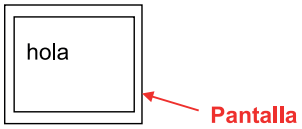
3. **Escribir** (" texto ", A)

   Si se desea acompañar el contenido de una celda con un texto, se emplea este formato. El identificador debe estar declarado y contener un dato.

En la Tabla 9 se muestran algunos ejemplos, en todos ellos se asume que los identificadores ya están declaradas.

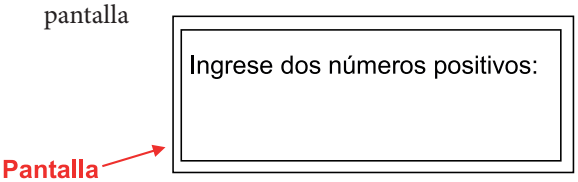
Tabla 9. Ejemplos de Instrucciones y ejecución

INSTRUCCIÓN	REPRESENTACIÓN DE LA EJECUCIÓN	
	Ejecución de las instrucciones	**Celda**
T ← **"hola"** **Escribir** (T)	Al ejecutarse se almacena en **T** el dato cadena "hola", Cuando se ejecuta la instrucción **Escribir** aparece en pantalla lo siguiente: Pantalla hola	T h o l a
G ← 535 **F** ← **"es"** **Escribir** (F, G)	En la **G** se almacena el valor 535 En la **F** se almacena el dato *es* Cuando se ejecuta la instrucción **Escribir** se muestra en pantalla: Pantalla es 535	G 535 F e s

Escribir (“hola”)	Cuando se ejecuta la instrucción **Escribir** aparece en pantalla lo siguiente: 

Escribir (“Ingrese dos números positivos:”)

Cuando se ejecuta la instrucción **Escribir** se muestra en pantalla



G ← 4.5

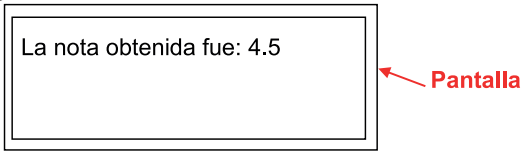
Se almacena 4.5 en **G**

G

4.5

Escribir (“La nota obtenida fue:”, **G)**

Cuando se ejecuta la instrucción **Escribir** aparece en pantalla lo siguiente:



D ← 3.5

Cuando se ejecuta se almacena en **D** 3.5

D

3.5

A ← 13

Cuando se ejecuta se almacena en **A** 13

A

13

B ← “sacaron”

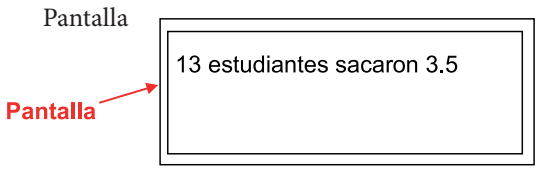
Cuando se ejecuta se almacena en **B** sacaron

B

s	a	c	a	r	o	n

Escribir (A , “estudiantes” , B, D)

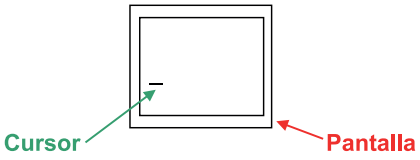
Cuando se ejecuta la instrucción **Escribir** se muestra en



Generalmente antes de una instrucción **Leer** va una instrucción **Escribir** esto con el fin de informar al usuario lo que se desea que éste ingrese. En el ejemplo del peso, edad y altura se creo la siguiente instrucción

Leer (peso, edad, altura)

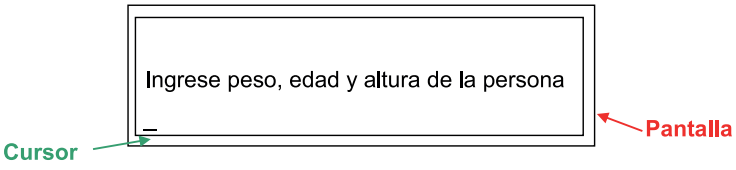
Cuando el computador llega a esta instrucción en pantalla aparece lo siguiente:



Justo antes de ejecutarse la instrucción **Leer**, en pantalla aparece el cursor titilando en espera a que usuario ingrese los datos. La pantalla con tan sólo un cursor titilando no ofrece información alguna al usuario acerca de qué debe hacer, es recomendado utilizar la instrucción Escribir para indicarle que datos suministrar. Agregando una instrucción **Escribir** antes de la instrucción **Leer** quedaría:

Escribir ("Ingrese peso, edad y altura de la persona")
Leer (peso, edad, altura)

De esta manera en pantalla aparece:



En la Tabla 10 se puede apreciar tres tipos de símbolos para la instrucción **Escribir**, el primero de ellos es idéntico al usado por la instrucción **Leer**, el segundo, es usado para imprimir y, el tercero, para mostrar en pantalla. El símbolo depende del programa que emplee para generar el diagrama de flujo.

Tabla 10. Representación de la instrucción Escribir en el diagrama de flujo.

PSEUDOCODIGO Instrucción	DIAGRAMA DE FLUJO
Escribir (A) **Escribir** ("El resultado es:", A)	1. Escribir (A) 2. Escribir (A) 3. Escribir (A) Escribir ("El resultado es:", A)

2.3 FORMATO GENERAL DEL PSEUDOCÓDIGO

Todo programa escrito en pseudocódigo está conformado por los siguientes tres bloques:

1. La Cabecera donde se coloca el nombre del programa escrito en pseudocódigo:

   Algoritmo_*nombre_del_programa*

2. El Bloque de Declaración es donde se informa al computador cuántas celdas y de qué tipo se van a necesitar, aquí se deben declarar todos los identificadores que se utilizarán en el transcurso de la ejecución del programa.

   El Bloque de Declaración puede contener dos secciones: Declaración de Variables y Declaración de Constantes.

a. *Declaración de variables*: comienza con la palabra reservada **var**, debajo de ella se debe colocar primero, el tipo de dato, seguido de dos puntos (**:**) y por último, la(s) celda(s) que se crearán.

```
var
        Tipo_de_dato : Listado_de_celdas
```

Si se desea declarar más de una variable del mismo tipo de dato, se deben separar con comas, por ejemplo:

```
var
        Entero : A, B
```

Lo anterior también se puede escribir como sigue:

```
var
        Entero : A
        Entero : B
```

El siguiente ejemplo muestra el uso de una celda para cada tipo de dato.

```
var
        Entero : A
        Decimal : B
        Caracter : C
        Booleano: G
```

Ningún nombre de variable se puede utilizar más de una vez. En el siguiente ejemplo la variable A se encuentra declarada dos veces, por lo tanto generará error en la ejecución del programa:

```
var
        Entero : A
        Caracter : A
        Booleano: G
```

Para declarar una variable cadena primero se le asigna un nombre y luego entre corchetes se coloca un número que representa la cantidad de celdas que usará la variable, por ejemplo:

```
var
        Cadena: D[10]
```

Las palabras **Decimal**, **Carácter**, **Entero**, **Cadena** y **Booleano** son palabras reservadas. El siguiente ejemplo muestra un error por usar una palabra reservada para una función a la cual no pertenece:

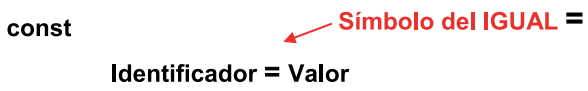
```
var
        Decimal : a, A, Entero
```

Existen dos maneras que ayudan en determinar cuántas variables y de qué tipos se necesitan declarar. La primera es la Técnica de las Preguntas que consiste en formularse dos preguntas cuyas respuestas ayudaran a construir la zona de Declaración de variables.

- *Pregúntese cuántos datos necesita conocer para resolver el problema*. La misma cantidad de datos obtenida en la respuesta corresponden a la cantidad de celdas que se deben crear para almacenarlos. A cada celda se le da un nombre y se evalúa qué tipo de dato debe contener.
- *Pregúntese cuántos cálculos necesita resolver con los datos capturados en la pregunta anterior*. La cantidad de cálculos a llevar a cabo es la misma cantidad de variables que se deben crear para almacenar los datos obtenidos al solucionar los mismos. A cada uno se le da un nombre y se evalúa de qué tipo de dato será el resultado obtenido.

La segunda técnica es la Técnica del Algoritmo, ésta consiste en guiarse por las palabras utilizadas en los pasos del algoritmo construido. En los ejercicios al final del capítulo se estudia en detalle cómo construir un algoritmo y, después, la aplicación de estas técnicas.

b. Declaración de constantes: comienza con la palabra reservada **const**, debajo de ella se debe colocar primero, el identificador, seguido del símbolo igual (=) y por último, el valor que se asignará a la celda. El valor es un dato, no una expresión aritmética que pueda generarlo.



Nunca se puede utilizar un identificador, que se encuentre en la declaración de constantes, dentro de los paréntesis en una instrucción **Leer** ya que este espacio está reservado únicamente para las celdas colocadas en la zona de declaración de variables.

Si se quiere declarar la constante PI se debe hacer lo siguiente:

```
const
        pi = 3.1416
```

Recuérdese que después del igual (=) debe ir el valor y no un expresión. No es válido lo siguiente en el mundo del pseudocódigo:

const

$raizdedos = \sqrt{2}$

$cociente = 4/5$

$porcen1 = 0.05\%$

Algunos ejemplos de declaración de constantes serían:

```
const
        pi = 3.1416
        raizdedos = 1.4142
        nombre = "pedro"
        letra = 's'
        porcent1 = 0.05
```

Cualquier variable declarada en la zona de declaración de variables NO puede ser utilizada en la de constantes. El siguiente ejemplo genera error al ser ejecutado puesto que A es al mismo tiempo una variable y una constante.

var

Entero **:**A
Decimal **:** B

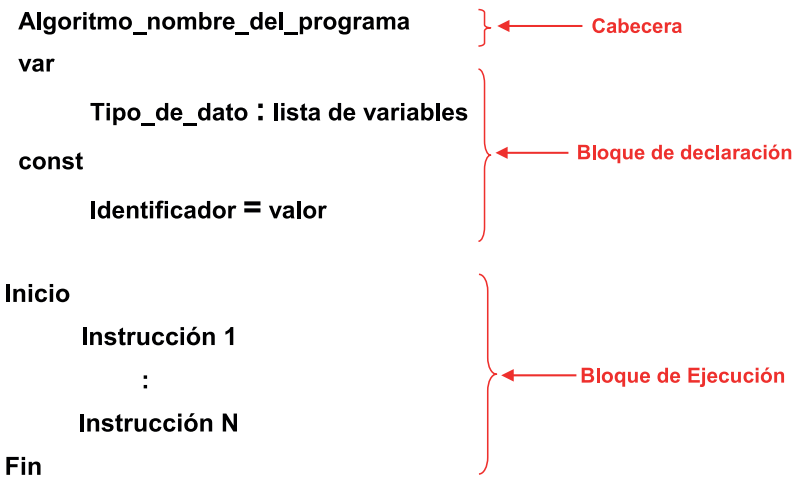
const

A = 3.456
Suma = 45

La sección de declaración de constantes es opcional.

3. El Bloque de Ejecución es el cuerpo de instrucciones que debe ejecutar el computador. Comienza y termina con las palabras reservadas **Inicio** y **Fin**, y en su interior podrá contener la repetición de las otras instrucciones las veces que sea necesario.

Entonces el formato general del pseudocódigo es:



2.4 EJERCICIOS

2.4.1 Ejercicios con Respuesta

1. Construya un algoritmo, pseudocódigo y diagrama de flujo tal, que conocida la base y altura de un rectángulo, calcule y muestre su área y perímetro.
2. Realice un algoritmo, pseudocódigo y diagrama de flujo tal que, dado el nombre de una persona (de máximo 8 caracteres), el apellido (de máximo 6), la edad en años y su peso en libras, calcule y muestre la edad en número de días, el peso en kilogramos, y el apellido seguido del nombre.
3. La potencia que consume un bombillo se expresa por medio de la fórmula **P = V×I**, donde **V** es el voltaje e **I** es la corriente. El voltaje está en función de la corriente **I** y la resistencia **R** del bombillo, por medio de la siguiente expresión: **V = I×R**. La potencia total entregada por la fuente es igual a la suma de las potencias de todos los bombillos. La figura siguiente muestra cuatro (4) bombillos, y ellos están conectados a una fuente. Construya un algoritmo, pseudocódigo y diagrama de flujo tal que, conocidos el valor de la corriente **I** y los valores de resistencia **R** - de cada bombillo -, calcule y muestre la potencia de cada bombillo y la potencia total entregada por la fuente, ver Ilustración 1.

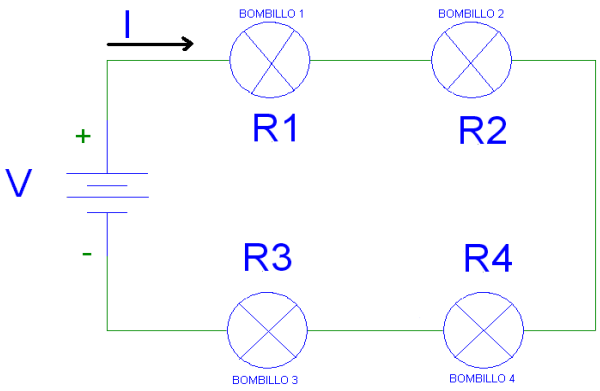


Ilustración 1. Circuito serie conformado por cuatro bombillos y una fuente de voltaje

4. En la tienda Wall Max existen los siguientes productos:

Artículo	Marca	Valor US$
Camisa	*T-shirt*	$15
Pantalón	*Lave`se*	$25 el par
Zapatos	*Acme*	$80
Ropa interior	*Proteggeme*	$45
Medias	*Ropita pa pie*	$10
DVD	*Pollito*	$70

La tienda ofrece un descuento del: 10% en las camisas y 10% en los pantalones por la compra de 3 de cada uno de ellos; 4% por la compra de dos pares de zapatos y el 50% de descuento por el tercer par; por la compra de cinco pares de medias el descuento es del 5% y 8% de descuento por comprar dos DVD. Construya un algoritmo, pseudocódigo y diagrama de flujo que muestre: el valor total de cada artículo que se compra, el valor de cada descuento obtenido, el valor total a pagar.

2.4.2 Ejercicios sin Respuesta

1. Pedro tiene el triple de la edad de Juan. Construya un algoritmo, pseudocódigo y diagrama de flujo tal, que dada la edad de Pedro, calcule y muestre la edad de Juan.
2. Construya un algoritmo, pseudocódigo y diagrama de flujo tal, que conocido el radio de un círculo, calcule y muestre su diámetro, circunferencia y área.
3. El supermercado Pepín se encuentra en promoción "miércoles de verduras y frutas"; descuento del 15% sobre las compras. Suponiendo que la suma de todas las verduras y frutas compradas fue **V,** construya un algoritmo, pseudocódigo y diagrama de flujo que calcule y muestre cuánto habría que pagar si hoy fuera miércoles.
4. Su profesor de programación ha sacado 4 notas a lo largo del semestre y es el momento de conocer su definitiva. Construya un algoritmo, pseudocódigo y diagrama de flujo tal, que dadas unas 4 notas, calcule y muestre el promedio de las mismas.
5. Se necesitan 20 ladrillos para cubrir un área de 1 m^2; suponiendo que hay que cubrir un área de **X** metros * **Y** metros; construya un algoritmo,

pseudocódigo y diagrama de flujo que calcule y muestre cuantos ladrillos se necesitarían en total para cubrir toda el área.

6. Usted saca un crédito en el banco a 5 años con un interés anual del 3%. Construya un algoritmo, pseudocódigo y diagrama de flujo tal, que conocido el monto inicial del préstamo, calcule y muestre cuanto debe pagar al cabo de los 5 años.

2.4.3 Respuesta a los Ejercicios

1. Construya un algoritmo, pseudocódigo y diagrama de flujo tal, que conocida la base y altura de un rectángulo, calcule y muestre su área y perímetro.

- Lo primero es comprender el problema, para ello se conceptualiza y determina el objetivo:

- Conceptualización: el problema consiste en obtener de una figura geométrica conocida - el rectángulo - su área y perímetro. Para ello se deben utilizar las ecuaciones:

$$A = B \times H \qquad P = 2(B + H)$$

Objetivo: calcular el valor del área y del perímetro de un rectángulo, utilizando la base y altura ingresadas por el usuario.

- Una vez comprendido el problema, se procede a construir el algoritmo:

Para resolver el problema se requiere que el usuario ingrese la base y la altura, por lo tanto deben ser capturados. Luego, se resuelven las ecuaciones del área y perímetro con los datos capturados. Y finalmente, se muestran los resultados. El algoritmo queda así:

1. Inicio
2. Capturar la base y altura del rectángulo
3. Resolver las ecuaciones del área y perímetro del rectángulo
   $$A = B \times H \qquad P = 2(B + H)$$
4. Mostrar los resultados del área y perímetro del rectángulo
5. Fin

- Una vez construido el algoritmo el siguiente paso es elaborar el pseudocódigo:

  Se determina el nombre que se va a dar al programa y se coloca en la cabecera:
  Algoritmo_Calculo_Area_y_Perimetro_Rectangulo

  Para determinar lo que se debe colocar en la zona de declaración de variables se tendrá en cuenta la técnica de las preguntas:

  ¿Cuántos datos necesita conocer para resolver el problema? Resolver el problema implica calcular el área y perímetro del rectángulo y para lograr esto se necesita conocer dos valores: *la* ***base*** *y la* ***altura,*** *e*ntonces se necesita crear dos variables el nombre de cada una de ellas será: base y altura.

  ¿Qué tipo de dato debe almacenar cada una de ellas? Base y altura son distancias, en este caso de una figura geométrica, y sus valores pueden incluir números después del punto decimal, por tanto, el tipo de dato de cada una de las variables es **Decimal**.

  ¿Cuántos cálculos necesita resolver con los datos capturados en la pregunta 1? Para solucionar el problema se necesita hallar el área y el perímetro del rectángulo y para ello es necesario hacer dos operaciones que son:

  $$A = B \times H \qquad P = 2(B + H)$$

  Como son dos operaciones se requiere de dos variables para almacenar los resultados, en una de ellas el área y en la otra el perímetro. Los nombres para cada una de las variables serán: área y perímetro.

  ¿Qué tipo de dato debe guardar cada variable? Las dos ecuaciones dependen de la base y la altura, por lo que las variables área y perímetro también son de tipo **Decimal**. La declaración de variables queda:

  var
  Decimal : base, altura, área, perímetro

En las ecuaciones a resolver el único valor que es constante durante la ejecución del programa es el número 2 que se encuentra en la siguiente expresión: $P = 2(B + H)$.

Se puede declarar como constante así:

```
const
    C1 = 2
```

Por último, se procede a transformar el algoritmo para construir el bloque de ejecución. El paso **2.** del algoritmo requiere la capturar datos, lo que corresponde a la instrucción **Leer** en pseudocódigo.

Leer (base, altura)

Se acompañada la instrucción **Leer** de una instrucción **Escribir,** esto con el fin de informarle al usuario lo que se desea que éste ingrese. Entonces las dos primeras instrucciones después de **Inicio** son:

Escribir ("Ingrese la base y la altura del rectángulo")
Leer (base, altura)

El paso **3.** implica la solución de las ecuaciones. Estas ecuaciones deben ser convertidas a expresiones algorítmicas para poder ser usadas dentro del programa. Teniendo en cuenta que las letras A, B, P y H se reemplazan por las variables área, base, perímetro y altura respectivamente, entonces las ecuaciones quedan:

$$area \leftarrow base * altura$$
$$perimetro \leftarrow 2 * (base + altura)$$

Como se declaró la constante C1 entonces la instrucción de asignación para hallar el perímetro queda:

$$perimetro \leftarrow C1 * (base + altura)$$

En el **4.** paso se debe visualizar los resultados del área y el perímetro y para ello se utiliza la instrucción **Escribir**:

Escribir ("El área y perímetro del rectángulo son:", area, perimetro)

Entonces el pseudocódigo quedaría:

```
Algoritmo_calculo_area_y_perimetro_rectangulo        ← Cabecera
var
    Decimal : base, altura, perímetro, area          } Bloque de
const                                                } Declaración
    C1 = 2
Inicio
    Escribir (" ingrese base y altura del rectángulo ")
    Leer (base, altura)
    area ← base * altura                             } Bloque
    perímetro ← C1 * ( base + altura)                } de
    Escribir ("El área y perímetro del rectángulo son:",  } Ejecución
    area, perimetro)
Fin
```

Comparando el algoritmo con el pseudocódigo tenemos:

Algoritmo	**Pseudocódigo**
	Algoritmo_calculo_area_y_perimetro_rectangulo
	var
	Decimal : base, altura, perímetro, area
	const
	C1 = 2
1. Inicio	→ **Inicio**
2. Capturar la base y altura del rectángulo	→ **Escribir** ("Ingrese base y altura del rectángulo") **Leer** (base, altura)
Resolver las ecuaciones del área y Perímetro del rectángulo $A = B \times H \quad P = 2(B + H)$	→ area ← base * altura perímetro ← C1 * (base + altura)
Mostrar los resultados del área y perímetro del rectángulo	→ **Escribir** ("El área y perímetro del rectángulo son:", area, perimetro)
5. Fin	→ **Fin**

Todo Diagrama de flujo comienza y termina con los símbolos de Inicio y Fin, no existe ningún símbolo para representar la cabecera ni la zona de declaración. Inicialmente se construirá el diagrama de flujo partiendo del pseudocódigo y, luego se hará los mismo pero esta vez usando el algoritmo, ver Ilustración 2

Pseudocódigo

Algoritmo_calculo_area_y_perimetro_rectangulo
var
    Decimal : base, altura, perímetro, área
const
    C1 = 2

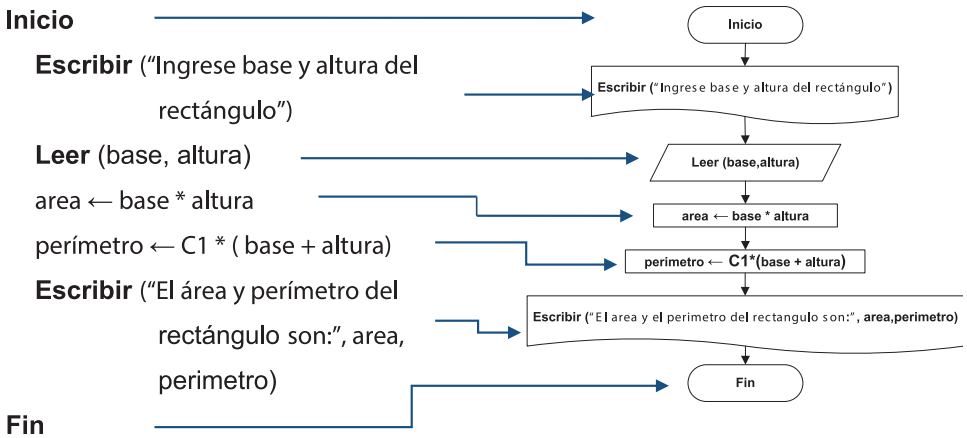


Ilustración 2. Relación entre el pseudocódigo y el diagrama de flujo

En forma general, realizar un Diagrama de flujo es muy sencillo tan solo se debe tener en cuenta las instrucciones y los símbolos asociadas a éstas así como también, el uso de conectores que los unan. La Ilustración 3 muestra cómo se obtiene el Diagrama de flujo a partir del Algoritmo.

Algoritmo

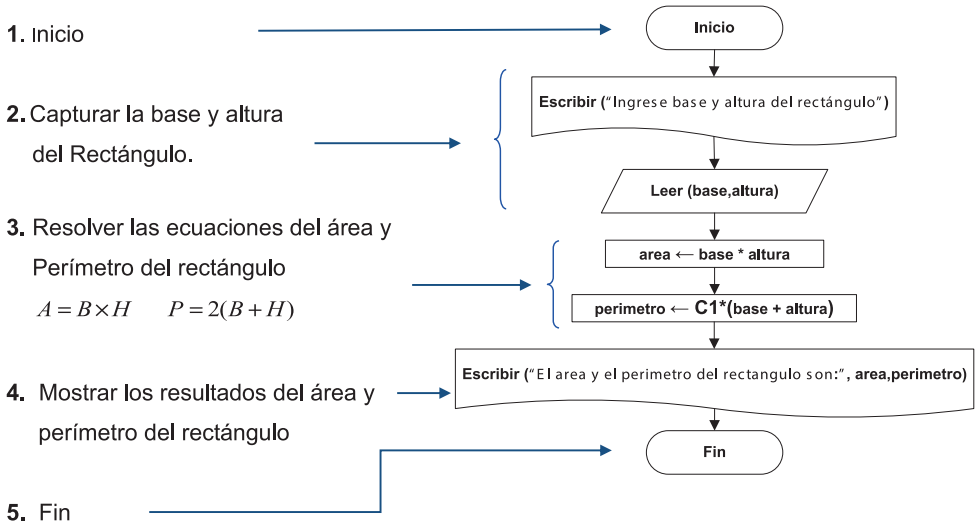


Ilustración 3. Relación entre el algoritmo y el diagrama de flujo.

5. Realice un algoritmo, pseudocódigo y diagrama de flujo tal que, dado el nombre de una persona (de máximo 8 caracteres), el apellido (de máximo 6), la edad en años y su peso en libras, calcule y muestre la edad en número de días, el peso en kilogramos, y el apellido seguido del nombre.

   Lo primero es conceptualizar el mundo del problema. Es necesario conocer los datos de nombre, apellido, peso y edad de una persona. Los últimos dos datos son importantes porque con ellos se debe llevar a cabo una conversión. Es necesario visualizar el apellido y luego el nombre contrario a la manera en que se capturan estos datos.

   El objetivo es encontrar y visualizar la edad en días, el peso en kilos así como también mostrar el apellido y el nombre de la persona. Para obtener la edad en días y el peso en kilos es necesario llevar a cabo una conversión, es decir:

$$Edad\ \ en\ \ dias = Edad\ \ en\ \ años \times \frac{365\ \ dias}{1\ \ año} \quad \Rightarrow \quad Edad\ \ en\ \ dias = Edad\ \ en\ \ años \times 365$$

$$Peso\ \ en\ \ kilos = Peso\ \ en\ \ libras \times \frac{1\ \ kilo}{2\ \ libras} \quad \Rightarrow Peso\ \ en\ \ kilos = \frac{Peso\ \ en\ \ libras}{2}$$

Ahora se construye el algoritmo:

1. Inicio.
2. Capturar el nombre, apellido, edad en años y peso en libras de una persona.
3. Convertir los años a días y las libras a kilos. Para obtener estos datos se utilizan las siguientes ecuaciones:

$$Edad \quad en \quad dias = Edad \quad en \quad años \times 365$$

$$Peso \quad en \quad kilos = \frac{Peso \quad en \quad libras}{2}$$

4. Visualizar la edad en número de días, el peso en kilos, el apellido y el nombre de la persona.
5. Fin.

Ahora se construye el pseudocódigo a partir del algoritmo. En el paso **2.** capturar implica el uso de dos instrucciones, **Escribir** y **Leer**. La palabra convertir, del paso **3.**, está asociada a la instrucción de asignación ya que por medio de ella se lleva a cabo el proceso de transformación de los datos ingresados por el usuario. Visualizar se realiza utilizando la instrucción **Escribir**.

Usando la **Técnica de las Preguntas** se tiene lo siguiente:

¿Cuántos datos necesita conocer para resolver el problema? Resolver el problema implica conocer el nombre, apellido, edad en años y peso en libras de una persona, sin estos datos el programador no podrá encontrar la solución. Es necesario capturar 4 datos por lo que se requieren cuatro variables para almacenarlos. Los identificadores de las cuatro variables son: nom, ape, edad_años y peso_libras respectivamente. Una vez conocido los nombres de las variables es necesario saber el tipo de las mismas; nom y ape son variables que guardarán el nombre (de 8 caracteres) y el apellido (máximo 6 caracteres) de la persona respectivamente, y como contienen más de una letra el tipo de dato para ellas es **Cadena**. La edad de una persona generalmente es un número que no incluye decimales por tanto el tipo de dato para edad_años es **Entero**. El peso de una persona puede contener números después del punto decimal por lo que la variable peso_libras es de tipo **Decimal**.

¿Cuántos cálculos se necesitan resolver con los datos capturados en la pregunta 1? Para resolver el ejercicio es necesario convertir la edad de años a días y el peso de libras a kilos, por lo que se requieren de dos operaciones y de dos variables que almacenen los resultados. Los identificadores para estas dos variables son: edad_días y peso_kilos respectivamente. Generalmente cuando se habla de cantidad de días se hace referencia a un número entero por lo que el tipo de dato para edad_días es **Entero**. El peso de una persona puede contener números después del punto decimal por lo que la variable peso_kilos es de tipo **Decimal**.

Con base en lo anterior la zona de declaración de variables queda:

var
Cadena : nom[8], ape[6]
Decimal : peso_libras, peso_kilos
Entero : edad_años, edad_días

Las instrucciones que reflejan los pasos **2.**, **3.** y **4.** del algoritmo son:

Escribir ("Ingrese el nombre, apellido, edad en años y peso en libras de la persona")
Leer (nom, ape, edad_años, peso_libras)
edad_días ← edad_años * 365
peso_kilos ← peso_libras / 2
Escribir ("La edad en días, el peso en kilos:", edad_días, peso_kilos)
Escribir ("el apellido y nombre de la persona son:", ape, nom)

A continuación la Ilustración 4 muestra el pseudocódigo completo y el correspondiente diagrama de flujo:

Pseudocódigo

```
Algoritmo_calculo_potencia_circuito_electrico
Var
   Cadena  : nom[8], ape[6]
   Decimal : peso_libras, peso_kilos
   Entero  : edad_años, edad_días
```

Inicio
Escribir (“ingrese el nombre, apellido, edad en años y peso en libras de la persona”)
Leer (nom, ape, edad_años, peso_libras)
edad_días ← edad_años * 365
peso_kilos ← peso_libras / 2
Escribir (“La edad en días, el peso en kilos:”, edad_días, peso_kilos)
Escribir (“el apellido y nombre de la persona son:”, ape, nom)
Fin

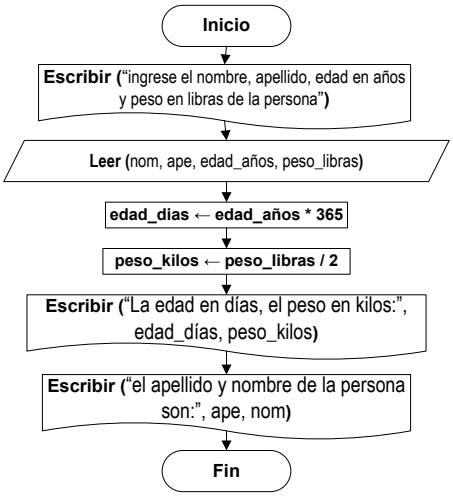


Ilustración 4. Relación entre el pseudocódigo y el diagrama de flujo.

6. La potencia que consume un bombillo se expresa por medio de la fórmula **P = V×I**, donde **V** es el voltaje e **I** es la corriente. El voltaje está en función de la corriente **I** y la resistencia **R** del bombillo, por medio de la siguiente expresión: **V = I×R**. La potencia total entregada por la fuente es igual a la suma de las potencias de todos los bombillos. La figura siguiente muestra cuatro (4) bombillos, y ellos están conectados a una fuente. Construya un algoritmo, pseudocódigo y diagrama de flujo tal que, conocidos el valor de la corriente **I** y los valores de resistencia **R** - de cada bombillo -, calcule y muestre la potencia de cada bombillo y la potencia total entregada por la fuente, ver Ilustración 5.

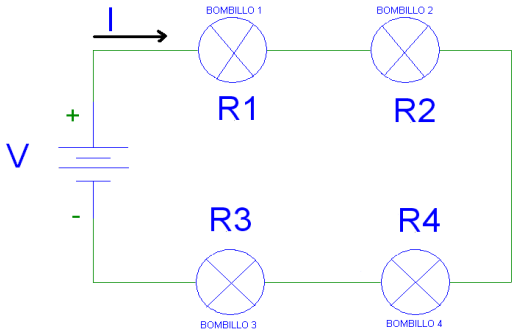


Ilustración 5. Circuito serie conformado por cuatro bombillos y una fuente de voltaje.

Conceptualización: el circuito contiene cuatro bombillos representados con las palabras R1, R2, R3 y R4 (la letra R es de resistencia) conectados a una fuente de voltaje V. La figura muestra que a los cuatro bombillos los atraviesa la corriente I, con un valor igual para cada uno de ellos y a su vez, está sale y retorna a la fuente V. Según el enunciado, multiplicar la corriente I por el valor de la resistencia de un bombillo produce el voltaje de ese bombillo en particular, además, multiplicar este voltaje por el valor de la corriente genera la potencia que consume el bombillo.

Objetivo: encontrar la potencia de cada bombillo así como también la potencia total entregada por la fuente y para ello, el enunciado establece que se conocen la corriente I que atraviesa el circuito y los valores de cada resistencia. Para calcular la potencia de cada bombillo se hace uso de

$$P = V \times I$$

En donde V es igual a:

$$V = I \times R$$

Se puede sustituir el valor de V en la primera ecuación por el que éste tiene en la segunda expresión, entonces

$$P = V \times I = (I \times R) \times R = I^2 \times R$$

Ahora se construye el algoritmo:

1. Inicio.
2. Capturar el valor de la corriente **I** y los valores de las resistencias **R1**, **R2**, **R3** y **R4** del circuito.
3. Calcular la potencia de cada bombillo y la potencia total de la fuente del circuito. La potencia de cada bombillo se puede obtener por el uso de

$$P = I^2 \times R$$

Donde se reemplaza el valor de **I** y **R** por el capturado en el punto 2, note que se deben hallar cuatro valores de potencia y para cada uno de ellos cambia el valor de **R** quedando igual el valor de **I**.

La potencia total se puede hallar de la siguiente manera:

$$P = P1 + P2 + P3 + P4$$

Donde P1, P2, P3 y P4 son las potencias del bombillo R1, R2, R3 y R4 respectivamente.

4. Mostrar la potencia de cada bombillo así como la potencia total del circuito.
5. Fin.

Con base en el algoritmo se construirá el pseudocódigo y para ello se hará uso de las palabras claves escritas en aquel con el fin de elaborar las instrucciones pertinentes. El nombre que se le dará a este programa será cálculo área y perímetro rectángulo, por tanto la **cabecera** queda:

Algoritmo_calculo_potencia_circuito_electrico

Para determinar cuántas variables y de qué tipo son necesarias declarar se usará en este caso la técnica del algoritmo, es decir:

El punto dos nos informa de cuántas variables se necesitan crear para poder almacenar los datos que se desean capturar. En este caso se necesitan de cinco variables, una para la corriente y las otras cuatro para los valores de las resistencias de los bombillos, a cada una de ellas se les asignan los siguientes identificadores **I**, **R1**, **R2**, **R3** y **R4** respectivamente. *¿Qué tipo de dato se guardará en* **I**, **R1**, **R2**, **R3** y **R4**? Los equipos eléctricos que miden estas variables registran valores reales por lo que cada una de estas variables debe ser de tipo **Decimal**, entonces hasta el momento la **Declaración de variables** tiene:

var

Decimal : I, R1, R2, R3, R4

El punto tres muestra cuántos cálculos se deben resolver, en este caso se deben llevar a cabo cinco cálculos, cuatro para las potencias de cada bombillo y uno para la potencia total, por tanto, es necesario declarar cinco variables a cada una de ellas se les asignan los siguientes identificadores **P1**, **P2**, **P3**, **P4** y **Pt**, el tipo de dato que éstas deben almacenar es **Decimal**.

