



UNIREMINGTON
CORPORACIÓN UNIVERSITARIA REMINGTON
RES. 2661 MEN JUNIO 21 DE 1996



FONDO EDITORIAL
REMINGTON

LÓGICA DE PROGRAMACIÓN CON PSEINT

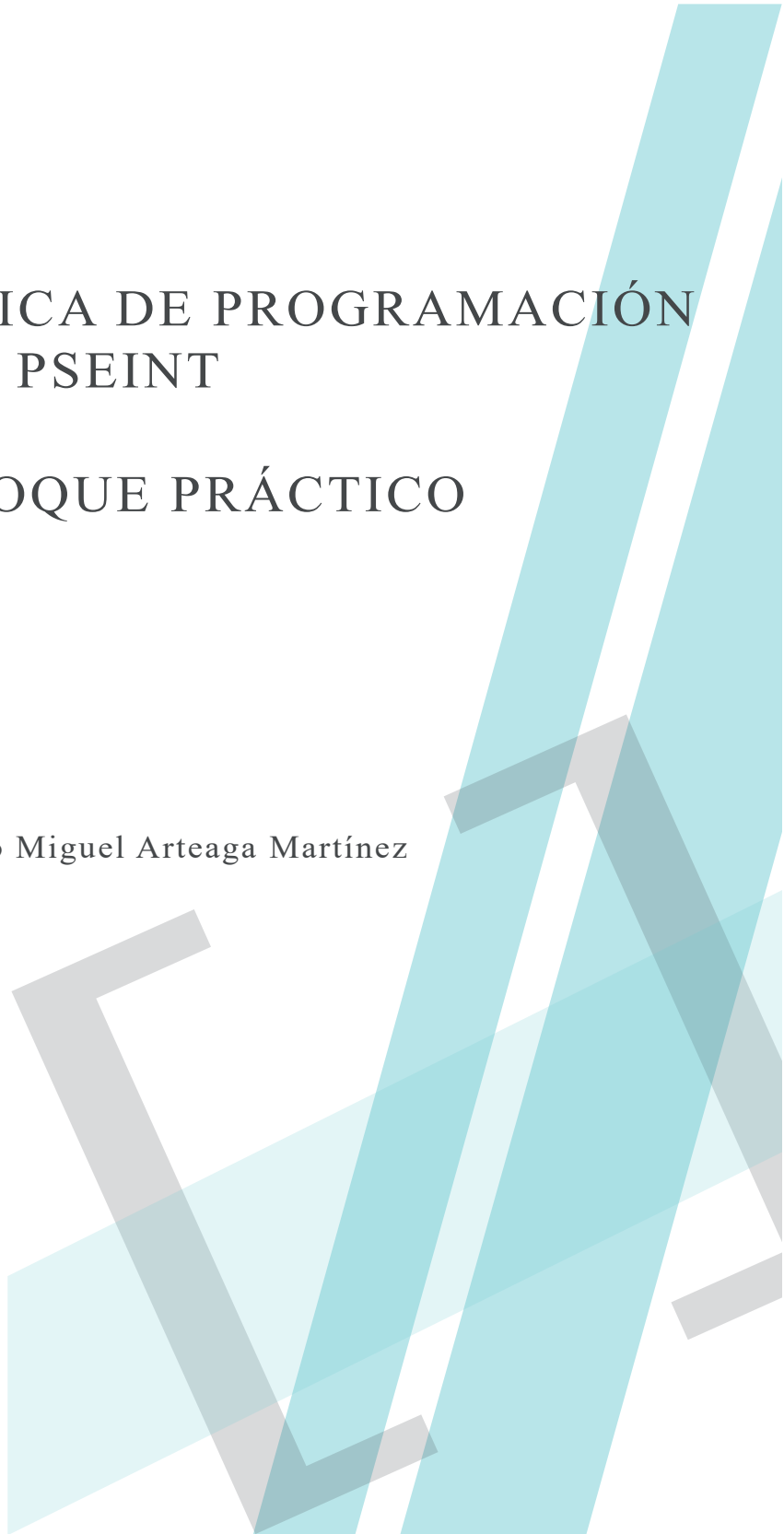
Enfoque práctico

Máximo Miguel Arteaga Martínez

LÓGICA DE PROGRAMACIÓN CON PSEINT

ENFOQUE PRÁCTICO

Máximo Miguel Arteaga Martínez



Lógica de programación con PSeInt. Enfoque práctico
© Corporación Universitaria Remington

Primera edición, diciembre 2023
ISBN 978-628-95852-4-7 (PDF - Internet)
<https://doi.org/10.22209/9786289585247>

Máximo Miguel Arteaga Martínez. Autor
Adriana Patricia Bustamante Fernández. Directora editorial
Viviana Díaz. Coordinadora de procesos editoriales
Alfonso Tobón Botero. Diseñador editorial
Jorge David Garcés Gómez. Diagramador, en LaTeX
Juan David Villa Rodríguez. Corrector de textos

Dirección de Bibliotecas y Fondo Editorial
Calle 51 nro. 51-27, Edificio Uniremington
PBX: (57) 604 322 10 00 / Ext.: 3505
Línea gratuita nacional: 018000 410 203
fondo.editorial@uniremington.edu.co
Medellín, Colombia

Nota legal Las opiniones expresadas en el presente texto no representan la posición oficial o institucional de la Corporación Universitaria Remington. Las citas realizadas y la originalidad de la obra son responsabilidad del autor; en consecuencia, la Corporación Universitaria Remington no será responsable ante terceros por el contenido técnico o ideológico expresado en el texto, ni asume responsabilidad alguna por las infracciones a las normas de propiedad intelectual.

Esta publicación se distribuye bajo una Licencia Creative Commons
«Reconocimiento-No Comercial-Compartir igual».



Arteaga Martínez, Máximo Miguel

Lógica de programación con Pseint. Enfoque práctico / Máximo Miguel Arteaga Martínez. - 1a. ed. -
Corporación Universitaria Remington, 2023
Número de páginas 209 p. ; tamaño 19.5 × 27.5 cm

ISBN: 978-628-95852-4-7

1. Programación lógica. 2. Algoritmos (Computadores). 3 programación modular.
4. programación (computadores electrónicos). I. Arteaga Martínez, Máximo Miguel. II. Tít.

CDD: 005.131 / A786

Dedicatoria

Para María Alejandra, mi hija, motor que me impulsa a seguir creciendo cada día.

Para Lidis Luz, mi esposa, que me ha apoyado incondicionalmente en el logro de
mis objetivos personales y profesionales.

Contenido

Introducción	12
1 CONCEPTOS BÁSICOS	14
1.1 ¿Qué es un algoritmo?	15
1.2 Análisis del problema	16
1.3 Desarrollo del algoritmo en pseudocódigo	17
1.4 Validación o prueba de escritorio	17
1.5 Variable	17
1.6 Constante	19
1.7 Expresiones	19
1.8 Tipos de datos	20
1.9 Operadores aritméticos o matemáticos	20
1.10 Operadores relacionales	21
1.11 Operadores lógicos	21
1.12 Pseudocódigo	21
1.13 Análisis del problema	22
1.14 Desarrollo del algoritmo	23
1.15 Verificación o validación del algoritmo	23
1.16 Instrucciones de declaración o definición de variables, entrada, salida, asignación y comentarios	24
1.17 Descargar, instalar y configurar PSeInt	26
1.18 Ejemplos resueltos	31
1.19 Ejercicios propuestos	57
2 CONDICIONALES Y CICLOS	58
2.1 Condicionales	59
2.2 Ejemplos resueltos	62
2.3 Ejercicios propuestos	80
2.4 Ciclos o bucles	81
2.5 Contadores	84
2.6 Acumuladores	84

2.7	Ejemplos resueltos	84
2.8	Ejercicios propuestos	107
3	PROGRAMACIÓN MODULAR	109
3.1	Funciones	111
3.2	Procedimientos	112
3.3	Parámetros	112
3.4	Variables globales y locales	113
3.5	Ejemplos resueltos	113
3.6	Ejercicios propuestos	145
4	ARREGLOS	146
4.1	Vector	147
4.2	Insertar un elemento en un vector	160
4.3	Eliminar un elemento de un vector	168
4.4	Métodos de ordenamiento	175
4.5	Métodos de búsqueda	188
4.6	Matriz	194
4.7	Ejercicios propuestos	204
	Referencias	206

Tablas

1.1	Identificación de variables Ejercicio1C1	32
1.2	Identificación de variables Ejercicio2C1	43
1.3	Identificación de variables Ejercicio3C1	46
1.4	Identificación de variables Ejercicio4C1	48
1.5	Identificación de variables Ejercicio5C1	51
1.6	Identificación de variables Ejercicio6C1	53
1.7	Identificación de variables Ejercicio7C1	55
2.1	Identificación de variables Ejercicio1C2	62
2.2	Identificación de variables Ejercicio2C2	63
2.3	Identificación de variables Ejercicio3C2	65
2.4	Identificación de variables Ejercicio4C2	68
2.5	Identificación de variables Ejercicio5C2	70
2.6	Identificación de variables Ejercicio6C2	72
2.7	Identificación de variables Ejercicio7C2	74
2.8	Identificación de variables Ejercicio8C2	77
2.9	Identificación de variables Ejercicio9C2	84
2.10	Identificación de variables Ejercicio10C2	87
2.11	Identificación de variables Ejercicio11C2	88
2.12	Identificación de variables Ejercicio12C2	91
2.13	Identificación de variables Ejercicio13C2	97
2.14	Identificación de variables Ejercicio14C2	101
2.15	Identificación de variables Ejercicio15C2	103
3.1	Identificación de variables Ejercicio1C3	113
3.2	Identificación de variables Ejercicio4C3	126
3.3	Identificación de variables Ejercicio5C3	131
3.4	Identificación de variables Ejercicio6C3	139
4.1	Identificación de variables Ejercicio3C4	152

Figuras

1.1	Opciones de sistemas operativos para instalar PSeInt	26
1.2	Descargar el instalador de pseudocódigo PSeInt	26
1.3	Asistente de instalación	27
1.4	Acuerdo de licencia de PSeInt	27
1.5	Carpeta o directorio de instalación	28
1.6	Completando el proceso de instalación de PSeInt	28
1.7	Entorno de desarrollo PSeInt	29
1.8	Configuración del entorno de desarrollo PSeInt	29
1.9	Opciones de configuración de PSeInt	30
1.10	Opciones del lenguaje	30
1.11	Algoritmo transcrito en el interpretador de pseudocódigo PSeInt	38
1.12	Ejecución Ejercicio1C1	38
1.13	Ejecución 2 Ejercicio1C1	40
1.14	Inicio ejecución paso a paso	40
1.15	Visualización mensaje Ingrese el valor 1	40
1.16	Entrada del dato val1	41
1.17	Visualización mensaje Ingrese el valor 2:	41
1.18	Entrada del dato val2	41
1.19	Visualización mensaje Ingrese el valor 3:	42
1.20	Entrada del dato val3	42
1.21	Visualización datos de salida parciales	42
1.22	Fin ejecución paso a paso	43
1.23	Ejecución Ejercicio2C1	45
1.24	Ejecución Ejercicio3C1	48
1.25	Ejecución Ejercicio4C1	51
1.26	Ejecución Ejercicio5C1	53
1.27	Ejecución Ejercicio6C1	55
1.28	Ejecución Ejercicio7C1	57
2.1	Ejecución 1 Ejercicio1C2	63
2.2	Ejecución 2 Ejercicio1C2	63

2.3	Ejecución 1 Ejercicio2C2	65
2.4	Ejecución 2 Ejercicio2C2	65
2.5	Ejecución 1 Ejercicio3C2	67
2.6	Ejecución 2 Ejercicio3C2	67
2.7	Ejecución 3 Ejercicio3C2	67
2.8	Ejecución 1 Ejercicio4C2	70
2.9	Ejecución 2 Ejercicio4C2	70
2.10	Ejecución 3 Ejercicio4C2	70
2.11	Ejecución 1 Ejercicio5C2	72
2.12	Ejecución 2 Ejercicio5C2	72
2.13	Ejecución 1 Ejercicio6C2	73
2.14	Ejecución 2 Ejercicio6C2	74
2.15	Ejecución 1 Ejercicio7C2	76
2.16	Ejecución 2 Ejercicio7C2	76
2.17	Ejecución 3 Ejercicio7C2	76
2.18	Ejecución Ejercicio9_1C2 ciclo PARA	85
2.19	Ejecución Ejercicio9_2C2 ciclo REPETIR	86
2.20	Ejecución Ejercicio9_3C2 ciclo MIENTRAS	87
2.21	Ejecución Ejercicio10C2	88
2.22	Ejecución Ejercicio11C2	90
3.1	Inicio ejecución del Ejercicio1C3	120
3.2	Validación en tiempo de ejecución del valor 1 Ejercicio1C3 para que no sea menor que 1	121
3.3	Validación en tiempo de ejecución del valor 1 Ejercicio1C3	121
3.4	Validación en tiempo de ejecución del valor 1 Ejercicio1C3 para que no sea mayor que 100	121
3.5	Validación en tiempo de ejecución del valor 1 Ejercicio1C3 ingreso de los valores 1 y 2 en el rango 1-100	122
3.6	Ejecución del Ejercicio2C3	124
3.7	Ejecución del Ejercicio3C3	125
3.8	Ingresar valor 1 y 2 Ejercicio4C3	130
3.9	Seleccionar la operación por realizar Ejercicio4C3	130
3.10	Resultado operación seleccionada Ejercicio4C3	130
4.1	Representación gráfica de un vector	147
4.2	Asignación de un nombre a un vector	148
4.3	Representación gráfica de posiciones de los elementos en un vector . .	148

4.4	Representación gráfica de asignación de valores a un vector	148
4.5	Ejecución Ejercicio1C4	150
4.6	Ejecución Ejercicio2C4	152
4.7	Inicio ejecución Ejercicio3C4	158
4.8	Ingreso de los datos del estudiante 1	158
4.9	Ingreso de los datos del estudiante 2	158
4.10	Ingreso de los datos del estudiante 3	159
4.11	Ingreso de los datos del estudiante 4	159
4.12	Ingreso de los datos del estudiante 5	159
4.13	Datos de salida Ejercicio3C4	160
4.14	Representación gráfica de un vector de 10 elementos lleno	161
4.15	Vector con cinco elementos, pero con capacidad para almacenar diez .	161
4.16	Inserción de un elemento al final	161
4.17	Vector con tres elementos, pero con capacidad para almacenar diez .	161
4.18	Desplazamiento de los elementos de un vector una posición a la derecha	162
4.19	Inserción de un elemento al principio de un vector	162
4.20	Vector con cuatro elementos, pero con capacidad para almacenar diez	162
4.21	Inserción de un elemento en una posición específica del vector	162
4.22	Vector con cinco elementos, pero con capacidad para almacenar diez .	163
4.23	Desplazar los elementos de un vector una posición a la derecha a partir de una posición específica	163
4.24	Insertar un elemento en una posición específica del vector	163
4.25	Vector con un elemento, pero con capacidad para diez	169
4.26	Eliminación de un elemento al final del vector con un solo elemento .	169
4.27	Vector con seis elementos, pero con capacidad para diez	169
4.28	Eliminación del último elemento de un vector con dos o más elementos	169
4.29	Vector con seis elementos, pero con capacidad para diez	170
4.30	Eliminación del primer elemento de un vector con dos o más elementos	170
4.31	Vector con cinco elementos, pero con capacidad para diez	170
4.32	Eliminación del último elemento de un vector	170
4.33	Proceso de ordenación de un vector de cinco elementos por burbuja .	176
4.34	Proceso de ordenación de un vector de cinco elementos por burbuja mejorado	178
4.35	Proceso de ordenación de un vector de cinco elementos por selección	179
4.36	Proceso de ordenación de un vector de cinco elementos por inserción .	181
4.37	Proceso de ordenación de un vector de cinco elementos por Shell . . .	182
4.38	Ejecución factorial iterativo	185

4.39	Ejecución factorial recursivo	185
4.40	Proceso de ordenación de un vector de cinco elementos por Quicksort	186
4.41	Vector de 11 elementos lleno y ordenado	190
4.42	Elemento central del vector	190
4.43	Elemento central subvector izquierdo	191
4.44	Elemento central del subvector anterior	191
4.45	Vector de 11 elementos ordenado	191
4.46	Elemento central del vector	191
4.47	Elemento central subvector derecho	191
4.48	Elemento central del subvector a la izquierda	192
4.49	Representación gráfica de una matriz	194
4.50	Representación gráfica de la asignación de un nombre a la matriz . .	194
4.51	Representación gráfica de las filas y columnas de una matriz	194
4.52	Representación gráfica de una matriz cuadrada de 4 por 4	195
4.53	Elementos de la diagonal principal de una matriz de 4 por 4	195
4.54	Elementos de la diagonal secundaria de una matriz de 4 por 4	196
4.55	Ejecución Ejercicio16C4	198
4.56	Ejecución Ejercicio17C4	203

Introducción

Quienes inician estudios en el programa de Ingeniería Informática o de Sistemas, Ciencia de la Computación, Ingeniería de Software, programas técnicos y tecnológicos afines, y, en general, cualquier persona que desee incursionar en el mundo del desarrollo de *software* (en este libro se usará el nombre *Ingeniería Informática* para referirnos a cualquiera de estas denominaciones), se enfrentan a un universo totalmente desconocido, como lo es la resolución de problemas a través del computador y, específicamente, a través del desarrollo de algoritmos en pseudocódigo.

Existe una gran dificultad para ellos a la hora de realizar el análisis de un problema determinado que permita proyectar una solución conceptual que lo resuelva. Se sientan frente al computador a digitar líneas de código para tratar de resolverlo utilizando el método de *prueba y error* (una mala práctica muy generalizada): es decir, van experimentando hasta conseguir la solución, lo cual puede desembocar en algunas de las siguientes situaciones:

- La solución dada no es óptima.
- Escriben demasiadas e innecesarias líneas de código.
- Definitivamente no logran resolver el problema, lo que genera angustia y frustración.

Esto conduce a que muchos deserten en los primeros dos o tres semestres de estudio. Entonces, con este libro esperamos que los aprendices desarrollen las habilidades y competencias en el análisis de problemas (entender, comprender y extraer los requerimientos), para que posteriormente puedan desarrollar un algoritmo en pseudocódigo que dé solución óptima al problema en estudio, además de crear una base sólida para la solución de problemas a fin de que avancen sin dificultad en el conocimiento de uno o más lenguajes de programación de alto nivel: Java, Python, PHP, C#, .Net, C++, entre otros que les sirvan para desarrollar aplicaciones web o móviles de calidad según la metodología de desarrollo de *software* y el modelo de calidad de *software* que seleccionen.

El objetivo de esta obra es proporcionarles a los lectores, sean estudiantes o no de Ingeniería Informática, las herramientas necesarias para aprender la lógica de programación. Los lectores aprenderán a realizar un análisis detallado de un problema identificando las variables necesarias y especificando su tipo de datos (entero, real, cadena, carácter o lógico), junto con una descripción clara y concisa. Este proceso garantiza que cualquier persona comprenda el significado de cada una de las variables identificadas. Posteriormente, deben relacionar los datos de entrada, los procesos (cálculos matemáticos necesarios para resolver el problema) y los datos de salida (resultados que el algoritmo debe proporcionar como solución al problema). Una vez completado este proceso, deben escribir el algoritmo en pseudocódigo, el cual puede ser ejecutado y validado para verificar que cumple con los requerimientos del problema y que produce los resultados esperados. Finalmente, viene una prueba de escritorio para garantizar la solidez del algoritmo.

Está dividido en cuatro capítulos. El capítulo I desarrolla los conceptos teóricos y la terminología propia del desarrollo de *software*, así como la fundamentación conceptual que les permita comprender mejor las temáticas subsiguientes; incluye un acercamiento al desarrollo de un algoritmo en pseudocódigo con estructuras secuenciales utilizando instrucciones para definir o declarar variables, instrucciones de entrada, salida y asignación que puede ser codificado y ejecutado en el interpretador de pseudocódigo PSeInt.

El capítulo II desarrolla los condicionales SI y SEGÚN, que permiten la toma de decisiones o bifurcación en la lógica de un algoritmo, y los ciclos o bucles MIENTRAS, REPETIR y PARA, los cuales sirven para repetir uno o más procesos mientras se cumpla una determinada condición. El capítulo III aborda el concepto de *programación modular*, la cual divide el problema en pequeños problemas. La solución de estos es realizada con unidades llamadas *funciones* y procedimientos diseñados para ejecutar una tarea específica. Incluye el manejo de parámetros por valor y referencia. Finalmente, el capítulo IV trabaja con los arreglos (vectores y matrices de dos dimensiones), que sirven para manipular de forma simultánea un grupo o conjunto de datos de un mismo tipo.

Así, este libro pretende que todo aquel que desee incursionar en el mundo de la programación alcance las habilidades y competencias en la resolución de problemas mediante su análisis, así como en el desarrollo modular de algoritmos en pseudocódigo con funciones, procedimientos y parámetros que funcionan de igual forma en los lenguajes de alto nivel.

Capítulo 1

CONCEPTOS BÁSICOS



Antes de entrar en detalles respecto de lo que concierne a la resolución de problemas a través de algoritmos en pseudocódigo, es preciso que se conozcan algunos términos claves.

1.1 ¿Qué es un algoritmo?

A diario las personas ejecutan acciones desde que se despiertan por las mañanas hasta que se van a la cama nuevamente por las noches. Las ejecutan en un orden lógico: por ejemplo, no van al lugar de trabajo para luego ducharse, vestirse y desayunar, porque la lógica indica que lo primero que deben hacer es ducharse, luego vestirse, desayunar y, ahora sí, dirigirse al lugar de trabajo. De acuerdo con Moreno (2012), “un algoritmo es una secuencia finita y ordenada de pasos para realizar una tarea en forma precisa” (p. 21), mientras que para Mancilla Herrera et al. (2015), “es un conjunto finito de reglas bien definidas en su lógica de control que permiten la solución de un problema en una cantidad finita de tiempo” (p. 6).

Según Murillo Escobar (2014), “un algoritmo es un conjunto de instrucciones combinadas de forma adecuada para resolver un determinado problema en una cantidad finita de tiempo” (párr. 11); en tanto para Juganaru Mathieu (2015), “un algoritmo constituye una lista bien definida, ordenada y finita de operaciones, que permite encontrar la solución a un problema determinado” (p. 2). Así que se puede decir, entonces, que *un algoritmo* es una secuencia de pasos lógicamente ordenados y finitos que dan solución a un problema determinado. Se componen de tres partes:

Las entradas. Toda información que se debe ingresar al algoritmo para que pueda realizar los procesos.

Los procesos. Operaciones o cálculos matemáticos que se deben desarrollar para darle solución al problema planteado.

Las salidas. Los resultados de los diferentes procesos llevados a cabo por el algoritmo.

La solución puede ser de dos tipos, dependiendo del problema:

Una solución que implique la descripción a través de frases y palabras; en este caso, se trata de un *algoritmo cualitativo*. Un ejemplo típico es el de una persona que va en una bicicleta de su casa al parque y en el transcurso del viaje se pincha. ¿Qué debe hacer?

Una posible solución, y decimos posible porque puede haber otras:

1. Se estaciona en la orilla de la vía.
2. Revisa la llanta pinchada para verificar si un objeto incrustado causó la pinchada, y luego retirarlo para que no le cause más daño al neumático.
3. Se dirige a una llantería.
4. Ya en la llantería, el encargado desmonta la llanta, retira el neumático, examina y determina el lugar del orificio. Luego, procede a marcarlo.
5. Prepara y coloca el parche.
6. Hace la prueba para verificar que el parche haya quedado correctamente adherido y no exista escape de aire.
7. Coloca el neumático en la llanta, la infla y procede a montarla nuevamente en la bicicleta.
8. Cancela el servicio.
9. Finalmente, la persona sigue su ruta hasta llegar al parque.