La declaración de variables usando la técnica del algoritmo es:

var

Decimal : I, R1, R2, R3, R4, P1, P2, P3, P4, Pt

Aunque la ecuación del cálculo de la potencia de cada bombillo existe el número dos, no se construirá la solución del ejemplo utilizando la declaración de constantes.

El punto dos del algoritmo contiene la palabra clave *Capturar*, la cual indica que se debe hacer uso de la instrucción **Leer**, pero la misma debe estar acompañada de la instrucción **Escribir**. Dentro de la instrucción **Leer** se deben colocar las variables que almacenaran los datos que se necesitan capturar y, se utiliza la instrucción **Escribir** con el fin de informar al usuario qué datos debe ingresar los cuales serán capturados cuando se ejecute la instrucción **Leer**. Las instrucciones quedan:

Escribir ("ingrese la corriente **I** y los valores de **R1**, **R2**, **R3** y **R4** del circuito")
Leer (I, R1, R2, R3, R4)

El punto tres del algoritmo informa sobre cuáles y cuántos cálculos se deben realizar para resolver el problema, este punto está relacionado con la instrucción de asignación. Las instrucciones quedan:

Aritmética	**Algorítmica**
$P1 = I^2 \times R1$	P1 ← I**2 * R1
$P2 = I^2 \times R2$	P2 ← I**2 * R2
$P3 = I^2 \times R3$	P3 ← I**2 * R3
$P4 = I^2 \times R4$	P4 ← I**2 * R4
$Pt = P1 + P2 + P3 + P4$	Pt ← P1 + P2 + P3 + P4

El punto cuatro del algoritmo permite la visualización de los resultados, para llevar a cabo esto es necesario el uso de la instrucción **Escribir**, las instrucciones quedan:

Escribir ("La potencia de los bombillos 1, 2, 3 y 4 del circuito es:", P1, P2, P3, P4)
Escribir ("La potencia total de la fuente es:", Pt)

El último punto del algoritmo es la terminación del mismo y para ello se utiliza la palabra reservada **Fin**.

A continuación se muestra el pseudocódigo junto al algoritmo y se hace énfasis en la relación entre ellos usando flechas las cuales ayudaran al lector a observar la correspondencia entre el algoritmo y el pseudocódigo. Al igual que la relación entre el pseudocódigo y el diagrama de flujo, ver Ilustración 6.

Algoritmo

1. Inicio

2. Capturar el valor de la corriente **I** y los valores de las resistencias **R1**, **R2**, **R3** y **R4** del circuito eléctrico.

3. Calcular la potencia de cada bombillo y la potencia total de la fuente del circuito eléctrico.

$P1 = I^2 \times R1 \quad P2 = I^2 \times R2$

$P3 = I^2 \times R3 \quad P4 = I^2 \times R4$

$Pt = P1 + P2 + P3 + P4$

4. Mostrar la potencia de cada bombillo así como la potencia total del circuito eléctrico.

5. Fin

Pseudocódigo

```
Algoritmo_calculo_potencia_circuito_electrico
var
    Decimal: I,R1,R2,R3,R4,P1,P2,P3,P4,Pt
Inicio
    Escribir ("ingrese la corriente I y los valores
              de R1, R2, R3 y R4 del circuito
              eléctrico")
    Leer ( I, R1, R2, R3, R4)
    P1 ← I * *2  *  R1
    P2 ← I * *2  *  R2
    P3 ← I * *2  *  R3
    P4 ← I * *2  *  R4
    Pt ← P1 + P2 + P3 + P4
    Escribir ("La potencia de los bombillos 1, 2,
              3 y 4 del
              circuito eléctrico es:", P1, P2, P3,
              P4)
    Escribir ("La potencia total de la fuente
              es:", Pt)
Fin
```

ALGORITMO usando ESC

```
Inicio
Capturar tres números enteros positivos.
Si ( A>=0 Y B >= 0 Y C>=0 ) entonces
 Si (A<> B Y A<>C Y B<>C) entonces
  Si (A > B Y A > C) entonces
      Mostrar el mayor es A
  De lo contrario
    Si (B > A Y B > C) entonces
        Mostrar el mayor es B
    De lo contrario
        Mostrar el mayor es C
    Fin_si
  Fin_si
 De lo contrario
  Mostrar existen números iguales
 Fin_si
De lo contrario
   Mostrar numero no valido
Fin_si
Fin
```

Pseudocódigo usando ESC

```
Algoritmo_encuentra_mayor
var
     Entero : A,B,C
Inicio
 Escribir("Ingrese tres números enteros positivos
          diferentes")
 Leer (A,B,C)
   Si (A>=0 Y B>=0 Y C>=0) entonces
    Si (A<>B Y A<>C Y B<>C) entonces
     Si (A > B Y A > C) entonces
        Escribir ("El numero mayor es:",A)
     De lo contrario
      Si (B > A Y B > C) entonces
        Escribir ("El numero mayor es:",B)
      De lo contrario
        Escribir ("El numero mayor es:",C)
      Fin_si
     Fin_si
    De lo contario
        Escribir ("Existen números iguales")
    Fin_si
   De lo contario
        Escribir ("numero no valido")
   Fin_si
Fin
```

Ilustración 6. Relación entre el pseudocódigo y el diagrama de flujo.

7. En la tienda Wall Max existen los siguientes productos:

Artículo	**Marca**	**Valor US$**
Camisa	*T-shirt*	$15
Pantalón	*Lave`se*	$25 el par
Zapatos	*Acme*	$80
Ropa interior	*Proteggeme*	$45
Medias	*Ropita pa pie*	$10
DVD	*Pollito*	$70

La tienda ofrece un descuento del: 10% en las camisas y 10% en los pantalones por la compra de 3 de cada uno de ellos; 4% por la compra de dos pares de zapatos y el 50% de descuento por el tercer par; por la compra de cinco pares de medias el descuento es del 5% y 8% de descuento por comprar dos DVD. Construya un algoritmo, pseudocódigo y diagrama de flujo que muestre: el valor total de cada artículo que se compra, el valor de cada descuento obtenido, el valor total a pagar.

Conceptualización: como en toda tienda, los clientes pueden o no comprar uno más productos y varios de cualquier artículo. El precio final a pagar depende de la cantidad que compre el cliente de cada artículo, por ejemplo, si se compran dos camisas, éstas tendrán un precio de US $30 dólares pero si se llevan 3, el valor final a pagar debe tener incluido el descuento del 10%. El único producto que no tiene descuento es la ropa interior. Un solo pantalón cuesta US $12.5 dólares.

Objetivo: encontrar y visualizar el valor de cada artículo que se compra, por ejemplo, un pantalón vale US $12.5 dólares, por dos son US $25 dólares. Además se debe obtener y mostrar el valor de cada descuento obtenido así como el valor total a pagar.

Obtener el descuento implica que el programa pueda reconocer la cantidad de productos que se compran de un artículo basándose en las características del mismo expresadas en el enunciado, pero para hallar este valor es necesario construir las ecuaciones pertinentes y para ello se presenta el siguiente análisis:

El enunciado establece que por la compra de 3 productos del artículo camisas o pantalones tiene un descuento del 10%, pero qué pasa cuando se compra 1 o dos camisas o incluso 6 o 9 de ellas? Si se compra una camisa el valor de este producto es de US $15 dólares; dos valen US $30 dólares; para 3 se debe aplicar el descuento del 10% (10% = 10/100 o 0.1), es decir:

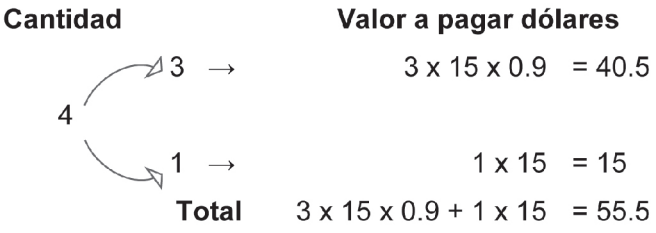
3 x 15	=	45	Valor a pagar sin descuento
3 x 15 x 10 / 100	=	4.5	Valor a pagar del descuento
3 x 15 – 3 x 15 x 10 / 100	=	40.5	Valor a pagar incluido el descuento

Por las tres camisas se debe pagar un total de US $40.5, este valor es equivalente a:

$$3 \times 15 - 3 \times 15 \times 0.1 = 3 \times 15\ (1 - 0.1) = 3 \times 15 \times 0.9 = 40.5$$

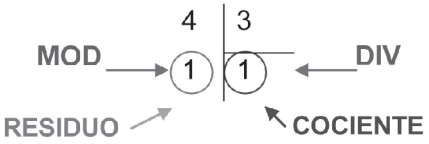
Donde 0.9 es igual a 90% que es el resultado de restarle al 100% el 10%. La expresión anterior permite calcular el valor a pagar por tres camisas incluido el descuento. En la expresión anterior los números 3 y 15 representan la cantidad de camisas y su precio por unidad respectivamente.

Cuando se compran 4 camisas, cómo se calcula el valor a pagar por esta cantidad? El programa construido debe detectar si en la cantidad de camisas compradas existen múltiplos o grupos de tres para aplicarles el descuento que ofrece la tienda; el número 4 se puede descomponer en 3 y 1, por lo que se aprecia que al llevar 4 camisas tres de éstas tienen el descuento del 10% y la cuarta no, es decir:



Entonces por cuatro camisas el cliente debe pagar $40.5 dólares.

Pero, cómo se puede obtener de un 4 un 3 y un 1? Como se busca encontrar cuantos grupos de 3 existen dentro de una cantidad determinada de camisas, se debe utilizar una operación aritmética que permita conseguir estos múltiplos de un valor dado de camisas y, la mejor manera para lograr esto es usando la división.



En la división anterior se muestra que el número 3 es el divisor y el 4, la cantidad de camisas, es el dividendo. Note que se obtiene un residuo y cociente de valor 1. Sin embargo, cómo se interpretan los resultados? El residuo, como en toda división, indica lo que queda, el resto, y en este ejemplo significa una camisa y al multiplicar el residuo por el precio de este artículo se obtiene:

1 x 15 = 15 dólares

Que es el valor de una unidad del artículo camisa. Para obtener el residuo se hace uso del operador **MOD** y se construye la siguiente expresión:

4 **MOD** 3 =1

Y la expresión que encuentra el valor a pagar por la cantidad de camisas que representan el residuo es:

4 **MOD** 3 * 15

Esta ecuación genera como resultado 15 que representa la cantidad de dinero que debe cancelar el cliente por la compra de una unidad del artículo camisa pero aún falta cobrar por las tres restantes. La expresión anterior permite encontrar el valor a pagar por una cantidad de camisas inferior a 3.

El resultado en el cociente representa la cantidad de grupos según el valor del divisor, es decir, cuatro dividido 3 genera un cociente de 1 que significa un grupo de 3, para el caso particular un grupo de 3 camisas y para éstas la tienda ofrece un descuento del 10%. El cociente se obtiene utilizando el operador **DIV** de la siguiente manera:

$$4 \textbf{ DIV } 3 = 1$$

Que representa un grupo de 3. Al multiplicar este valor por el divisor se obtiene la cantidad total de camisas que conforma el grupo, es decir:

$$4 \textbf{ DIV } 3 * 3 = 3$$

Por esta cantidad la tienda ofrece un descuento del 10%, entonces para obtener el valor final a pagar por estas tres camisas se multiplica el resultado anterior por el precio y por 0.9, es decir:

$$4 \textbf{ DIV } 3 * 3 * 15 * 0.9 = 40.5$$

Que representa el mismo valor que se obtuvo por la compra de 3 camisas incluido el descuento. El valor total a pagar por las cuatro camisas se obtiene de sumar:

$$4 \textbf{ MOD } 3 * 15$$

con

$$4 \textbf{ DIV } 3 * 3 * 15 * 0.9$$

Es decir,

$$4 \textbf{ MOD } 3 * 15 + 4 \textbf{ DIV } 3 * 3 * 15 * 0.9 = 55.5$$

Para cantidades superiores o iguales a 3 se observa en la Tabla 11:

Tabla 11. Análisis de las cantidades de camisas para el ejemplo 4.

Cantidad		Valor a pagar dólares	
3		3 x 15 x 0.9	= 40.5
4	3	3 x 15 x 0.9	= 40.5
	1	1 x 15	= 15
	Total	3 x 15 x 0.9 + 1 x 15	= 55.5

5	3	3 x 15 x 0.9		= 40.5
	2	2 x 15		= 30
		Total	3 x 15 x 0.9 + 2 x 15	= 70.5
6	3	3 x 15 x 0.9		= 40.5
	3	3 x 15 x 0.9		= 40.5
		Total	3 x 15 x 0.9 + 3 x 15 x 0.9	= 81
7	6	6 x 15 x 0.9		= 81
	1	1 x 15		= 15
		Total	3 x 15 x 0.9 + 2 x 15	= 96

Fuente: Autores

En la Tabla 11 que para una cantidad superior o igual a 3 se descompone ésta en dos valores y uno de ellos, el del múltiplo de 3, está presente en todas las expresiones que determinan el valor total a pagar por la compra.

Para una cantidad de cinco camisas y utilizando el análisis previo se tiene que la ecuación que permite encontrar el valor a total a pagar es:

$$5 \textbf{ MOD } 3 * 15 + 5 \textbf{ DIV } 3 * 3 * 15 * 0.9 = 70.5$$

La única diferencia entre las ecuaciones que encuentran el valor total a pagar para 4 y 5 camisas, es el valor que representa la cantidad, es decir, el 4 o el 5, y es exactamente este dato el que siempre está cambiando y es ingresado por el usuario, entonces para generalizar la expresión que permite hallar el valor final a pagar incluido el descuento por la compra de camisas es:

$$Vtpc \leftarrow CC \textbf{ MOD } 3 * 15 + CC \textbf{ DIV } 3 * 3 * 15 * 0.9$$

Donde CC es la variable que almacena el valor de la cantidad de camisas ingresado por el usuario y Vtpc es la variable que almacena el valor total a pagar

por las camisas. La siguiente tabla muestra la comprobación de la ecuación anterior.

Tabla 12. Comprobación de la ecuación Vtpc para el ejemplo 4.

	Primer término		**Segundo término**		
Cantidad	CC **MOD** 3 * 15	**Resultado**	CC **DIV** 3 * 3 * 15 * 0.9	**Resultado**	**Vtpc**
0	0 **MOD** 3 * 15	0	0 **DIV** 3 * 3 * 15 * 0.9	0	0
1	1 **MOD** 3 * 15	15	1 **DIV** 3 * 3 * 15 * 0.9	0	15
2	2 **MOD** 3 * 15	30	2 **DIV** 3 * 3 * 15 * 0.9	0	30
3	3 **MOD** 3 * 15	0	3 **DIV** 3 * 3 * 15 * 0.9	40.5	40.5
4	4 **MOD** 3 * 15	15	4 **DIV** 3 * 3 * 15 * 0.9	40.5	55.5
5	5 **MOD** 3 * 15	30	5 **DIV** 3 * 3 * 15 * 0.9	40.5	70.5
6	6 **MOD** 3 * 15	0	6 **DIV** 3 * 3 * 15 * 0.9	81	81

Fuente: Autores

Los resultados obtenidos con la ecuación de Vtpc coinciden con los de la Tabla 12 así como con los del análisis previo de este ejemplo. Obsérvese como el término que incluye el operador **MOD** permite obtener el precio a pagar para una cantidad de 1 o 2 productos sea que éstos representen la cuantía de la compra ó los artículos que "sobran" de aquellos que tienen descuento (por ejemplo, de 4 artículos uno de ellos no tiene descuento).

La ecuación general del descuento para las camisas es:

dtoc ←CC **DIV** 3 * 3 * 15 * 0.1

Aunque el análisis anterior se realizó para las camisas, para los pantalones la tienda Wall Mart ofrece el mismo tipo de descuento por lo que utilizando los mismos criterios se puede concluir que la expresión que permite obtener el valor total a pagar por la compra de pantalones es:

Vtpp ← CP **MOD** 3 * 12.5 + CP **DIV** 3 * 3 * 12.5 * 0.9

Esta expresión se diferencia de Vtpc en el precio unitario por pantalón. Vtpp y CP son las variables que almacenan el valor total a pagar por pantalones y la cantidad de pantalones respectivamente.

Para los artículos medias y DVD las expresiones son las siguientes:

Vtpm ← CM **MOD** 5 * 10 + CM **DIV** 5 * 5 * 10 * 0.95

Vtpdvd ← CDVD **MOD** 2 * 70 + CDVD **DIV** 2 * 2 * 70 * 0.92

En estas ecuaciones Vtpm, Vtpdvd, CM y CDVD son valor total a pagar medias, valor total a pagar dvd, cantidad de medias y cantidad de dvd respectivamente. En estas expresiones que se usa 0.95 y 0.92 para representar el 5% y 8% respectivamente además, obsérvese que el divisor para Vtpm es el 5 y para Vtpdvd el 2 que representan la cantidad de productos de medias y dvd para los cuales la tienda ofrece descuento.

Las ecuaciones de descuento para las camisas, pantalones, medias y dvd son:

dtoc ←CC **DIV** 3 * 3 * 15 * 0.1
dtop ←CP **DIV** 3 * 3 * 12.5 * 0.1
dtom ←CM **DIV** 5 * 5 * 10 * 0.05
dtodvd ←CDVD **DIV** 2 * 2 * 70 * 0.08

La tienda ofrece un descuento especial para el articulo *Ropita pa pie*, 4% por la compra de dos pares de zapatos y el 50% de descuento por el tercer par y el programa construido debe entregar el valor a pagar por cualquier cantidad comprada de este articulo. El siguiente análisis muestra como se obtienen los términos de la expresión con la cual se halla el valor total a pagar por zapatos.

Si se compra un par de zapatos el valor de este producto es de US $70 dólares; para dos pares se debe aplicar el descuento del 4% (4% = 4/100 o 0.04), es decir:

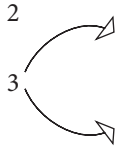
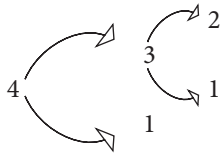
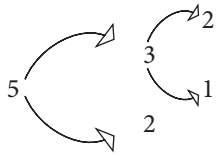
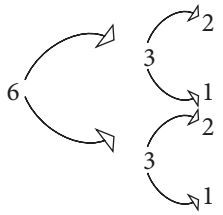
2 x 70	=	140	Valor a pagar sin descuento
2 x 70 x 4 / 100	=	5.6	Valor a pagar del descuento
2 x 70 – 2 x 70 x 4 / 100	=	134.4	Valor a pagar incluido el descuento

Para tres pares la expresión debe incluir el 50% (50% = 50/100 o 0.5) de descuento para el tercer par y el 4% para los otros dos pares, es decir

2 x 70	=	140	Valor a pagar sin descuento
2 x 70 x 4 / 100	=	5.6	Valor a pagar del descuento
2 x 70 – 2 x 70 x 4 / 100	=	134.4	Valor a pagar incluido el descuento
1 x 70 x 0.5	=	35	Valor del tercer par
2 x 70 x 0.96 + 1 x 70 x 0.5	=	169.4	Valor a pagar incluido el descuento

La Tabla 13 muestra cómo se discrimina el cobro según la cantidad de zapatos comprados.

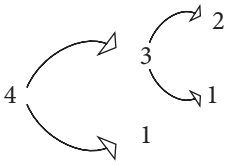
Tabla 13. Análisis de las cantidades de zapatos para el ejemplo 4.

Cantidad		Valor a pagar dólares	
1		1 x 70	= 70
2		2 x 70 x 0.96	= 134.4
			
3	2	2 x 70 x 0.96	= 134.4
	1	1 x 70 x 0.5	= 35
	Total	2 x 70 x 0.96 + 1 x 70 x 0.5	= 169.4
			
4	3 (2, 1)	2 x 70 x 0.96	= 134.4
	1	1 x 70 x 0.5	= 35
		1 x 70	= 70
	Total	2 x 70 x 0.96 + 1 x 70 x 0.5 + 1 x 70	= 239.4
			
5	3 (2, 1)	2 x 70 x 0.96	= 134.4
	2	1 x 70 x 0.5	= 35
		2 x 70 x 0.96	= 134.4
	Total	2 x 70 x 0.96 + 1 x 70 x 0.5 + 2 x 70 x 0.96	= 303.8
			
6	3 (2, 1)	2 x 70 x 0.96	= 134.4
	3 (2, 1)	1 x 70 x 0.5	= 35
		2 x 70 x 0.96	= 134.4
		1 x 70 x 0.5	= 35
	Total	2 x (2 x 70 x 0.96) + 2 x (1 x 70 x 0.5)	= 338.4

El valor a pagar por dos pares de zapatos incluido el descuento se puede representar de la siguiente manera:

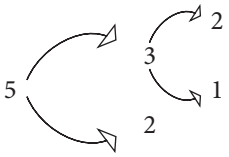
$$2 \text{ x } 70 - 2 \text{ x } 70 \text{ x } 4 / 100 = 2 \text{ x } 70 \ (1 - 0.04) = 2 \text{ x } 70 \text{ x } 0.96$$

De la Tabla 13 que para una cantidad de 3 o más existe en común grupos de 3 pares entre los cuales se encuentra el tercer par que tiene el descuento del 50%, los otros dos pares son cobijados por el descuento del 4% de la tienda. Por ejemplo, cuatro pares se pueden descomponer en 3 y 1, es decir:

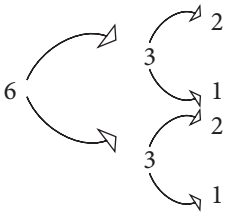


Dos pares con 4% de descuento	2 x 70 x 0.96	= 134.4
Tercer par 50% de descuento	1 x 70 x 0.5	= 35
"Par libre" no tiene descuento	1 x 70	= 70
Total	2 x 70 x 0.96 + 1 x 70 x 0.5 + 1 x 70	= 239.4

Para cinco y seis pares se tiene lo siguiente:



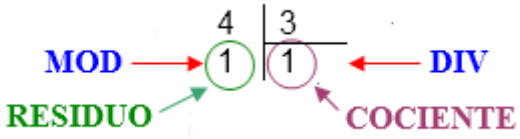
Dos pares con 4% de descuento	2 x 70 x 0.96	= 134.4
Tercer par 50% de descuento	1 x 70 x 0.5	= 35
"Pares libres" no tienen descuento	2 x 70 x 0.96	= 134.4
Total	2 x 70 x 0.96 + 1 x 70 x 0.5 + 2 x 70 x 0.96	= 303.8



Dos pares con 4% de descuento	2 x 70 x 0.96	= 134.4
Tercer par 50% de descuento	1 x 70 x 0.5	= 35
Dos pares con 4% de descuento	2 x 70 x 0.96	= 134.4
Tercer par 50% de descuento	1 x 70 x 0.5	= 35
Total	2 x (2 x 70 x 0.96) + 2 x (1 x 70 x 0.5)	= 338.8

En todos los casos de cualquier cantidad de pares de zapatos, la misma se puede descomponer en grupos de 3, pero por qué esta valor? Se escoge este número debido al descuento del 50% del tercer par, por lo que siempre que se encuentre un grupo de tres se puede garantizar que dentro de éste se halla el par con el 50% de descuento y los pares restantes dentro de este grupo tendrán el 4%, esto significa entonces que se debe construir una expresión que permita obtener el 3 para así generar el valor a pagar por esta cantidad.

Para obtener un 3 y un 1 de un cuatro se realiza el siguiente procedimiento:



Siempre que el **DIV** genere un valor diferente a cero, se asegura que existe un grupo superior o igual a 3 y con certeza se debe cobrar por dos pares de zapatos incluyendo el 4% de descuento en éstos, y el tercer par a la mitad de precio. La expresión para dos pares es

$$4 \textbf{ DIV } 3 * 2 * 70 * 0.96 = 134.4$$

Y para el tercer par es

$$4 \textbf{ DIV } 3 * 70 * 0.5 = 35$$

Según la Tabla 13 para cantidades superiores o iguales a 3 pares de zapatos siempre se puede encontrar grupos de 3 por lo que las dos expresiones anteriores se pueden generalizar para cualquier cantidad de zapatos mayor o igual a 3 pero solo para encontrar el valor a pagar por el grupo, entonces por el grupo de 3 que hacen parte de los cuatro pares el costo a pagar es:

$$4 \textbf{ DIV } 3 * 2 * 70 * 0.96 + 4 \textbf{ DIV } 3 * 70 * 0.5 = 169.4$$

Una compra de cinco pares de zapatos, incluye según la Tabla 13 a un grupo de 3 y "dos pares" libres, por tanto, para este grupo de 3 la expresión es

$$5 \textbf{ DIV } 3 * 2 * 70 * 0.96 + 5 \textbf{ DIV } 3 * 70 * 0.5 = 169.4$$

Las expresiones para los grupos de 3 que hacen parte de la cantidad 4 o 5 pares de zapatos solo se diferencian en el valor 4 en el primer caso y cinco en el último. Para seis pares existen dos grupos de 3 lo que significa que se repite dos veces las expresiones obtenidas para un grupo de 3, la ecuación para 6 pares es

$$6 \textbf{ DIV } 3 * 2 * 70 * 0.96 + 6 \textbf{ DIV } 3 * 70 * 0.5 = 338.4$$

En la expresión anterior el **DIV** genera el 2 logrando esto la duplicidad de cada término en la ecuación.

Entonces para un grupo de 3 se puede generalizar la expresión que permite calcular el precio a pagar por estos tres pares de la siguiente forma:

$$CZ \textbf{ DIV } 3 * 2 * 70 * 0.96 + CZ \textbf{ DIV } 3 * 70 * 0.5$$

Donde CZ representa la variable donde se almacena la cantidad de zapatos ingresada por el usuario.

Hasta el momento la ecuación obtenida permite hallar el valor a pagar por un grupo de 3 en una cantidad de 3 o más pares de zapatos sin embargo aún falta encontrar la expresión para los "pares libres".

Cuatro pares de zapatos incluyen un "par libre", en la operación de división el **MOD** muestra que queda un "par libre" el cual no tiene descuento alguno y por el mismo se debe cancelar US $70 dólares, este valor se obtiene de la siguiente manera

4 **MOD** 3 * 70 = 70

Para cinco pares de zapatos el **MOD** muestra que quedan dos "pares libres" y, según la oferta de la tienda, para esta cantidad se ofrece un descuento del 4% por lo que estos "pares libres" son cobijados por la promoción, entonces la ecuación que permite obtener el valor a pagar es

5 **MOD** 3 * 70 * 0.96 = 134.4

De la Tabla 13 que solo se generan dos tipos de valores para los "pares libres" y son 1 y 2, por lo que se necesita una ecuación que permita hallar el valor a pagar por estos residuos y satisfaga para ambas cantidades. Obsérvese que uno de los residuos es par, por lo que se aprovecha esta característica en la ecuación final construyendo dentro de ella el operador que admita solo pares, y para ello se hace uso del **DIV** 2, entonces

5 **MOD** 3 **DIV** 2 * 2 * 70 * 0.96 = 134.4

5 **MOD** 3 da como resultado 2 luego este junto con el **DIV** 2 genera 1 que significa un par, este valor se multiplica por 2 obteniéndose el 2, número que indica un par de zapatos a los cuales se les aplica luego el precio y el descuento del 4%. La anterior expresión funciona cuando el **DIV** genera un valor diferente de cero significa entonces que el **MOD** genera un valor par, por tanto, solo sirve para cantidades de zapatos impares y superiores a 3.

Para el valor 1 se cambia el **DIV** por el **MOD** quedando

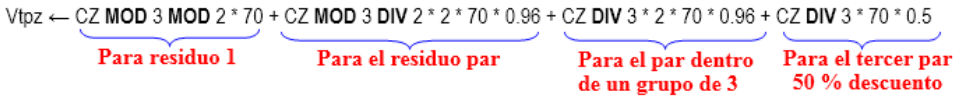
4 **MOD** 3 **MOD** 2 * 70 = 70

Esta ecuación trabaja cuando el primer **MOD** genera como resultado un 1, esto indica que funciona para cualquier cantidad de zapatos que satisfaga el primer **MOD**.

Si se generalizan las expresiones para cantidades de zapatos cuyo valor son 1 y 2 se tiene lo siguiente

CZ **MOD** 3 **DIV** 2 * 2 * 70 * 0.96	para el residuo par
CZ **MOD** 3 **MOD** 2 * 70	para residuo 1

Entonces la expresión que permite obtener el valor a pagar para cualquier cantidad de zapatos incluido el descuento es



Donde CZ es la variable que almacena el valor de la cantidad de zapatos ingresado por el usuario y Vtpz es la variable que almacena el valor total a pagar por este artículo incluido el descuento. En la construcción de Vtpz se elaboró los términos de la misma partiendo de aquellas cantidades de zapatos que son iguales o superiores a 3 y, que los términos para los residuos en cantidades iguales a 4 o 5 pares satisfacen también para la compra de uno o dos pares de zapatos. La Tabla 14 muestra la comprobación de la ecuación anterior.

La expresión para el descuento de los zapatos es

dtoz ← CZ **MOD** 3 **DIV** 2 * 2 * 70 * 0.04 + CZ **DIV** 3 * 2 * 70 * 0.04 + CZ **DIV** 3 * 70 * 0.5

En esta expresión que se incorpora el número 0.04 que corresponde al 4% valor del descuento para dos pares de zapatos.

Las ecuaciones que permiten resolver el problema de la tienda Wall Mart son:

Vtpc ← CC **MOD** 3 * 15 + CC **DIV** 3 * 3 * 15 * 0.9
Vtpp ← CP **MOD** 3 * 12.5 + CP **DIV** 3 * 3 * 12.5 * 0.9
Vtpm ← CM **MOD** 5 * 10 + CM **DIV** 5 * 5 * 10 * 0.95
Vtpdvd ← CDVD **MOD** 2 * 70 + CDVD **DIV** 2 * 2 * 70 * 0.92

Vtpz ← CZ **MOD** 3 **MOD** 2 * 70 + CZ **MOD** 3 **DIV** 2 * 2 * 70 * 0.96 + CZ **DIV** 3 * 2 * 70 * 0.96 + CZ **DIV** 3 * 70 * 0.5

Y para los descuentos son:

dtoc ←CC **DIV** 3 * 3 * 15 * 0.1 dtop ←CP **DIV** 3 * 3 * 12.5 * 0.1
dtom ←CM **DIV** 5 * 5 * 10 * 0.05 dtodvd ←CDVD **DIV** 2 * 2 * 70 * 0.08
dtoz ← CZ **MOD** 3 **DIV** 2 * 2 * 70 * 0.04 + CZ **DIV** 3 * 2 * 70 * 0.04 + CZ **DIV** 3 * 70 * 0.5

Tabla 14. Comprobación de la ecuación Vtpz para el ejemplo 4.

	Primer termino		**Segundo termino**		**Tercer termino**		**Cuarto termino**		
Cant.	CZ **MOD** 3 **MOD** 2 * 70	=?	CZ **MOD** 3 **DIV** 2 * 2 * 70 * 0.96	=?	CZ **DIV** 3 * 2 * 70 * 0.96	=?	CZ **DIV** 3 * 70 * 0.5	**=?**	**Vtpz**
0	0 **MOD** 3 **MOD** 2 * 70	0	0 **MOD** 3 **DIV** 2 * 2 * 70 * 0.96	0	0 **DIV** 3 * 2 * 70 * 0.96	0	0 **DIV** 3 * 70 * 0.5	0	0
1	1 **MOD** 3 **MOD** 2 * 70	70	1 **MOD** 3 **DIV** 2 * 2 * 70 * 0.96	0	1 **DIV** 3 * 2 * 70 * 0.96	0	1 **DIV** 3 * 70 * 0.5	0	70
2	2 **MOD** 3 **MOD** 2 * 70	0	2 **MOD** 3 **DIV** 2 * 2 * 70 * 0.96	134.4	2 **DIV** 3 * 2 * 70 * 0.96	0	2 **DIV** 3 * 70 * 0.5	0	134.4
3	3 **MOD** 3 **MOD** 2 * 70	0	3 **MOD** 3 **DIV** 2 * 2 * 70 * 0.96	0	3 **DIV** 3 * 2 * 70 * 0.96	134.4	3 **DIV** 3 * 70 * 0.5	35	169.4
4	4 **MOD** 3 **MOD** 2 * 70	70	4 **MOD** 3 **DIV** 2 * 2 * 70 * 0.96	0	4 **DIV** 3 * 2 * 70 * 0.96	134.4	4 **DIV** 3 * 70 * 0.5	35	239.4
5	5 **MOD** 3 **MOD** 2 * 70	0	5 **MOD** 3 **DIV** 2 * 2 * 70 * 0.96	134.4	5 **DIV** 3 * 2 * 70 * 0.96	134.4	5 **DIV** 3 * 70 * 0.5	35	303.8
6	6 **MOD** 3 **MOD** 2 * 70	0	6 **MOD** 3 **DIV** 2 * 2 * 70 * 0.96	0	6 **DIV** 3 * 2 * 70 * 0.96	268.8	6 **DIV** 3 * 70 * 0.5	70	338.8
7	7 **MOD** 3 **MOD** 2 * 70	70	7 **MOD** 3 **DIV** 2 * 2 * 70 * 0.96	0	7 **DIV** 3 * 2 * 70 * 0.96	268.8	7 **DIV** 3 * 70 * 0.5	70	408.8

En la Tabla 14 que el tercer y cuarto término de la ecuación para el cálculo de Vtpz generan un valor cuando la cantidad de zapatos es mayor o igual a 3 además, los términos primer y segundo sirven para obtener el valor a pagar tanto por una cantidad inferior a 3 así como para los mayores a este número excepto los múltiplos de 3.

Una vez obtenidas las ecuaciones que permiten resolver el problema se puede continuar con los ***Elementos Involucrados***, es decir, especificar los elementos activos (usuarios) y pasivos (recursos disponibles y/o modificables) que están involucrados en el problema. Elementos Activos, el usuario que ingresará los

datos de las cantidades de cada artículo que desea comprar. Elementos pasivos, las ecuaciones anteriormente descritas.

Todas las expresiones que resuelven el ejemplo 4 están construidas con operadores algorítmicos.

Ahora se aplican las herramientas de la sección 4 para construir el algoritmo.