Como se puede apreciar, se realizó una descripción de una serie de pasos lógicos, ordenados y finitos, que le dan solución al problema planteado (algoritmo) a través de frases y palabras.

La segunda alternativa se refiere al uso de procesos (cálculos o fórmulas matemáticas). En este caso, se conocen como ***algoritmos cuantitativos***. No ofrecemos aquí un ejemplo porque es preciso conocer otros conceptos y términos antes de entrar en la resolución de problemas de este tipo (este libro se centra en la resolución de problemas cuantitativos).

La solución de un problema mediante la lógica de programación implica tres pasos fundamentales, que se deben seguir en estricto orden:

1. Análisis del problema.
2. Desarrollo del algoritmo en pseudocódigo.
3. Validación o prueba de escritorio del algoritmo.

1.2 Análisis del problema

Implica entender y comprender al 100% el problema que se pretende resolver. Esta etapa es fundamental: de ella depende el éxito en la siguiente fase.

1.3 Desarrollo del algoritmo en pseudocódigo

Codificar o transcribir en sentencias o instrucciones el análisis que se le ha hecho al problema planteado. Un algoritmo, como cualquier lenguaje de programación, contiene sentencias o instrucciones de entradas, salidas, asignación, condicionales y ciclos, así como variables, constantes, expresiones, contadores, acumuladores, operadores matemáticos, lógicos y relacionales. Cada uno de estos, y otros conceptos, será desarrollado a medida que avance la lectura y estudio del libro.

1.4 Validación o prueba de escritorio

Consiste en realizar un seguimiento paso a paso (sentencia por sentencia) de un algoritmo para determinar si los resultados arrojados son los esperados. De no ser así, se procede a aplicar los respectivos ajustes. La prueba de escritorio se repite hasta conseguir el resultado esperado. Así que esta ayuda a determinar si el algoritmo está bueno o no. Más adelante, cuando estemos desarrollando algoritmos, veremos cómo aplicar una prueba de escritorio.

1.5 Variable

Es un ente que identifica a una característica o propiedad específica de un determinado elemento dentro de un problema. Su valor puede variar en el transcurso de la ejecución del algoritmo; por ejemplo...

Para identificar un estudiante del curso Lógica de Programación, se tiene:

Tipo de documento de identificación del estudiante, número de identificación, nombres y apellidos, y un código o ID asignado por la institución de educación; por lo tanto, se tienen las siguientes variables:

Tipo de identificación, es decir, cédula de ciudadanía, tarjeta de identidad, cédula de extranjería o pasaporte. Para referirse al tipo de identificación de un estudiante, se puede usar una de las siguientes variables:

tipo_Identificacion

tipoIdentificacion

tipoDocumento

clase_Documento

Cualquiera de las anteriores se refiere al tipo de identificación de un estudiante.

Usualmente se utilizan nombres abreviados y sin tilde, así:

tipo.Ident
tipoIdent
tipo.Doc
tipoDoc
cla.Doc
claDoc

La siguiente variable es el número de identificación, que se puede designar por:

nro.Ident
nroIdent
nro.Doc
nroDoc

La siguiente variable es el nombre o nombres del estudiante, y se puede definir como:

nomb.Est
nombEst

Continúa la variable para los apellidos del estudiante, que se define como:

ape.Est
apeEst

Finalmente, el código o ID asignado por la institución, y que es único para cada estudiante:

cod.Est
codEst
id.Est
idEst

Como regla general, para definir variables se usará la notación Lower Camel Case, que consiste en escribir toda la palabra inicial en minúsculas y las siguientes palabras con la primera letra en mayúscula:

Tipo de documento	tipDoc
Nombres del estudiante	nomEst
Apellidos del estudiante	apeEst
Código del estudiante	codEst
Sueldo básico	sueBas
Sueldo neto	sueNet

1.6 Constante

Es un ente que identifica a una característica específica de un determinado elemento en un problema. Su valor nunca cambia en el transcurso de la ejecución del algoritmo, es decir, permanece fijo; por ejemplo, el valor constante PI, que se designa como:

$Pi = 3.141593$, cuya asignación algorítmica se realiza como:

$Pi \leftarrow 3.141593$

Otro ejemplo es el valor de la hora cátedra que pagan a un profesor en una institución según el grado educativo: profesional 15.000 pesos, especialista 25.000, magíster 50.000 y doctor 100.000. En este caso se identifican cuatro constantes, a saber:

$valHorPro \leftarrow 15000$

$valHorEsp \leftarrow 25000$

$valHorMag \leftarrow 50000$

$valHorDoc \leftarrow 100000$

Una variable o constante puede tener valores numéricos dentro de su nombre:

iva10

iva16

iva5

Indica IVA del 10 %, IVA del 16 % e IVA del 5 %, respectivamente.

Otro ejemplo puede ser:

Por10

Por20

Por30

Por40

Porcentaje del 10, del 20, del 30 y del 40, respectivamente.

No deben contener caracteres especiales (como `¡` `”` `#` `$` `%` `&` `¿` `?`) porque generan errores y confusiones en el lector.

1.7 Expresiones

Son una combinación entre variables, constantes, operadores y paréntesis; por ejemplo: $a + (b * 5)/c$

1.8 Tipos de datos

En términos generales, en el mundo del desarrollo de *software* se trabaja con un conjunto de datos llamados *primitivos*, los cuales sirven para indicar el tipo de información que puede almacenar una variable o constante, y las operaciones que se pueden llevar a cabo con estas; específicamente, en la lógica de programación están los siguientes tipos de datos:

Enteros

Son todos aquellos valores que no tienen punto decimal o flotante, pueden ser positivos o negativos y se incluye el cero: 10, 120, 1200, etc.

Reales

Son todos aquellos valores que tienen punto decimal o flotante, pueden ser positivos o negativos y se incluye el cero: 4.5, 3.1416, 0.83, etc.

Cadenas

También conocidas como *alfanuméricos*, están conformadas por letras, o por una combinación entre letras, números, signos y símbolos, y van entre comillas dobles; por ejemplo: “Calle 107 # 125-48A”, “Hola mundo”.

Caracteres

Están conformados por un solo carácter, van entre comillas dobles, y pueden ser letras, números, signos o símbolos; por ejemplo: “a”, “x”, “3”, “#”.

Lógicos

Constituyen un tipo de dato especial que solo puede almacenar uno de dos valores: falso o verdadero.

1.9 Operadores aritméticos o matemáticos

Permiten hacer operaciones o cálculos matemáticos con los valores de las variables o constantes.

+	Suma
-	Resta
*	Multiplicación
/	División
\wedge	Potencia
% o Mod	Resto o residuo de la división entre enteros

1.10 Operadores relacionales

Se usan para establecer una relación entre dos o más variables de un mismo tipo.

>	Mayor que
<	Menor que
\geq	Mayor igual
\leq	Menor igual
\neq	Diferente
=	Igual

1.11 Operadores lógicos

Se usan para establecer una relación entre dos o más variables de tipo lógico.

& o Y	Conjunción
o O	Disyunción
\sim o No	Negación

1.12 Pseudocódigo

Oviedo Regino (2012) define el *pseudocódigo* como “la representación de los pasos del algoritmo a través de palabras, utilizando una nomenclatura estandarizada para denotar el significado de cada paso” (p. 69). Por su parte, Robledano (2019) lo define como “una forma de expresar los distintos pasos que va a realizar un programa, de la forma más parecida a un lenguaje de programación” (párr. 2); en tanto GameDev-Traum (2023) indica que “es una herramienta que permite representar instrucciones de código, pero sin usar un lenguaje de programación formal, es un punto intermedio entre el lenguaje coloquial y el lenguaje de programación” (párr. 1).

De acuerdo con las anteriores definiciones, se puede decir que el pseudocódigo permite escribir algoritmos en un lenguaje natural en español utilizando las mismas

estructuras que implementan los lenguajes de programación de alto nivel.

Para escribir algoritmos en pseudocódigo, se debe hacer el análisis del problema previamente a fin de posteriormente desarrollar el algoritmo en el interpretador de pseudocódigo PSeInt.

1.13 Análisis del problema

Es el primer paso en la solución de un problema mediante el desarrollo de un algoritmo, y exige reconocer detalladamente aquel problema que se quiere solucionar identificando todos los requerimientos a fin de darle una solución óptima. Consiste en:

Identificar las variables y constantes que intervienen en el problema

Se debe indicar por cada variable el tipo de dato (entero, real, cadena, carácter o lógico), el nombre de la variable y una descripción detallada de cada una:

Tipo	Variable	Descripción
Cadena	nomEst	Nombre del estudiante
Real	notFin	Nota final o definitiva

Datos de entrada

Información que se debe ingresar al algoritmo para que realice los procesos y arroje los resultados (datos de salida). Es necesario relacionar las variables separadas por coma.

Procesos

Son las operaciones o cálculos matemáticos requeridos para darle solución al problema planteado. Se dividen en:

Procesos parciales. Son las operaciones o cálculos matemáticos que se efectúan por cada carga o ingreso de datos (datos de entrada) hecho.

Procesos totales. Son las operaciones o cálculos matemáticos que se efectúan luego de finalizar la carga o ingreso de datos (datos de entrada). Se efectúan una sola vez.

Datos de salida

Son los resultados que arroja el algoritmo, provenientes de los diferentes procesos. Se relacionan las variables separadas por coma. Se dividen en:

Salidas parciales. Son los datos que se deben mostrar por cada carga o ingreso de datos (datos de entrada) que se le haga al algoritmo.

Salidas totales. Son los datos que se deben mostrar luego de finalizar la carga o ingreso de datos (datos de entrada). Se muestran una sola vez.

1.14 Desarrollo del algoritmo

Como se indicó anteriormente, un *algoritmo* es una secuencia de pasos lógicamente ordenados y finitos que dan solución a un problema determinado. Para su desarrollo se llevará a cabo en pseudocódigo, con la herramienta PSeInt, un *software* libre y gratuito que se distribuye bajo licencia GPL (General Public License).

De acuerdo con su autor, PSeInt es...

... Una herramienta para asistir a un estudiante en sus primeros pasos en programación, mediante un simple e intuitivo pseudolenguaje en español que permite centrar su atención en los conceptos fundamentales de la algoritmia computacional, minimizando las dificultades propias de un lenguaje real y proporcionando un entorno de trabajo con numerosas ayudas y recursos didácticos (Novara, 2003a, párr. 1).

Se puede descargar PSeInt desde <https://PSeInt.sourceforge.net/> para Windows, Linux y Mac. Más adelante explicaremos su proceso de instalación y configuración.

El algoritmo será desarrollado teniendo en cuenta el análisis previo del problema, en el cual se identifican las variables y constantes, se determinan los datos de entrada, los procesos parciales y totales, así como los datos de salida parciales y totales. Seguidamente, se debe realizar la verificación del algoritmo, conocida también como *prueba de escritorio*.

1.15 Verificación o validación del algoritmo

Consiste en hacerle un seguimiento manual, paso a paso (línea por línea), al código del algoritmo simulando datos de entrada, con el fin de comprobar que los procesos arrojan los resultados esperados. De no ser así, será necesario revisar el análisis del problema y el código del algoritmo para aplicar las respectivas correcciones y repetir la prueba de escritorio hasta obtener los datos de salida esperados o correctos. Por ello es importante que a cada algoritmo se le haga esta prueba.

Una vez terminado el análisis del problema, desarrollado el algoritmo y validado

mediante la prueba de escritorio, se procede a la fase de codificación en un lenguaje de programación de alto nivel, como Java, Python, C#, PHP, .Net, C, C++, etc.

En términos generales, un algoritmo tiene la siguiente estructura:

Algoritmo NombreDelAlgoritmo

```
Acción 1
Acción 2
Acción 3
.
.
.
Acción N
```

FinAlgoritmo

Como regla general, para nombrar un algoritmo se usará la notación Upper Camel Case, en la cual la letra inicial de cada palabra comienza con mayúscula y el resto de la palabra queda en minúscula: NombreDelAlgoritmo, NominaEmpleados, etc.

Inicialmente se desarrollarán algoritmos sencillos en pseudocódigo, con declaración o definición de las variables, entrada de datos, salida de datos, asignación y comentarios.

1.16 Instrucciones de declaración o definición de variables, entrada, salida, asignación y comentarios

Definir

Se utiliza para declarar o definir una o más variables separadas por coma; sintaxis:

```
Definir variable1, variable2, variable3, ..., variableN Como [Entera | Real |
Cadena | Lógica]
```

Por ejemplo:

```
Definir sueBas, sueNet Como Real
Definir nomEst, apeEst Como Cadena
Definir valor1, valor2 Como Entero
Definir sexo Como Caracter
Definir sw Como Logico
```


Leer

Se utiliza para ingresar los datos de entrada en un algoritmo; sintaxis:

```
Leer <variable>
```

Por ejemplo:

```
Leer nomEst
```

```
Leer proNot
```

La primera instrucción captura o lee el nombre de un estudiante; la segunda el promedio de notas.

Escribir

Se utiliza para visualizar información: puede ser un mensaje entre comillas dobles, el contenido de una variable o ambos; sintaxis:

Visualizar un mensaje

```
Escribir "Mensaje a visualizar"
```

Visualizar el contenido o valor de una variable

```
Escribir Variable
```

Visualizar un mensaje y el contenido de una variable

```
Escribir "Mensaje a visualizar", Variable
```

Ejemplos:

```
Escribir "Hola mundo"
```

```
Escribir proNot
```

```
Escribir "Promedio de nota: ", proNot
```

Asignación

Es la acción de darle valor a una variable o constante. Se realiza con el símbolo de flecha izquierda; por ejemplo, *asignar el valor 5 a la nota 1*.

```
nota1 ← 5
```

También se utiliza para asignarle a una variable el resultado de una operación matemática; por ejemplo, *asignar a la variable suma el resultado del valor 1 más el valor 2*:

```
suma ← valor1 + valor2
```

Comentarios

Un comentario es un mensaje que sirve para documentar el código, y se realiza anteponiendo `//` y el comentario deseado. No es tenido en cuenta al momento de ejecutarse el algoritmo:

```
//Definición o declaración de variables
//Datos de entrada
//Procesos parciales
//Salidas parciales
```

1.17 Descargar, instalar y configurar PSeInt

PSeInt se puede descargar en la siguiente URL: <https://PSeInt.sourceforge.net/>. Se debe seleccionar el sistema operativo sobre el cual se va a instalar (Windows, Linux o Mac). En este caso, clic sobre Windows (**figura 1.1**).

Figura 1.1: Opciones de sistemas operativos para instalar PSeInt



Nota: Sistemas operativos en los cuales se puede instalar PSeInt.

Se selecciona la opción “Descargar instalador para Microsoft Windows” (**figura 1.2**).

Figura 1.2: Descargar el instalador de pseudocódigo PSeInt



Nota: Descarga del instalador de pseudocódigo PSeInt para Windows.

Con la acción anterior se descarga el archivo ejecutable a la carpeta que se tenga por defecto (generalmente, “Descargas”). Para su instalación se hace doble clic sobre el archivo descargado y se responde “Sí” a la pregunta “¿Quieres permitir que esta

aplicación realice cambios?”. Se visualiza la siguiente ventana y se hace clic en el botón “Siguiete” (figura 1.3).

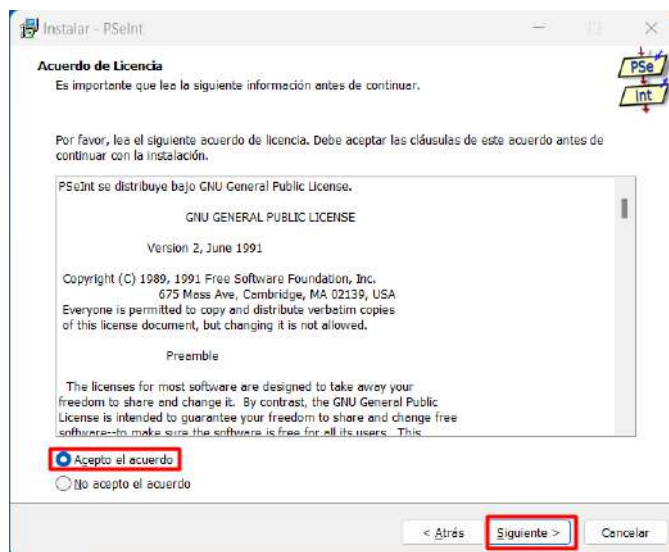
Figura 1.3: Asistente de instalación



Nota: Inicio del proceso de instalación de PSeInt en Windows.

En la siguiente ventana se acepta el acuerdo de licencia y se hace clic en el botón “Siguiete” (figura 1.4).

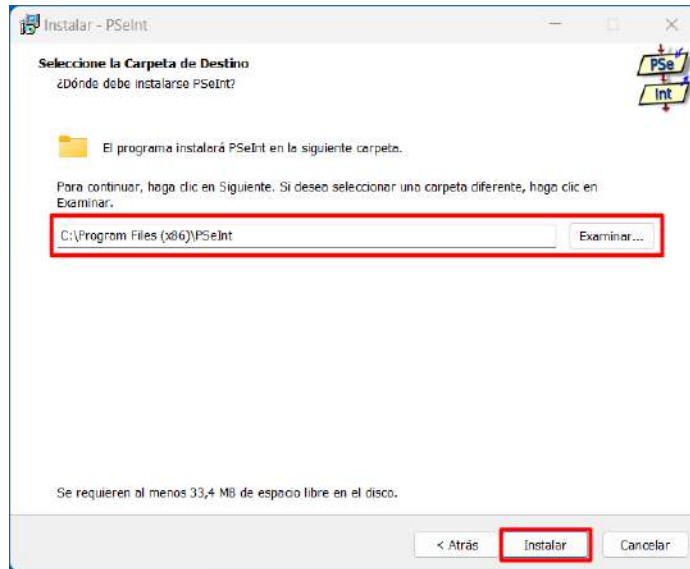
Figura 1.4: Acuerdo de licencia de PSeInt



Nota: Se debe aceptar el acuerdo de licencia para continuar con el proceso de instalación de PSeInt.

En la siguiente ventana se selecciona la carpeta de destino, en la cual se va a instalar PSeInt (se sugiere dejarla por defecto), y se hace clic en el botón “Instalar” (figura 1.5).

Figura 1.5: Carpeta o directorio de instalación



Nota: Seleccionar la carpeta en la que se instalará PSeInt (se sugiere dejarla por defecto).

Al finalizar la instalación se visualiza la siguiente ventana (figura 1.6).

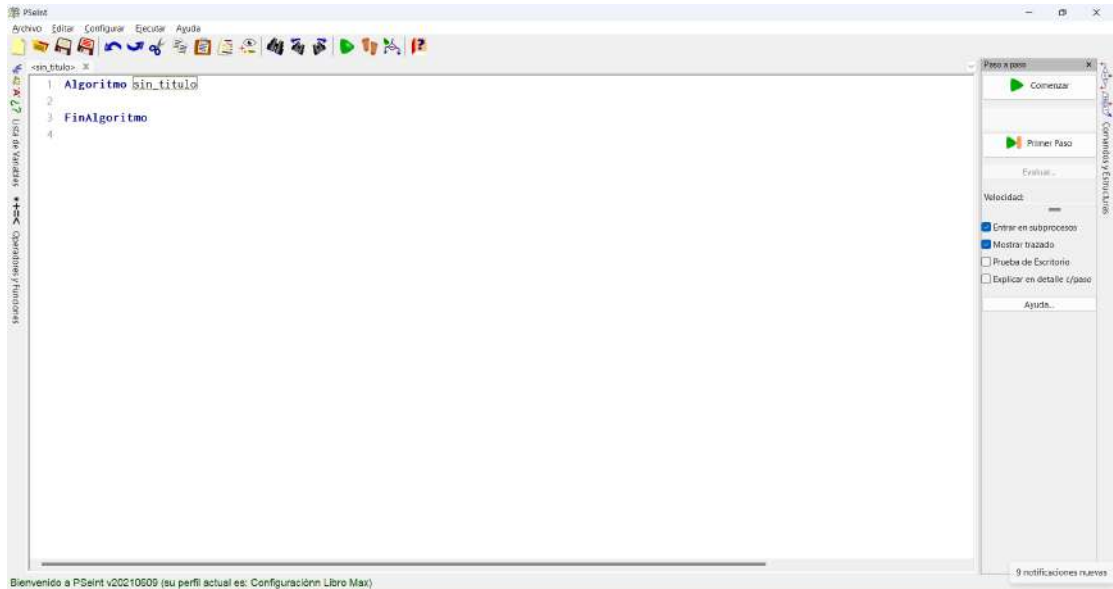
Figura 1.6: Completando el proceso de instalación de PSeInt



Nota: Finalizando proceso de instalación de PSeInt

Se hace clic en el botón “Siguiente” y se carga el editor de PSeInt, en el cual se va a digitar el algoritmo (figura 1.7).

Figura 1.7: Entorno de desarrollo PSeInt

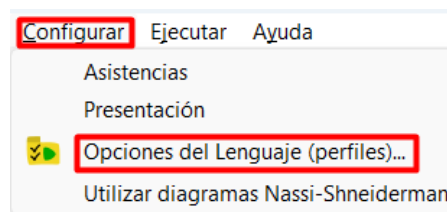


Nota: Entorno de desarrollo PSeInt.

Al pasar el puntero del ratón por cada ícono se visualiza la acción que cada uno realiza. En la parte superior está el “Menú principal” con las opciones “Archivo”, “Editar”, “Configurar”, “Ejecutar” y “Ayuda”.

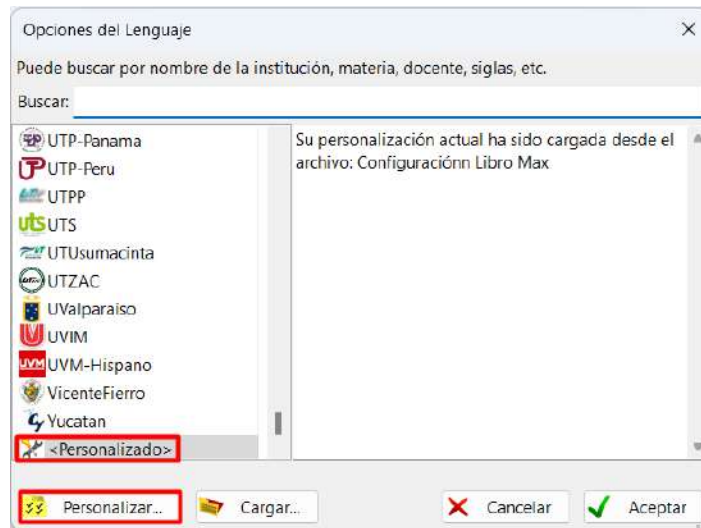
El siguiente paso es configurar el entorno. Para ello se hace clic en la opción “Configurar” / “Opciones del lenguaje (perfiles)” (figura 1.8).

Figura 1.8: Configuración del entorno de desarrollo PSeInt



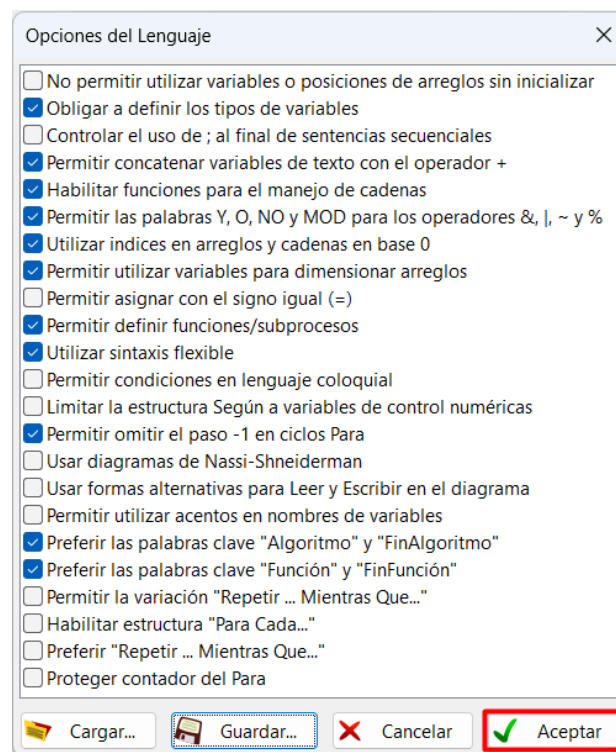
Nota: Configuración del editor de PSeInt.