El algoritmo del ejemplo es:

1. Inicio.
2. Capturar la cantidad de camisas, pantalones, medias, zapatos, DVD y ropa interior que el cliente desea comprar en la tienda Wall Max.
3. Calcular el valor total de cada artículo que se compra, el valor de cada descuento obtenido y el valor total a pagar con base en lo que el cliente desea conseguir de la tienda Wall Max.
4. Visualizar el valor total de cada artículo que se compra, el valor de cada descuento obtenido y el valor total a pagar con base en lo que el cliente desea conseguir de la tienda Wall Max.
5. Fin.

Entonces el pseudocódigo queda:

Algoritmo_tienda_Wall_Max
var
Entero**:** CC, CP, CM, CZ, CDVD, CRI, Vtri
Decimal**:** Vtpc, Vtpp, Vtpm, Vtpz, Vtpdvd, Vtri, dtoc, dtop, dtom, dtoz, dtodvd, Vt
Inicio
Escribir ("Ingrese la cantidad de camisas, pantalones, medias, zapatos, DVD y ropa interior que desea comprar en la tienda Wall Mart")
Leer (CC, CP, CM, CZ, CDVD, CRI)
Vtpc ← CC **MOD** 3 * 15 + CC **DIV** 3 * 3 * 15 * 0.9
Vtpp ← CP **MOD** 3 * 12.5 + CP **DIV** 3 * 3 * 12.5 * 0.9
Vtpm ← CM **MOD** 5 * 10 + CM **DIV** 5 * 5 * 10 * 0.95
Vtpdvd ← CDVD **MOD** 2 * 70 + CDVD **DIV** 2 * 2 * 70 * 0.92
Vtri ← CRI * 45
Vtpz ← CZ **MOD** 3 **MOD** 2 * 70 + CZ **MOD** 3 **DIV** 2 * 2 * 70 * 0.96 + CZ

DIV 3 * 2 * 70 * 0.96 + CZ **DIV** 3 * 70 * 0.5
dtoc ←CC **DIV** 3 * 3 * 15 * 0.1
dtop ←CP **DIV** 3 * 3 * 12.5 * 0.1
dtom ←CM **DIV** 5 * 5 * 10 * 0.05
dtodvd ←CDVD **DIV** 2 * 2 * 70 * 0.08
dtoz ← CZ **MOD** 3 **DIV** 2 * 2 * 70 * 0.04 + CZ **DIV** 3 * 2 * 70 * 0.04 + CZ **DIV** 3 * 70 * 0.5
Vt ← Vtpc + Vtpp + Vtpm + Vtdvd + Vtpz + Vtri
Escribir ("El valor total a pagar por camisas, pantalones, medias, zapatos, DVD y ropa interior de su compra en la tienda Wall Mart es:", Vtpc, Vtpp, Vtpm, Vtpz, Vtpdvd, Vtri)
Escribir ("El valor de descuento de las camisas, pantalones, medias, zapatos y DVD de su compra en la tienda Wall Mart es:", dtoc, dtop, dtom, dtoz, dtodvd)
Escribir ("El valor total a pagar por su compra en la tienda Wall Mart es:", Vt)
Fin

En la siguiente Ilustración 7 se aprecia el diagrama de flujo para este ejemplo.

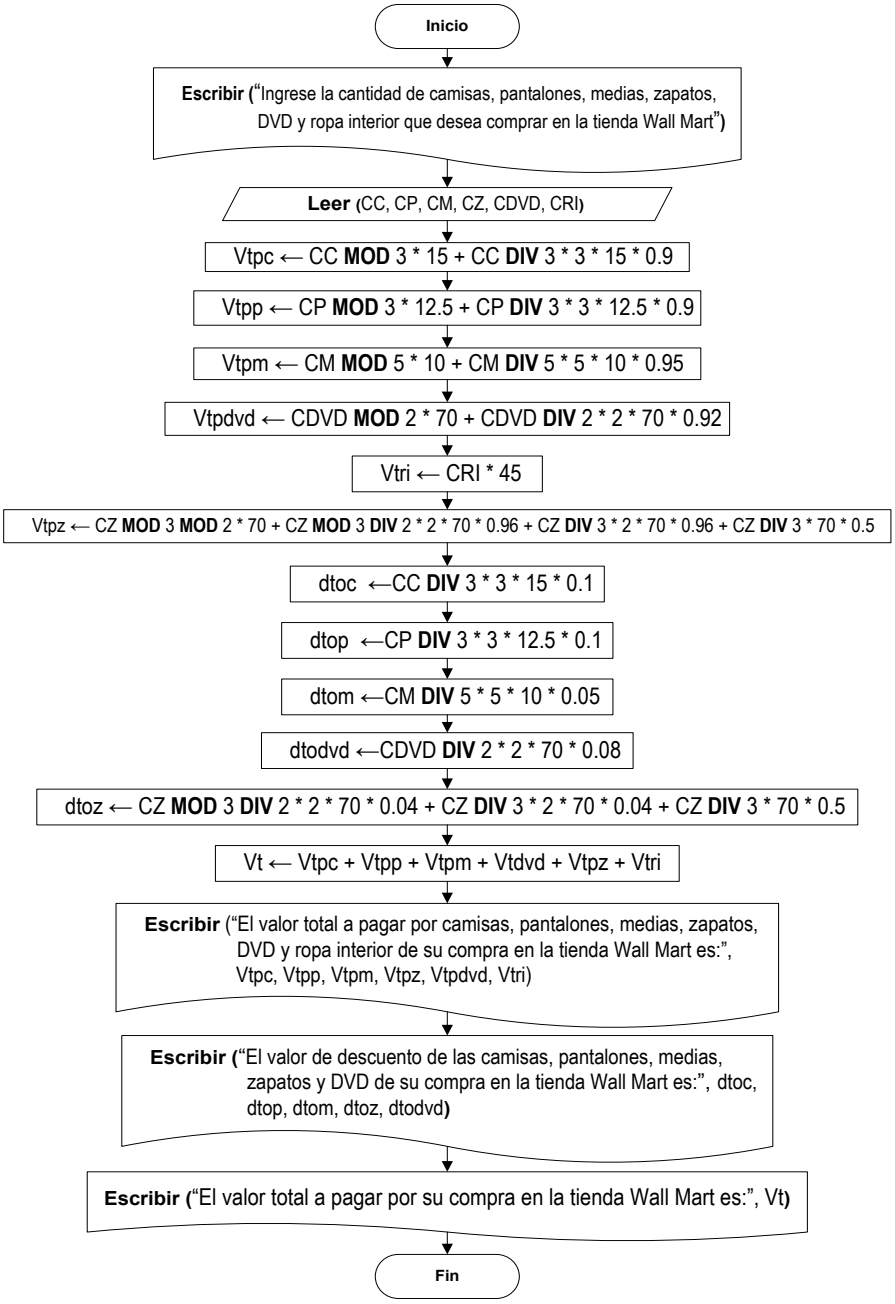


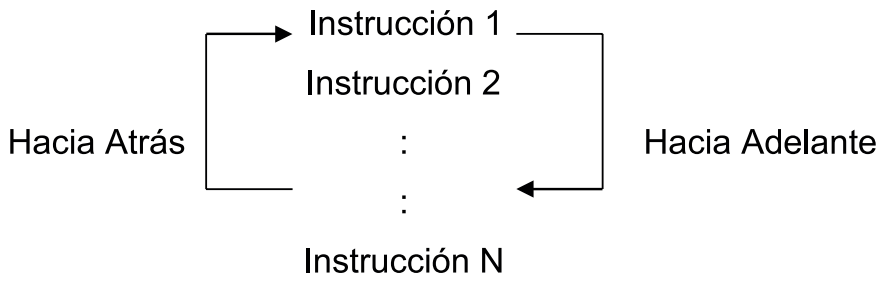
Ilustración 7. Diagrama de flujo para el ejemplo 4.

Capítulo 3

3. INSTRUCCIONES DE SELECCIÓN

En pseudocódigo existe un tipo de instrucción llamada de Bifurcación. Una instrucción de bifurcación obliga a tomar un camino basado en la evaluación de una condición.

Las instrucciones de bifurcación se subdividen en: Estructuras de Selección y Estructuras de Repetición. En el caso de las estructuras de selección, las instrucciones se ejecutan hacia adelante, siguiendo la secuencia del cuerpo de instrucciones del programa; en las estructuras de repetición las instrucciones se pueden ejecutar hacia adelante o hacia atrás dentro del bloque de ejecución del programa.



3.1 ESTRUCTURAS DE SELECCIÓN SIMPLE

La estructura de selección simple, tiene el siguiente formato general en el pseudocódigo, ver Ilustración 8:

Si (**condición**) **entonces**
Instrucción 1
Instrucción 2
:
:
Instrucción N
Fin_si

Ilustración 8. Formato general pseudocódigo para la Estructura de Selección Simple.

La estructura de selección simple contiene tres palabras reservadas: **Si**, **entonces** y **Fin_si**. Siempre comienza con la palabra **Si** y termina con **Fin_si**. Además, contiene un cuerpo de instrucciones entre ambas palabras, y una **condición** dentro de paréntesis. La **condición** al ser evaluada debe siempre generar un valor de tipo booleano: **verdadero** o **falso**.

La **condición** se puede construir de dos formas:

- Usando los operadores relacionales y/o operadores lógicos en una expresión que al ser resuelta genera un valor booleano
- O usando una variable de tipo booleano.

En ambos casos, las variables empleadas en la construcción de la **condición** deben estar declaradas y contener un valor antes de ser evaluadas.

Cuando la **condición** genera un resultado **verdadero** automáticamente se ejecuta de una manera secuencial el cuerpo de instrucciones, al ejecutarse la última instrucción -Instrucción N- el programa continúa con el **Fin_si**, dando por terminada la ejecución del **Si**. A continuación ejecutará las instrucciones que se encuentren después del **Fin_si**. Véase Ilustración 9.

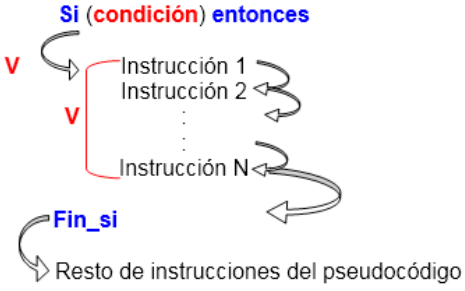


Ilustración 9. Funcionamiento de la estructura de selección simple cuando la condición da Verdadero.

Cuando la **condición** genera un resultado Falso automáticamente se salta a la palabra reservada **Fin_si**, se termina la ejecución del **Si** y se sigue con la ejecución del resto de instrucciones del programa, ver Ilustración 10.

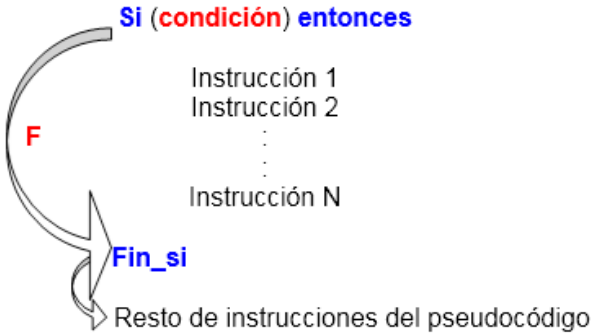


Ilustración 10. Funcionamiento de la estructura de selección simple cuando la condición da Falso.

La Ilustración 11 muestra el formato general en el diagrama de flujo para la estructura de selección simple.

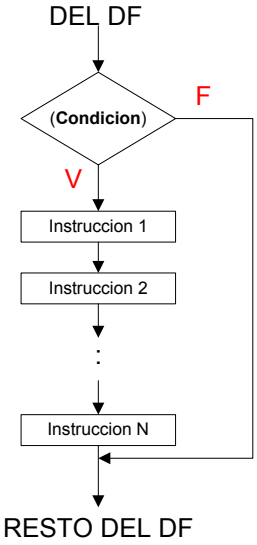


Ilustración 11. Formato general Diagrama de Flujo para las estructura de selección simple.

Note que en el diagrama de flujo de la Ilustración 11 no se coloca las palabras reservadas **Si**, **entonces** y **Fin_si**. El **Fin_si** se representa por la unión de dos flechas. En el camino del Falso no existe instrucción alguna y llega direct-

amente a la flecha que sale de la ejecución de la última instrucción para así continuar con la ejecución del resto de instrucciones. Las palabras DEL DF y RESTO DEL DF indican que existen otras porciones del diagrama de flujo que no se muestra aquí.

3.2 ESTRUCTURAS DE SELECCIÓN COMPUESTA

La estructura de selección compuesta tiene el siguiente formato general en el pseudocódigo, ver Ilustración 12:

Si (**condición**) **entonces**

Instrucción 1
Instrucción 2
:
:
Instrucción N

De lo contrario

Instrucción O
Instrucción P
:
:
Instrucción Z

Fin_si

Ilustración 12. Formato general pseudocódigo para la Estructura de Selección Compuesta.

La Estructura de Selección Compuesta contiene cuatro palabras reservadas: **Si**, **entonces**, **De lo contrario** y **Fin_si**. Siempre comienza con la palabra **Si** y termina con **Fin_si** y contiene dos cuerpos de instrucciones: el primero entre las palabras reservadas **Si** y **De lo contrario** y el segundo, entre las palabras **De lo contrario** y **Fin_si**. Además, contiene una **condición** dentro de paréntesis. La **condición** al ser evaluada debe siempre generar un valor booleano: verdadero o falso.

La siguiente figura ilustra su funcionamiento. Cuando la **condición** genera un resultado **verdadero** se ejecuta el primer cuerpo de instrucciones. El segundo cuerpo de instrucciones se ejecuta cuando la **condición** genera un resultado **falso, ver** Ilustración 13.

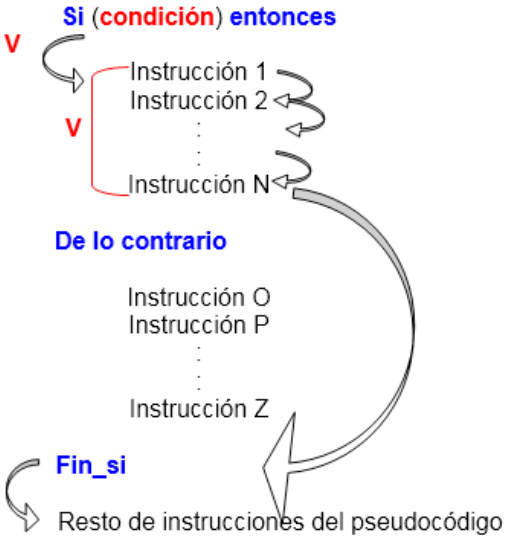


Ilustración 13. Funcionamiento de la estructura de selección compuesta cuando la condición da verdadero.

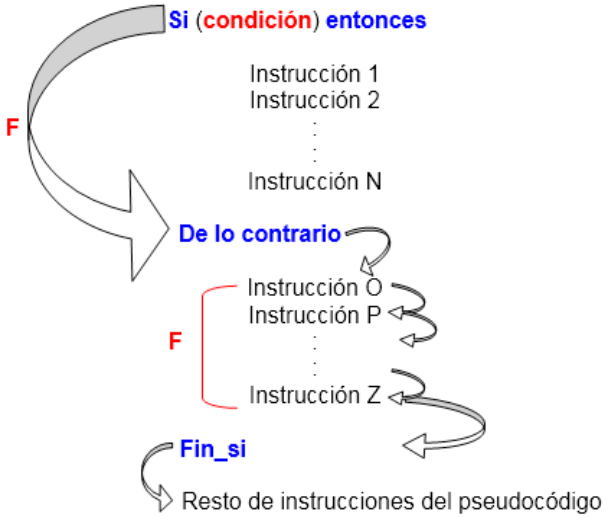


Ilustración 14. Funcionamiento de la estructura de selección compuesta cuando la condición da falso.

La Ilustración 13 y la Ilustración 14, muestran que después de ejecutar la última instrucción del cuerpo de instrucciones correspondiente, el programa salta inmediatamente a ejecutar el **Fin_si** y sigue con la ejecución del resto de instrucciones.

La **condición** de la estructura de selección compuesta se construye de igual forma que la **condición** de la estructura de selección simple.

La siguiente Ilustración 15 muestra el formato general en el diagrama de flujo para la estructura de selección compuesta.

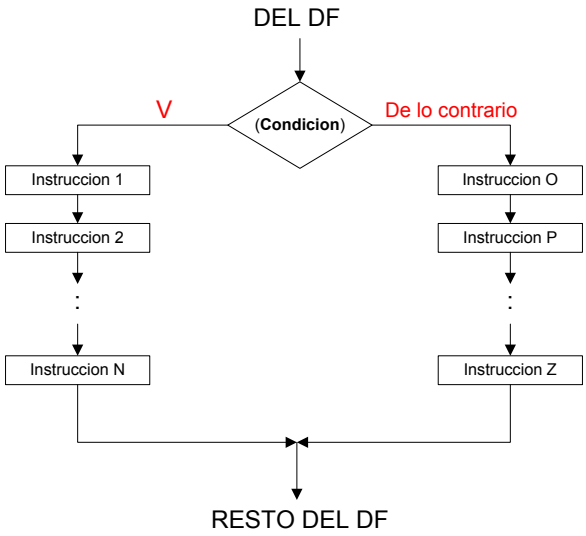


Ilustración 15. Formato general diagrama de flujo estructura de selección compuesta.

3.3 ESTRUCTURAS DE SELECCIÓN MÚLTIPLE

La estructura de selección múltiple tiene el siguiente formato general en el pseudocódigo, ver Ilustración 16:

Si (**selector**) **igual**

Valor 1 **:** Instrucción 1
Instrucción 2
:
Instrucción N

```
        Valor 2 :        Instrucción 1
                         Instrucción 2
                               :
                         Instrucción N

                  :              :
        Valor N :        Instrucción 1
                         Instrucción 2
                               :
                         Instrucción N

   De lo contrario :     Instrucción 1
                         Instrucción 2
                               :
                         Instrucción N

Fin_si
```

Ilustración 16. Formato general pseudocódigo para la estructura de selección múltiple.

La estructura de selección múltiple utiliza un **selector** en vez de una **condición**. Está conformada por las palabras reservadas: **Si**, **igual**, **De lo contario** y **Fin_si**. Entre las palabras **Si** y **Fin_si,** al lado izquierdo de dos puntos, se coloca toda la gama de valores posibles que puede generar el **selector**. Al lado derecho de los dos puntos se colocan las instrucciones que se deben ejecutar cuando exista una igualdad entre el valor generado por el **selector** y el valor que se encuentra al lado izquierdo de los dos puntos. Algunos autores utilizan **Según sea**, **haga** y **Fin_según_sea** en vez de **Si**, **igual** y **Fin_si**.

El **selector** es una variable o una expresión conformada por operadores que al ser resuelta genera un valor. El valor y el tipo de dato se comparan con Valor 1, Valor 2,…, Valor N y si es igual a alguno de ellos, el programa ejecuta el cuerpo de instrucciones que se encuentra a su lado. Cuando se ejecuta la última instrucción, el programa automáticamente salta a ejecutar el **Fin_si** y luego sigue con el resto de instrucciones del programa.

Si el valor generado por el **selector** no se encuentra dentro de la gama de valores posibles, automáticamente el programa busca la opción **De lo contrario** y ejecuta el cuerpo de instrucciones asociado a ella. El **De lo contrario** es opcional, esto significa que no es obligatorio colocarlo.

La(s) variable(s) que forman el **selector** deben: estar declaradas, contener un valor y coincidir en tipo con la gama de valores posibles.

La siguiente Ilustración 17 muestra uno de los posibles funcionamientos de la estructura de selección múltiple:

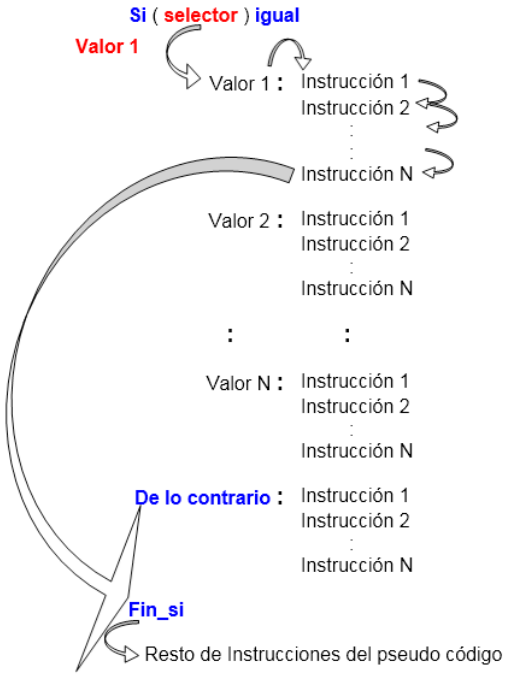


Ilustración 17. Funcionamiento de la estructura de selección múltiple cuando selector genera Valor 1.

La siguiente Ilustración 18 muestra el funcionamiento cuando **selector** genera un dato que no coincide con ninguno de los valores, Valor 1, Valor 2, … , Valor N.

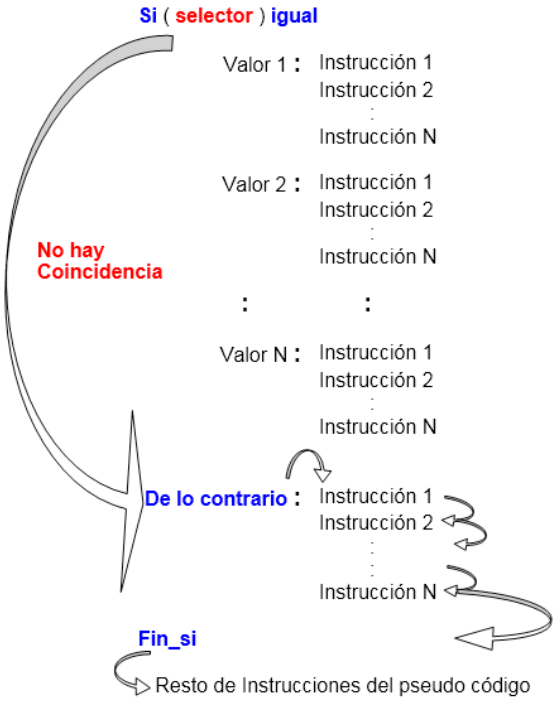


Ilustración 18. Funcionamiento cuando selector genera un valor que no coincide con ninguno de los valores dentro de la estructura de selección múltiple.

La siguiente Ilustración 19 muestra el funcionamiento cuando **selector** genera un dato que no coincide con ninguno de los valores, Valor 1, Valor 2, … , Valor N y no existe **De lo contrario**.

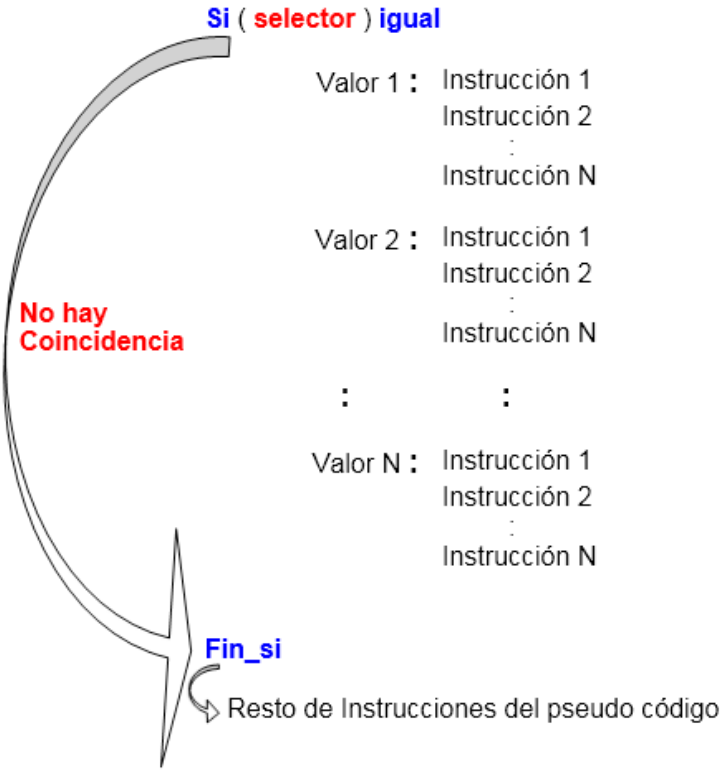


Ilustración 19. Funcionamiento cuando selector genera un valor que no coincide con ninguno de los valores y tampoco existe el De lo contario.

Al no existir coincidencia salta automáticamente al **Fin_si**, cerrándose la estructura, luego se sigue la ejecución del resto de instrucciones del pseudocódigo.

La siguiente Ilustración 20 muestra el formato general en el diagrama de flujo para la estructura de selección múltiple.

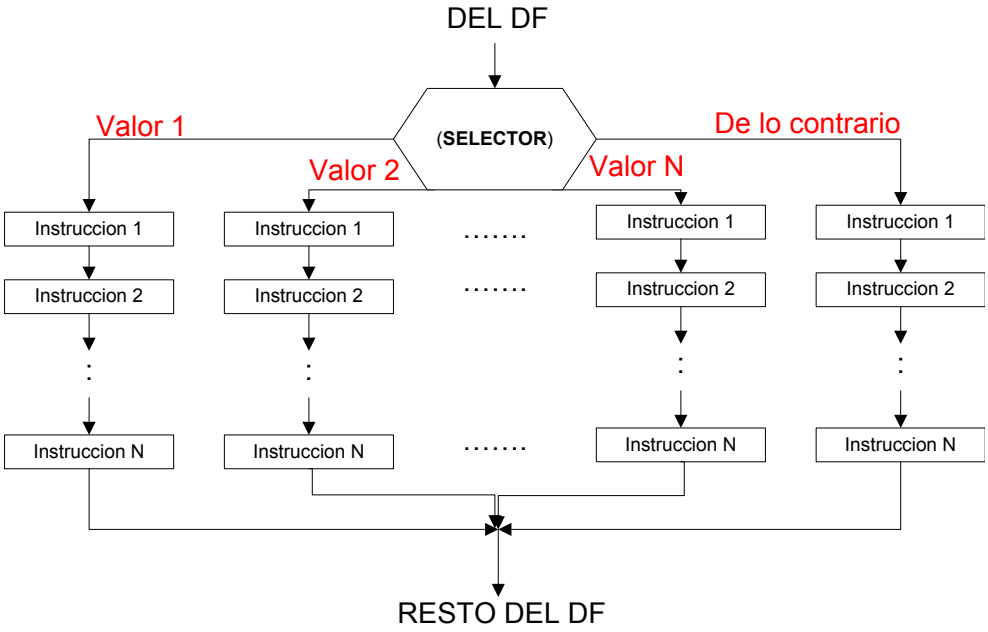


Ilustración 20. Formato general Diagrama de Flujo para la estructura de selección múltiple.

3.4 ESTRUCTURAS DE SELECCIÓN ANIDADAS

La estructura de selección anidada no tiene un formato general en el pseudocódigo, ni en el diagrama de flujo. Consiste en utilizar las estructuras anteriormente mencionadas dentro de alguna o varias de ellas mismas, por ejemplo, utilizar varias estructuras de selección simple dentro de otra estructura de selección simple, o dentro de una estructura de selección compuesta, o dentro de una estructura de selección múltiple, o viceversa. No existe una normal general en la construcción de las expresiones de selección anidadas ni en la cantidad de expresiones de selección que se pueden colocar.

Aspectos a tener en cuenta para construir una expresión de selección anidada:

- Deben existir tantos **Fin_si** como **Si** existan. Debe tenerse presente que un **Fin_si** nunca cierra más de un **Si**.
- El **Fin_si** más interno le corresponderá única y exclusivamente al último **Si** más interno.

- Claridad de dónde colocar cada **Si** y cada **Fin_si**, sino se ubican bien algunos **Si** podrían depender de otros **Si** y por tanto no ejecutarse apropiadamente o nunca ejecutarse.

Las siguientes ilustraciones muestran algunos ejemplos:

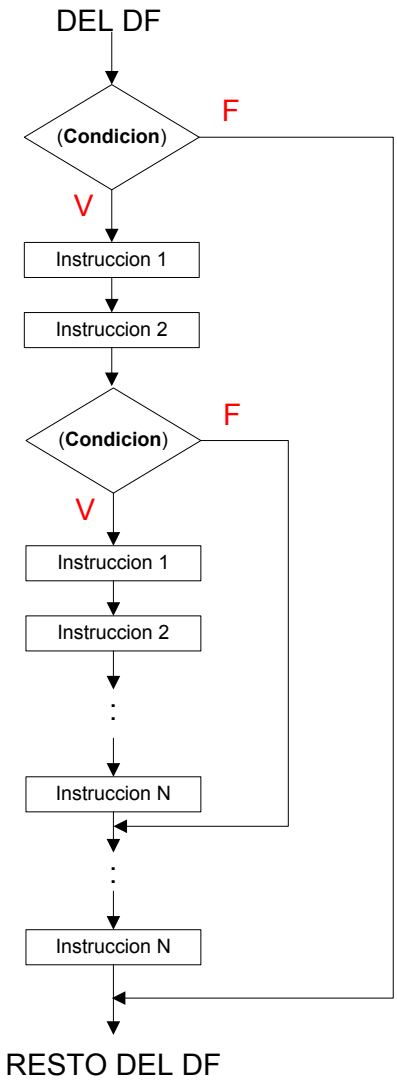


Ilustración 21. Estructura de selección anidada formada por una expresión de selección simple dentro de otra.

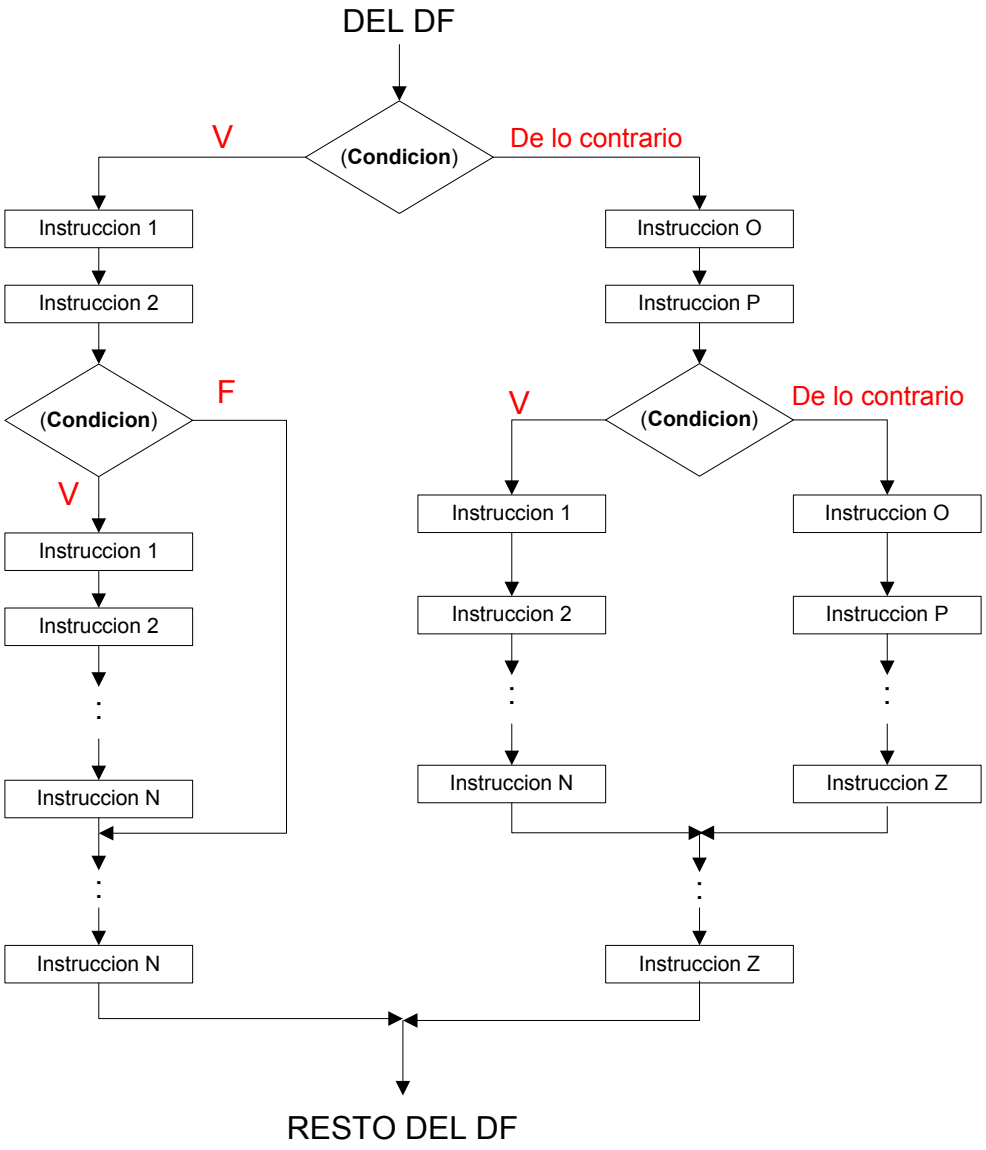


Ilustración 22. Estructura de selección anidada conformada por una estructura de selección simple y una estructura de selección compuesta dentro de una estructura de selección compuesta.

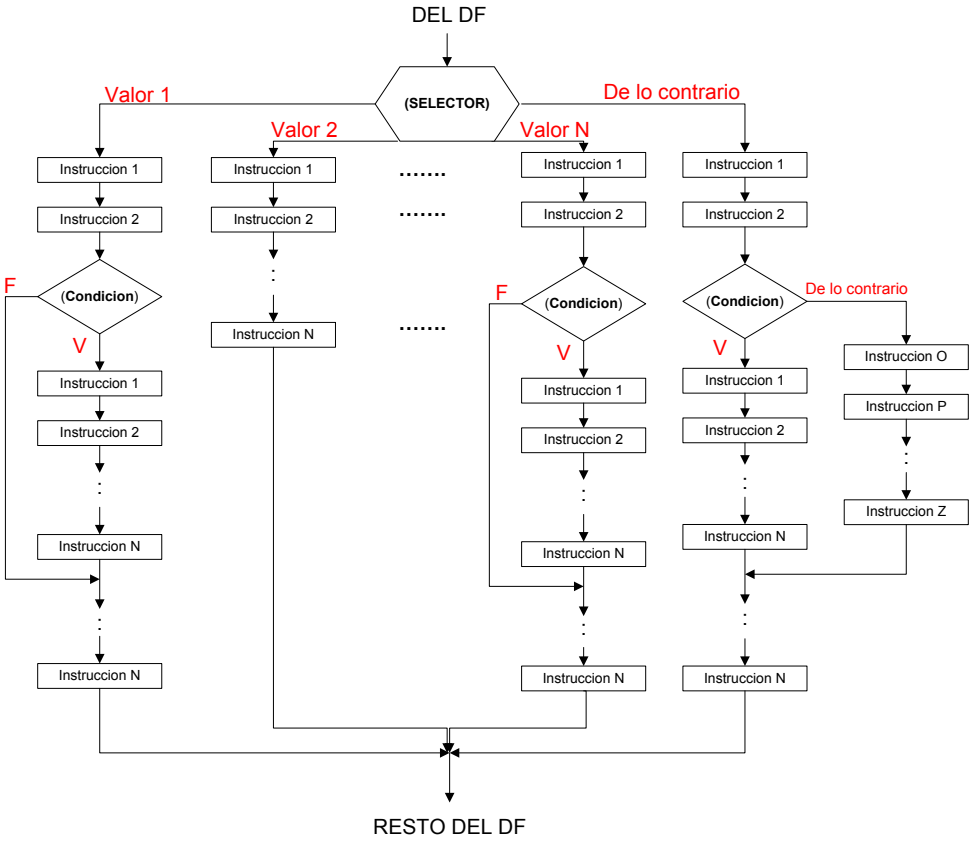


Ilustración 23. Estructura de selección anidada conformada por dos estructuras de selección simple, una estructura de selección compuesta dentro de una estructura de selección múltiple.