En la siguiente ventana se selecciona “Personalizado” y se hace clic en el botón “Personalizar” (figura 1.9).

Figura 1.9: Opciones de configuración de PSeInt

Nota: Personalización de la configuración de PSeInt.

Se visualiza la siguiente ventana (figura 1.10).

Figura 1.10: Opciones del lenguaje

Nota: Opciones de configuración de PSeInt.

Se debe verificar para seleccionar únicamente las opciones indicadas en la figura 1.10, y hacer clic en el botón “Guardar”; finalmente, se hace clic en el botón “Aceptar”.

Asimismo, se puede cargar la configuración compartida en el archivo “Configuración PSeInt” haciendo clic en el botón “Cargar”; luego se selecciona el archivo, se hace clic en el botón “Abrir” y, finalmente, clic en el botón “Aceptar”.

Descarga los ejemplos desarrollados en el libro en el siguiente enlace:

<https://doi.org/10.22209/9786289585247>

1.18 Ejemplos resueltos

1. Realizar un algoritmo en pseudocódigo que lea tres números. Calcule e imprima la suma, el producto y el promedio de estos.

Lo primero es el análisis del problema, que, como se indicó, debe identificar las variables que intervienen en el problema, especificando el tipo de dato (entero, real, cadena, carácter o lógico) y una descripción.

Para identificar correctamente las variables, se procede a leer detenidamente el enunciado del problema, las veces que sea necesario hasta comprenderlo en su totalidad.

“Realizar un algoritmo en pseudocódigo que lea tres números”. En este fragmento del enunciado se pide “que lea tres números”, lo cual indica, entonces, que se requieren tres variables, las cuales pueden ser estas: val1, val2 y val3.

Luego continúa el enunciado indicando “. . . Calcule e imprima la suma, el producto y el promedio de estos”. Como se aprecia, se debe efectuar la suma, el producto y el promedio de los tres valores, por lo cual hacen falta tres variables más: suma, producto, promedio.

Ahora, para identificar el tipo de dato de cada una de estas variables, se plantea el siguiente interrogante: ¿qué valores pueden almacenar las variables val1, val2 y val3? Pueden ser datos enteros, como 10, 20, 250, o reales, como 10.25, 20.85, 250.15. En este caso, para que la solución sea la óptima, se recomienda usar un tipo de dato real, ya que no se limita solo a ingresar datos enteros.

Como valor 1, valor 2 y valor 3 son reales, también lo serán la suma, el producto y el promedio, debido a que el resultado de estas operaciones será también un real, por ejemplo:

La suma de $10.25 + 20.85 + 250.15 = 281.25$

El producto de $10.25 * 20.85 * 250.15 = 52562.76875$

El promedio de $281.25/3 = 93.75$

La identificación de las variables se efectuará en la **tabla 1.1**

Tabla 1.1: Identificación de variables Ejercicio1C1

Tipo	Variable	Descripción
Real	val1	Valor 1
Real	val2	Valor 2
Real	val3	Valor 3
Real	suma	Suma de los 3 valores o números
Real	producto	Producto de los 3 valores o números
Real	promedio	Promedio de los 3 valores o números

Una vez identificadas las variables, se deben determinar los datos de entrada necesarios para que el algoritmo realice los procesos de suma, producto y promedio de los tres valores. En este caso, los datos requeridos son los tres valores que se almacenarán en las variables *val1*, *val2* y *val3*, respectivamente. Se relacionan separados por coma.

Datos de entrada: *val1*, *val2*, *val3*

Lo siguiente es determinar los procesos parciales y totales (fórmulas matemáticas). En estos primeros ejercicios no hay procesos totales ni datos de salida totales: son ejercicios sencillos para ir comprendiendo la lógica de la resolución de un problema a través de un algoritmo en pseudocódigo; más adelante, a partir de la unidad siguiente, cuando abordemos los condicionales y los ciclos o bucles, trabajaremos con estos. De momento, indicaremos N/A (no aplica).

Procesos parciales:

suma \leftarrow *val1* + *val2* + *val3*

producto \leftarrow *val1* * *val2* * *val3*

promedio \leftarrow *suma*/3

Procesos totales: N/A

El paso siguiente es indicar los datos de salida parciales y totales, relacionando las variables separadas por coma.

Datos de salida parciales: *suma*, *producto*, *promedio*

Datos de salida totales: N/A

Resumiendo, se tiene entonces:

Identificación de variables

Ver **tabla 1.1**

Datos de entrada: val1, val2, val3

Procesos parciales:

$suma \leftarrow val1 + val2 + val3$

$producto \leftarrow val1 * val2 * val3$

$promedio \leftarrow suma/3$

Procesos totales: N/A

Datos de salida parciales: suma, producto, promedio

Datos de salida totales: N/A

Después, corresponde codificar el algoritmo en el interpretador de pseudocódigo PSeInt. Inicialmente se desarrolla de forma textual, así:

- a. Se declaran o definen las variables.
- b. Se procede a capturar o ingresar los datos de entrada.
- c. Se realizan los procesos parciales.
- d. Se visualizan los datos de salida parciales.

Nótese que, como no hay procesos ni salidas totales, no se tienen en cuenta para esta solución.

Algoritmo Ejercicio1C1

```
//Definir o declarar las variables
Definir val1, val2, val3, suma, producto, promedio Como Real

//Datos de entrada
Escribir "Ingrese el valor 1: "
Leer val1
Escribir "Ingrese el valor 2: "
Leer val2
Escribir "Ingrese el valor 3: "
Leer val3

//Procesos parciales
suma ← val1 + val2 + val3
producto ← val1 * val2 * val3
promedio ← suma / 3

//Datos de salida parciales
Escribir "La suma de los 3 valores es igual a : ", suma
```

```
    Escribir "El producto de los 3 valores es igual a: ", producto
    Escribir "El promedio de los 3 valores es igual a: ",
        Redon(promedio*100) / 100
```

```
FinAlgoritmo
```

Prueba de escritorio

Para realizarla se escriben las variables una debajo de la otra, en el orden en que están dentro del algoritmo; se les van asignando valores según el caso:

```
val1:
val2:
val3:
suma:
producto:
promedio:
```

Se sigue el algoritmo, línea por línea, así:

```
Escribir "Ingrese el valor 1: "
Leer val1
```

En este caso se omite la línea de código Escribir "Ingrese el valor 1: " porque es un mensaje de visualización para orientar al usuario sobre la acción que debe ejecutar: debe ingresar un valor, supongamos el 5; entonces, se le asigna este valor a la variable val1:

```
val1: 5
val2:
val3:
suma:
producto:
promedio:
```

Las siguientes líneas de código son:

```
Escribir "Ingrese el valor 2: "
Leer val2
```

Esto indica que el usuario debe ingresar el segundo valor, por ejemplo, 10; se le asigna a la variable val2:

```
val1: 5
val2: 10
```

val3:
suma:
producto:
promedio:

Las siguientes líneas de código son:

Escribir “Ingrese el valor 3: ”

Leer val3

Esto indica que el usuario debe ingresar el tercer valor; por ejemplo, 2, y se le asigna a la variable val3:

val1: 5
val2: 10
val3: 2
suma:
producto:
promedio:

La siguiente línea de código es:

suma ← val1 + val2 + val3

Significa, entonces, que a la variable suma se le asigna el resultado de sumar los valores $5 + 10 + 2$, que son los de las variables val1, val2 y val3, respectivamente:

val1: 5
val2: 10
val3: 2
suma: 17
producto:
promedio:

La siguiente línea de código es:

producto ← val1 * val2 * val3

Ahora se realiza el producto de $5 * 10 * 2$, y se le asigna a la variable producto.

val1: 5
val2: 10
val3: 2
suma: 17
producto: 100
promedio:

La siguiente línea de código es:

```
promedio ← suma / 3
```

En este caso, como suma es igual a 17 y se ingresaron tres valores, entonces se procede a dividir 17 / 3, y el resultado se le asigna a la variable promedio:

```
val1: 5  
val2: 10  
val3: 2  
suma: 17  
producto: 100  
promedio: 5.6666666667
```

La siguiente línea de código es:

```
Escribir "La suma de los 3 valores es igual a :", suma
```

La acción que ejecuta esta línea de código es visualizar el mensaje "La suma de los 3 valores es igual a :", y el valor de la variable suma; aparece en pantalla lo siguiente:

```
La suma de los 3 valores es igual a : 17
```

Para las siguientes dos líneas de código:

```
Escribir "El producto de los 3 valores es igual a :", producto  
Escribir "El promedio de los 3 valores es igual a :", promedio
```

Se visualiza en pantalla lo siguiente:

```
El producto de los 3 valores es igual a: 100  
El promedio de los 3 valores es igual a: 5. 6666666667
```

Como se aprecia, el promedio tiene 10 cifras decimales. Se puede modificar la última línea de código: Escribir "El promedio de los 3 valores es igual a :", promedio por:

```
Escribir "El promedio de los 3 valores es igual a :", Redon(promedio*100)/100
```

La función "Redon" redondea el entero más cercano de un determinado valor; por ejemplo:

```
Promedio ← 5.6666666667  
Escribir Redon(promedio)
```

Se obtiene como resultado 6, que es el entero más cercano al valor 5.6666666667. Si se desea obtener dos cifras decimales, entonces se multiplica por 100 la variable promedio y el resultado se divide entre 100:

Escribir `Redon(promedio * 100) / 100`, que da como resultado 5.67

Para obtener tres cifras decimales, se multiplica por 1000 y se divide entre 1000:

Escribir `Redon(promedio * 1000) / 1000`, que da como resultado 5.667

Si se desea solo una cifra decimal, se multiplica por 10 y se divide entre 10:

Escribir `Redon(promedio * 10) / 10`, que da como resultado 5.7

En resumen, la prueba de escritorio queda así:

```
val1: 5
val2: 10
val3: 2
suma: 17
producto: 100
promedio: 5.67
```

Las salidas quedan como:

```
5 + 10 + 2 = 17
5 * 10 * 2 = 100
17 / 3 = 5.67
```

Si se realizan estas operaciones a través de una calculadora o una hoja de Excel, el resultado será exactamente el mismo, lo cual indica que el algoritmo es correcto. De no ser así, se debe revisar el análisis del problema para identificar dónde está el error, y seguidamente analizar el algoritmo. Es necesario repetir la prueba de escritorio hasta conseguir los resultados esperados.

Para transcribir el algoritmo que se realizó, se procede, entonces, a abrir PSeInt. Se reemplaza `sin_titulo` por el nombre dado `Ejercicio_1_C1`:

```
Algoritmo Ejercicio1C1
FinAlgoritmo
```

Se transcriben todas las demás líneas de código (**figura 1.11**).

Figura 1.11: Algoritmo transcrito en el interpretador de pseudocódigo PSeInt

```
Algoritmo Ejercicio1C1
//Definir o declarar las variables
Definir val1, val2, val3, suma, producto, promedio Como Real

//Datos de entrada
Escribir "Ingrese el valor 1: "
Leer val1
Escribir "Ingrese el valor 2: "
Leer val2
Escribir "Ingrese el valor 3: "
Leer val3

//Procesos parciales
suma ← val1 + val2 + val3
producto ← val1 * val2 * val3
promedio ← suma / 3

//Datos de salida parciales
Escribir "La suma de los 3 valores es igual a   : ", suma
Escribir "El producto de los 3 valores es igual a: ", producto
Escribir "El promedio de los 3 valores es igual a: ", Redon(promedio*100)/100
FinAlgoritmo
```

Hay varias formas de ejecutar el algoritmo:

Se presiona la tecla F9

Se hace clic en la opción “Ejecutar” / “Ejecutar”

Se hace clic en el ícono ejecutar 

El resultado que se obtiene al ejecutarlo se aprecia en la **figura 1.12**.

Figura 1.12: Ejecución Ejercicio1C1

```
*** Ejecución Iniciada. ***
Ingrese el valor 1:
> 5
Ingrese el valor 2:
> 10
Ingrese el valor3 3:
> 2
La suma de los 3 valores es igual a   : 17
El producto de los 3 valores es igual a: 100
El promedio de los 3 valores es igual a: 5.67
*** Ejecución Finalizada. ***
```

Como se puede apreciar, los valores de los datos de entrada están debajo del mensaje:

Ingrese el valor 1:

> 5

Si se desea que el 5 esté al frente del mensaje, ingrese el valor 1; se modifica la línea de código Escribir "Ingrese el valor 1: " por:

Escribir sin saltar "Ingrese el valor 1: "

Lo mismo se hace con las otras dos líneas de código de los datos de entrada, y queda como:

Escribir Sin Saltar "Ingrese el valor 2: "

Escribir Sin Saltar "Ingrese el valor 3: "

Finalmente, el algoritmo queda:

Algoritmo Ejercicio1C1

```
//Definir o declarar las variables
Definir val1, val2, val3, suma, producto, promedio Como Real

//Datos de entrada
Escribir Sin Saltar "Ingrese el valor 1: "
Leer val1
Escribir Sin Saltar "Ingrese el valor 2: "
Leer val2
Escribir Sin Saltar "Ingrese el valor 3: "
Leer val3

//Procesos parciales
suma ← val1 + val2 + val3
producto ← val1 * val2 * val3
promedio ← suma / 3

//Datos de salida parciales
Escribir "La suma de los 3 valores es igual a : ", suma
Escribir "El producto de los 3 valores es igual a: ", producto
Escribir "El promedio de los 3 valores es igual a: ", Redon (promedio*100)
/100
```

FinAlgoritmo

Al ejecutarlo, se tiene (**figura 1.13**):


Figura 1.13: Ejecución 2 Ejercicio1C1

```

*** Ejecución Iniciada. ***
Ingrese el valor 1: > 5
Ingrese el valor 2: > 10
Ingrese el valor3 3: > 2
La suma de los 3 valores es igual a      : 17
El producto de los 3 valores es igual a: 100
El promedio de los 3 valores es igual a: 5.67
*** Ejecución Finalizada. ***

```

Otra forma de ejecutar (correr) el algoritmo es paso a paso, “para ver qué instrucciones se ejecutan y en qué orden, y observar cómo cambian los contenidos de las variables” (Novara, 2003b, párr. 8). Se puede hacer de la siguiente forma:

- Clic en el menú “Ejecutar” / “Ejecutar paso a paso”
- Clic en el ícono paso a paso 

La secuencia de ejecución paso a paso es la siguiente:

Algoritmo Ejercicio1C1

El algoritmo comienza su ejecución (**figura 1.14**).

Figura 1.14: Inicio ejecución paso a paso

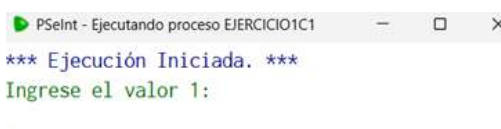
Pasa a la siguiente línea de código, donde se definen o declaran las variables val1, val2, val3, suma, producto, promedio de tipo real.

Definir val1, val2, val3, suma, producto, promedio Como Real

Pasa a la siguiente línea de código:

Escribir Sin Saltar "Ingrese el valor 1: "

Esto permite visualizar el mensaje entre comillas dobles (**figura 1.15**).

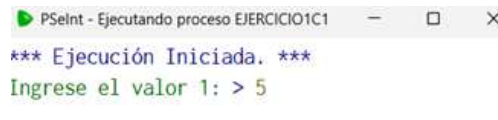
Figura 1.15: Visualización mensaje Ingrese el valor 1

La ejecución continúa con la línea:

```
Leer val1
```

El usuario ingresa un valor y presiona la tecla Enter; por ejemplo, el valor 5 (figura 1.16).

Figura 1.16: Entrada del dato val1

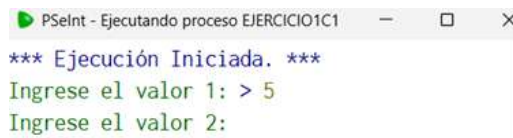


Continúa la ejecución con la siguiente línea de código:

```
Escribir Sin Saltar "Ingrese el valor 2: "
```

Se visualiza el mensaje entre comillas dobles en la ventana de ejecución (figura 1.17).

Figura 1.17: Visualización mensaje Ingrese el valor 2:

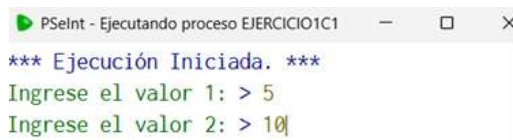


La ejecución continúa con la línea:

```
Leer val2
```

El usuario ingresa un valor y presiona la tecla Enter; por ejemplo, el valor 10 (figura 1.18).

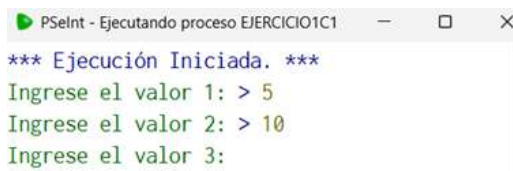
Figura 1.18: Entrada del dato val2



La ejecución continúa con la línea:

```
Escribir Sin Saltar "Ingrese el valor 3: "
```

Se visualiza el mensaje entre comillas dobles en la ventana de ejecución (figura 1.19).

Figura 1.19: Visualización mensaje Ingrese el valor 3:


```

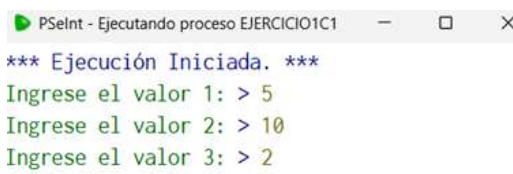
PSeInt - Ejecutando proceso EJERCICIO1C1
*** Ejecución Iniciada. ***
Ingrese el valor 1: > 5
Ingrese el valor 2: > 10
Ingrese el valor 3:

```

La ejecución continúa con la línea:

```
Leer val3
```

El usuario ingresa un valor y presiona la tecla Enter; por ejemplo, el valor 2 (figura 1.20).

Figura 1.20: Entrada del dato val3


```

PSeInt - Ejecutando proceso EJERCICIO1C1
*** Ejecución Iniciada. ***
Ingrese el valor 1: > 5
Ingrese el valor 2: > 10
Ingrese el valor 3: > 2

```

La ejecución continúa con las líneas de código:

```

suma <- val1 + val2 + val3
producto <- val1 * val2 * val3
promedio <- suma / 3

```

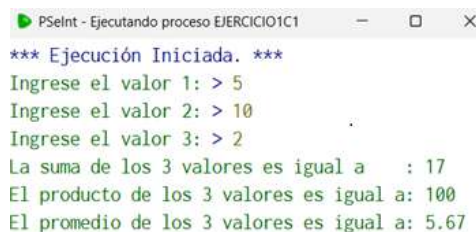
En la cual se calculan la suma, el producto y el promedio, respectivamente. Los resultados se guardan en las variables suma, producto y promedio, y a continuación se ejecutan las siguientes líneas de código:

```

Escribir "La suma de los 3 valores es igual a : ", suma
Escribir "El producto de los 3 valores es igual a: ", producto
Escribir "El promedio de los 3 valores es igual a: ", Redon(promedio*100)/100

```

La pantalla muestra el resultado de sumar los tres valores ingresados, el producto y el promedio (figura 1.21).

Figura 1.21: Visualización datos de salida parciales


```

PSeInt - Ejecutando proceso EJERCICIO1C1
*** Ejecución Iniciada. ***
Ingrese el valor 1: > 5
Ingrese el valor 2: > 10
Ingrese el valor 3: > 2
La suma de los 3 valores es igual a : 17
El producto de los 3 valores es igual a: 100
El promedio de los 3 valores es igual a: 5.67

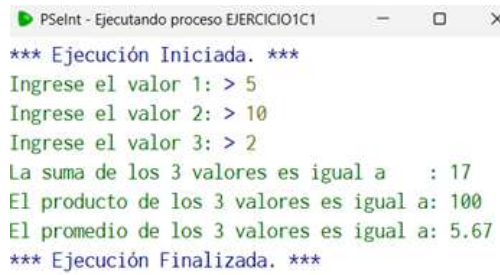
```

Finalmente, se ejecuta la línea de código:

FinAlgoritmo

Y se termina la ejecución paso a paso del algoritmo (**figura 1.22**).

Figura 1.22: Fin ejecución paso a paso



```

PSeInt - Ejecutando proceso EJERCICIO1C1
*** Ejecución Iniciada. ***
Ingrese el valor 1: > 5
Ingrese el valor 2: > 10
Ingrese el valor 3: > 2
La suma de los 3 valores es igual a : 17
El producto de los 3 valores es igual a: 100
El promedio de los 3 valores es igual a: 5.67
*** Ejecución Finalizada. ***

```

Para más detalle sobre la forma de ejecución, se sugiere ir a la opción “Ayuda” de PSeInt: “Ayuda” / “Índice” / “Seguimiento de algoritmo”.

2. Realizar un algoritmo en pseudocódigo para convertir grados Celsius a grados Kelvin y Fahrenheit.

Los pasos que se deben seguir son:

- a. Hacer el análisis del problema (identificación de variables, datos de entrada, procesos parciales, procesos totales, datos de salida parciales y datos de salida totales).
- b. Codificar el algoritmo en PSeInt.
- c. Llevar a cabo la prueba de escritorio.

Análisis del problema

Al leer detenidamente el problema planteado, se puede ver claramente que intervienen tres variables: grados Celsius, grados Kelvin y grados Fahrenheit; por tanto, son las variables que se requieren (**tabla 1.2**).

Tabla 1.2: Identificación de variables Ejercicio2C1

Tipo	Variable	Descripción
Entero	graCel	Grados Celsius (centígrados)
Real	graKel	Grados Kelvin
Real	graFah	Grados Fahrenheit

Seguidamente se determinan los datos de entrada, aquí se plantea el siguiente interrogante, ¿qué datos se requieren ingresar al algoritmo para realizar los procesos?, como respuesta se tiene entonces que se requiere leer o ingresar los grados Celsius para convertirlos, posteriormente, a grados Kelvin y Fahrenheit.

Datos de entrada: graCel

A continuación, se ejecutan los procesos parciales y totales. Los procesos parciales son todos aquellos cálculos necesarios por cada carga o entrada de datos (datos de entrada), y los totales son los que se realizan al finalizar la entrada de datos (tienen la particularidad de que se efectúan una sola vez).

Observaciones

- Para convertir grados Celsius a grados Kelvin y Fahrenheit, se tienen las siguientes fórmulas matemáticas:

$$\text{Grados Kelvin} = \text{grados Celsius} + 273.15$$

$$\text{Grados Fahrenheit} = (\text{grados Celsius} * 9/5) + 32$$

- Como se indicó en el ejemplo anterior, en estos ejercicios iniciales no hay procesos totales ni datos de salida totales.

Procesos parciales:

$$graKel \leftarrow graCel + 273.15$$

$$graFah \leftarrow (graCel * 9/5) + 32$$

Procesos totales: N/A

Datos de salida parciales: graCel, graFah

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Los pasos para seguir son:

- a. Definir o declarar las variables.
- b. Ingresar los datos de entrada.
- c. Realizar los procesos parciales.
- d. Visualizar los datos de salida parciales.

Algoritmo Ejercicio2C1

```
//Definir o declarar las variables
Definir graCel Como Entero
Definir graKel, GraFah Como Real
```

```

//Datos de entrada
Escribir Sin Saltar "Grados Celsius: "
Leer graCel

//Procesos parciales
graKel ← graCel + 273.15
graFah ← (graCel * 9/5) + 32

//Datos de salida parciales
Escribir "Los ", graCel, "grados Celsius equivalentes a ", graKel, "grados
Kelvin"
Escribir "Los ", graCel, "grados Celsius equivalentes a ", graFah, "grados
Fahrenheit"

FinAlgoritmo

```

Nota. Obsérvese que los datos de salida parciales se han acompañado de mensajes explicativos para que los usuarios entiendan los resultados. Esto es así porque los desarrolladores de *software* crean sistemas de información para que los usen otras personas, y, por ello, tanto las entradas como las salidas deben tener mensajes claros.