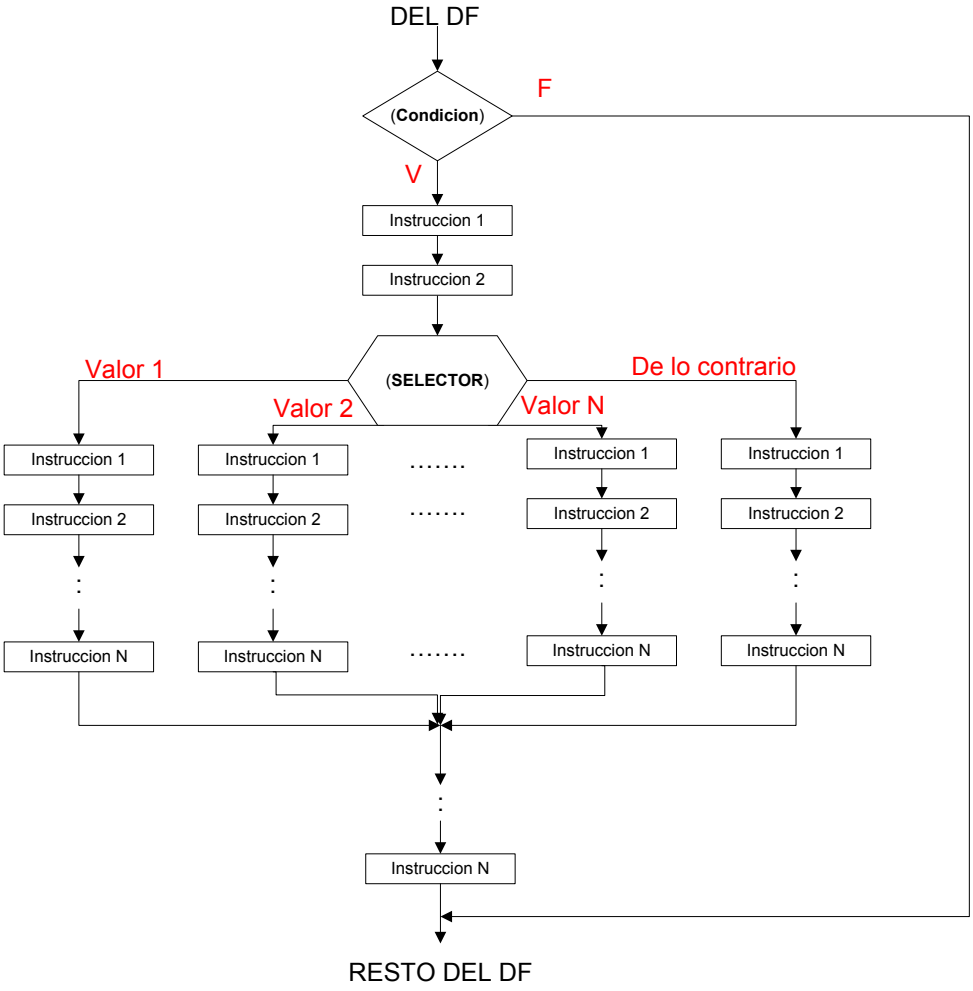


Ilustración 24. Estructura de selección anidada conformada por una estructura de selección múltiple dentro de una estructura de selección simple.

3.5 CONSTRUCCIÓN DE ESTRUCTURAS DE SELECCIÓN

A continuación se presentan una serie de ejemplos, a manera de guía, para ayudar a comprender al lector cuándo es necesario el uso de una estructura de selección y cómo construir su **condición**.

1. Si la palabra **SI** se encuentra en el enunciado y está vinculada directamente con el problema, inmediatamente se debe pensar en el uso de una estructura de selección. Algunos enunciados proveen de los elementos necesarios para construir la condición y éstos están vinculados directamente con la palabra **SI** y con el problema.

Construya un pseudocódigo tal que conocido el sueldo de un empleado, calcule el incremento del mismo basado en lo siguiente: **si** el *sueldo es inferior a $100000, aumente el 15%*; **si** el *sueldo está entre $100000 y $1.000.000, incluidos, aumente el 8%* y; **si** el *sueldo supera el $1.000.000, aumente el 3%*.

La frase donde se encuentra la palabra **SI** en el ejemplo nos muestra la **condición** y se puede construir teniendo en cuenta lo que dice la misma. Por ejemplo, "Si sueldo es inferior a 100000", indica que la **condición** es: Sueldo < 100000, y parte de la estructura de selección sería:

Si (Sueldo < 100000) **entonces**…

Las instrucciones que se colocan después del **entonces** son todas aquellas que deben ejecutar cuando la **condición** sea verdadera en este caso, el cálculo del aumento del sueldo.

El mismo procedimiento se puede aplicar con las otras frases para construir las demás condiciones.

2. Si en el enunciado no aparece la palabra **SI** y se deduce del mismo que el problema se resuelve haciendo comparaciones, entonces se debe hacer uso de una expresión de selección. En este caso, la **condición** contendrá los identificadores que contienen los números o valores a comparar.

Dados tres números enteros positivos diferentes construya un pseudocódigo que permita determinar cuál es el mayor de ellos.

Para resolver el problema se debe comparar cada número con los otros dos por lo cual se hace necesario el uso de una estructura de selección.

El enunciado comienza con "dados tres números…", lo cual significa que los genera el usuario y éstos deben guardarse en sus respectivas variables. Para construir la condición se usan los identificadores con el correspondiente operador relacional. Por ejemplo una **condición** podría ser:

Si (A > B Y A > C) **entonces**…

La anterior **condición** genera un resultado verdadero cuando el número almacenado en la variable A es mayor que los otros dos y la instrucción que se coloca después del **entonces** es la de mostrar en pantalla el número A como mayor.

A continuación se muestra el fragmento de pseudocódigo que contiene la estructura de selección completa:

```
Si ( A > B   Y   A > C) entonces
        Escribir (" El mayor es:", A)
De lo contrario
        Si ( B > A   Y  B > C) entonces
                Escribir (" El mayor es:", B)
        De lo contrario
                Escribir (" El mayor es:", C)
        Fin_si
Fin_si
```

3. Las estructuras de selección múltiple se emplean cuando el enunciado muestra que al evaluar una misma expresión o variable se generan diferentes valores y, a cada valor generado le corresponde la ejecución de una acción diferente. Sin embargo, cualquier estructura de selección podría ser empleada para el mismo fin, simplemente se obtendría una solución más extensa.

Construya un pseudocódigo que permita realizar operaciones aritméticas elementales, según sea la clave ingresada:

CLAVE	**OPERACIÓN**
+	SUMA
-	RESTA
*	MULTIPLICACION
/	DIVISION

Observe que en la variable llamada CLAVE se almacenará un dato tipo carácter y podrá contener cualquiera de los cuatro mostrados en la tabla. Por ejemplo, cuando CLAVE contenga símbolo **+** se debe realizar una suma. La **condición** para este ejemplo sería:

Si (CLAVE) **igual** …

Después del **igual** se colocan toda la gama de valores posibles que puede almacenar la variable CLAVE (+, -, *, /).

A continuación se muestran dos posibles soluciones para el mismo problema, usando estructuras de selección diferentes:

```
Si ( CLAVE) igual
 " + " :suma ← A + B
        Escribir ( " La operación", CLAVE,
        "da:", suma)
 " – " :   resta ← A - B
        Escribir ( " La operación", CLAVE,
        "da:", resta)
 " * " :   mult ← A * B
        Escribir("La operación", CLAVE,
        "da:", mult)
 " / " :   divi ← A / B
        Escribir("La operación", CLAVE,
        "da:", divi)
De lo contrario:Escribir("Clave no valida ")
Fin_si
```

```
Si ( CLAVE = " + ") entonces
  suma ← A + B
  Escribir ( " La operación", CLAVE, "da:",
             suma)
Fin_si
Si ( CLAVE = " – ") entonces
  resta ← A - B
  Escribir ( " La operación", CLAVE, "da:",
             resta)
Fin_si
Si ( CLAVE = " * ") entonces
  mult ← A * B
  Escribir ( " La operación", CLAVE, "da:",
             mult)
Fin_si
Si ( CLAVE = " / ") entonces
   divi ← A / B
   Escribir("La operación", CLAVE, "da:",
            divi)
De lo contrario
   Escribir  ( "Clave no valida ")
Fin_si
```

4. Si el problema se resuelve con la ayuda de expresiones aritméticas, se debe identificar qué valores pueden generar errores aritméticos que hagan que la expresión no pueda calcularse. Si se encuentra al menos un valor que genere error se debe incluir una estructura de selección.

- En el siguiente ejemplo, si la variable C toma el valor de cero, la operación no tiene solución, existe un error de división por cero.

$$\frac{A+B}{C}$$

Los valores de C para los cuales la operación aritmética no genera error son todos aquellos para los cuales se satisface C < > 0, la **condición** sería entonces:

Si (C < > 0) **entonces**

A continuación se muestra el fragmento de pseudocódigo correspondiente a la estructura de selección simple, que soluciona este problema.

Si (C < > 0) **entonces**
    Resultado ← (A + B) / C
    Escribir (" El valor de la expresión es:", Resultado)
Fin_si

En caso de querer mostrar un mensaje de error, se puede emplear una estructura de selección compuesta como muestra el siguiente fragmento de pseudocódigo:

Si (C < > 0) **entonces**
    Resultado ← (A + B) / C
    Escribir (" El valor de la expresión es:", Resultado)
De lo contrario
    Escribir (" El valor ingresado para C es incorrecto")
Fin_si

- Otro error son las raíces cuadradas de números negativos. La siguiente expresión aritmética posee variables dentro de una raíz cuadrada, si $B+3*C<0$, la raíz no tiene solución por tanto, se debe usar una estructura de selección.

$$\sqrt{(B+3*C)}$$

Los valores para los cuales la operación aritmética no genera error son todos aquellos que satisfacen $B+3*C>0$, parte de la estructura de selección sería:

Si ($B+3*C>0$) **entonces**…

Las instrucciones que se colocan después del **entonces**, son todas aquellas que se deben ejecutar cuando la **condición** genera un resultado **verdadero**, en los casos anteriores, la expresión matemática a resolver.
A continuación se muestra el fragmento de pseudocódigo que representa la estructura de selección completa:

Si ($B+3*C>0$) **entonces**
    Resultado ← ($B+3*C$) ** 0.5

Escribir (“ El valor de la expresión es:”, Resultado)
Fin_si

5. Consiste en encontrar todos los valores no deseados que puedan generar resultados con errores de interpretación. Son los más difíciles de encontrar y normalmente no se hallan explícitamente en los enunciados. Los valores que crean los casos críticos son generados por errores de digitación del usuario, por suposición de datos que nunca deben ser ingresados pero que pueden ser digitados, etc.

Los casos críticos implican que el programador debe pensar en lo “obvio” y en las “suposiciones” para así evitar que ellas aparezcan en algún momento dado de la ejecución del programa y generen error en la interpretación de los resultados obtenidos.

Si se encuentra al menos un caso crítico, se debe usar una estructura de selección y la **condición** estará formada por una expresión que contenga todos los valores posibles que generen error.

Conocido el sueldo de un empleado, construya un pseudocódigo que calcule e imprima su nuevo sueldo, sabiendo que si el sueldo del empleado es inferior a $100000 se le debe aumentar el 15%.

Dentro del enunciado se observa la **SI**, se debe construir una estructura de selección cuya condición sería la siguiente:

Sueldo < 100000

Al construir las expresiones aritméticas se comprueba que ninguna generaría error independientemente del valor del Sueldo.

NSueldo = Sueldo + Aumento
Aumento = Sueldo x 0.15

Sin embargo, un sueldo negativo generaría un error de interpretación. Se construye una estructura de selección cuya condición sería:

Si (Sueldo > = 0) **entonces**

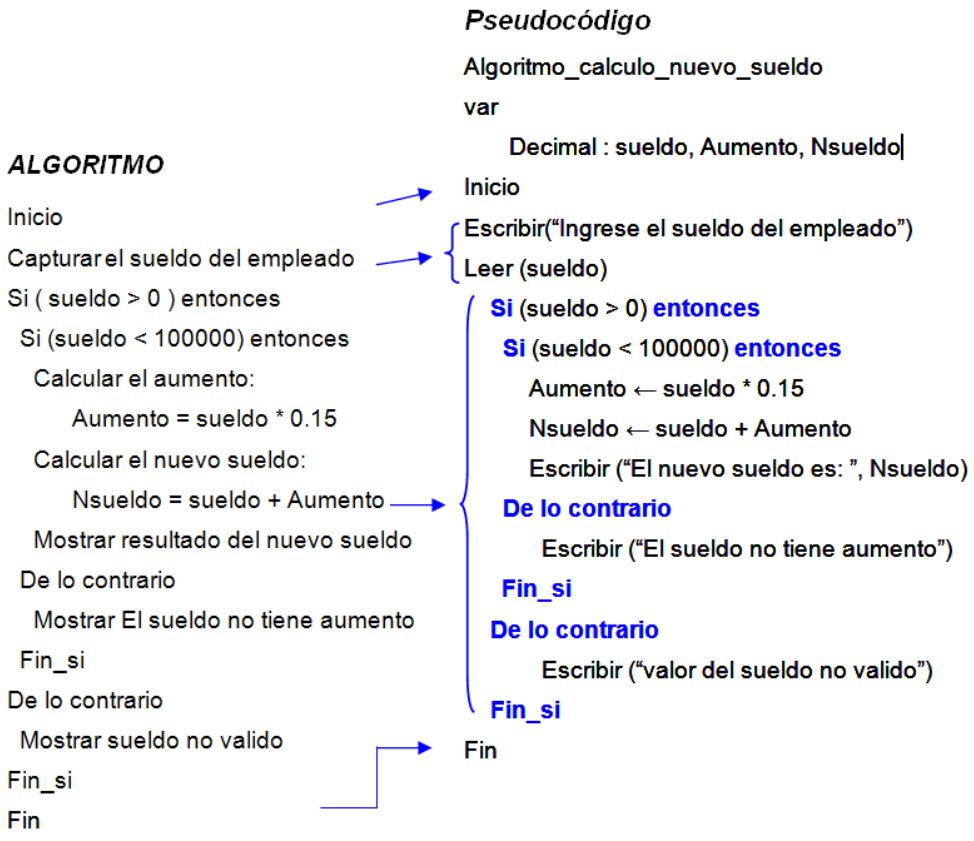
Normalmente si se encuentran casos críticos, las estructuras de selección deben colocarse inmediatamente después de haber sido capturado el dato que se

evaluará en la **condición**; en el ejemplo anterior, inmediatamente después de que el usuario ingrese el Sueldo.

A continuación se muestra el fragmento de pseudocódigo que representa la estructura de selección completa:

```
Si (Sueldo > = 0) entonces
        Si (Sueldo < 100000) entonces
                Aumento ← Sueldo * 0.15
                NSueldo ← Sueldo + Aumento
                Escribir (" El nuevo sueldo es:", NSueldo)
        Fin_si
Fin_si
```

El algoritmo completo y su equivalente en pseudocódigo quedarían como sigue:



La siguiente solución muestra un error común en la aplicación de las estructuras de selección.

Pseudocódigo –versión dos–

```
Algoritmo_calculo_nuevo_sueldo
var
    Decimal : sueldo, Aumento, Nsueldo
Inicio
Escribir("Ingrese el sueldo del empleado")
Leer (sueldo)
Si(sueldo>0 Y sueldo<100000) entonces
    Aumento ← sueldo * 0.15
    Nsueldo ← sueldo + Aumento
De lo contario
    Escribir("El sueldo no tiene aumento")
Fin_si
Escribir ("El nuevo sueldo es: ", Nsueldo)
Fin
```

La primera versión del pseudocódigo cumple con los requerimientos establecidos en el algoritmo, en la segunda versión se reúnen las dos condiciones en una sola. Cuando no se cumple esta condición, no se calcula Nsueldo por lo tanto, no habrá valor alguno para visualizar al ejecutarse la penúltima instrucción, lo cual genera un error.

En la siguiente Ilustración 25 se muestra el diagrama de flujo para este problema relacionándolo con el pseudocódigo por medio de flechas.

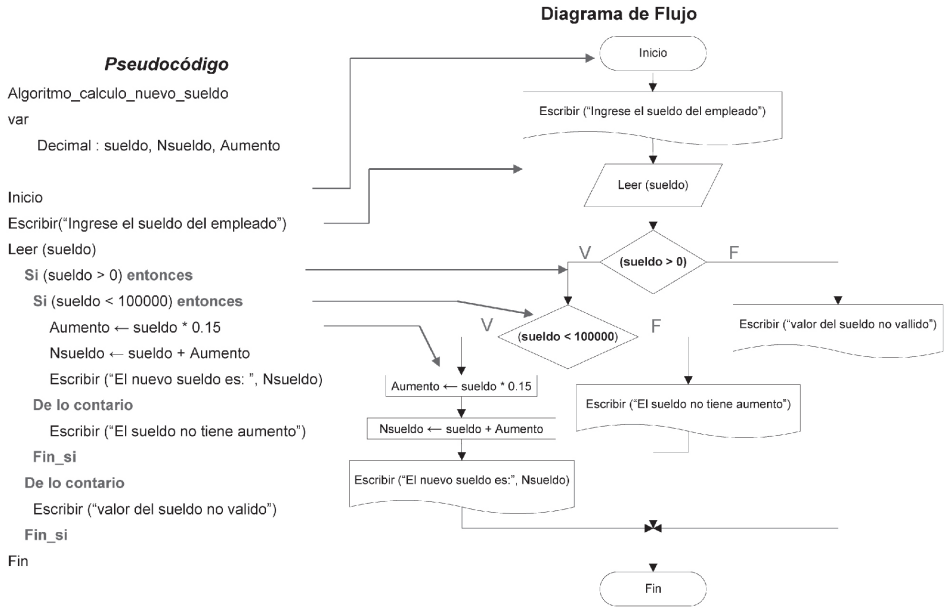


Ilustración 25. Relación entre pseudocódigo y diagrama de flujo usando estructuras de control.

3.6 EJERCICIOS

3.6.1 Ejercicios con Respuesta

1. Dados tres números enteros positivos diferentes construya un pseudocódigo que permita determinar cuál es el mayor de ellos.

2. Construya un pseudocódigo que permita realizar operaciones aritméticas elementales, según sea la clave ingresada:

CLAVE	OPERACIÓN
+	SUMA
-	RESTA
*	MULTIPLICACION
/	DIVISION

Cada operación requiere de dos operandos –números–. Muestre la clave ingresada y el resultado de la operación.

3. Una compañía dedicada al alquiler de automóviles cobra un monto fijo de $300 para los primeros 300 Km. de recorrido. Para más de 300 Km. y hasta 1000 Km., cobra un monto adicional de $15 por cada kilómetro en exceso sobre 300. Para más de 1000 Km. cobra un monto adicional de $10 por cada kilómetro en exceso sobre 1000. Los precios no incluyen el 16% del impuesto al valor agregado, IVA. Diseñe un pseudocódigo que determine el monto a pagar por el alquiler de un vehículo y el monto incluido del impuesto.

3.6.2 Ejercicios sin Respuesta

4. Dado cualquier par de números enteros construya un pseudocódigo que reste siempre el menor del mayor.

5. Construya un pseudocódigo que indique si un número es par o impar.

6. Si representamos los días de la semana con dígitos tendremos que el 1 corresponde al lunes, el 2 al martes y así sucesivamente. Dado un entero cualquiera, construya un pseudocódigo que muestre el nombre del día de la semana al que corresponde.

7. Construya un pseudocódigo que dada una letra cualquiera indique si se trata de una vocal o no.

8. Construya un pseudocódigo que redondee un número es decir, que convierta un número decimal a su equivalente entero.

9. Construya un pseudocódigo que dado dos valores **X** y **Y**, verifique si las siguientes ecuaciones arrojan el mismo resultado: a) **X** + 3**Y**
   b) 2**X** – 5**Y**

3.6.3 Respuesta a los Ejercicios

1. Dados tres números enteros positivos diferentes construya un pseudocódigo que permita determinar cuál es el mayor de ellos.

   Primero se determina si es necesario usar alguna estructura de selección. Aunque el enunciado no contiene la palabra SI, pide encontrar el mayor de tres números, para ello es necesario comparar un número con los dos restantes y, determinar si éste es o no mayor. Como se requiere el uso de comparaciones para resolver el problema entonces se debe usar una estructura de selección. Las **condiciones** de cada una de las estructuras es:

   Si (A > B Y A > C) **entonces**
   Si (B > A Y B > C) **entonces**
   Si (C > A Y C > B) **entonces**

   Se deben usar a lo máximo tres comparaciones (tres condiciones) una para cada número. En las variables A, B y C se guardaran los números ingresados por el usuario.

   No se necesitan estructuras de selección múltiple porque no existen más de dos caminos a escoger con una misma expresión. Y el ejercicio no exige el uso de operaciones aritméticas por lo no habrán errores de este tipo.

   El ejercicio específica "números enteros positivos" por lo que no son válidos los números negativos, la ***condición*** de la estructura quedaría:

   Si (A >= 0 Y B >= 0 Y C >=0) **entonces**

Además, el ejercicio exige hallar el mayor de tres números diferentes por lo que los números no son válidos si son iguales, se requiere de la siguiente estructura de selección:

Si (A <> B Y A <> C Y B <> C) **entonces**

Estas estructuras se colocan justo después de capturarse los tres números.

Para este ejercicio se mostraran dos versiones del algoritmo y de pseudocódigo, la primera usa estructuras de selección simples.

ALGORITMO usando ESS

```
Inicio
Capturar tres números enteros positivos.
Si (A>=0 Y B >= 0 Y C>=0) entonces
 Si (A<> B Y A<>C Y B<>C) entonces
  Si (A > B Y A > C) entonces
   Mostrar el mayor es A
  Fin_si
  Si (B > A Y B > C) entonces
   Mostrar el mayor es B
  Fin_si
  Si (C > A Y C > B) entonces
   Mostrar el mayor es C
  Fin_si
 Fin_si
 Si (A = B O A = C O B = C)entonces
  Mostrar existen números iguales
 Fin_si
Fin_si
Si (A < 0 O B < 0 O C < 0) entonces
    Mostrar numero no valido
Fin_si
Fin
```

Pseudocódigo usando ESS

```
Algoritmo_encuentra_mayor
var
     Entero : A,B,C
Inicio
 Escribir("Ingrese tres números enteros positivos
          diferentes")
 Leer (A,B,C)
 Si (A >= 0 Y B >= 0 Y C >= 0) entonces
    Si (A<>B Y A<>C Y B<>C) entonces
       Si (A > B Y A > C) entonces
          Escribir ("El numero mayor es:",A)
       Fin_si
       Si (B > A Y B > C) entonces
          Escribir ("El numero mayor es:",B)
       Fin_si
       Si (C > A Y C > B) entonces
          Escribir ("El numero mayor es:",C)
       Fin_si
    Fin_si
    Si ( A = B O A = C O B = C) entonces
       Escribir ("Existen números iguales")
    Fin_si
 Fin_si
 Si ( A < 0 O B < 0 O C < 0) entonces
       Escribir ("numero no valido")
 Fin_si
Fin
```

Se requieren de 7 estructuras de selección para resolver el problema, dos de ellas son utilizadas para informar al usuario que ocurre cuando ingresa un dato que no es valido (números negativos) ó cuando digita dos o tres números iguales. La ausencia de estas dos estructuras no afecta la solución del problema pero es importante incluirlas en la solución.

La segunda versión utiliza estructuras de selección compuesta. En la construcción del pseudocódigo el programador es libre de escoger la solución que utilizará, generalmente se usa aquella que tenga menos líneas de código.

ALGORITMO usando ESC

```
Inicio
Capturar tres números enteros positivos.
Si ( A>=0 Y B >= 0 Y C>=0 ) entonces
 Si (A<> B Y A<>C Y B<>C) entonces
  Si (A > B Y A > C) entonces
      Mostrar el mayor es A
  De lo contrario
    Si (B > A Y B > C) entonces
        Mostrar el mayor es B
    De lo contrario
        Mostrar el mayor es C
    Fin_si
  Fin_si
 De lo contrario
  Mostrar existen números iguales
 Fin_si
De lo contrario
   Mostrar numero no valido
Fin_si
Fin
```

Pseudocódigo usando ESC

```
Algoritmo_encuentra_mayor
var
     Entero : A,B,C
Inicio
 Escribir("Ingrese tres números enteros positivos
          diferentes")
 Leer (A,B,C)
   Si (A>=0 Y B>=0 Y C>=0) entonces
    Si (A<>B Y A<>C Y B<>C) entonces
     Si (A > B Y A > C) entonces
        Escribir ("El numero mayor es:",A)
     De lo contrario
      Si (B > A Y B > C) entonces
        Escribir ("El numero mayor es:",B)
      De lo contrario
        Escribir ("El numero mayor es:",C)
      Fin_si
     Fin_si
    De lo contario
        Escribir ("Existen números iguales")
    Fin_si
   De lo contario
        Escribir ("numero no valido")
   Fin_si
Fin
```

Comparando las dos soluciones se puede apreciar que aquella que utiliza estructuras de selección compuesta tiene ventajas por que usa menos líneas de código y, contiene tan sólo 4 estructuras de selección. Se puede aprovechar la zona del **Falso** para determinar cuándo el número mayor es el almacenado en la variable C, obsérvese que se llega a este **De lo contrario** cuando se han descartado las opciones anteriores.

Es importante recordar que un **Si** sólo puede tener un **De lo contrario** y debe tener un **Fin_si**, si esto no se tiene en cuenta se podría construir la solución de un problema utilizando más de un **De lo contrario** dentro de un solo **Si**.

A continuación la Ilustración 26 y la Ilustración 27 muestra la solución del problema usando diagramas de flujo.

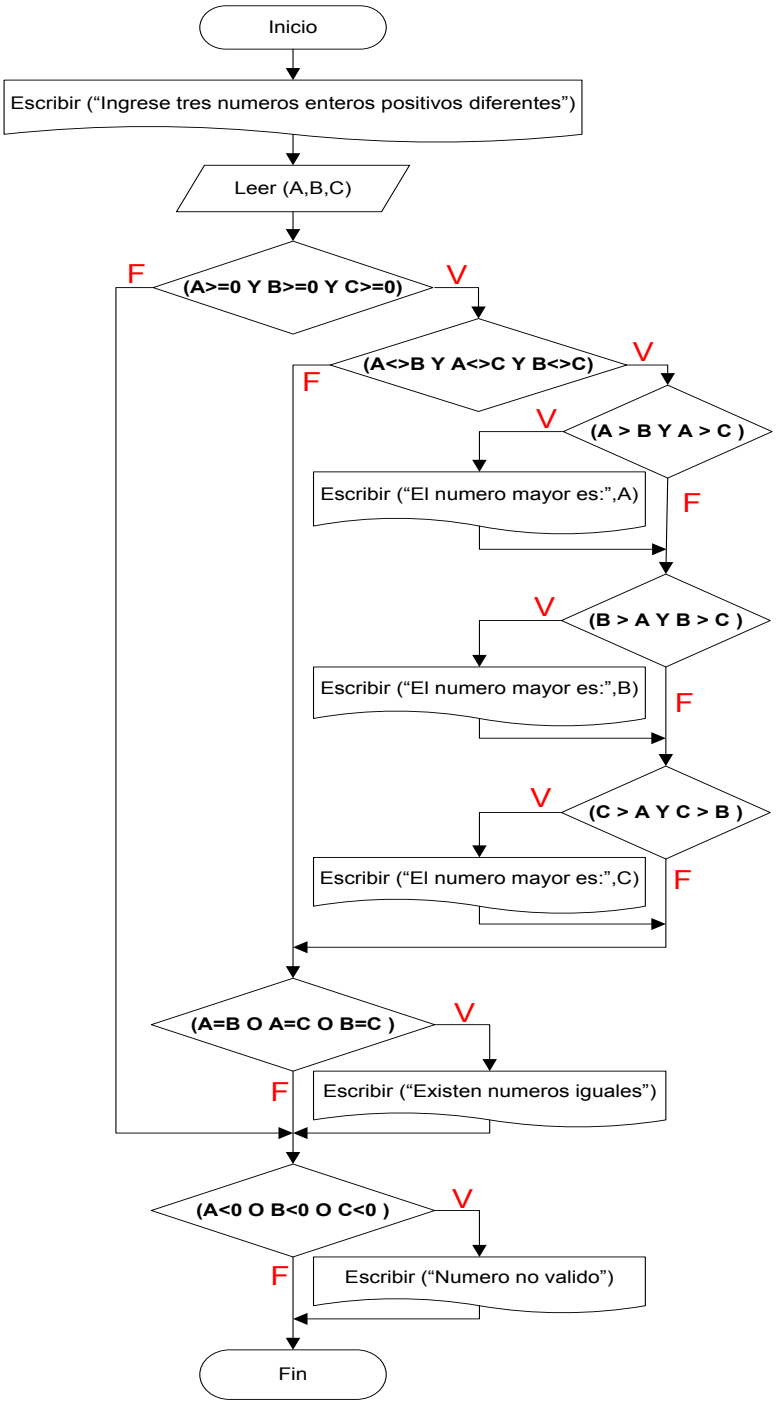


Ilustración 26. Diagrama de flujo usando estructuras de selección simples.

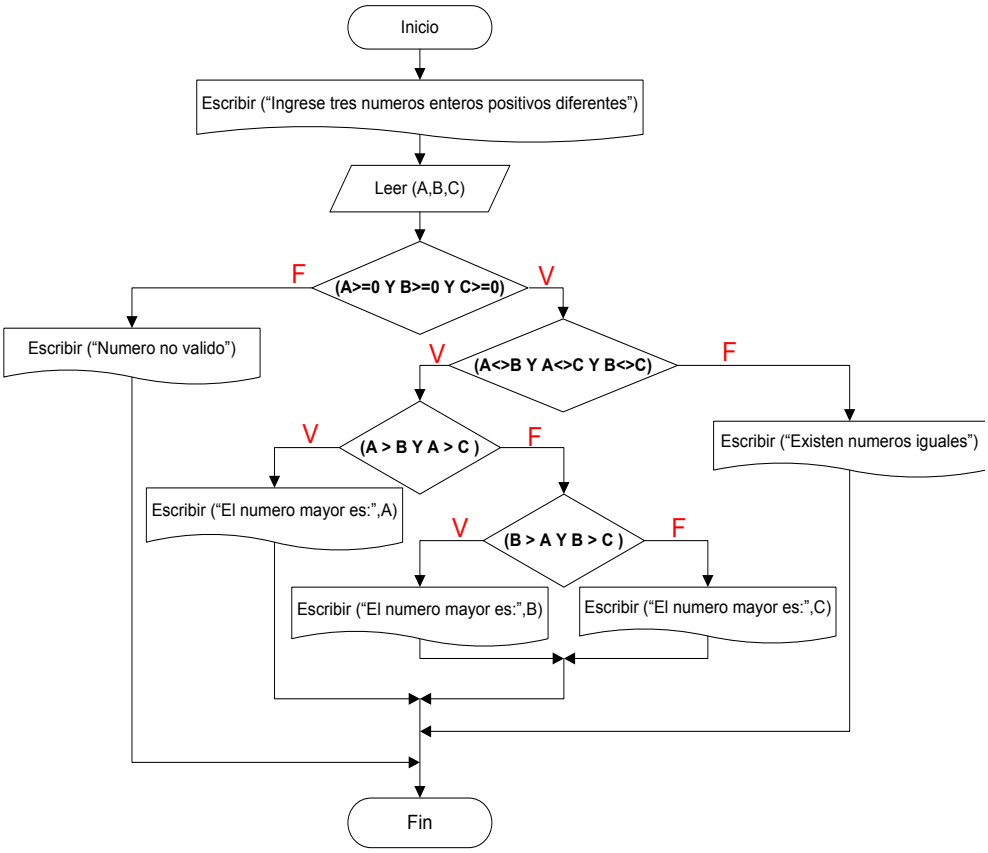


Ilustración 27. Diagrama de flujo estructuras de selección compuestas.

2. Construya un pseudocódigo que permita realizar operaciones aritméticas elementales, según sea la clave ingresada:

CLAVE	OPERACIÓN
+	SUMA
-	RESTA
*	MULTIPLICACION
/	DIVISION

Cada operación requiere de dos operandos –números–. Muestre la clave ingresada y el resultado de la operación.

El enunciado carece de la palabra SI sin embargo, para poder llevar a cabo la operación SUMA, es necesario determinar que la clave ingresada es el símbolo + y, para ello, se debe comparar el dato dado por el usuario con el símbolo correspondiente. Como se deben hacer comparaciones se escribe una condición, es decir:

Si (clave = "+") **entonces**
Si (clave = "–") **entonces**
Si (clave = "*") **entonces**
Si (clave = "/") **entonces**

Note que dependiendo de un solo dato ingresado por el usuario se pueden escoger uno de entre cuatro caminos (las cuatro operaciones) por lo que una estructura de selección múltiple podría ser más apropiada:

Si (clave) **igual**

"+" **:** operación suma
"–" **:** operación resta
"*" **:** operación multiplicación
"/" **:** operación división

Las operaciones para resolver el ejercicio son:

sum = A + B
res = A – B
mul = A * B
divi = A / B

Sólo en la operación de división se puede generar un error aritmético, si el usuario ingresa un cero en la variable B, la condición queda:

Si (B <> 0) **entonces**

En la ejecución del pseudocódigo el usuario puede ingresar cualquier número sea entero o decimal, además, las condiciones limitaran la solución a sólo cuatro claves conocidas por lo que una clave diferente no genera resultados que no se pueden interpretar.

Según el análisis anterior, se necesitan dos estructuras de selección para resolver el problema, una de ellas para impedir que se ingrese el número cero y otra, para permitir realizar una operación dependiendo de la clave ingresada por el usuario.

El problema se resolverá usando las dos opciones, primero con las estructuras de selección simple o compuestas y, luego con la estructura de selección múltiple. El lector apreciará ambas soluciones y podrá observar las diferencias en las mismas.

ALGORITMO usando ESS

```
Inicio
Capturar clave.
Si (clave = " + ") entonces
  Mostrar ingrese dos números
  Capturar dos números
  Calcular suma
    suma = A + B
  Mostrar el resultado de la operación
            y la clave
Fin_si
Si (clave = "– ") entonces
  Mostrar ingrese dos números
  Capturar dos números
  Calcular resta
      resta = A – B
  Mostrar el resultado de la operación
            y la clave
Fin_si
Si (clave = " * ") entonces
  Mostrar ingrese dos números
```

Pseudocódigo usando ESS

```
Algoritmo_encuentra_mayor
var
      Decimal : A,B, sum, res, mult, divi
      Caracter : clave
Inicio
 Escribir("Ingrese clave")
 Leer (clave)
 Si (clave = "+") entonces
     Escribir ("Ingrese dos números")
     Leer (A,B)
     sum ← A + B
     Escribir ("La clave", clave, "genera",sum)
 Fin_si
 Si (clave = "–") entonces
     Escribir ("Ingrese dos números")
     Leer (A,B)
     res ← A – B
     Escribir ("La clave", clave, "genera",res)
 Fin_si
 Si (clave = "*") entonces
     Escribir ("Ingrese dos números")
     Leer (A,B)
     mult ← A * B
```

```
  Mostrar ingrese dos números
  Capturar dos números
  Calcular multiplicación
     mult = A * B
  Mostrar el resultado de la operación
           y la clave
Fin_si
Si (clave = " / ") entonces
  Mostrar ingrese dos números
      Capturar dos números
      Si ( B < > 0) entonces
        Calcular división
           divi = A / B
           Mostrar el resultado de la
                    operación y la clave
      Fin_si
      Si (B = 0) entonces
         Mostrar Clave no valida
      Fin_si
    Fin_si
    Si(clave<>"+" O clave<>"–" 0 clave<>"*"
       O clave<>"/") entonces
       Mostrar clave no valida
    Fin_si
    Fin
      Escribir ("La clave", clave, "genera",mult)
Fin_si
Si (clave = "/") entonces
    Escribir ("Ingrese dos números")
    Leer (A,B)
    Si (B <> 0) entonces
       divi ← A / B
       Escribir ("La clave", clave, "genera",divi)
       Fin_si
       Si (B = 0) entonces
          Escribir ("Numero no valido")
       Fin_si
    Fin_si
    Si (clave<>" + " O clave<>" – " O clave<>" * "
       clave<>" / ") entonces
       Escribir ("Clave no valida")
    Fin_si
    Fin
```

A continuación la Ilustración 28 se muestra el Diagrama de Flujo respectivo.

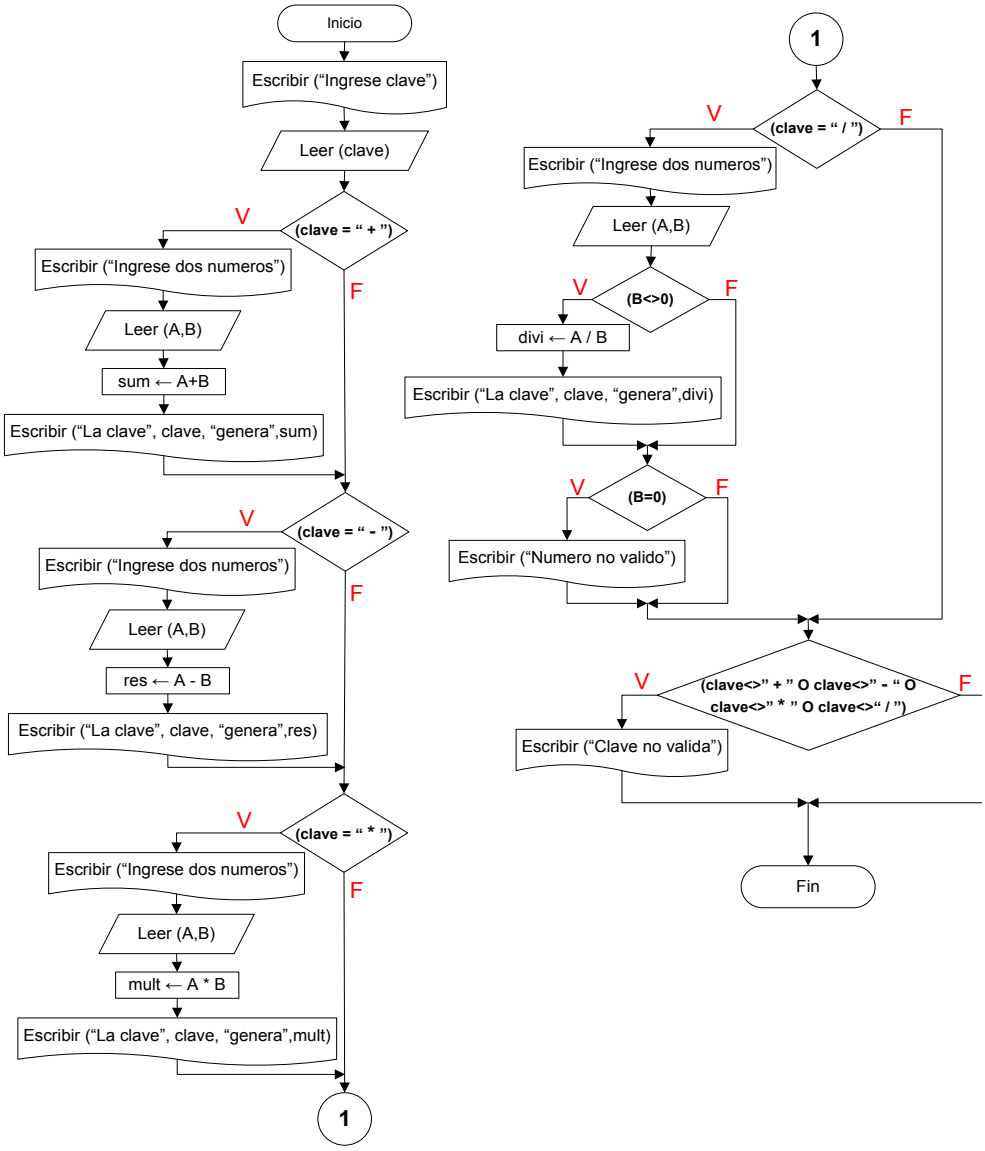


Ilustración 28. Solución utilizando estructuras de selección simple.

ALGORITMO usando ESC

```
Inicio
Capturar clave.
Si (clave = " + ") entonces
  Mostrar ingrese dos números
  Capturar dos números
  Calcular suma
    suma = A + B
  Mostrar el resultado de la operación
          y la clave
De lo contrario
  Si (clave = "– ") entonces
    Mostrar ingrese dos números
    Capturar dos números
    Calcular resta
      resta = A – B
    Mostrar el resultado de la operación
          y la clave
  De lo contrario
    Si (clave = " * ") entonces
      Mostrar ingrese dos números
      Capturar dos números
      Calcular multiplicación
        mult = A * B
      Mostrar el resultado de la
            operación y la clave
    De lo contrario
      Si (clave = " / ") entonces
        Mostrar ingrese dos números
```

Pseudocódigo usando ESC

```
Algoritmo_encuentra_mayor
var
    Decimal : A,B, sum, res, mult, divi
    Caracter : clave
Inicio
 Escribir("Ingrese clave")
 Leer (clave)
 Si (clave = "+") entonces
    Escribir ("Ingrese dos números")
    Leer (A,B)
    sum ← A + B
    Escribir ("La clave", clave, "genera",sum)
 De lo contrario
   Si (clave = "–") entonces
      Escribir ("Ingrese dos números")
      Leer (A,B)
      res ← A – B
      Escribir ("La clave", clave, "genera",res)
   De lo contrario
     Si (clave = "*") entonces
       Escribir ("Ingrese dos números")
       Leer (A,B)
       mult ← A * B
       Escribir ("La clave", clave, "genera",mult)
     De lo contrario
       Si (clave = "/") entonces
          Escribir ("Ingrese dos números")
          Leer (A,B)
          Si (B <> 0) entonces
             divi ← A / B
             Escribir ("La clave", clave, "genera",divi)
```

```
        Capturar dos números
        Si ( B < > 0) entonces
          Calcular división
          divi = A / B
          Mostrar el resultado de la
              operación y la clave
        De lo contrario
          Mostrar numero no valido
        Fin_si
      De lo contrario
        Mostrar clave no valida
      Fin_si
    Fin_si
  Fin_si
Fin_si
Fin
```

```
        De lo contrario
          Escribir ("Numero no valido")
        Fin_si
      De lo contrario
        Escribir ("clave no valida")
      Fin_si
    Fin_si
  Fin_si
Fin_si
Fin
```

A continuación la Ilustración 29 muestra el Diagrama de Flujo respectivo.

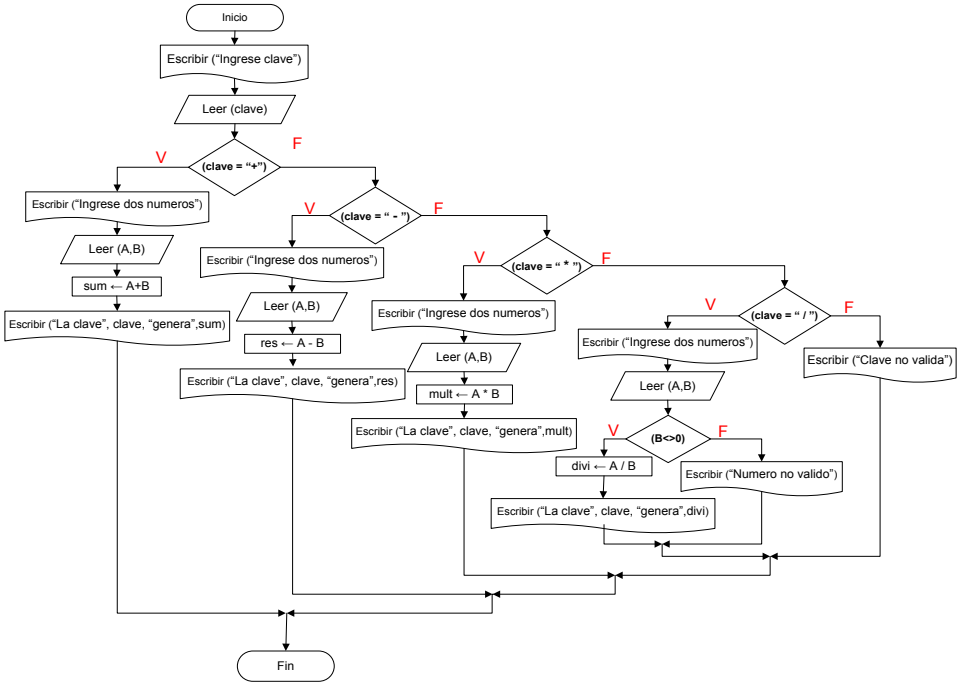


Ilustración 29. Solución utilizando estructuras de selección compuesta.

ALGORITMO usando ESM

```
Inicio
Capturar clave.
Si (clave ) igual
 " + " : Mostrar ingrese dos números
        Capturar dos números
        Calcular suma
   suma = A + B
        Mostrar el resultado de la
        operación y la clave
 "– " : Mostrar ingrese dos números
        Capturar dos números
        Calcular resta
        resta = A – B
        Mostrar el resultado de la
        operación y la clave
" * " : Mostrar ingrese dos números
        Capturar dos números
        Calcular multiplicación
          mult = A * B
        Mostrar el resultado de la
              operación y la clave
 " / " : Mostrar ingrese dos números
        Capturar dos números
        Si ( B < > 0) entonces
          Calcular división
          divi = A / B
          Mostrar el resultado de la
               operación y la clave
         De lo contrario
           Mostrar numero no valido
         Fin_si
  De lo contrario : Mostrar clave no valida
  Fin_si
  Fin
```

Pseudocódigo usando ESM

```
Algoritmo_encuentra_mayor
var
     Decimal : A,B, sum, res, mult, divi
     Caracter : clave
Inicio
 Escribir("Ingrese clave")
 Leer (clave)
 Si (clave) igual
          " + " :  Escribir ("Ingrese dos números")
                   Leer (A,B)
                   sum ← A + B
                   Escribir ("La clave", clave, "genera",sum)
          " – " :  Escribir ("Ingrese dos números")
                   Leer (A,B)
                   res ← A – B
                   Escribir ("La clave", clave, "genera",res)
          " * " :  Escribir ("Ingrese dos números")
                   Leer (A,B)
                   mult ← A * B
                   Escribir ("La clave", clave, "genera",mult)
          " / " :  Escribir ("Ingrese dos números")
                   Leer (A,B)
                   Si (B <> 0) entonces
                    divi ← A / B
                    Escribir ("La clave", clave, "genera",divi)
                   De lo contrario
                    Escribir ("Numero no valido")
                   Fin_si
De lo contrario : Escribir ("clave no valida")
Fin_si
Fin
```

El uso de una estructura de selección múltiple permite construir un programa más corto. La siguiente Ilustración 30 muestra el diagrama de flujo respectivo.

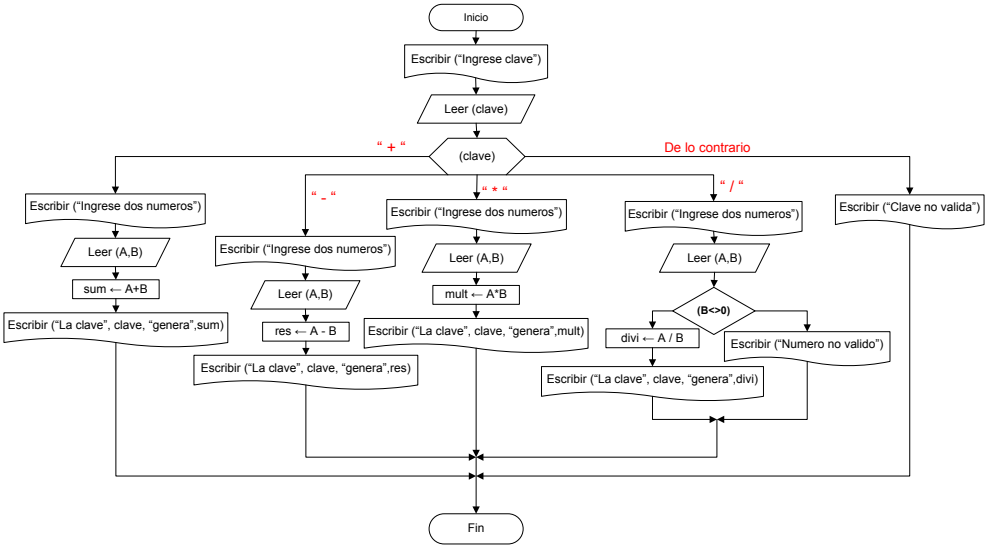


Ilustración 30. Diagrama de flujo usando estructura de selección múltiple.

3. Una compañía dedicada al alquiler de automóviles cobra un monto fijo de $300 para los primeros 300 Km. de recorrido. Para más de 300 Km. y hasta 1000 Km., cobra un monto adicional de $15 por cada kilómetro en exceso sobre 300. Para más de 1000 Km. cobra un monto adicional de $10 por cada kilómetro en exceso sobre 1000. Los precios no incluyen el 16% del impuesto al valor agregado, IVA. Diseñe un pseudocódigo que determine el monto a pagar por el alquiler de un vehículo y el monto incluido del impuesto.

El ejercicio exige determinar el monto a pagar por kilómetro recorrido dependiendo además del rango en que se encuentre según la distancia recorrida, es decir, si se utilizó el automóvil por 315 km se debe pagar por esta cantidad $300 por los primeros 300 km y $15 por cada kilómetro por encima de 300, o sea, $15 x 15 = $225, para dar un total de $300 + $225 = $525.

La siguiente Ilustración 31 muestra las ecuaciones que se deben usar para resolver el problema, las mismas dependen de la distancia recorrida.

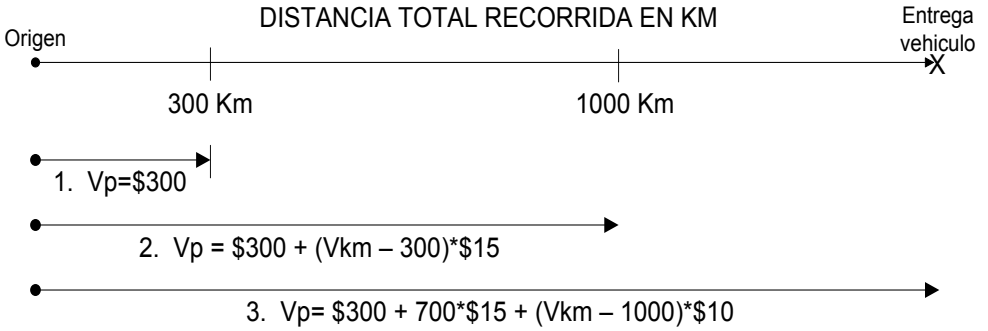


Ilustración 31. Análisis del ejercicio 3.

En la figura 3.24 **Vp** es la variable donde se almacena el valor a pagar y, **Vkm** es la variable que contiene el valor de los kilómetros recorridos. Note de la figura anterior que $525 se obtuvo de la ecuación 2, es decir:

$$Vp = 300 + (315 - 300) * 15$$
$$Vp = 525$$

Vkm se reemplazó por 315 que es la distancia del ejemplo.
La ecuación 1 muestra que un recorrido entre 0 y 300Km debe pagar un valor de $300. Cuando se supera la distancia de 300Km, en el análisis del valor total a pagar se tiene en cuenta los $300 por los primeros 300Km ya que este valor es fijo para este recorrido; el valor a pagar por encima de los 300Km (pero inferior a los 1000Km) dependerá del trayecto recorrido y para obtener ese valor se resta de la distancia los primeros 300Km pues éstos ya están incluidos en los $300, es decir:

$$Vp = 300 + (Vkm - 300) * 15$$

El primer número 300 representan el monto a pagar por la distancia recorrida de 300Km; el paréntesis refleja la distancia restante, por encima de los primeros 300Km, cuyo resultado se multiplica por 15 que es el valor a pagar por cada kilómetro en exceso sobre 300.

Utilizando un análisis similar se obtiene la ecuación 3 de la Figura 3.24, en donde 300 + 700*15 representa el monto a pagar por los primeros 1000Km y, (Vkm–1000)*10 simboliza el dinero a pagar por los kilómetros en exceso sobre 1000.

En la solución del problema, el programador deberá solicitar al usuario la distancia recorrida.

De la Figura 3.24 se puede apreciar que existen 3 maneras de obtener el monto a pagar por la distancia recorrida, el primero de ellos es usado cuando el vehículo ha recorrido una distancia entre 0 y 300Km, el segundo para recorridos entre 300 y 1000Km y el último para trayectos superiores a los 1000Km; para escoger cuál de ellas se debe utilizar es necesario construir estructuras de selección, quedando:

Si (Vkm <= 300) **entonces**
  Vp = 300

Si (Vkm >300 Y Vkm <= 1000) **entonces**
  Vp = 300 + (Vkm – 300)*15

Si (Vkm > 1000) **entonces**
  Vp = 300 + 700*15 + (Vkm – 1000)*10

Al ser Vkm la distancia recorrida por el automóvil, la misma no puede ser negativa por tanto es necesario una estructura de selección quedando:

Si (Vkm > 0) **entonces**

Ingresar un valor mayor a 0Km implica que el programa deba generar un mensaje de texto informando el monto a pagar por el alquiler del vehículo. 0Km no es un dato inválido pero tampoco se debe cancelar dinero alguno por el alquiler del automóvil. Para los valores que no cumplen la condición el programa debe informar al usuario del ingreso de datos inválidos.

Según el análisis anterior, se necesitan cuatro estructuras de selección para resolver el problema, una de ellas para impedir que se ingrese un número negativo y, las otras, para escoger la ecuación pertinente que permita obtener el valor a pagar por el usuario según la distancia recorrida. En cada condición el uso de operadores relacionales crea un rango de valores para la variable **Vkm** representando una gran cantidad de posibilidades para cada estructura de selección.

A continuación se elabora el algoritmo y el pseudocódigo. El problema se resuelve usando las dos opciones, primero con las estructuras de selección simples y luego con las estructuras de selección compuestas.

ALGORITMO usando ESS

```
Inicio
Capturar distancia recorrida.
Si (distancia recorrida>0) entonces
  Si (distancia recorrida < 300) entonces
      Vp = 300
  Fin_si
  Si (distancia recorrida >= 300 Y
    distancia recorrida < 1000) entonces
      Vp = 300 + (distancia recorrida –
        300)*15
  Fin_si
  Si (distancia recorrida >1000) entonces
      Vp = 300 + 700 *15 + (distancia
        recorrida –  300)*15
  Fin_si
  Mostrar el valor a pagar por el alquiler
Fin_si
Si(distancia recorrida < 0) entonces
    Mostrar dato no valido
Fin_si
Fin
```

Pseudocódigo usando ESS

```
Algoritmo_encuentra_mayor
var
     Decimal : Vkm, Vp
Inicio
 Escribir("Ingrese distancia recorrida")
 Leer (Vkm)
 Si (Vkm >0) entonces
    Si (Vkm <= 300) entonces
      Vp = 300
    Fin_si
    Si (Vkm > 300 Y Vkm <= 1000) entonces
      Vp = 300 + (Vkm – 300)*15
    Fin_si
    Si (Vkm > 1000) entonces
       Vp = 300 + 700 * 15 + (Vkm – 1000)*10
    Fin_si
    Escribir ("El valor a pagar por el alquiler es", Vp)
 Fin_si
 Si (Vkm <=0) entonces
    Escribir ("Dato no valido")
 Fin_si
Fin
```

ALGORITMO usando ESC

```
Inicio
Capturar distancia recorrida.
Si (distancia recorrida>0) entonces
 Si (distancia recorrida < 300) entonces
    Vp = 300
 De lo contrario
   Si (distancia recorrida >= 300 Y
    distancia recorrida < 1000) entonces
     Vp = 300 + (distancia recorrida –
      300)*15
   De lo contrario
     Vp = 300 + 700 *15 + (distancia
      recorrida –  300)*15
   Fin_si
 Fin_si
 Mostrar el valor a pagar por el alquiler
De lo contrario
 Mostrar dato no valido
Fin_si
Fin
```

Pseudocódigo usando ESC

```
Algoritmo_encuentra_mayor
var
     Decimal : Vkm, Vp
Inicio
 Escribir("Ingrese distancia recorrida")
 Leer (Vkm)
 Si (Vkm >0) entonces
    Si (Vkm <= 300) entonces
       Vp = 300
    De lo contrario
       Si (Vkm > 300 Y Vkm <= 1000) entonces
          Vp = 300 + (Vkm – 300)*15
       De lo contrario
          Vp = 300 + 700 * 15 + (Vkm – 1000)*10
       Fin_si
    Fin_si
    Escribir ("El valor a pagar por el alquiler es", Vp)
 De lo contrario
    Escribir ("Dato no valido")
 Fin_si
Fin
```

Las siguientes ilustraciones muestran los diagramas de flujo respectivos.

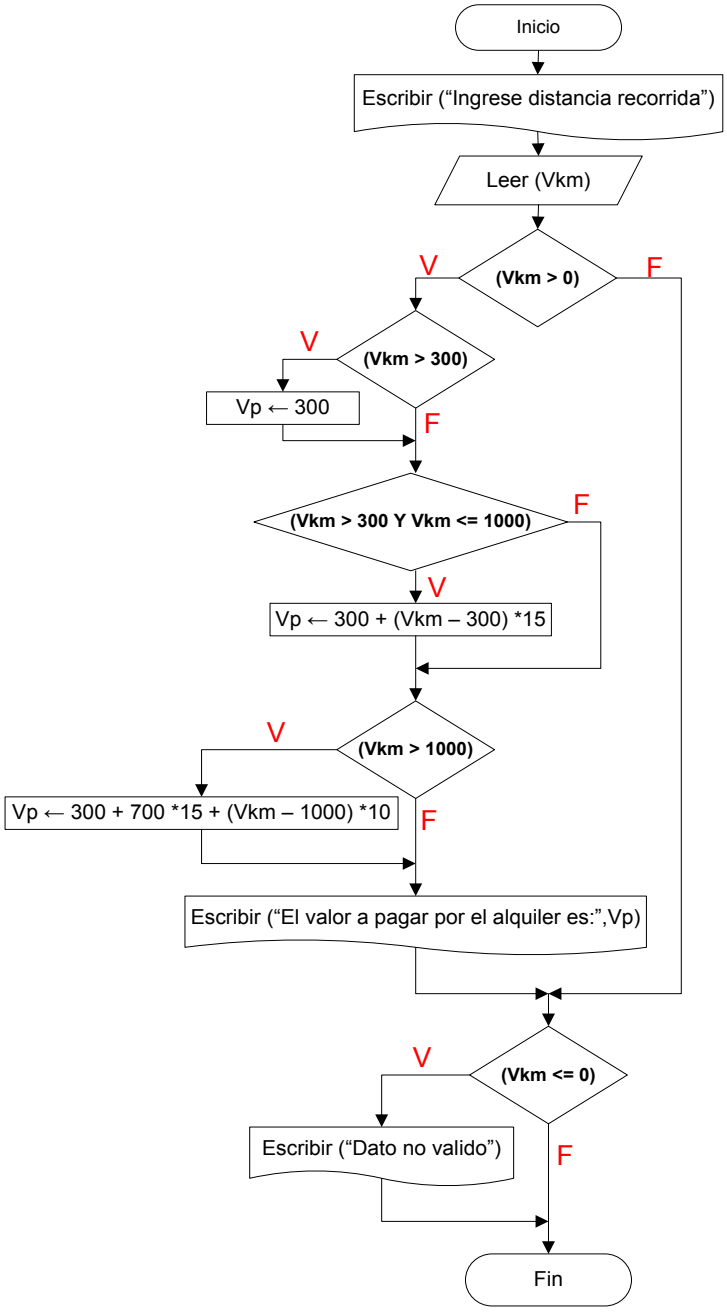


Ilustración 32. Diagrama de flujo usando estructuras de selección simples.

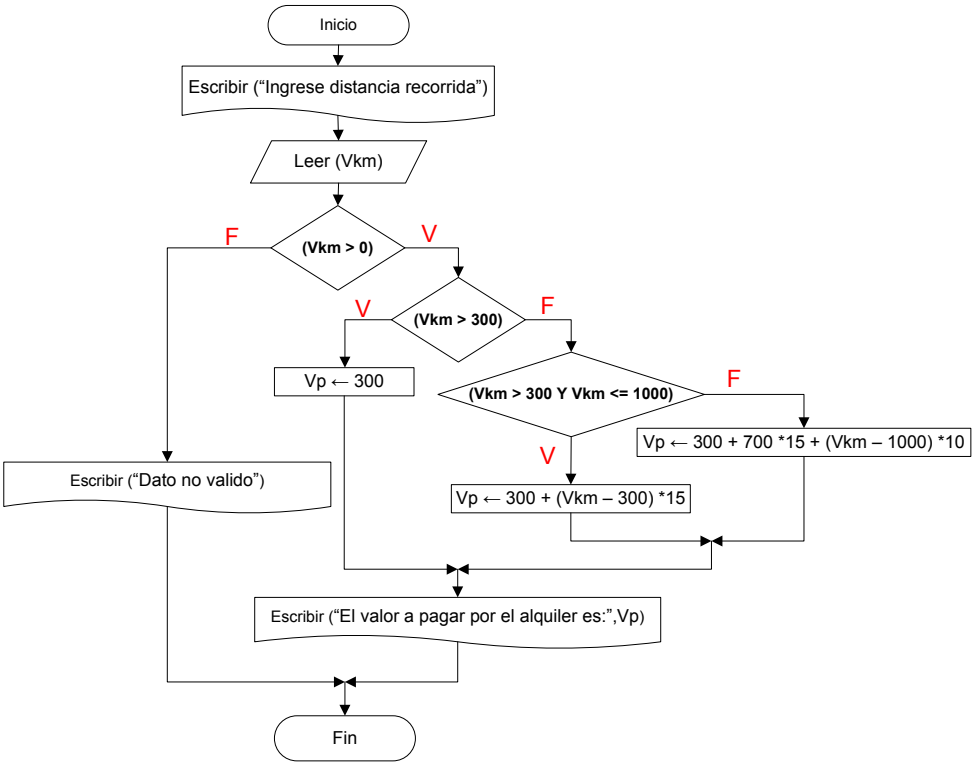


Ilustración 33. Diagrama de flujo usando estructuras de selección compuestas.

Sólo se requieren 3 símbolos de decisión para resolver el problema esto debido a la ventaja del uso de las estructuras de selección compuestas, la construcción del diagrama de flujo resulta más sencilla y corta.

Capítulo 4

4. ESTRUCTURAS DE REPETICIÓN

Hasta ahora todo lo visto permite construir pseudocódigos que se ejecutan tan sólo una vez, es decir, se ejecutan una a una las instrucciones de una manera secuencial hasta llegar a la ejecución de la instrucción **Fin**, sin embargo existen problemas que requieren repetir grupos de instrucciones, por ejemplo, programas que constantemente están requiriendo el ingreso de una cierta cantidad de datos.

Suponga que una compañía tiene 30 empleados y se desea calcular el aumento del sueldo de cada uno de ellos, se debe construir un programa en el cual se pueda ingresar el sueldo de cada uno de los empleados. Con las instrucciones hasta ahora conocidas se produciría un programa muy extenso, la solución es emplear estructuras de repetición.

Las estructuras de repetición y las estructuras de selección pertenecen a las instrucciones de bifurcación y aunque ambas usan una condición para determinar la acción a ejecutar, las estructuras de repetición se diferencian de las de selección porque permiten que ciertas instrucciones se ejecuten más de una vez hasta que se satisfaga la condición. Las estructuras de repetición también son conocidas como ciclos o bucles.

4.1 ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA

La estructura de repetición Mientras Haga tiene el siguiente formato general en el pseudocódigo, ver Ilustración 34:

Mientras (**condición**) **haga**

Instrucción 1
Instrucción 2
:
Modificador de Condición
Instrucción N
Fin_mientras

Ilustración 34. Formato general pseudocódigo para la estructura de repetición Mientras Haga.

La estructura de repetición Mientras Haga contiene tres palabras reservadas: **Mientras**, **haga** y **Fin_mientras**. Comienza con la palabra **Mientras** y termina con **Fin_mientras**. Además, contiene un cuerpo de instrucciones entre ambas palabras, y una **condición** dentro de paréntesis. La **condición** al ser evaluada debe siempre generar un valor de tipo booleano: **verdadero** o **falso**. Dentro del cuerpo de instrucciones debe contener una en particular, el Modificador de Condición, esta instrucción se encarga de cambiar el contenido de la variable que se encuentra en la **condición**, esto permite que en algún momento se termine la ejecución del ciclo. La ausencia o mala construcción del Modificador de Condición puede ocasionar ciclos infinitos – ciclos que nunca terminan – y por tanto, se pierde el control de la ejecución del programa.

La **condición** se puede construir de dos formas:

- Usando los operadores relacionales y/o operadores lógicos en una expresión que al ser resuelta genera un valor booleano
- O usando una variable de tipo booleano.

En ambos casos, las variables empleadas en la construcción de la **condición** deben estar declaradas y contener un valor antes de ser evaluadas.

Cuando la **condición** genera un resultado **verdadero** automáticamente se ejecuta de una manera secuencial el cuerpo de instrucciones, incluida la instrucción llamada Modificador de Condición; al ejecutarse la última instrucción -Instrucción N- la ejecución del programa se devuelve a evaluar de nuevo la **condición**, si ésta genera un resultado **verdadero**, ingresa nuevamente a la estructura de repetición y ejecuta todo el cuerpo de instrucciones, incluido el Modificador de Condición; este proceso se repite cada vez que la **condición** genere un resultado verdadero como se muestra en la Figura 4.2.

Cada ingreso al cuerpo de instrucciones de toda estructura de repetición se le conoce como Iteración, la cantidad de iteraciones que se lleve a cabo en un programa dependerá del límite puesto en la **condición**. Con cada iteración el contenido de la variable que se encuentra en la **condición** va cambiando debido a la ejecución de la instrucción Modificador de Condición hasta que alcance, sobrepase o sea igual al límite, cuando esto ocurre se termina la ejecución de la estructura de repetición.

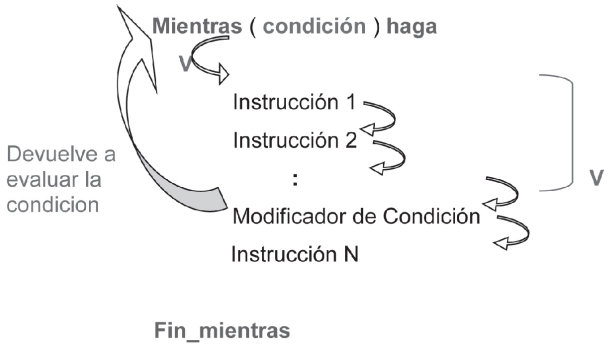


Figura 4.2 Funcionamiento de la estructura de repetición Mientras Haga cuando la **condición** da **verdadero**.

Cuando la **condición** genera un resultado **falso**, la ejecución del programa salta a la palabra reservada **Fin_mientras** cerrando la estructura de repetición y luego la ejecución continua con las instrucciones siguientes, en la figura 4.3 se muestra este comportamiento.

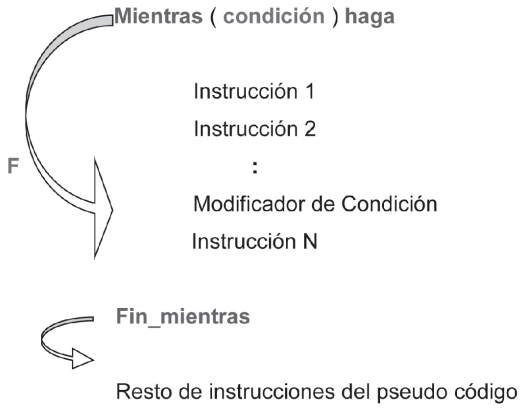


Figura 4.3 Funcionamiento de la estructura de repetición Mientras Haga cuando la **condición** da **falso**.

La siguiente figura muestra el formato general en el diagrama de flujo para de la estructura de repetición Mientras Haga.