Prueba de escritorio

```

graCel: 35
graKel: 35 + 273.15 = 308.15
graFah: (35 * 9/5) + 32 = 95

```

Al ejecutar el algoritmo, se obtiene el siguiente resultado (**figura 1.23**):

Figura 1.23: Ejecución Ejercicio2C1

```

*** Ejecución Iniciada. ***
Grados Celsius: > 35
Los 35 grados Celsius equivalentes a 308.15 grados Kelvin
Los 35 grados Celsius equivalentes a 95 grados Fahrenheit
*** Ejecución Finalizada. ***

```

Esto indica que el algoritmo es correcto, por cuanto coinciden los datos de salida de la prueba de escritorio con los datos de salida del algoritmo al momento de ejecutarlo.

- Una persona invierte su capital en un banco y desea saber cuánto dinero ganará después de un mes si la entidad paga a razón del 2% mensual.

Análisis del problema

Las variables que se pueden identificar son el capital invertido, la ganancia y la tasa de interés (tabla 1.3).

Tabla 1.3: Identificación de variables Ejercicio3C1

Tipo	Variable	Descripción
Real	capInv	Capital invertido
Real	tasInt	Tasa de interés
Real	gan	Ganancia en un mes

Datos de entrada: capInv

Procesos parciales:

$tasInt \leftarrow 2$

$gan \leftarrow (capInv * tasInt)/100$

Procesos totales: N/A

Datos de salida parciales: gan

Datos de salida totales: N/A

Observaciones

Algunos dirán que se puede omitir la variable *tasInt* y reemplazarla directamente por 2 en la fórmula, así:

$gan \leftarrow (capInv * 2)/100$

Incluso, algunos irían más allá y la plantearían como:

$gan \leftarrow capInv * 0.02$

Las dos opciones son correctas y dan el mismo resultado, pero tienen las siguientes desventajas:

- Es preferible usar variables antes que valores constantes, debido a que, si las condiciones o requerimientos cambian, se debe ir a cada línea de código donde se use este valor para actualizar; en tanto que, si se usa una variable, solo se debe actualizar la línea donde se inicializa la variable, y ese cambio se verá reflejado en todo el algoritmo.
- Hacer uso de $capInv * 0.02$ implica que el programador primero tuvo que realizar la operación $2/100$ para obtener 0.02. Es ideal plantear las operaciones y dejar que el computador haga los cálculos: esto garantiza que no se cometan errores.

Es recomendable hacer uso de variables y no de valores constantes en los distintos procesos, sean parciales o totales.

Desarrollo del algoritmo en PSeInt

Los pasos para seguir son:

- a. Definir o declarar las variables.
- b. Inicializar las variables.
- c. Ingresar los datos de entrada.
- d. Realizar los procesos parciales.
- e. Visualizar los datos de salida parciales.

Nótese que en este ejercicio hay un paso adicional: inicializar variables, en este caso, `tasInt`, y se le asigna 2, equivalente al 2%, de acuerdo con el enunciado del problema.

Algoritmo Ejercicio3C1

```
//Definir o declarar las variables
Definir capInv, tasInt, gan Como Real

//Inicializar las variables
tasInt ← 2

//Datos de entrada
Escribir Sin Saltar "Capital invertido: "
Leer capInv

//Procesos parciales
gan ← (capInv * tasInt) / 100

//Datos de salida parciales
Escribir "Los ", capInv, "pesos a una tasa de interés del ", tasInt, "%
        tuvo un rendimiento de $", gan, "pesos"
```

FinAlgoritmo

Prueba de escritorio

capInv: 2000000

tasInt: 2

gan: $(2000000 * 2) / 100 = 40000$

Al ejecutar el algoritmo, se obtiene el siguiente resultado (**figura 1.24**):

Figura 1.24: Ejecución Ejercicio3C1

```

*** Ejecución Iniciada. ***
Capital invertido: > 2000000
Los 2000000 pesos a una tasa de interés del 2% tuvo un rendimiento de $40000 pesos
*** Ejecución Finalizada. ***

```

Al comparar el resultado de la prueba de escritorio con el de la ejecución, se aprecia que son exactamente iguales, así que el algoritmo es correcto.

- Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas. Él desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que hizo en el mes y el total que recibirá en dicho periodo.

Análisis del problema

Las variables que intervienen en el problema son sueldo base y porcentaje de comisión, y, al tener un porcentaje de comisión, hay que calcular el valor de esa comisión (otra variable). Como son tres ventas, entonces se tienen tres variables: venta 1, venta 2 y venta 3; el total recibido al mes corresponde al sueldo neto. Por lo tanto, en la identificación de las variables se tiene (**tabla 1.4**):

Tabla 1.4: Identificación de variables Ejercicio4C1

Tipo	Variable	Descripción
Real	sueBas	Sueldo base o básico
Real	porCom	Porcentaje de comisión
Real	valCom	Valor de la comisión
Real	valVen1	Valor de la venta 1
Real	valVen2	Valor de la venta 2
Real	valVen3	Valor de la venta 3
Real	sueNet	Sueldo neto

Datos de entrada: sueBas, valVen1, valVen2, valVen3

Procesos parciales:

$porCom \leftarrow 10$

$valCom \leftarrow ((valVen1 + valVen2 + valVen3) * porCom) / 100$

$sueNet \leftarrow sueBas + valCom$

Procesos totales: N/A

Datos de salida parciales: valCom, sueNet

Datos de salida totales: N/A

Observaciones

Si se analiza el proceso $\text{valCom} \leftarrow ((\text{valVen1} + \text{valVen2} + \text{valVen3}) * \text{porCom}) / 100$, se debe tener en cuenta que al principio hay doble paréntesis, porque primero se desea sumar el valor de las tres ventas para luego multiplicarlas por el porcentaje de comisión y, finalmente, dividir entre 100. Al momento de efectuar las operaciones, es necesario considerar el orden de prioridad de ellas para agrupar los términos. El orden de prioridad es:

- () y [] Paréntesis y corchetes
- ^ Potencia
- * y / Multiplicaciones y divisiones
- + y - Sumas y restas

Cuando hay dos operaciones del mismo nivel seguidas, por ejemplo, $4 + 5 - 10$, estas se resuelven de izquierda a derecha, es decir, primero se suma $4 + 5$ y al resultado se le resta 10:

$$4 + 5 = 9$$

$$9 - 10 = -1$$

Para el siguiente caso, $4 * (10 - 5)$, primero se realiza la operación del paréntesis $(10 - 5)$, y el resultado se multiplica por 4:

$$4 * (10 - 5) = 4 * (5) = 20$$

Si se tuviera $4 * 10 - 5$, primero se realiza la multiplicación y luego la diferencia:

$$4 * 10 - 5 = 40 - 5 = 35$$

Como se puede apreciar, los resultados son totalmente distintos; de allí la importancia de agrupar correctamente las operaciones según se requiera, y esto lo indicará el contexto del problema que se esté resolviendo.

Desarrollo del algoritmo en PSeInt

Los pasos para seguir son:

- a. Definir o declarar las variables.
- b. Inicializar las variables.
- c. Ingresar los datos de entrada.
- d. Realizar los procesos parciales.
- e. Visualizar los datos de salida parciales.

Algoritmo Ejercicio4C1

```
//Declarar o definir las variables
Definir sueBas, porCom, valCom, valVen1, valVen2, valVen3, sueNet Como
    Real

//Inicializar variables
porCom ← 10

//Datos de entrada
Escribir Sin Saltar "Sueldo básico: "
Leer sueBas
Escribir Sin Saltar "Valor venta 1: "
Leer valVen1
Escribir Sin Saltar "Valor venta 2: "
Leer valVen2
Escribir Sin Saltar "Valor venta 3: "
Leer valVen3

//Procesos parciales
valCom ← ((valVen1 + valVen2 + valVen3) * porCom) / 100
sueNet ← sueBas + valCom

//Datos de salida parciales
Escribir "Valor de la comisión: ", valCom
Escribir "Sueldo neto : ", sueNet
```

FinAlgoritmo

Prueba de escritorio

sueBas: 2000000

porCom: 10

valVen1: 850000

valVen2: 1200000

valVen3: 980000

$\text{valCom: } ((850000 + 1200000 + 980000) * 10) / 100 = 303000$
 $\text{sueNet: } 2000000 + 303000 = 2303000$

Al ejecutar el algoritmo, se obtiene el siguiente resultado (**figura 1.25**).

Figura 1.25: Ejecución Ejercicio4C1

```

*** Ejecución Iniciada. ***
Sueldo básico: > 2000000
Valor venta 1: > 850000
Valor venta 2: > 1200000
Valor venta 3: > 980000
Valor de la comisión: 303000
Sueldo neto      : 2303000
*** Ejecución Finalizada. ***

```

El resultado arrojado por el algoritmo coincide con la prueba de escritorio, lo que indica que es correcto.

- Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuánto deberá pagar finalmente por esta.

Análisis del problema

Una de las variables que se pueden identificar es el porcentaje de descuento. Como se tiene este porcentaje, se debe calcular el valor por descontar, que se convierte en otra de las variables; el valor por descontar se calcula, a su vez, sobre el valor de compra. Y, finalmente, está el valor o total a pagar. Así, las variables por identificar son (**tabla 1.5**):

Tabla 1.5: Identificación de variables Ejercicio5C1

Tipo	Variable	Descripción
Real	porDes	Porcentaje de descuentos
Real	valDes	Valor del descuento
Real	valCom	Valor de la compra
Real	valPag	Valor para pagar

Datos de entrada: valCom

Procesos parciales:

$\text{porDes} \leftarrow 15$

$\text{valDes} \leftarrow (\text{valCom} * \text{porDes}) / 100$

$valPag \leftarrow valCom - valDes$

Procesos totales: N/A

Datos de salida parciales: porDes, valDes, valPag

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Los pasos para seguir son:

- a. Definir o declarar las variables.
- b. Inicializar las variables.
- c. Ingresar los datos de entrada.
- d. Realizar los procesos parciales.
- e. Visualizar los datos de salida parciales.

Algoritmo Ejercicio5C1

```
//Definición o declaración de las variables
Definir porDes, valDes, valCom, valPag Como Real

//Inicializar las variables
porDes ← 15

//Datos de entrada
Escribir Sin Saltar "Valor de la compra: "
Leer valCom

//Procesos parciales
valDes ← (valCom * porDes) / 100
valPag ← valCom - valDes

//Datos de salida parciales
Escribir "Porcentaje descuento: ", porDes
Escribir "Valor descontado : ", valDes
Escribir "Valor a pagar : ", valPag
```

FinAlgoritmo

Prueba de escritorio

porDes: 15

valCom: 750000

valDes: $(750000 * 15) / 100 = 112500$

valPag: $750000 - 112500 = 637500$

Al ejecutar el algoritmo, se obtiene el siguiente resultado (**figura 1.26**).

Figura 1.26: Ejecución Ejercicio5C1

```

*** Ejecución Iniciada. ***
Valor de la compra: > 750000
Porcentaje descuento: 15
Valor descontado    : 112500
Valor a pagar       : 637500
*** Ejecución Finalizada. ***

```

El resultado arrojado por el algoritmo coincide con la prueba de escritorio, lo que indica que es correcto.

6. Un estudiante desea saber cuál será su calificación final en el curso de Algoritmos, con los siguientes ítems de calificaciones:

Primer parcial: 20 %

Segundo parcial: 20 %

Práctica: 35 %

Parcial final: 25 %

Análisis del problema

Al realizar la lectura del problema, se pueden identificar las siguientes variables: calificación final o nota definitiva, primer parcial, segundo parcial, práctica, parcial final. Como todos estos ítems tienen porcentajes, es preferible hacer uso de una variable para cada uno de ellos, de manera que, si se modifica un dato posteriormente, solo se tenga que actualizar la inicialización de las variables. Estas son, entonces, el porcentaje del primer parcial, del segundo parcial, de la práctica y del parcial final (**tabla 1.6**).