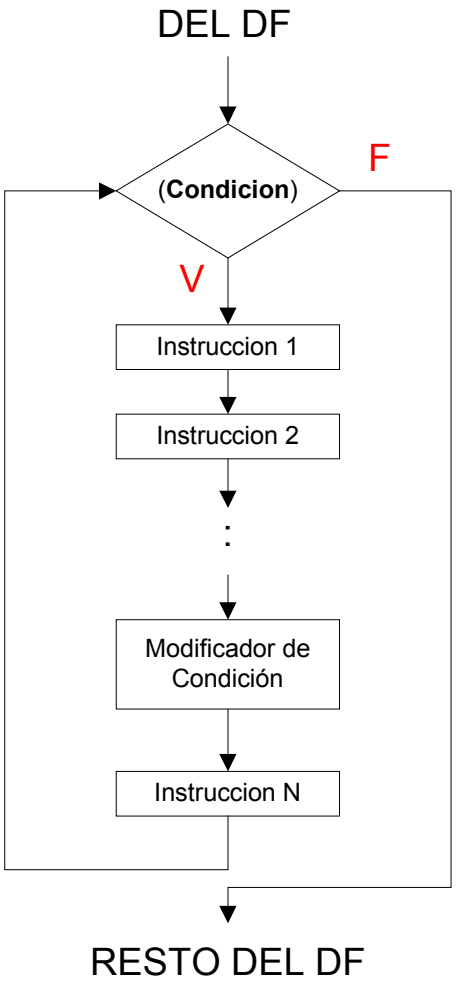


Figura 4.4 Formato general diagrama de flujo para la estructura de repetición Mientras Haga.

En la Figura 4.4 no se aprecia las palabras reservadas **Mientras**, **haga** y **Fin_mientras**, éstas no tienen representación en el diagrama de flujo. Como se observa en el diagrama de flujo, en el camino del **falso** no existe instrucción alguna y llega directamente para continuar con la ejecución del resto de instrucciones – RESTO DEL DF –.

4.2 ESTRUCTURA DE REPETICIÓN HAGA MIENTRAS

La estructura de repetición Haga Mientrastiene el siguiente formato general en el pseudocódigo:

Haga

Instrucción 1
Instrucción 2
:
Modificador de Condición
Instrucción N

Mientras (condición)

Figura 4.5 Formato general pseudocódigo para la estructura de repetición Haga Mientras.

La estructura de repetición Haga Mientrascontiene dos palabras reservadas: **Haga** y **Mientras**. Comienza con la palabra **Haga** y termina con **Mientras** y dentro de éstas se encuentra el cuerpo de instrucciones el cual incluye el Modificador de Condición. Además, contiene una **condición** dentro de paréntesis. La **condición** al ser evaluada debe siempre generar un valor de tipo booleano: **verdadero** o **falso**. En la **condición** se deben usar los operadores relacionales y/o lógicos, y sus variables deben estar declaradas y contener algún dato.

Al ejecutarse la palabra reservada **Haga**, el programa interpreta que se está ingresando a una estructura de repetición, luego se sigue con la ejecución de cada una de las instrucciones que hacen parte del cuerpo, incluida el Modificador de Condición, al ejecutarse ésta se modifica el contenido de la variable que se encuentra en la **condición**. Después de la ejecución de la instrucción N, se evalúa la **condición**, si ésta genera un resultado **verdadero**, la ejecución del programa salta automáticamente a la primera instrucción que se encuentra después de la palabra reservada **Haga** y este proceso se repite toda vez que la **condición** genere un resultado **verdadero**.

En la estructura de repetición Haga Mientrasse evalúa la **condición** al final, por lo que todas las instrucciones que hacen parte de la estructura se ejecutan antes de la condición, es decir, primero se ejecuta y luego se pregunta. Esa es la principal diferencia con la estructura de repetición Mientras Haga.

La siguiente figura muestra el funcionamiento de la estructura de repetición Haga Mientrascuando la **condición** genera un resultado **verdadero**.

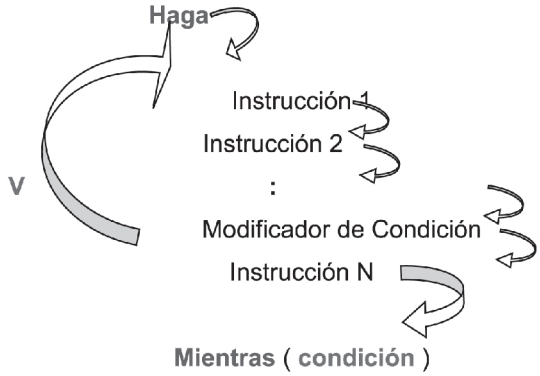


Figura 4.6 Funcionamiento de la estructura de repetición Haga Mientrascuando la **condición** da **verdadero**.

Cuando la **condición** genera **falso** se termina la ejecución de la estructura y se continúa con la ejecución del resto de instrucciones del pseudocódigo. La siguiente figura muestra este comportamiento.

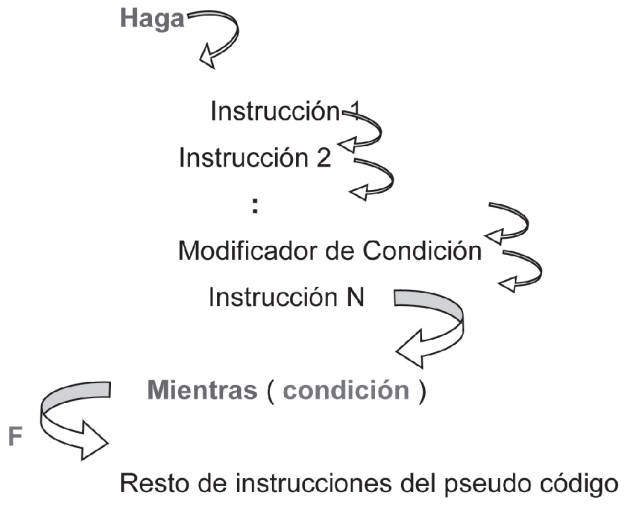


Figura 4.7 Funcionamiento de la estructura de repetición Haga Mientrascuando la **condición** da **falso**.

La siguiente figura muestra el formato general del diagrama de flujo para la estructura de repetición Haga Mientras.

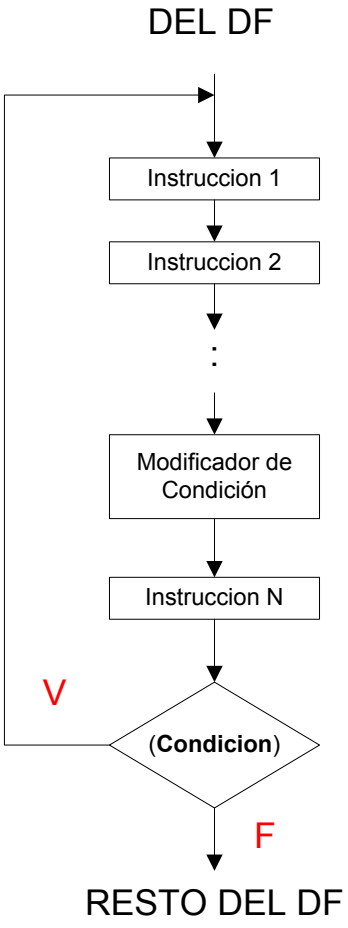


Figura 4.8 Formato general diagrama de flujo para la estructura de repetición Haga Mientras.

En la figura 4.8 no se aprecia las palabras reservadas **Haga** y **Mientras**, éstas no tienen representación en el diagrama de flujo. Como se observa el camino del **falso** va directamente a continuar con la ejecución del resto de instrucciones – RESTO DEL DF –.

4.3 ESTRUCTURA DE REPETICIÓN REPITA HASTA

La estructura de repetición Repita Hasta tiene el siguiente formato general en el pseudocódigo:

Repita

Instrucción 1
Instrucción 2
:
Modificador de Condición
Instrucción N

Hasta (**condición**)

Figura 4.9 Formato general pseudocódigo para la Estructura de Repetición Repita Hasta.

La estructura de repetición Repita Hasta contiene dos palabras reservadas: **Repita** y **Hasta**. Comienza con la palabra **Repita** y termina con **Hasta** y dentro de éstas se encuentra el cuerpo de instrucciones el cual incluye el Modificador de Condición. Además, contiene una **condición** dentro de paréntesis. La **condición** al ser evaluada debe siempre generar un valor de tipo booleano: **verdadero** o **falso**. En la **condición** se deben usar los operadores relacionales y/o lógicos, y sus variables deben estar declaradas y contener algún dato.

Igual que en la estructura de repetición Repita Hasta, la estructura de repetición Repita Hasta ejecuta las instrucciones asociadas a la estructura por lo menos una vez y luego pregunta. Sin embargo, en una estructura de repetición Repita Hasta se repite la ejecución del cuerpo de instrucciones, cuando la **condición** genera **falso** como resultado.

El Modificador de Condición cumple las mismas funciones en esta estructura de repetición como en las anteriores.

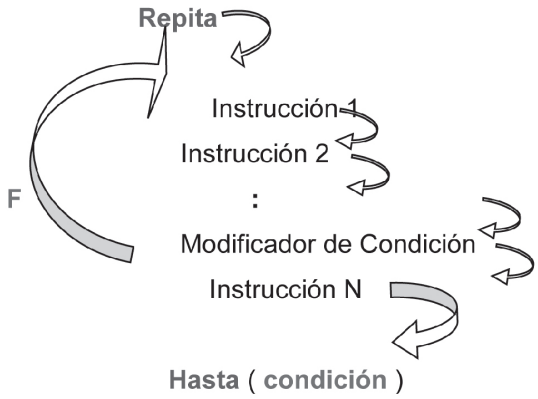


Figura 4.10 Funcionamiento de la estructura de repetición Repita Hasta cuando la **condición** da **falso**.

Cuando la **condición** genera un resultado **verdadero**, termina la ejecución de la estructura de repetición Repita Hasta y se continua con la ejecución del resto de instrucciones del pseudocódigo, como muestra la Figura 4.11.

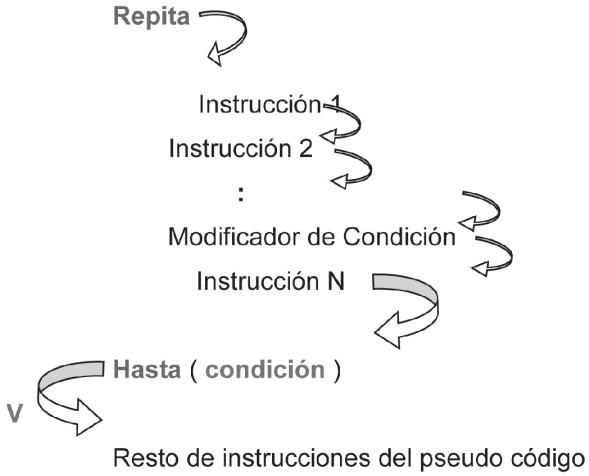


Figura 4.11 Funcionamiento de la estructura de repetición Repita Hasta cuando la **condición** da **verdadero**.

En la figura 4.12 se puede apreciar el formato general del diagrama de flujo para la estructura de repetición Repita Hasta.

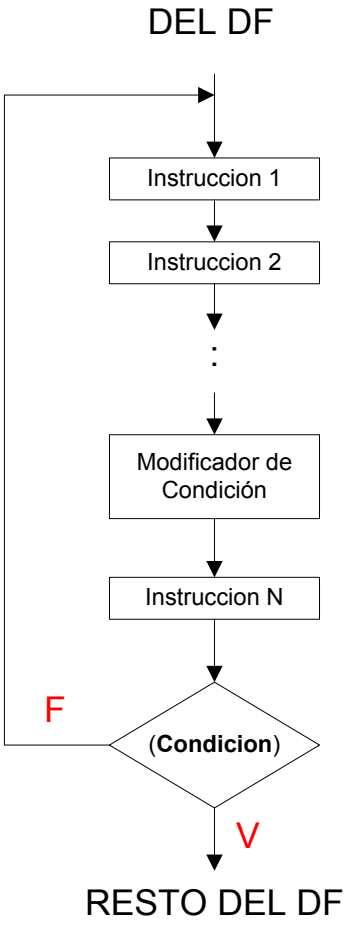


Figura 4.12 Formato general diagrama de flujo para la estructura de repetición Repita Hasta.

Note que la única diferencia entre las figuras 4.8 y 4.12 es la ubicación de las letras **V** y **F**. Igual que en la Figura 4.8, la Figura 4.12 no posee las palabras reservadas **Repita** y **Hasta**.

4.4 ESTRUCTURA DE REPETICIÓN PARA

La Estructura de Repetición Para tiene el siguiente formato general en el pseudocódigo:

Para V ← Vi hasta Vf (INC o DEC) haga

Instrucción 1
Instrucción 2
:
Instrucción N

Fin_para

Figura 4.13 Formato general pseudocódigo para la Estructura de Repetición Para.

La estructura de repetición Para contiene las siguientes palabras reservadas: **Para**, **hasta**, **haga** y **Fin_para**. Comienza con la palabra **Para** y termina con **Fin_para** y dentro de éstas se encuentra el cuerpo de instrucciones. Está conformada por una Inicialización (**V ← Vi**), una variable final **Vf** con la cual **V** comparara su valor, lo que correspondería a la **condición**, y dos palabras reservadas - **INC**, **DEC** - que permiten el incremento o decremento de la variable **V**.

La estructura de repetición Para es la única estructura de repetición que no contiene entre sus instrucciones la llamada Modificador de Condición, ésta es reemplazada por las palabras reservadas **INC** – incremento – o **DEC** – decremento – las cuales se colocan directamente dentro del ciclo Para.

En la Figura 4.13, **Vi** y **Vf** significan Valor Inicial y Valor Final respectivamente y son variables que contienen valores enteros por tanto, la estructura de repetición Para es utilizada únicamente para aquellos problemas donde **V** desea alcanzar el valor de **Vf** y superarlo. Las variables **Vi**, **V** y **Vf** deben estar declaradas y **Vi** y **Vf** deben contener valores antes de ejecutar la estructura de repetición. Se puede utilizar la estructura de repetición Para sin las variables **Vi** y **Vf**, simplemente colocando los valores numéricos respectivos, si este es el caso no se necesita declarar las dos variables.

Cuando se ingresa por primera vez a la estructura de repetición Para y sólo en esta única ocasión, a **V** se le asigna el valor de **Vi** y este valor se compara con el que contiene **Vf** de la siguiente manera:

1. Cuando se utiliza el operador menor o igual que (≤) para comparar a **V** con **Vf**, es decir se realiza **V ≤ Vf,** si el resultado es **verdadero**, se ingresa a la estructura de repetición y se ejecuta el cuerpo de instrucciones asociado a ésta. Después de la última instrucción se incrementa el valor de **V**, para lograr esto se usa la palabra reservada **INC,** si se desea incrementar de uno en uno se coloca **INC** 1 – la ausencia de esta palabra también puede ser interpretada como un incremento de uno en uno –, en forma general, el número que acompaña la palabra **INC** indica el valor del incremento, **INC** 1 se puede representar como:

INC 1 es lo mismo que V ← V + 1

Una vez se incrementa a **V** se compara su nuevo valor con el de **Vf** es decir, **V ≤ Vf**, si el resultado es **verdadero** se ejecuta nuevamente el cuerpo de instrucciones, luego se ejecuta el **INC** y, después se compara el nuevo valor de **V** con **Vf**, este proceso se repite cada vez que **V ≤ Vf** genere un resultado **verdadero**. En la Figura 4.14 se aprecia este comportamiento.

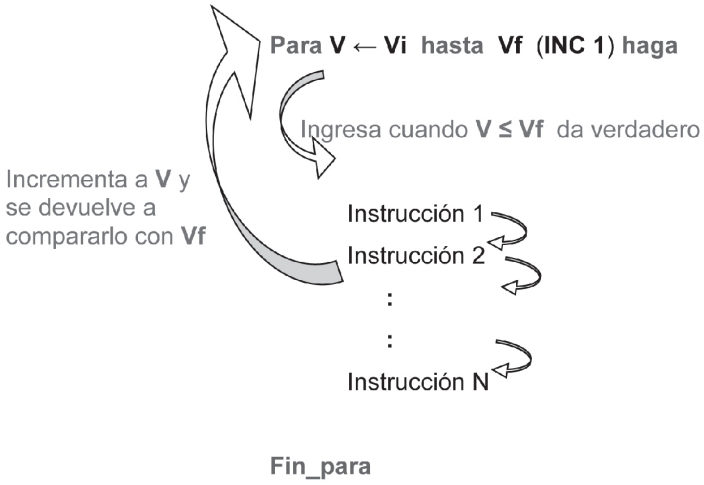


Figura 4.14 Funcionamiento de la estructura de repetición Para usando **INC** cuando **V ≤ Vf** da **verdadero**.

Cuando **V** supera el valor de **Vf**, la comparación **V ≤ Vf** genera un resultado **falso**, la ejecución del programa salta automáticamente a **Fin_para** y lo ejecuta cerrando el ciclo Para y se continua con la ejecución del resto de instrucciones del pseudocódigo, como muestra la Figura 4.15.

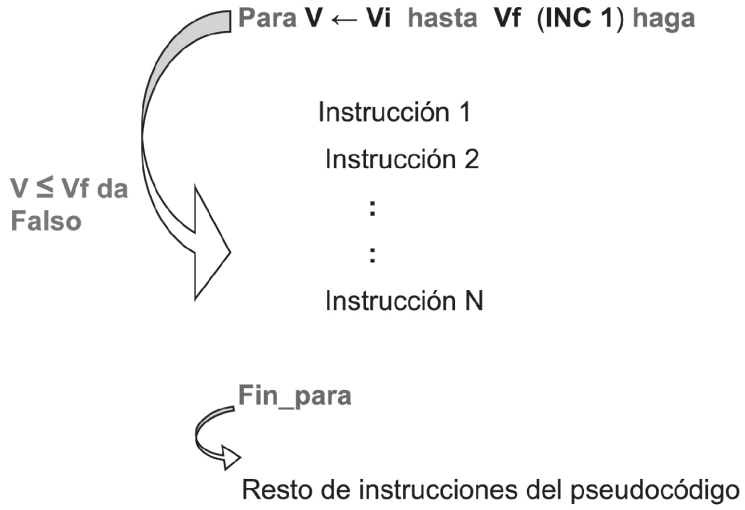


Figura 4.15 Funcionamiento de la estructura de repetición Para usando **INC** cuando **V ≤ Vf** da **falso**.

Si **Vi** es menor que **Vf** es recomendable nunca usar el decremento, **DEC**, ya que esto provocará que el valor de **V** disminuya en vez de aumentar por tanto, la **ERP** nunca terminaría su ejecución.

2. Cuando se utiliza el operador mayor o igual que (≥) para comparar a **V** con **Vf**, es decir se realiza **V ≥ Vf,** si el resultado es **verdadero**, se ingresa a la estructura de repetición y se ejecuta el cuerpo de instrucciones asociado a ésta, después de ejecutar la última instrucción se decrementa el valor de **V**, para lograr esto se usa la palabra reservada **DEC** si se desea decrementar de uno en uno se coloca **DEC** 1, en forma general, el número que acompaña la palabra **DEC** indica el valor del decremento, **DEC** 1 se puede representar como:

DEC 1 es lo mismo que V ← V – 1

Una vez se decrementa a **V** se compara su nuevo valor con el de **Vf** es decir, **V ≥ Vf**, si el resultado es verdadero se ejecuta nuevamente el cuerpo de instrucciones de la **ER**, luego se ejecuta el **DEC** y, después se compara el nuevo valor de **V** con **Vf**, este proceso se repite cada vez que **V ≥ Vf** genere un resultado verdadero. En la Figura 4.16 se aprecia este comportamiento.

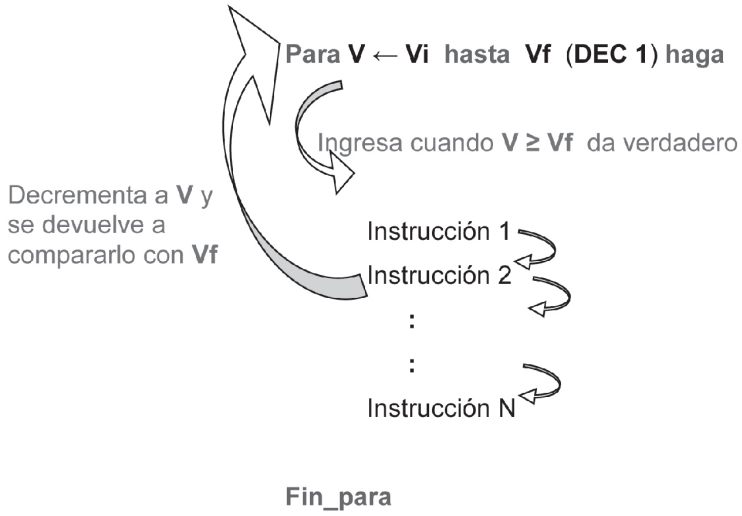


Figura 4.16 Funcionamiento de la estructura de repetición Para usando **DEC** cuando **V ≥ Vf** da **verdadero**.

Cuando **V** es inferior al valor de **Vf**, la comparación **V ≥ Vf** genera un resultado **falso**, la ejecución del programa salta automáticamente a **Fin_para** y lo ejecuta cerrando el ciclo Para y se continua con la ejecución del resto de instrucciones del pseudocódigo, como muestra la Figura 4.17.

Si **Vi** es mayor que **Vf** no se debe usar el incremento, **INC**, ya que esto provocará que el valor de **V** aumente en vez de disminuir por tanto, la estructura de repetición Para nunca terminaría su ejecución.

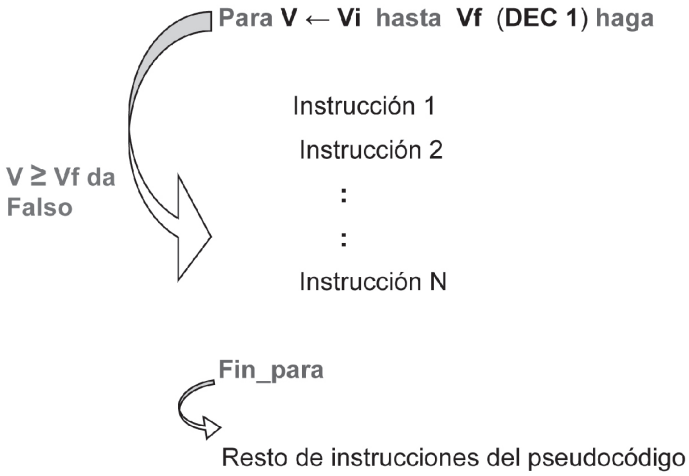


Figura 4.17 Funcionamiento de la estructura de repetición Para usando **DEC** cuando **V ≥ Vf** da **falso**.

La siguiente figura muestra el formato general del diagrama de flujo para la Estructura de Repetición Para.

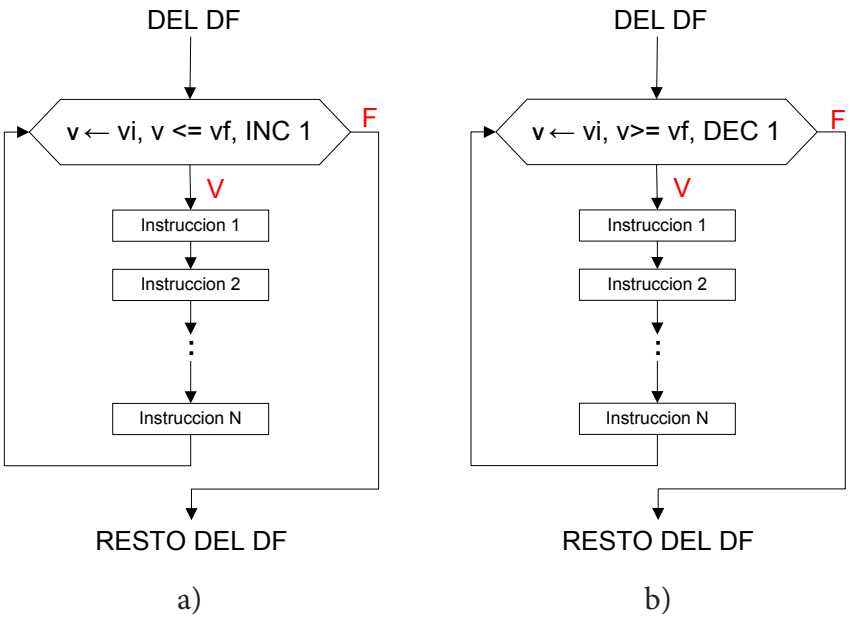


Figura 4.18 Formato general diagrama de flujo para la estructura de repetición Para. a) Con **V <= Vf**, b) Con **V >= Vf**

4.5 CONSTRUCCIÓN DE ESTRUCTURAS DE REPETICIÓN

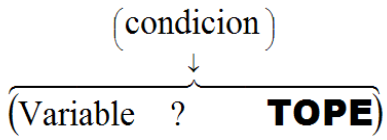
Si se necesita realizar una operación o cálculo más de una vez, se debe utilizar una estructura de repetición. Aquello que se debe realizar más de una vez representa el cuerpo de instrucciones de la estructura de repetición.

Conocidas las notas finales de un curso de 30 estudiantes, construya un programa que encuentre el promedio de notas del salón.

Para calcular el promedio del salón es necesario conocer todas las notas y para ello es necesario pedir la nota de cada estudiante, este proceso se debe hacer 30 veces por tanto, se necesita una estructura de repetición.

A continuación se presenta una guía para ayudar al lector a comprender cómo construir la **condición**.

En la **condición** debe existir una variable que contendrá el dato a comparar con el valor al cual se desea llegar. El valor deseado o el valor al cual se desea llegar son conocidos como **Tope**. En la estructura de repetición Para el valor de la variable depende de la palabra reservada **INC** o **DEC**. Por tanto, hasta el momento se conoce que la **condición** está conformada de la siguiente forma:



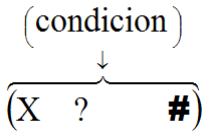
La **condición** debe generar uno de dos valores posibles **verdadero** o **falso** por lo que el símbolo de interrogación indica que en esa posición se debe colocar un operador relacional y/o lógico.

Cuando el Tope es de tipo **Numérico** existen dos casos:

1. Cuando un número en el enunciado representa la cantidad de veces que hay que realizar la misma operación. En este caso se debe construir una instrucción llamada **Contador**, que normalmente cuenta de uno en uno:

$$\mathbf{X \leftarrow X + 1}$$

La variable que aparece en el contador es la misma variable que se coloca en la **condición** entonces, hasta ahora la **condición** tiene la siguiente forma:



Donde el símbolo de numeral representa un Tope numérico. La variable **X** debe estar declarada y contener un dato. Asignar a **X** un valor de arranque antes de ejecutarse el contador se conoce como Inicialización y tiene generalmente la forma:

$$\mathbf{X \leftarrow 0} \quad \text{o} \quad \mathbf{X \leftarrow 1}$$

La Inicialización se coloca antes de la estructura de repetición por lo que los formatos del pseudocódigo para las estructuras de repetición Mientras Haga, Haga Mientras y Repetir Hasta quedan como se muestra a continuación:

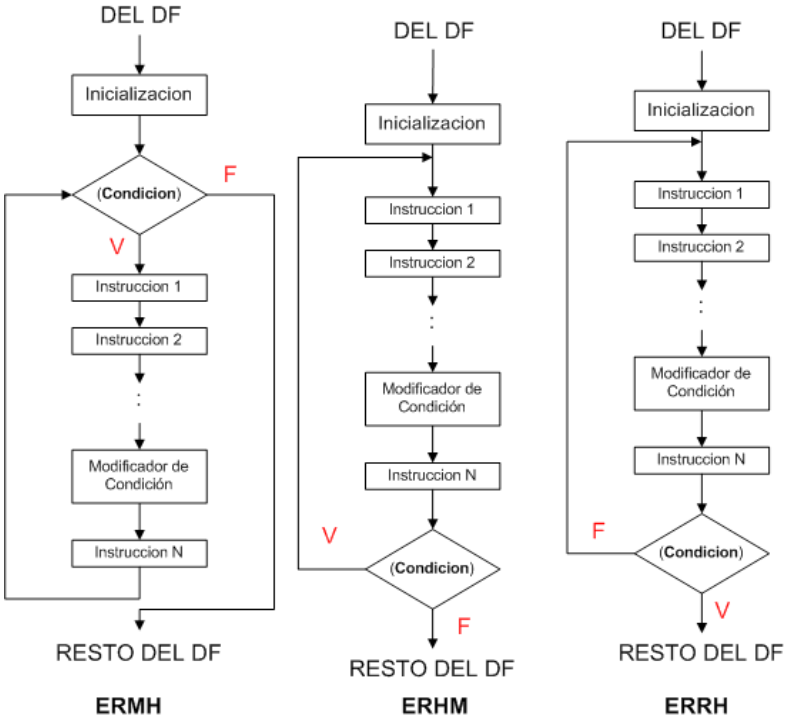


Figura 4.19 Formatos de las estructuras de repetición Mientras Haga, Haga Mientras y Repetir Hasta con Inicialización.

Hasta el momento se tiene para cualquier estructura de repetición excepto la Para, lo siguiente:

Mientras Haga	**Haga Mientras**	**Repita Hasta**
Inicialización	Inicialización	Inicialización
Mientras (***Condición***) **haga**	**Haga**	**Repita**
Instrucción 1	Instrucción 1	Instrucción 1
Instrucción 2	Instrucción 2	Instrucción 2
:	:	:
:	:	:
Modificador de Condición	Modificador de Condición	Modificador de Condición
Instrucción N	Instrucción N	Instrucción N
Fin_Mientras	**Mientras** (***Condición***)	**Hasta** (***Condición***)

Figura 4.20 Formatos de las Mientras Haga, Haga Mientras y Repetir Hasta con Inicialización en el pseudocódigo.

Reemplazando lo hasta ahora visto, se tiene

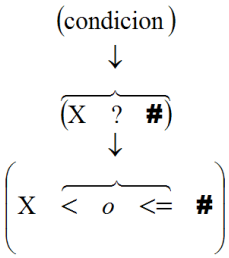
Mientras Haga	**Haga Mientras**	**Repita Hasta**
X ← 0	**X ← 0**	**X ← 0**
Mientras (**X ? TOPE**) **haga**	**Haga**	**Repita**
Instrucción 1	Instrucción 1	Instrucción 1
Instrucción 2	Instrucción 2	Instrucción 2
:	:	:
:	:	:
X ← X + 1	**X ← X + 1**	**X ← X + 1**
Instrucción N	Instrucción N	Instrucción N
Fin_Mientras	**Mientras** (**X ? TOPE**)	**Hasta** (**X ? TOPE**)

Figura 4.21 Formatos de las Mientras Haga, Haga Mientras y Repetir Hasta con la instrucción de Inicialización y del Contador en el pseudocódigo.

El Tope en este caso, representa la cantidad de veces que se debe ejecutar la estructura de repetición, por lo que, el Tope es el valor al cual **X** desea llegar y debe ser inferior a Tope.

Basándose en lo anterior, el operador que reemplazaría al símbolo de interrogación dentro de la condición es un operador relacional: el menor que - **<** - o,

el menor o igual que - ≤ -. El operador puesto en la **condición** debe permitir que se pueda ingresar a la estructura de repetición, a ***condición*** se convierte en:



Supóngase que se desea mostrar en pantalla cinco veces la frase "hola", el trozo de pseudocódigo que realiza esto es el siguiente:

	Prueba de escritorio		
X ← 0	X	**Visualización**	**Comentarios**
Mientras (X < 5) haga	0	hola	Cumple; ejecuta Escribir; incrementa a X
Escribir ("hola")	1	hola	Cumple; ejecuta Escribir; incrementa a X
X ← X + 1	2	hola	Cumple; ejecuta Escribir; incrementa a X
Fin_mientras	3	hola	Cumple; ejecuta Escribir; incrementa a X
	4	hola	Cumple; ejecuta Escribir; incrementa a X
	5		No cumple, X tiene el mismo valor del **Tope**

En el ejemplo anterior, usando el operador menor que, la estructura de repetición se ejecutó 5 veces, lo estipulado en el **TOPE**. Cuando la variable **X** tiene el valor de 5 la **condición** no se cumple y se termina la ejecución de la estructura de repetición Mientras Haga.

Cambiando a menor o igual se tiene:

	Prueba de escritorio		
X ← 0	X	**Visualización**	**Comentarios**
Mientras (X <= 5) haga	0	hola	Cumple; ejecuta Escribir; incrementa a X
Escribir ("hola")	1	hola	Cumple; ejecuta Escribir; incrementa a X
X ← X + 1	2	hola	Cumple; ejecuta Escribir; incrementa a X
Fin_mientras	3	hola	Cumple; ejecuta Escribir; incrementa a X
	4	hola	Cumple; ejecuta Escribir; incrementa a X
	5	hola	Cumple; ejecuta Escribir; incrementa a X
	6		No cumple, X supera el valor del **TOPE**

Resolviendo el mismo ejemplo pero usando el operador menor o igual que, se visualizó seis veces la palabra *hola* no cumpliendo esto con el enunciado, por tanto, el operador menor o igual que, no satisface la solución del problema con una Inicialización de **X** en 0. Sin embargo, al cambiar la Inicialización a 1 y dejando el operador menor o igual que, se tiene:

	Prueba de escritorio		
X ← 1	X	**Visualización**	**Comentarios**
Mientras (X <= 5) haga	1	hola	Cumple; ejecuta Escribir; incrementa a X
Escribir ("hola")	2	hola	Cumple; ejecuta Escribir; incrementa a X
X ← X + 1	3	hola	Cumple; ejecuta Escribir; incrementa a X
Fin_mientras	4	hola	Cumple; ejecuta Escribir; incrementa a X
	5	hola	Cumple; ejecuta Escribir; incrementa a X
	6		No cumple, X supera el valor del **TOPE**

Con el ejemplo anterior se concluye lo siguiente para estructura de repetición Mientras Haga, estructura de repetición Haga Mientrasy estructura de repetición Repetir Hasta, si se usa:

- **X ← 0** se utiliza el operador es **<**
- **X ← 1** se utiliza el operador es **<=**

Reemplazando lo hasta ahora visto, se tiene

Mientras Haga	**Haga Mientras**	**Repetir Hasta**
X ← 0	**X ← 0**	**X ← 0**
Mientras (**X < TOPE**) **haga**	**Haga**	**Repita**
Instrucción 1	Instrucción 1	Instrucción 1
Instrucción 2	Instrucción 2	Instrucción 2
:	:	:
:	:	:
X ← X + 1	**X ← X + 1**	**X ← X + 1**
Instrucción N	Instrucción N	Instrucción N
Fin_Mientras	**Mientras** (**X < TOPE**)	**Hasta** (**X >= TOPE**)

Figura 4.22 Formatos de las estructura de repetición Mientras Haga, Haga Mientras y Repetir Hasta cuando el Tope es numérico, con Inicialización, Contador y condición.

Cuando se usa una estructura de repetición Repetir Hasta si se inicializa con 0 se debe usar el operador mayor o igual que (véase la Figura 4.22), y el símbolo > cuando se inicializa con 1, esto porque se busca que al menos se genere la repetición de la misma una vez.

Cuando se usa la estructura de repetición Para debe tenerse en cuenta lo siguiente: el valor de la Inicialización debe guardarse en la variable **Vi**. El Tope se coloca donde va la variable **Vf**. **INC** o **DEC** cumplirán con la función del Contador. La siguiente figura muestra la aplicación del Tope numérico en la estructura de repetición Para.

```
Para X ← Vi hasta Vf (INC) haga
  Instrucción 1
  Instrucción 2
      :
      :
  Instrucción N
Fin_para
```

a)

```
Para X ← 1 hasta TOPE (INC) haga
  Instrucción 1
  Instrucción 2
      :
      :
  Instrucción N
Fin_para
```

b)

Figura 4.23 Formato de la estructura de repetición Para con Tope numérico. a) Usando las variables Vi y Vf. b) Usando los valores inicial y final.

Normalmente el valor inicial, **Vi**, es 1 pero puede cambiar dependiendo de las características del problema. Se pueden mezclar los casos a) y b) para construir la estructura de repetición Para.

Cuando el número en el enunciado representa la terminación del programa. Por ejemplo, un juego de computadora que termina cuando el usuario presiona la tecla "5", en este caso el programa se ejecuta muchas veces y sólo termina su ejecución cuando se presiona el número 5.

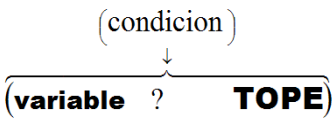
En este caso NO se usa Contador, y el Modificador de Condición depende del uso de dos instrucciones como se muestra a continuación:

Modificador de Condición →
- **Escribir** ("aquí se coloca un texto que pide se ingrese un número para terminar o continuar ")
- **Leer** (***variable***)

Figura 4.24 Instrucciones que conforman el Modificador de Condición cuando el Tope es numérico.

Como se observa en la Figura 4.24, en la instrucción **Escribir** se debe colocar un texto que informe al usuario la necesidad de ingresar un número particular para terminar o continuar con la ejecución del programa. Inmediatamente el usuario ingresa el número, éste es capturado por la instrucción **Leer** y es almacenado en **variable**, ésta es exactamente la misma que aparece en la **condición**. El usuario es quien decide la continuación o no de la ejecución de la estructura de repetición.

La ***condición*** tiene la forma



Es necesario que se satisfagan dos cosas: primero, conocer el operador que reemplazará al símbolo de interrogación y, segundo, que la **variable** tenga un dato almacenado.

Como la estructura de repetición Mientras Haga evalúa la **condición** al comienzo, en **variable** se puede almacenar un dato de dos maneras posibles:

- Inicializándolo con un valor que permita que se ejecute la instrucción de repetición al menos una vez, es decir, debe ser diferente al Tope. Cuando se ejecuta las instrucciones que conforman el Modificador de Condición, se le da el "poder" al usuario para que él decida sobre la continuación de la ejecución de la estructura de repetición.

```
Mientras Haga

Inicialización
Mientras ( variable ? TOPE) haga
  Instrucción 1
  Instrucción 2
       :
       :
  Escribir ("numero a pedir")
  Leer (variable)
  Instrucción N
Fin_Mientras
```

Figura 4.25 Formatos de las estructuras de repetición Mientras Haga, Haga Mientras y Repetir Hasta con las instrucciones que conforman el Modificador de Condición.

- Arrancando por la decisión del Usuario. En este caso se deben colocar las instrucciones del Modificador de Condición también antes de la estructura de repetición. Si el usuario desea que se ejecute la estructura de repetición, él debe ingresar un dato diferente Tope.

```
Mientras Haga

 Escribir ("numero a pedir")
 Leer (variable)
Mientras ( variable <> TOPE) haga
 Instrucción 1
 Instrucción 2
      :
      :
 Escribir ("numero a pedir")
 Leer (variable)
 Instrucción N
Fin_Mientras
```

Figura 4. 27 Formatos de la estructura de repetición Mientras Haga cuando el Tope es numérico, con las instrucciones que conforman el Modificador de Condición, la **condición** construida y el arranque por decisión del Usuario.

Las estructura de repetición Haga Mientras y Repetir Hasta no necesitan Inicialización ya que estas estructuras evalúan la **condición** al final, y cuando esto ocurre **variable** ya contiene un valor que ha sido suministrado por el usuario.

Mientras Haga	**Haga Mientras**	**Repetir Hasta**
variable ← numero diferente a TOPE		
Mientras (*variable* <> **TOPE**) **haga**	**Haga**	**Repita**
Instrucción 1	Instrucción 1	Instrucción 1
Instrucción 2	Instrucción 2	Instrucción 2
:	:	:
:	:	:
Escribir ("numero a pedir")	**Escribir** ("numero a pedir")	**Escribir** ("numero a pedir")
Leer (*variable*)	**Leer** (*variable*)	**Leer** (*variable*)
Instrucción N	Instrucción N	Instrucción N
Fin_Mientras	**Mientras** (*variable* <> **TOPE**)	**Hasta** (*variable* = **TOPE**)

Figura 4.26 Formatos de las estructuras de repetición MIENTRAS HAGA, HAGA MIENTRAS y REPETIR HASTA cuando el Tope es numérico

Cuando el Tope es de tipo **carácter**, **cadena** o **booleano**, como en el caso de "Digite SALIR para terminar", el usuario es quien decide la terminación del programa. Esta categoría no usa Contador y la construcción del Modificador de Condición depende del uso de dos instrucciones como se muestra a continuación:

Modificador de Condición →
- **Escribir** ("aquí se coloca un texto que pide se ingrese un carácter, cadena o booleano para terminar o continuar ")
- **Leer** (***variable***)

Figura 4.28 Instrucciones que conforman el Modificador de Condición cuando el Tope es NO Numérico.

El funcionamiento de este Modificador de Condición es similar al que se vio anteriormente y sólo se diferencia de éste por que el usuario debe ingresar un **carácter**, **cadena** o **booleano** para la terminación de la estructura de repetición.

Igual que en el caso de Tope numérico, la estructura de repetición Mientras Haga debe inicializar la variable de la **condición** antes de arrancar el ciclo. Puede ser iniciada por el programador o por el Usuario. Las estructuras de repetición Haga Mientras y Repetir Hasta no necesitan de Inicialización ya que cuando se evalúa la variable en la **condición** ya posee un valor digitado por el Usuario. A continuación se ilustran los dos casos:

Mientras Haga	**Haga Mientras**	**Repetir Hasta**
variable ← carácter, cadena o booleano		
Mientras (***variable*** <> **TOPE**) **haga**	**Haga**	**Repita**
Instrucción 1	Instrucción 1	Instrucción 1
Instrucción 2	Instrucción 2	Instrucción 2
:	:	:
:	:	:
Escribir ("cadena, carácter o booleano a pedir")	**Escribir** ("cadena, carácter o boolenao a pedir")	**Escribir** ("cadena, carácter o boolenao a pedir")
Leer (***variable***)	**Leer** (***variable***)	**Leer** (***variable***)
Instrucción N	Instrucción N	Instrucción N
Fin_Mientras	**Mientras** (***variable*** <> **TOPE**)	**Hasta** (***variable*** = **TOPE**)

Figura 4.29 Formatos de las estructura de repetición Mientras Haga, Haga Mientras y Repetir Hasta cuando el Tope es NO numérico, con las instrucciones que conforman el Modificador de condición, la **condición** construida y la Inicialización para Mientras Haga**.**

Mientras Haga	**Haga Mientras**	**Repetir Hasta**
Escribir ("cadena, carácter o booleano a pedir")		
Leer (***variable***)		
Mientras (***variable*** **<> TOPE**) **haga**	**Haga**	**Repita**
Instrucción 1	Instrucción 1	Instrucción 1
Instrucción 2	Instrucción 2	Instrucción 2
:	:	:
:	:	:
Escribir ("cadena, carácter o booleano a pedir")	**Escribir** ("cadena, carácter o booleano a pedir")	**Escribir** ("cadena, carácter o booleano a pedir")
Leer (***variable***)	**Leer** (***variable***)	**Leer** (***variable***)
Instrucción N	Instrucción N	Instrucción N
Fin_Mientras	**Mientras** (***variable*** **<> TOPE**)	**Hasta** (***variable*** **= TOPE**)

Figura 4.30 Formatos de las estructura de repetición Mientras Haga, Haga Mientras y Repetir Hasta cuando el Tope es NO numérico, con las instrucciones que conforman el Modificador de condición, la **condición** construida y el Arranque por decisión del Usuario para Mientras Haga**.**

En las figuras 4.29 y 4.30 se puede apreciar que las estructura de repetición Haga Mientras y Repetir Hasta no tienen diferencias entre sí y éstas se diferencian de las figuras 4.27 y 4.26 en que en las primeras solicitan del usuario un **carácter**, **cadena** o **booleano** para la terminación de la estructura de repetición mientras que en las segundas solicitan un número.

A continuación se explicarán los conceptos vistos hasta aquí en el libro, a través de un ejemplo. Escriba un pseudocódigo tal, que dados N números enteros, determine cuántos de ellos son pares y cuántos impares.

Primero se determina si es necesario usar alguna estructura de selección. Para determinar si un número es par, éste se debe comparar por medio de alguna operación empleando el número dos, por ejemplo:

Num **MOD** 2 = 0,

donde Num representa el número ingresado por Usuario. Cuando esta igualdad se cumple, indica que el valor que contiene Num es par. Cuando

Num **MOD** 2 < > 0

se cumple, indica que el valor que contiene Num es impar.

La condición sería: **Si** (Num MOD 2 <> 0) **entonces**

Las operaciones no generan ningún error aritmético sin embargo, hay un caso crítico, la cantidad de números debe ser positiva, la condición para este caso es:

Si (N > 0) **entonces**

donde N representa la cantidad de números.

Ahora se revisará el uso de estructuras de repetición. El enunciado dice ".. dados N números enteros", esto significa primero, que el usuario es quien ingresará la cantidad de datos y el valor de los mismos, y segundo, que a cada dato ingresado se le debe determinar si es par o impar y este análisis se debe hacer N veces, por tanto, se concluye que es necesario el uso de una estructura de repetición.

Primero se identifica el Tope, según el enunciado se ingresarán N números enteros, con base en esto se concluye que el Tope es numérico. A cada número ingresado se le debe determinar si es par o impar esto indica que el proceso de comparación se debe realizar N veces por tanto, en este ejemplo N es el Tope.

Según lo anterior se construye el *Contador*

$$\mathbf{X \leftarrow X + 1}$$

La **condición** y la Inicialización queda:

X ← 0
Mientras (X < N) **haga**

Como en la **condición** existe la variable N, es necesario capturar este valor antes de que se ejecute ésta y para ello se usa las siguientes instrucciones:

Escribir ("Ingrese la cantidad de números")
Leer (N)

Dentro de la estructura de repetición Mientras Haga se debe capturar un número por vez y luego analizarlo para determinar si es par o impar.

Ahora se construye el Algoritmo, y una vez construido éste, se elabora el pseudocódigo y el Diagrama de Flujo respectivo.

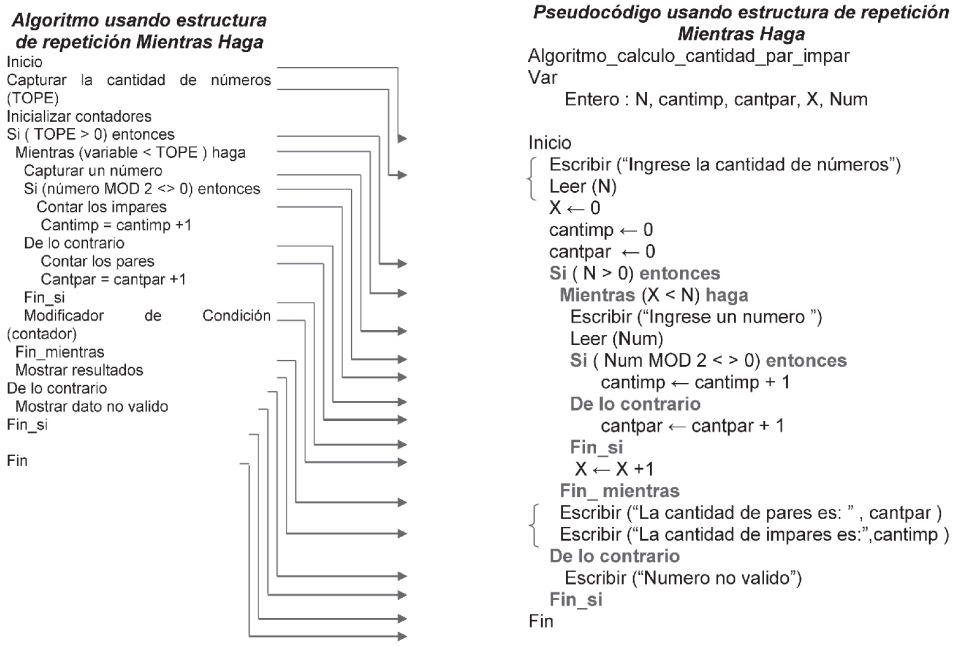


Figura 4.31 Relación entre el algoritmo y el pseudocódigo del ejemplo.

Pseudocódigo usando estructura de repetición

Mientras Haga

```
Algoritmo_calculo_cantidad_par_impar
Var
    Entero : N, cantimp, cantpar, X, Num
Inicio
  Escribir ("Ingrese la cantidad de números")
  Leer (N)
  X ← 0
  cantimp ← 0
  cantpar ← 0
  Si ( N > 0) entonces
   Mientras (X < N) haga
    Escribir ("Ingrese un numero ")
    Leer (Num)
    Si ( Num MOD 2 < > 0) entonces
        cantimp ← cantimp + 1
    De lo contrario
        cantpar ← cantpar + 1
    Fin_si
    X ← X +1
   Fin_ mientras
   Escribir ("La cantidad de pares es: " , cantpar )
   Escribir ("La cantidad de impares es:",cantimp )
  De lo contrario
    Escribir ("Numero no valido")
  Fin_si
Fin
```

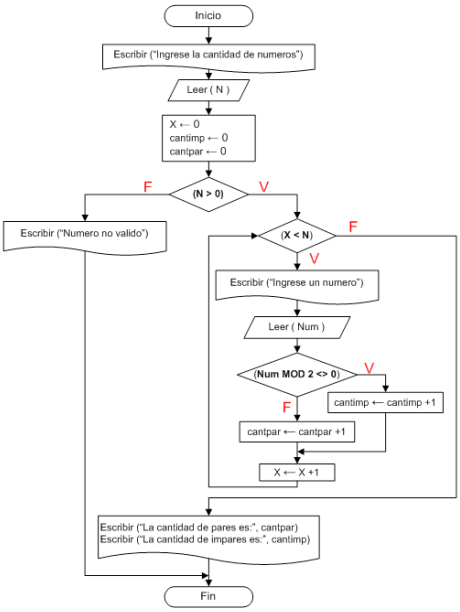


Figura 4.32 Relación entre el diagrama de flujo y pseudocódigo del ejemplo.

El mismo ejemplo se puede resolver utilizando las otras Estructuras de Repetición.

Pseudocódigo usando Haga Mientras

```
Algoritmo_calculo_cantidad_par_impar
Var
    Entero : N, cantimp, cantpar, X, Num
Inicio
Escribir ("Ingrese la cantidad de números")
Leer (N)
X ← 0
cantimp ← 0
cantpar ← 0
Si ( N > 0) entonces
  Haga
    Escribir ("Ingrese un numero ")
    Leer (Num)
    Si ( Num MOD 2 < > 0) entonces
      cantimp ← cantimp + 1
    De lo contrario
      cantpar ← cantpar + 1
    Fin_si
    X ← X +1
  Mientras (X < N)
Escribir ("La cantidad de pares es:" , cantpar )
Escribir("La cantidad de impares es:",cantimp )
De lo contrario
  Escribir ("Numero no valido")
  Fin_si
Fin
```

Pseudocódigo usando Repetir Hasta

```
Algoritmo_calculo_cantidad_par_impar
Var
    Entero : N, cantimp, cantpar, X, Num
Inicio
Escribir ("Ingrese la cantidad de números")
Leer (N)
X ← 0
cantimp ← 0
cantpar ← 0
Si ( N > 0) entonces
  Repita
    Escribir ("Ingrese un numero ")
    Leer (Num)
    Si ( Num MOD 2 < > 0) entonces
      cantimp ← cantimp + 1
    De lo contrario
      cantpar ← cantpar + 1
    Fin_si
    X ← X +1
  Hasta (X >= N)
Escribir ("La cantidad de pares es:" , cantpar )
Escribir("La cantidad de impares es:",cantimp )
De lo contrario
  Escribir ("Numero no valido")
  Fin_si
Fin
```

Pseudocódigo usando Para

```
Algoritmo_calculo_cantidad_par_impar
Var
    Entero : N, cantimp, cantpar, X, Num
Inicio
Escribir ("Ingrese la cantidad de números")
Leer (N)
cantimp ← 0
cantpar ← 0
Si ( N > 0) entonces
  Para X← 1 hasta N INC 1 haga
    Escribir ("Ingrese un numero ")
    Leer (Num)
    Si ( Num MOD 2 < > 0) entonces
      cantimp ← cantimp + 1
    De lo contrario
      cantpar ← cantpar + 1
    Fin_si
  Fin_Para
  Escribir ("La cantidad de pares es:" , cantpar )
  Escribir("La cantidad de impares es:",cantimp )
De lo contrario
  Escribir ("Numero no valido")
Fin_si
Fin
```

Figura 4.33 Solución del ejemplo utilizando estructura de repetición Haga Mientras, Repetir Hasta y Para.

4.6 CASOS ESPECIALES

Siempre que se construya una estructura de repetición es importante tener en cuenta el control de la misma, recuérdese que para ello se debe incluir el Modificador de Condición dentro del cuerpo de la estructura sin embargo, en el desarrollo de un pseudocódigo o programa en general, el programador puede construir una estructura de repetición que se ejecute infinitamente o por el contrario, que nunca se ejecute, el primer caso se conoce como Ciclo Infinito y el segundo como Ciclo Nulo.

Normalmente un Ciclo Infinito puede ocurrir por alguno de los siguientes errores:

a. Cuando el programador no tuvo en cuenta en la construcción del pseudocódigo la creación del Modificador de Condición perdiendo el control total del programa en la ejecución de la estructura de repetición, el siguiente trozo de pseudocódigo muestra esto:

```
X ← 1
Mientras (X < 5) haga
  Escribir ("Hola")
Fin_mientras
```

```
X ← 1
Haga
  Escribir ("Hola")
Mientras (X < 5)
```

```
X ← 1
Repita
  Escribir ("Hola")
Hasta (X >= 5)
```

```
X ← 1
Mientras (X<>5) haga
  Escribir ("Hola")
Fin_mientras
```

```
X ← 1
Haga
  Escribir ("Hola")
Mientras (X <> 5)
```

```
X ← 1
Repita
  Escribir ("Hola")
Hasta (X=5)
```

```
X ← "n"
Mientras (X<>"S" O X<>"s") haga
  Escribir ("Hola")
Fin_mientras
```

```
X ← "n"
Haga
  Escribir ("Hola")
Mientras (X<>"S" O X<>"s")
```

```
X ← "n"
Repita
  Escribir ("Hola")
Hasta (X = "S" O X = "s")
```

Figura 4.34 Ejemplos de Ciclo Infinito por ausencia del Modificador de Condición para las diferentes estructuras de repetición

b. Cuando el programador no construye correctamente el Modificador de Condición, es decir, no tuvo en cuenta el operador aritmético correcto (+ o –) para el Tope numérico o, no incluyó en la instrucción Leer la variable que se encuentra en la condición para el Tope.

```
X ← 6                          X ← 5                          X ← 1
Mientras (X > 5) haga          Haga                           Repita
 Escribir ("Hola")              Escribir ("Hola")              Escribir ("Hola")
 X ← X + 1                      X ← X – 1                      X ← X – 1
Fin_mientras                   Mientras (X < 5)               Hasta (X >= 5)
```

a)

```
X ← 1                          X ← 1                          X ← 1
Mientras (X <> 5) haga         Haga                           Repita
 Escribir ("Hola")              Escribir ("Hola")              Escribir ("Hola")
 Escribir ("Ingrese 5 para      Escribir ("Ingrese 5 para      Escribir("Ingrese 5 para
terminar")                     terminar")                     terminar")
 Leer (Num)                     Leer (Num)                     Leer (Num)
Fin_mientras                   Mientras (X <> 5)              Hasta (X = 5)
```

b)

```
X ← "n"                        X ← "n"                        X ← "n"
Mientras (X<>"S" O             Haga                           Repita
X<>"s") haga                    Escribir ("Hola")              Escribir ("Hola")
 Escribir ("Hola")              Escribir ("Ingrese S para      Escribir("Ingrese S para ter-
 Escribir ("Ingrese S para      terminar")                     minar")
terminar")                      Leer (Res)                     Leer (Res)
 Leer (Res)                    Mientras (X<>"S" O             Hasta (X = "S" O X = "s")
Fin_mientras                   X<>"s")
```

c)

Figura 4.35 Ejemplo de Ciclo Infinito cuando no se construye correctamente el Modificador de Condición para las diferentes estructuras de repetición.

c. Cuando el programador no incluye el Modificador de Condición dentro de la estructura de repetición pero si en el programa.

```
X ← 10
Mientras (X > 5) haga
  Escribir ("Hola")
Fin_mientras
X ← X – 1
```

```
X ← 1
Haga
  Escribir ("Hola")
Mientras (X < 5)
  X ← X + 1
```

```
X ← 1
Repita
  Escribir ("Hola")
Hasta (X >= 5)
X ← X + 1
```

a)

```
X ← 1
Mientras (X <> 5) haga
 Escribir ("Hola")
Fin_mientras
 Escribir ("Ingrese 5 para terminar")
 Leer (X)
```

```
X ← 1
Haga
 Escribir ("Hola")
Mientras (X <> 5)
 Escribir ("Ingrese 5 para terminar")
 Leer (X)
```

```
X ← 1
Repita
 Escribir ("Hola")
Hasta (X = 5)
  Escribir("Ingrese 5 para terminar")
 Leer (X)
```

b)

```
X ← "n"
Mientras (X<>"S" O X<>"s") haga
 Escribir ("Hola")
Fin_mientras
 Escribir ("Ingrese S para terminar")
 Leer (X)
```

```
X ← "n"
Haga
 Escribir ("Hola")
Mientras (X<>"S" O X<>"s")
  Escribir ("Ingrese S para terminar")
 Leer (X)
```

```
X ← "n"
Repita
 Escribir ("Hola")
Hasta (X = "S" O X = "s")
 Escribir("Ingrese S para terminar")
 Leer (X)
```

c)

Figura 4.39 Ejemplo de Ciclo Infinito cuando el Modificador de Condición está por fuera de la estructura de repetición**.**

Un Ciclo Nulo ocurre en aquellas estructuras donde se pregunta antes de ingresar a la estructura de repetición y sucede cuando no se cumple la condición, provocando que la estructura nunca se ejecute. En estructura de repetición Haga Mientras y Repetir Hasta por la forma en que se construyen no se puede dar el Ciclo Nulo ya que en estas se ingresa primero y se pregunta después.

Otra manera de Ciclo Nulo ocurre cuando se utiliza el arranque por decisión del Usuario, y éste digita un valor que hace que no se ingrese a la estructura de repetición.

```
X ← 10
Mientras (X < 5) haga
  Escribir ("Hola")
  X ← X + 1
Fin_mientras
```
a)

```
Escribir ("Ingrese 5 para terminar")
Leer (X)
Mientras (X <> 5) haga
  Escribir ("Hola")
  Escribir ("Ingrese 5 para terminar")
  Leer (X)
Fin_mientras
```
b)

```
Escribir ("Ingrese S para terminar")
Leer (X)
Mientras (X <> "S" O X <> "s" ) haga
  Escribir ("Hola")
  Escribir ("Ingrese S para terminar")
  Leer (X)
Fin_mientras
```
c)

Figura 4.40 Ejemplos de Ciclo Nulo

4.7 EJERCICIOS

4.7.1 Ejercicios con Respuesta

1. Construya un pseudocódigo tal que lea un número entero N, muestre la cantidad de términos y el resultado de la siguiente serie:

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\ldots\pm\frac{1}{N}$$

2. Construya un pseudocódigo tal que encuentre y muestre todos los enteros positivos, comenzando desde el cero, que satisfacen la siguiente expresión:

$$P^3+Q^4-2*P^2 \quad \langle \quad 680$$

4.7.2 Ejercicios sin Respuesta

3. Construya un pseudocódigo que eleve un número **X** a una potencia dada, usando sucesivas multiplicaciones.

4. Construya un pseudocódigo que calcule el factorial de un número.

5. Construya un pseudocódigo que dado un número N, indique si se trata de un número primo o o no.

6. Dada una secuencia de N números, construya un pseudocódigo que muestre por pantalla el resultado de la Sumatoria de los números.

7. Usted acaba de compra el Chimbis, dentro del cual se venden 10 variedades de productos. Decide por estrategia de mercadeo colocar en la entrada los productos con precios más bajos, así que construye un pseudocódigo que le ayude a ordenar ascendentemente por precio todos los productos que hay en bodega y los muestre por pantalla.

4.7.3 Respuesta a los Ejercicios

1. Construya un pseudocódigo tal que lea un número entero N, muestre la cantidad de términos y el resultado de la siguiente serie:

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\ldots\pm\frac{1}{N}$$

Note de la serie anterior que N se encuentra en el denominador y que el mismo se incrementa de uno en uno además, los números pares están precedidos por el signo menos y los impares con el más. N representa la cantidad de números y ésta es ingresada por el usuario. Nótese además que si el usuario ingresa el número cero (0), indicando cero términos de la serie, el pseudocódigo en su ejecución no debe mostrar ni valor alguno ni termino de la serie. El primer término (término 1) es el número 1.

El incremento de uno en uno ofrece la pista para la construcción de la serie y, para llevar a cabo esto, se utiliza la instrucción conocida como el *contador*, es decir:

$$X \leftarrow X + 1$$

Esta instrucción permitirá incrementar de uno en uno la variable *X* que se utilizará para la construcción de la serie. Recuérdese que para ejecutar esta instrucción X debe estar declarada y contener un dato por lo cual se debe inicializar esta variable. El *contador* cumple tres funciones en este ejemplo, primero ofrece el número que se ubicará en el denominador, segundo, informa sobre los pares para así incluir el signo respectivo en la expresión y, por último, lleva la cuenta de la cantidad de veces que se debe realizar o ejecutar la Estructura de Repetición.

Como los pares llevan el signo menos se puede incorporar este signo con base en descubrir cuando la variable X es par, para ello se hace uso de la siguiente expresión:

$$X \quad MOD \quad 2 = 0$$

La anterior ecuación indica que se debe llevar a cabo una comparación para poder descubrir cuando el contenido de X es par, es decir, si el resultado de X **MOD** 2 es igual a 0 el número es par, nótese que ser par implica que el resultado de la comparación debe ser verdadero. X **MOD** 2 = 0 es la condición de una Estructura de Selección Compuesta.

Si el usuario ingresa el número 3, el pseudocódigo construido debe mostrar la serie de la siguiente manera:

$$1 - \frac{1}{2} + \frac{1}{3}$$

Y a su vez debe visualizar el resultado de esa suma, es decir, 0.8333333

Para obtener el valor anterior, el programa construido debe restar ½ de 1 y el resultado debe ser sumado a **1/3**. En el pseudocódigo existe una instrucción que permite conservar resultados para que éstos puedan ser usados nueva-

mente con otros valores y así obtener el valor final, la misma es conocida con el nombre de *acumulador*, y se construye de la siguiente manera:

$$sum \leftarrow sum + X$$

La variable *sum* contendrá el resultado de la ejecución de la instrucción anterior, conservando así el valor que después se sumara con el contenido de X generándose un nuevo dato para *sum*. Recuérdese que en el pseudocódigo ejecutar una instrucción de asignación implica que primero se debe resolver el lado derecho de la flecha y el resultado se almacena en el lado izquierdo. Con base en esto, antes de ejecutarse el *acumulador* primero se debe asignar un valor a la variable que se encuentra al lado izquierdo de la flecha, si esto no se hace, *sum* no tendría dato alguno produciéndose un error en la ejecución de esta instrucción. El valor inicial de *sum* debe ser cero para que el mismo no afecte el resultado de la operación suma.

Nótese que el *acumulador* es muy similar al *contador*, aquella se diferencia de ésta en el uso de dos variables mientras que en el *contador* se utiliza un número como segundo operando.

Al igual que en el *contador*, en la construcción del *acumulador* se debe tener en cuenta el uso de una misma variable la cual se debe encontrar a ambos lados de la flecha.

En la solución del ejemplo 4.2 es necesario el uso de dos acumuladores, el primero para los signos positivos y, el segundo, para los negativos, quedando:

$$sum \leftarrow sum + \frac{1}{X}$$

$$sum \leftarrow sum - \frac{1}{X}$$

Si X **MOD** 2 =0 se utiliza el *acumulador* con el signo menos, de lo contrario se usa la otra expresión.

Para resolver el ejemplo primero se aplican las herramientas de la sección 4.1.5 para determinar si es necesario usar alguna **ES**, por tanto se tiene:

Herramienta #

1. No se cumple

2. Para determinar si un numero es par, éste se debe comparar por medio de la siguiente expresión

X **MOD** 2 = 0

Cuando esta igualdad se cumple, indica que el valor que contiene X es par, de lo contrario, el contenido de X es impar.
Nótese como con el uso de comparaciones (utilización del operador =) se puede resolver el problema por tanto, esta herramienta se cumple y se necesita usar una **ES**, la condición es

Si (X **MOD** 2 = 0) **entonces**

Si se cumple esta condición se debe ejecutar el acumulador con el signo menos. Para esta herramienta se puede utilizar **ESS** o **ESC.**

3. Nótese que la serie utiliza signos positivos para los pares y negativos para los impares; conocer si es par o impar depende de una única expresión, X **MOD** 2, un resultado diferente a cero indica que el numero contenido en X es impar, de lo contrario es par. Se puede utilizar una **ESM** para resolver este ejercicio quedando la misma de la siguiente manera:

Si (X **MOD** 2) **igual**

0 **:** operaciones para par
1 **:** operaciones para impar

Nótese que existen dos maneras para resolver el ejemplo 4.2, el programador podrá escoger cualquiera de ellas, en este texto se llevará a cabo la solución utilizando primero **ESS** o **ESC** y después **ESM**.

4. Operaciones Aritméticas

Las operaciones para resolver el ejercicio son:

$$sum \leftarrow sum + \frac{1}{X}$$

$$sum \leftarrow sum - \frac{1}{X}$$

Nótese que estas ecuaciones generan error aritmético cuando X contiene el valor cero, para resolver esta inquietud se puede escoger uno de dos caminos, primero, se inicializa a X en cero pero se coloca el *contador* antes de la ejecución de alguna de las instrucciones del *acumulador* o, segundo, se da un valor inicial a X de 1 y se coloca al *contador* después de la ejecución del *acumulador*.

4. Casos Críticos

Se cumple por tanto se requiere una **ES**. La cantidad de números debe ser positiva, la condición para este caso es:

Si (N > 0) **entonces**

Donde N representa la cantidad de números, información ingresada por el usuario. Esta estructura debe colocarse inmediatamente después de capturar el valor de N.

Ahora se aplican las herramientas de la sección 4.1.5 para determinar si es necesario usar alguna **ER**, por tanto se tiene:

Herramienta #

1. Nótese del enunciado del ejemplo 4.2 que según la cantidad ingresada por el usuario, se deben realizar en igual cantidad las sumas y restas, es decir, si el usuario ingresa el número 4, primero el pseudocódigo en su ejecución debe mostrar

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}$$

Pero después debe visualizar el resultado de esta expresión (sumas y restas) y la misma depende de la cantidad de elementos que el usuario desea por lo que, el calculo a realizar está en función de N, por tanto, se concluye que es necesario el uso de una **ER**.

2. Sobre la ***condición***

Primero se identifica el **TOPE**, según el enunciado se ingresará un número que representa la cantidad de términos a visualizar y sobre éstos se realizará un cálculo. La cantidad de elementos a sumar o restar depende directamente de la cantidad de términos a visualizar que a su vez es el valor ingresado por el usuario y almacenado en la variable N, entonces N representa la cantidad de veces que se debe realizar la operación, con base en esto se concluye que el **TOPE** es numérico y **caso 1**. En este ejemplo N es el **TOPE**. Según lo anterior se construye el *contador*

$$\mathbf{X \leftarrow X + 1}$$

Para un **TOPE** numérico **caso 1** se puede utilizar cualquier Estructura de Repetición. Según la figura 4.22 para la **ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA**, la ***condición*** y la *Inicialización* queda

X ← 0

Mientras (X < N) **haga**

Como en la ***condición*** existe la variable N, es necesario capturar este valor antes de que se ejecute ésta y para ello se usa las siguientes instrucciones

Escribir ("Ingrese la cantidad de términos de la serie a visualizar y obtener
su resultado")
Leer (N)

Después de capturar a N, se coloca la **condición** del caso crítico.
Dentro de la **ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA** se debe construir la serie y después obtener el resultado de la misma.

Ahora se aplican las herramientas de la sección 4.1.6 y 4.1.6 para construir el Algoritmo, una vez construido éste, se elaborará el pseudocódigo. Primero se resolverá el ejemplo utilizando **ESS**, luego usando **ESC** y por ultimo **ESM**, se elaborará el pseudocódigo con la Estructura de Repetición **Para**.

Solución al ejemplo 4.2 usando **ESS** y **ERP**

Herr. ***ALGORITMO usando ESS***

1. Inicio

2. Capturar la cantidad de términos de la serie a visualizar (TOPE).

3. Inicializar acumulador
Si (TOPE>0) entonces
Para X = 1 hasta TOPE haga
Si (X **MOD** 2 = 0) entonces
Crear serie con negativos
sum = sum – 1/X
visualizar termino
Fin_si
Si (X **MOD** 2 <> 0) entonces
Crear serie con positivos
sum = sum + 1/X
visualizar termino
Fin_si
Fin_para
Mostrar el resultado de la serie
Fin_si
Si(TOPE <= 0) entonces
Mostrar dato no valido
Fin_si

4. Fin

Pseudocódigo usando ESS y ERP

```
Algoritmo_muestra_terminos_y_resultado_serie
var
    Decimal : sum
    Entero: X,N
Inicio
Escribir(" Ingrese la cantidad de términos de la serie a
                visualizar y obtener su resultado ")
Leer (N)
sum ← 0
Si (N >0) entonces
   Escribir ("La serie es:")
   Para X← 1 hasta N haga
     Si (X MOD 2 = 0) entonces
        sum ← sum – 1/X
        Escribir ( "– 1/", X)
     Fin_si
     Si (X MOD 2 <> 0) entonces
        sum ← sum + 1/X
        Si (X =1) entonces
           Escribir ( "1")
        Fin_si
        Si (X <>1) entonces
           Escribir ( "+ 1/", X)
        Fin_si
     Fin_si
   Fin_para
   Escribir ("El resultado de la serie es", sum)
Fin_si
Si (N<=0) entonces
   Escribir ("Dato no valido")
Fin_si
Fin
```

Nótese que en la construcción del pseudocódigo, en la visualización de los términos con signo positivo se hizo uso de dos Estructuras de Selección, esto es con el fin de mostrar el primer elemento de la serie, el número 1, sin que éste aparezca junto a los operadores + y /. La segunda instrucción Escribir se utiliza para que aparezca una sola vez la frase *La serie es:*, si esta instrucción se ubica dentro de la Estructura de Repetición, la misma aparecerá la cantidad de veces que representa el **TOPE**.

A continuación se resuelve el ejemplo 4.2 usando **ESC**.

Solución al ejemplo 4.2 usando **ESM** y **ERP**

Herr. ***ALGORITMO usando ESM***

1. Inicio

2. Capturar la cantidad de términos de la serie a visualizar (TOPE).

3. Si (TOPE>0) entonces
  Para X = 1 hasta TOPE haga
    Si (X **MOD** 2) igual
      0: Crear serie con negativos
        sum ← sum – 1/X
        visualizar termino
      1: Crear serie con positivos
        sum ← sum + 1/X
        visualizar termino
    Fin_si
  Fin_para
  Mostrar el resultado de la serie
  De lo contrario
  Mostrar dato no valido
  Fin_si

4. Fin

Pseudocódigo usando ESM y ERP

```
Algoritmo_muestra_terminos_y_resultado_serie
var
    Decimal : sum
    Entero: X,N
Inicio
Escribir(" Ingrese la cantidad de términos de la serie a
            visualizar y obtener su resultado ")
Leer (N)
sum ← 0
Si (N >0) entonces
   Escribir ("La serie es:")
   Para X← 1 hasta N haga
     Si (X MOD 2) igual

        0: sum ← sum – 1/X
           Escribir ( "– 1/", X)
        1: sum ← sum + 1/X
           Si (X ) igual
                1: Escribir ( "1")
           De lo contrario: Escribir ( "+ 1/", X)
           Fin_si
     Fin_si
   Fin_para
   Escribir ("El resultado de la serie es", sum)
De lo contrario
   Escribir ("Dato no valido")
Fin_si
Fin
```

Los operadores +, – y / que se encuentran dentro de las instrucciones Escribir sirven para se visualicen cuando éstas se ejecutan esto con el fin de construir la serie pedida.

A continuación se resuelve el ejemplo 4.2 usando **ESM**.

Solución al ejemplo 4.2 usando **ESM** y **ERP**

Herr. ***ALGORITMO usando ESM***

1. Inicio
2. Capturar la cantidad de términos de la serie a visualizar (TOPE).
3. Si (TOPE>0) entonces
   Para X = 1 hasta TOPE haga
   Si (X **MOD** 2) igual
   0: Crear serie con negativos
   sum ← sum – 1/X
   visualizar termino
   1: Crear serie con positivos
   sum ← sum + 1/X
   visualizar termino
   Fin_si
   Fin_para
   Mostrar el resultado de la serie
   De lo contrario
   Mostrar dato no valido
   Fin_si
4. Fin

Pseudocódigo usando ESM y ERP

```
Algoritmo_muestra_terminos_y_resultado_serie
var
    Decimal : sum
    Entero: X,N
Inicio
Escribir(" Ingrese la cantidad de términos de la serie a
            visualizar y obtener su resultado ")
Leer (N)

sum ← 0
Si (N >0) entonces
   Escribir ("La serie es:")
   Para X← 1 hasta N haga
     Si (X MOD 2) igual
        0: sum ← sum – 1/X
           Escribir ( "– 1/", X)
        1: sum ← sum + 1/X
           Si (X ) igual
                1: Escribir ( "1")
           De lo contrario: Escribir ( "+ 1/", X)
           Fin_si
     Fin_si
   Fin_para
   Escribir ("El resultado de la serie es", sum)
De lo contrario
   Escribir ("Dato no valido")
Fin_si
Fin
```

Nótese en todas las soluciones al ejemplo 4.2 que únicamente se visualiza el número 1 cuando X contiene el valor de 1 y además, el ingreso a esta zona del programa ocurre cuando X **MOD** 2 no es igual a cero, recuérdese que se evalúa al contenido de X para mostrar el 1 o el termino positivo de la serie mas no el resultado de la operación Modulo.

En la última solución del ejemplo 4.2 se puede apreciar el uso de dos **ESM**, la primera para permitir la construcción de la serie para los términos pares e impares y la segunda, para visualizar los elementos positivos de la serie, en esta última se evalúa el contenido de X que determinará cual de las dos instrucciones Escribir se debe ejecutar. Nótese como se puede utilizar una **ESM** con solo dos caminos a escoger sin embargo, las ventajas de esta estructura se desaprovechan, es decir, para dos caminos en donde se puede escoger uno de ellos es más recomendable el uso de una **ESC**, aunque no tener en cuenta esto no significa que se genere error en la ejecución del pseudocódigo. A continuación se muestran los diagramas de flujo para cada una de las soluciones presentadas al ejemplo 4.2.

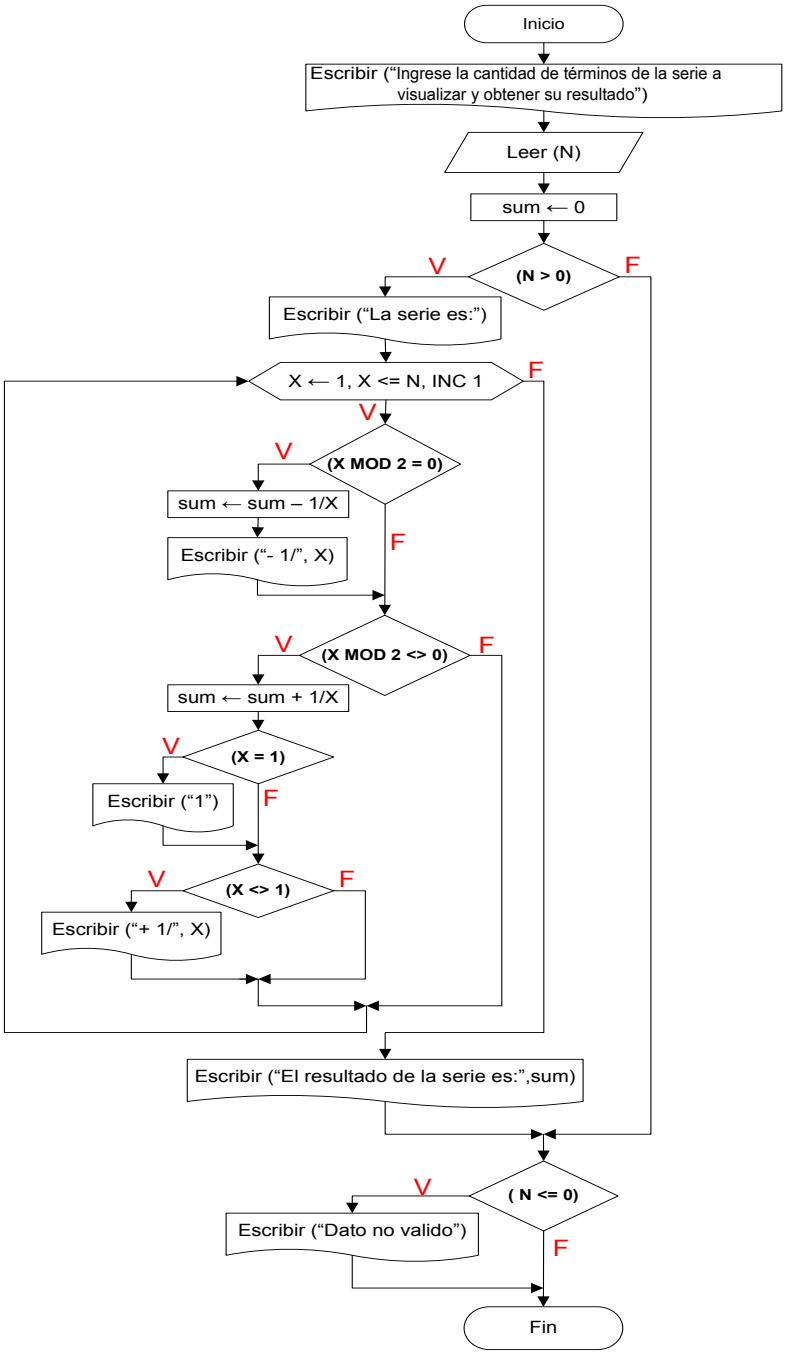


Figura 4.33 Solución del ejemplo 4.2 usando **ESS** y **ERP**.

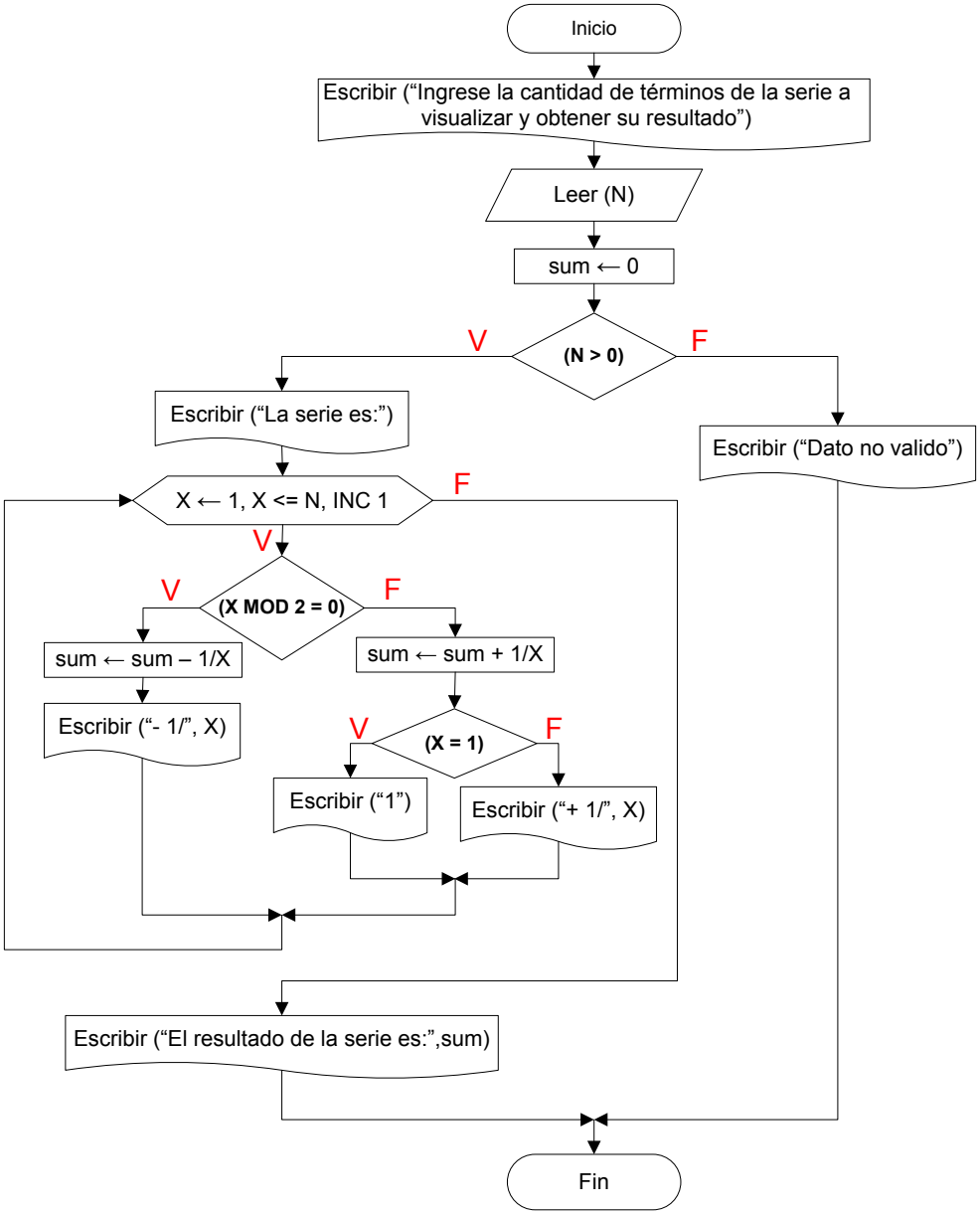


Figura 4.34 Solución del ejemplo 4.2 usando **ESC** y **ERP**.

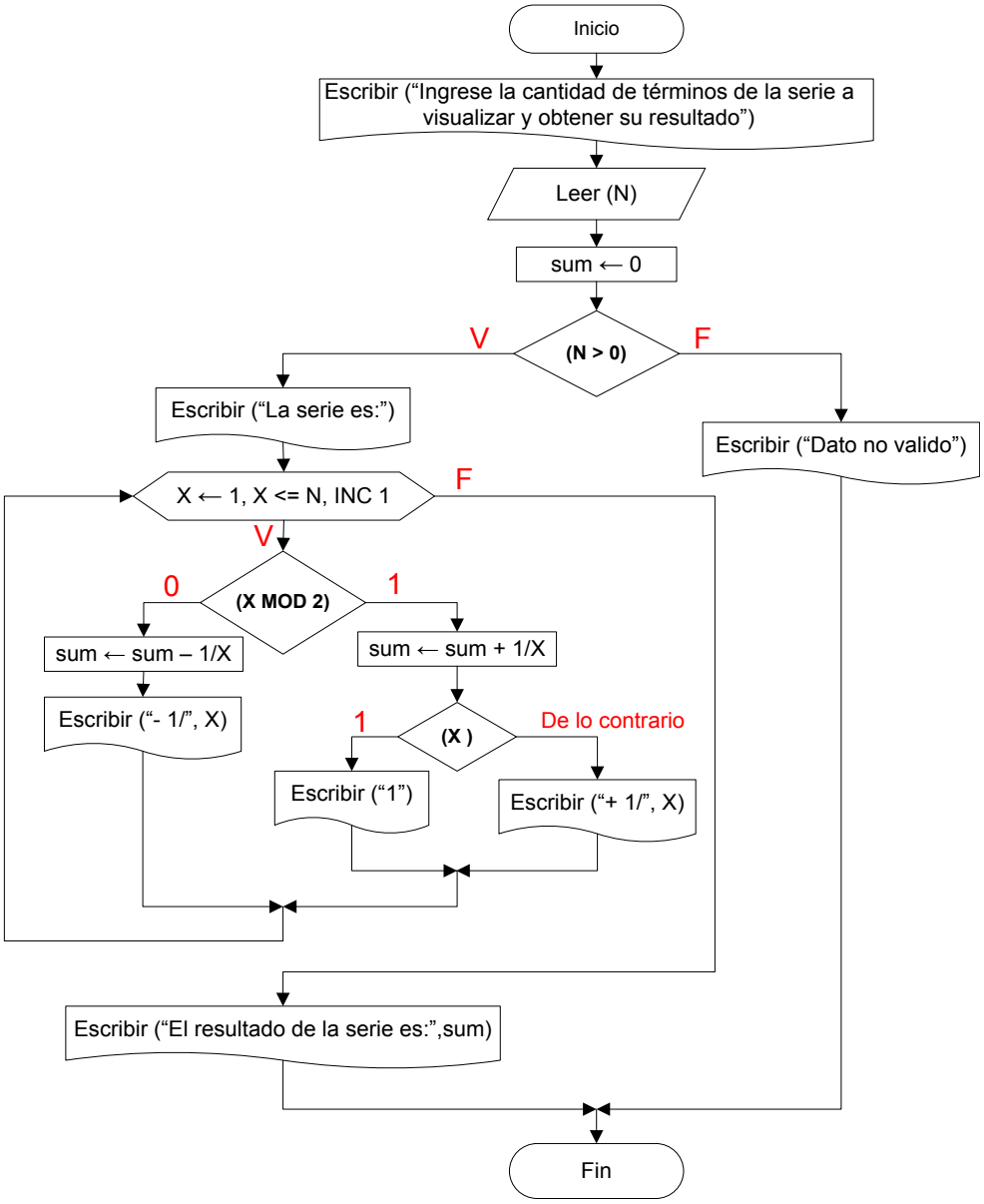


Figura 4.35 Solución del ejemplo 4.2 usando **ESM** y **ERP**.

Note que las diferencias visuales entre las figuras 4.34 y 4.35 no son muy perceptibles, tan solo se ha cambiado las letras V y F por 0 y 1 o por 1 y De lo contrario así como también se ha creado el selector, pero el comportamiento es similar.

2. Construya un pseudocódigo tal que encuentre y muestre todos los enteros positivos, comenzando desde el cero, que satisfacen la siguiente expresión:

$$P^3 + Q^4 - 2 * P^2 \quad \langle \quad 680$$

La solución busca que se reemplacen tanto P como Q por números enteros de forma que al evaluarse la inecuación anterior la misma se cumpla y, si esto ocurre se deben mostrar estos valores. Nótese que pueden surgir muchas posibles combinaciones de forma que se satisfaga la anterior desigualdad, por ejemplo, P=Q=0 produce el mismo resultado que P=0 y Q=1, es decir, satisfacen la inecuación anterior.

Para cada valor de P y Q se debe evaluar la expresión indicando esto que el proceso de comprobación de la desigualdad se lleva a cabo muchas veces y el mismo termina cuando la inecuación no se satisface.

Nótese que para P=0 existen varios valores de Q que satisfacen la inecuación sugiriendo esto que la mejor opción de resolver el ejemplo 4.3 es darle un valor a P y a Q de arranque y crear las líneas de pseudocódigo necesarias para que cambie Q, dejando a P fijo temporalmente, hasta que la desigualdad no se satisfaga para después incrementar el valor de P y repetir el proceso de nuevo. Obsérvese que el punto de partida y de parada de la solución al ejemplo es la inecuación anterior.

Ahora se aplican las herramientas de la sección 4.1.5 para determinar si es necesario usar alguna **ES**, entonces:

Herramienta #	
1.	No se cumple
2.	En efecto se debe realizar una comparación, es decir, resolver el ejemplo implica la comprobación de la desigualdad por lo que se podría pensar en el uso de una **ES** para llevar a cabo esto, sin embargo, la inecuación se debe evaluar muchas veces y en este tipo de ejemplo en particular es más recomendable no utilizar **ES** ya que no se conoce el valor exacto de veces que se debe comprobar la inecuación.
3.	No se cumple pues el ejemplo no ofrece varios caminos a escoger.
4.	Operaciones Aritméticas Las operación para resolver el ejercicio es: $P^3 + Q^4 - 2 * P^2 \quad \langle \quad 680$ La misma no genera error aritmético para cualquier valor de P y Q, por tanto esta herramienta no se cumple.
4.	Casos Críticos Como es el programa construido que debe generar los resultados para P y Q, éstos deben ser positivos y es el programador quien crea estos valores por lo que esta herramienta no se cumple.

Ahora se aplican las herramientas de la sección 4.1.5 para determinar si es necesario usar alguna **ER**, entonces:

Herramienta #	
1.	Según el ejemplo 4.3 se debe evaluar varias veces la desigualdad, por realizar esta labor más de una vez entonces es necesario el uso de una **ER**.

2. Sobre la ***condición***

Primero se identifica el **TOPE**, según el enunciado quién determina la cantidad de veces que se debe ejecutar una **ER** es la condición generando verdadero o falso al ser evaluada, como se debe comprobar la inecuación cada vez que existe un nuevo valor para P o Q, entonces la mejor manera de detener la ejecución o no de la **ER** es colocando como condición la desigualdad, pero, esta expresión está conformada por dos variables, P y Q, las cuales deben contener un valor antes de ser evaluada por lo que inicialmente a P y Q se les debe dar un valor de arranque, para este caso es 0.

Nótese que se debe ejecutar la **ER** un numero finito de veces determinado por el cumplimiento de la desigualdad, esta cantidad es desconocida y solo podrá ser establecida en el momento de ejecución del pseudocódigo, sin embargo, el lector como ejercicio puede descubrir hasta que valor máximo para P y Q se cumple la inecuación permitiendo esto la construcción de una condición para las **ER**, pero esto implica que el programa construido solo funcione para el ejemplo en particular y, no para cualquier modificación del mismo, por tanto en este ejemplo se desarrollará una solución general y no particular (aquella en la cual se conoce el tope para P y Q).

Como se debe evaluar un número finito de veces la inecuación, la misma se considera como **TOPE** numérico **caso 1** y, se utilizan dos contadores, uno para P y el otro para Q de la siguiente manera:

P ← P + 1 **Q ← Q + 1**

Para un **TOPE** numérico **caso 1** se puede utilizar cualquier Estructura de Repetición sin embargo, para resolver este ejemplo no se puede utilizar la **ERP** ya que ésta no permite construir una condición como la inecuación lo requiere. Según la figura 4.22 para la **ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA**, la ***condición*** y la *Inicialización* queda

P ← 0
Q ← 0
Mientras ($P^3 + Q^4 - 2 * P^2 \quad \langle \quad 680$) **haga**

En la solución del ejemplo se utilizaran dos **ER** una para P y la otra para Q, en cada una de ellas se afectara la variable pertinente, pero las dos tendrán la misma condición.

Recuérdese que en la solución de este ejemplo no se requiere que el usuario ingrese valor alguno, éstos son construidos por la ejecución del programa creado por el programador.

Este ejemplo muestra un caso particular del **TOPE** numérico **caso 1**, en donde el usuario no ingresa dato alguno pero se requiere generar muchos valores dependiendo de la comprobación de una desigualdad.

Ahora se aplican las herramientas de la sección 4.1.6 para construir el Algoritmo, una vez construido éste, se elaborará el pseudocódigo. Primero se resolverá el ejemplo utilizando **ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA**, luego usando **ESTRUCTURA DE REPETICIÓN HAGA MIENTRAS**y por último **ESTRUCTURA DE REPETICIÓN REPETIR HASTA**.

Solución al ejemplo 4.3 usando **ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA**

Herr. ***ALGORITMO***

1. Inicio

2. No aplica.

3. Inicializar contadores

Mientras($P^3 + Q^4 - 2P^2 \langle \quad 680$) haga

Mientras($P^3 + Q^4 - 2P^2 \langle \quad 680$) haga

Mostrar los valores de P y Q

Incrementar a Q

Q = Q+1

Fin_mientras

Incrementar a P

P = P+1

Inicializar a Q en cero

Fin_mientras

4. Fin

Pseudocódigo usando ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA

```
Algoritmo_muestra_terminos_P_y_Q
var
    Entero: P,Q
Inicio
 P ← 0
 Q ← 0
 Escribir ( "Los valores de P y Q son: ")
 Mientras (P ** 3 + Q ** 4 – 2 * P ** 2 < 680) haga
   Mientras (P ** 3 + Q ** 4 – 2 * P ** 2 < 680) haga

    Escribir ( P, Q)
    Q ← Q + 1
   Fin_mientras
   P ← P + 1
   Q ← 0
 Fin_mientras
Fin
```

Solución al ejemplo 4.3 usando **ESTRUCTURA DE REPETICIÓN HAGA MIENTRAS**

Herr. ***ALGORITMO***

1. Inicio
2. No aplica.
3. Inicializar contadores
   Haga
     Haga
       Mostrar los valores de P y Q
       Incrementar a Q
         Q = Q+1
     Mientras($P^3 + Q^4 - 2P^2 \langle \quad 680$)
     Incrementar a P
       P = P+1
     Inicializar a Q en cero
   Mientras($P^3 + Q^4 - 2P^2 \langle \quad 680$)
4. Fin

Pseudocódigo usando ESTRUCTURA DE REPETICIÓN HAGA MIENTRAS

```
Algoritmo_muestra_terminos_P_y_Q
var
     Entero: P,Q
Inicio
 P ← 0
 Q ← 0
 Escribir ( "Los valores de P y Q son: ")
 Haga
   Haga
     Escribir ( P, Q)
     Q ← Q + 1
   Mientras (P ** 3 + Q ** 4 – 2 * P ** 2 < 680)
   P ← P + 1
   Q ← 0
 Mientras (P ** 3 + Q ** 4 – 2 * P ** 2 < 680)
Fin
```

Nótese en ambas soluciones que se inicializa dos veces la variable Q, la primera como valor de arranque y la segunda, como reinicio del valor más bajo que puede tomar Q.

Solución al ejemplo 4.3 usando
ESTRUCTURA DE REPETICIÓN REPETIR HASTA

Herr. ***ALGORITMO***

1. Inicio

2. No aplica.

3. Inicializar contadores
Repita
Repita
Mostrar los valores de P y Q
Incrementar a Q
Q = Q+1
Hasta ($P^3 + Q^4 - 2P^2 \rangle =$ 680)
Incrementar a P
P = P+1
Inicializar a Q en cero
Hasta ($P^3 + Q^4 - 2P^2 \rangle =$ 680)

4. Fin

Pseudocódigo usando ESTRUCTURA DE REPETICIÓN REPETIR HASTA

```
Algoritmo_muestra_terminos_P_y_Q
var
    Entero: P,Q
Inicio
 P ← 0
 Q ← 0
 Escribir ( "Los valores de P y Q son: ")
 Repita
   Repita
    Escribir ( P, Q)
    Q ← Q + 1
   Hasta (P ** 3 + Q ** 4 – 2 * P ** 2 >= 680)
   P ← P + 1
   Q ← 0
 Hasta (P ** 3 + Q ** 4 – 2 * P ** 2 >= 680)
Fin
```

La primera instrucción Escribir es utilizada para que cuando se ejecute aparezca el contenido de la misma y a través de ésta se informe al usuario sobre los valores de P y Q que satisfacen la desigualdad, los cuales se muestran cuando se ejecuta la instrucción Escribir (P,Q).

Nótese que terminar de ejecutar la Estructura de Repetición más interna implica que el valor de Q sea tal que al ser reemplazado en la desigualdad en esta estructura, produce que la misma no se satisfaga, y cuando esto ocurre, se ha encontrado el valor más alto de Q para un P determinado, sin embargo, si Q no toma un nuevo valor una vez se termine la ejecución de la **ER** mas interna, ocasionará que no se cumpla la condición de la Estructura de Repetición mas externa, provocando esto que no se encuentren todos los valores de P, por

tanto, es importante inicializar en cero a Q antes de que el programa retorne a evaluar la condición mas externa.

Las soluciones del ejemplo 4.3 entregan todas las parejas de P y Q que satisfacen la desigualdad

$$P^3 + Q^4 - 2P^2 \langle \quad 680$$

Esto significa que para un P determinado, en la ejecución del programa se encontraran y visualizaran todos los que Q que junto a P satisfacen la inecuación.

La figura siguiente muestra el diagrama de flujo respectivo.

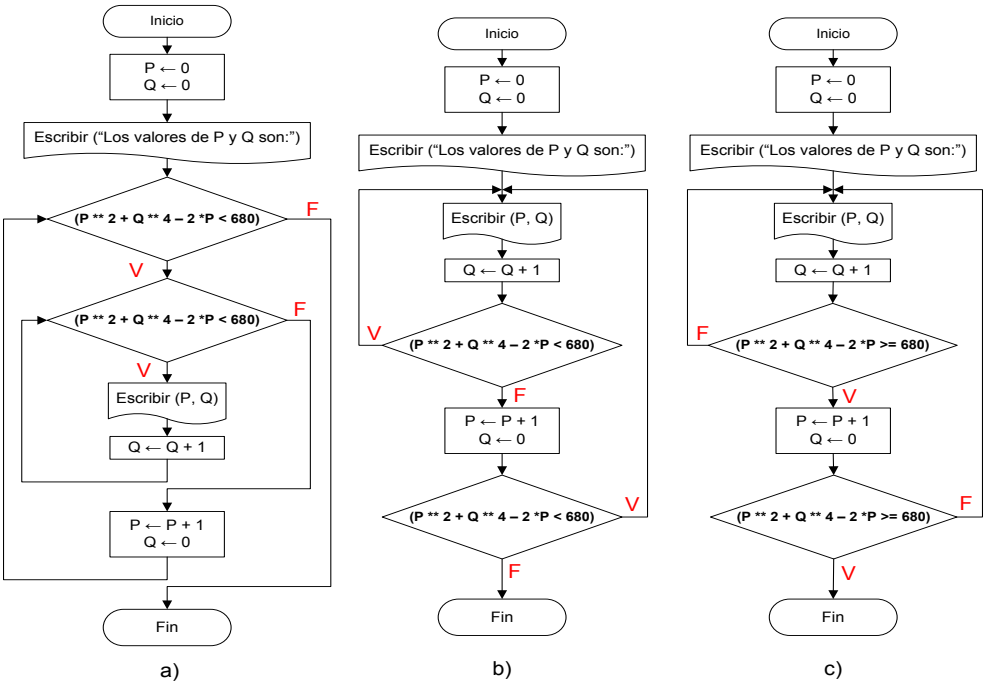


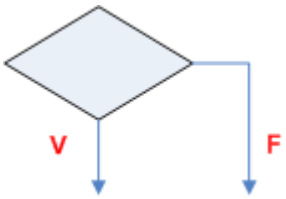
Figura 4.36 Solución del ejemplo 4.3 usando Estructuras de Repetición.
Mientras haga - **ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA**
Haga Mientras - **ESTRUCTURA DE REPETICIÓN HAGA MIENTRAS**
Repita Hasta - **ESTRUCTURA DE REPETICIÓN REPETIR HASTA**

ANEXOS

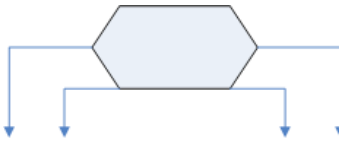
ANEXO 1

SÍMBOLOS UTILIZADOS EN LOS DIAGRAMAS DE FLUJO

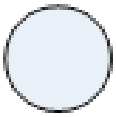
Representación del Símbolo	Explicación del Símbolo
	Utilizado para marcar el *Inicio* y *Fin* de todo Diagrama de Flujo. Asociado con las Instrucciones **Inicio** y **Fin** del Pseudocódigo.
	Utilizado para representar la *entrada y visualización de datos*. Este símbolo está asociado con las instrucciones **Leer** y **Escribir** del Pseudocódigo. Palabras afines para el uso de este símbolo son para: **Entrada – Leer -** **Visualización – Escribir -** Capturar. Mostrar. Conocer. Visualizar. Ingresar. Imprimir.
	Utilizado para representar *procesos*, es decir, llevar a cabo operaciones con los datos. Este símbolo está asociado con la instrucción de **Asignación** del Pseudocódigo. Palabras afines para el uso de este símbolo son: Calcular. Resolver.



Utilizado para representar la *Decisión*. Dentro del rombo se coloca una *condición* y ésta al ser evaluada genera uno de dos resultados posibles – Verdadero o Falso -, según el resultado es el camino o rama que se toma. Este símbolo está asociado con las instrucciones de **Bifurcación** – Estructuras de Selección y de Repetición-**.**



Utilizado para representar la *Decisión Múltiple*. Dentro del rombo se coloca un *selector* y éste almacena un valor, dependiendo del valor almacenado se tomará la rama respectiva. Una de sus ramas es conocida como *De lo contrario* y es opcional su uso. Este símbolo está asociado con las instrucciones de **Bifurcación** y únicamente con la Estructura de Selección Múltiple**.**



Utilizado para representar *conexión dentro de una misma página*. Al existir varios de estos símbolos en una misma pagina, se recomienda el uso de un número dentro del círculo**.** Este conector se usa cuando no hay mas espacio para el Diagrama de Flujo, se llega a él por medio de una flecha y es el último símbolo a colocar. Para continuar con el Diagrama de Flujo, se comienza con el conector y de él sale una flecha que se dirige al siguiente símbolo del Diagrama de Flujo.

Utilizado para representar *conexión entre páginas diferentes*. Solo puede existir uno de estos símbolos por página. Si en la construcción de un Diagrama de Flujo se requiere el uso de varios de estos conectores, se recomienda el uso de un número dentro de la figura. Este conector se usa cuando no hay más espacio para el Diagrama de Flujo, se llega a él por medio de una flecha y es el último símbolo a colocar. Para continuar con el Diagrama de Flujo, se comienza con el conector y de él sale una flecha que se dirige al siguiente símbolo del Diagrama de Flujo.

Para indicar la *dirección del flujo* del Diagrama de Flujo, se deben utilizar flechas las cuales deben ser rectas, horizontales y verticales. No se permite el uso de flechas inclinadas y ni cruzadas. Dos flechas, de una manera independiente, no pueden llegar a un mismo símbolo.
Cada flecha sale del centro del símbolo y llega al centro del símbolo siguiente.

Utilizado para representar *Llamado a Subrutina*. En él se encontrará un pequeño programa que resuelve un subproblema del problema principal.

Utilizado para representar *Impresión de un resultado*. Dentro de él se coloca lo que el programador desea sea impreso. Este símbolo está asociado con la instrucción **Escribir** del Pseudocódigo.
Palabras afines para el uso de este símbolo son:
Imprimir
Mostrar.
Visualizar.
Escribir.

Utilizado para *mostrar un resultado en pantalla*. Dentro de él se coloca lo que el programador desea sea impreso. Este símbolo está asociado con la instrucción **Escribir** del Pseudocódigo.

Utilizado para representar *Entrada manual de datos*.

BIBLIOGRAFÍA

Becerra Santamaria, C. A. (2001). *Los 600 Principales Metodos del Java 2a. Edición.* Bogotá, Colombia: Kimpres Ltda.

Findlay, W. (1984). *Pascal, Programación Metódica.* (L. J. Cearra Zabala, Trad.) Madrid, España: Rueda.

Joyanes Aguilar, L. (2008). *Fundamentos de Programación: Algoritmos, estrucutras de datos y objetos* (4ta edición ed.). Madrid, España: Mc Graw-Hill Interamericana.

Joyanes Aguilar, L., & Zahonero Martinez, I. (2001). *Programación en C: Metodología, algoritmos y estructuras de datos y objetos.* Madrid, España: Mc Graw-Hill Interamericana.

Joyanes Aguilar, L., Rodríguez Baena, L., & Fernández Azuela, M. (1996). *Fundamentos de Programación: Libro de Problemas.* Madrid, España: Mc Graw-Hill Interamericana.

Juganaru Mathieu, M. (2014). *Introducción a la Programación.* México D.F., México: Grupo Editorial Patria.

Martín Quetglás, G., Toledo Lobo, F., & Cerverón Lleó, V. (2002). Fundamentos de Informática y Programación. Valencia, España.

Pareja Flores, C. (1997). *Desarrollo de algoritmos y Técnicas de Programación en Pascal.* Madrid, España: RA-MA.

Rodríguez A., M. Á. (1991). *Metodología de Programación a través de Pseudocódigo* (1era Edición ed.). McGraw Hill.

Rothwell, T., & Youngman, J. (2007). *The GNU C Reference Manual.* Free Software Foundation.

Tucker, A., & Noonan, R. (2003). *Lenguajes de programación : principios y paradigmas.* Madrid, España: McGraw-Hill Interamericana.

VILLALOBOS SALCEDO, J. A., & CASALLAS GUTIERREZ, R. (2006). *Fundamentos De Programación: Aprendizaje Activo Basado En Casos.* México: Pearson Education.

Weiss, M. A., Marroquín, O., Segura, C., & Verdejo, J. (2000). *Estructura de datos en Java : compatible con Java 2.* (P. Educación, Ed.) Madrid, España.