Tabla 1.6: Identificación de variables Ejercicio6C1

Tipo	Variable	Descripción
Real	notDef	Nota definitiva
Real	priPar	Primer parcial
Real	segPar	Segundo parcial
Real	pra	Práctica
Real	parFin	Parcial final
Real	porPriPar	Porcentaje primer parcial
Real	porSegPar	Porcentaje segundo parcial
Real	porPra	Porcentaje práctica
Real	porParFin	Porcentaje parcial final

Datos de entrada: priPar, segPar, pra, parFin

Procesos parciales:

$porPriPar \leftarrow 20$

$porSegPar \leftarrow 20$

$porPra \leftarrow 35$

$porParFin \leftarrow 25$

$notDef \leftarrow ((priPar * porPriPar)/100) + ((segPar * porSegPar)/100)$
 $+ ((pra * porPra)/100) + ((parFin * porParFin)/100)$

Procesos totales: N/A

Datos de salida parciales: notDef

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio6C1

```
//Definición o declaración de las variables
Definir notDef, priPar, segPar, pra, parFin, porPriPar, porSegPar, porPra,
porParFin Como Real

//Inicializar las variables
porPriPar ← 20
porSegPar ← 20
porPra ← 35
porParFin ← 25

//Datos de entrada
Escribir Sin Saltar "Primer parcial : "
Leer priPar
Escribir Sin Saltar "Segundo parcial: "
Leer segPar
Escribir Sin Saltar "Práctica : "
Leer pra
Escribir Sin Saltar "Parcial final : "
Leer parFin

//Procesos parciales
notDef ← ((priPar * porPriPar) / 100) + ((segPar * porSegPar) / 100)
+ ((pra * porPra) / 100) + ((parFin * porParFin) / 100)

//Datos de salida parciales
Escribir "Nota definitiva : ", notDef
```

FinAlgoritmo

Prueba de escritorio**porPriPar:** 20**porSegPar:** 20**porPra:** 35**porParFin:** 25**priPar:** 4.2**segPar:** 2.5**pra:** 3.7**parFin:** 4.5**notDef:** $((4.2 * 20)/100) + (2.5 * 20)/100 + ((3.7 * 35)/100) + ((4.5 * 25)/100)$ **notDef:** $0.84 + 0.5 + 1.295 + 1.125 = 3.76$

Al ejecutar el algoritmo, se obtiene el siguiente resultado (**figura 1.27**).

Figura 1.27: Ejecución Ejercicio6C1

```

*** Ejecución Iniciada. ***
Primer parcial : > 4.2
Segundo parcial: > 2.5
Práctica       : > 3.7
Parcial final  : > 4.5
Nota definitiva : 3.76
*** Ejecución Finalizada. ***

```

El resultado arrojado por el algoritmo coincide con la prueba de escritorio, lo que indica que es correcto.

- Determinar el porcentaje de hombres y de mujeres presentes en el curso de Algoritmos, si se conoce el número de hombres y mujeres que tiene.

Análisis del problema

Las variables que se pueden identificar son porcentaje de hombres, porcentaje de mujeres, número de hombres y número de mujeres. Para calcular el porcentaje de hombres y de mujeres es necesario conocer el total de estudiantes inscritos en el curso (**tabla 1.7**).

Tabla 1.7: Identificación de variables Ejercicio7C1

Tipo	Variable	Descripción
Real	porHom	Porcentaje de hombres
Real	porMuj	Porcentaje de mujeres
Entero	numHom	Número de hombres
Entero	numMuj	Número de mujeres
Entero	totEst	Total de estudiantes

Datos de entrada: numHom, numMuj

Procesos parciales:

$totEst \leftarrow numHom + numMuj$

Un porcentaje se calcula de la siguiente forma:

El total de estudiantes corresponde al 100%, ¿a qué porcentaje corresponde el número de hombres?

$$\begin{array}{lcl} totEst & \longrightarrow & 100\% \\ numHom & \longrightarrow & X? \end{array}$$

La X se reemplaza por la variable porHom:

$$porHom \leftarrow (numHom * 100) / totEst$$

De la misma forma se plantea el porcentaje de mujeres, con sus respectivas variables:

$$porMuj \leftarrow (numMuj * 100) / totEst$$

Procesos totales: N/A

Datos de salida parciales: totEst, porHom, porMuj

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio7C1

```
//Definición o declaración de las variables
Definir porHom, porMuj, numHom, numMuj, totEst Como Real

//Datos de entrada
Escribir Sin Saltar "Número de estudiantes hombres: "
Leer numHom
Escribir Sin Saltar "Número de estudiantes mujeres: "
Leer numMuj

//Procesos parciales
totEst ← numHom + numMuj
porHom ← (numHom * 100) / totEst
porMuj ← (numMuj * 100) / totEst

//Datos de salida parciales
Escribir "Total estudiantes del curso: ", totEst
Escribir "porcentaje de hombres : ", Redon(porHom * 100) / 100
Escribir "Porcentaje de mujeres : ", Redon(porMuj * 100) / 100
```

FinAlgoritmo

Prueba de escritorio**numHom:** 15**numMuj:** 22**totEst:** $15 + 22 = 37$ **porHom:** $(15 * 100)/37 = 40,54054$ **porMuj:** $(22 * 100)/37 = 59,45945$

Al ejecutar el algoritmo, se obtiene el siguiente resultado (**figura 1.28**).

Figura 1.28: Ejecución Ejercicio7C1

```

*** Ejecución Iniciada. ***
Número de estudiantes hombres: > 15
Número de estudiantes mujeres: > 22
Total estudiantes del curso: 37
porcentaje de hombres      : 40.54
Porcentaje de mujeres      : 59.46
*** Ejecución Finalizada. ***

```

Como se aprecia, los resultados son exactos, pero en la ejecución se redondean los porcentajes a dos cifras decimales. El porcentaje de hombres es igual porque después del 40,54 sigue cero; en cambio, en el de las mujeres, después de 59,45 sigue 9. En la estadística, si la cifra es mayor o igual que 5, se aproxima al siguiente valor; en este caso, como es 9, el 5 pasa a ser 6: 59,46.

1.19 Ejercicios propuestos

Para cada uno de los siguientes ejercicios, realizar el análisis del problema, el algoritmo en PSeInt y la prueba de escritorio.

1. Calcular el área y perímetro de un triángulo.
2. Calcular el radio de un círculo.
3. Calcular el interés que una persona debe pagar por un capital a una tasa de interés N.
4. Para conformar una empresa se unen tres socios, y cada uno aporta un capital distinto; se desea conocer el porcentaje de inversión de cada uno.
5. Calcular la función cuadrática $X = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$; para la raíz cuadrada, usar la función RC(X) o RAIZ(X), que se puede consultar en la ayuda de PSeInt.

Capítulo 2

CONDICIONALES Y CICLOS



Las estructuras condicionales y los ciclos son de mucha importancia en el desarrollo de *software* porque permiten realizar validaciones para la toma de decisiones, así como repetir un conjunto de instrucciones o sentencias un número determinado de veces, o hasta que se cumpla cierta condición.

2.1 Condicionales

Son usados para seleccionar “la ruta que debe tomar la ejecución de instrucciones de un algoritmo, o también el flujo que debe llevar el control de ejecución cuando se presentan tomas de decisiones” (Oviedo Regino, 2012, p. 67). Según propone Geomatemáticas (2005), las estructuras condicionales “comparan una variable contra otro(s) valor(es), para que en base al resultado de esta comparación, se siga un curso de acción dentro del programa, la comparación se puede hacer contra otra variable o contra una constante, según se necesite” (párr. 1).

Los condicionales son esenciales en la programación porque sirven para validar contenidos o tomar decisiones a la hora de seguir uno u otro curso o flujo de acción en la ejecución de un algoritmo o programa. Se tiene la estructura SI y SEGÚN, con las cuales se pueden formar condicionales simples, compuestos, anidados y múltiples.

Condicionales simples y compuestos

Los condicionales simples y compuestos se realizan con la estructura SI. Tienen la siguiente sintaxis:

Condicional simple

Si <Condición> **Entonces**

Acción 1

Acción 2

...

Acción N

FinSi

Si la condición se cumple (es verdadera), se ejecutan las sentencias o instrucciones dentro del cuerpo Si... FinSi; si no se cumple (es falsa), la ejecución se salta a la línea de código después del FinSi.

Condicional compuesto

Si <Condición> Entonces

Acción 1
Acción 2
...
Acción N

SiNo

Acción 1
Acción 2
...
Acción N

FinSi

En este caso se valida la condición: si se cumple (es verdadera), se ejecutan las sentencias o instrucciones dentro del cuerpo Si... SiNo; si no se cumple (es falsa), se ejecutan las sentencias o instrucciones dentro del cuerpo SiNo... FinSi.

Condicionales múltiples

Se pueden realizar con el condicional SI o SEGÚN, permiten validar o comparar distintas condiciones ejecutándose para cada una de estas un grupo de sentencias o instrucciones, tienen la siguiente sintaxis.

Si <Condición 1> Entonces

Acción 1
Acción 2
...
Acción N

SiNo

Si <Condición 2> Entonces

Acción 1
Acción 2
...
Acción N

SiNo

Si <Condición 3> Entonces

Acción 1
Acción 2
...
Acción N

```

SiNo
    Acción 1
    Acción 2
    ...
    Acción N
FinSi

```

```

FinSi

```

```

FinSi

```

De esta forma se pueden agregar las condiciones necesarias de acuerdo con los requerimientos del problema, teniendo en cuenta que cada SI debe tener un SiNo y un FinSi.

La otra estructura es SEGÚN, la cual tiene la siguiente sintaxis:

```

Según <Variable> Hacer

```

```

    <Lista 1>:
        Acción 1
        Acción 2
        ...
        Acción N

```

```

    <Lista 2>:
        Acción 1
        Acción 2
        ...
        Acción N

```

```

    ...

```

```

    <Lista N>:
        Acción 1
        Acción 2
        ...
        Acción N

```

```

De otro modo:
    Acción 1
    Acción 2
    ...
    Acción N

```

```

FinSegún

```

En la sintaxis, las listas pueden ser de tipo entero (1, 2, 3, 4, etc.) o de tipo carácter (“A”, “B”, “C”, “D”, “E”, etc.).

La sección “**De otro modo**” es opcional: se ejecuta en caso de que ninguna de las opciones anteriores se cumpla.

2.2 Ejemplos resueltos

1. Realizar un algoritmo que lea o capture dos valores. Si el valor 1 es mayor o igual al valor 2, hacer la suma de estos.

Análisis del problema

Tabla 2.1: Identificación de variables Ejercicio1C2

Tipo	Variable	Descripción
Real	val1	Valor 1
Real	val2	Valor 2
Real	suma	Suma de los dos números

Datos de entrada: val1, val2

Procesos parciales:

$suma \leftarrow val1 + val2$

Procesos totales: N/A

Datos de salida parciales: suma

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio1C2

```
//Definición o declaración de las variables
Definir val1, val2, suma Como Real

//Datos de entrada
Escribir Sin Saltar "Valor No. 1: "
Leer val1
Escribir Sin Saltar "Valor No. 2: "
Leer val2

//Procesos parciales
Si val1 >= val2 Entonces
    suma ← val1 + val2
```

```

//Datos de salida parciales
Escribir "Suma = ", suma
FinSi
FinAlgoritmo

```

Prueba de escritorio

Val1:	5	2
Val2:	2	5
Suma:	7	—

Al ejecutar el algoritmo, se obtiene el siguiente resultado (**figuras 2.1 y 2.2**).

Figura 2.1: Ejecución 1 Ejercicio1C2

```

*** Ejecución Iniciada. ***
Valor No. 1: > 5
Valor No. 2: > 2
Suma = 7
*** Ejecución Finalizada. ***

```

Figura 2.2: Ejecución 2 Ejercicio1C2

```

*** Ejecución Iniciada. ***
Valor No. 1: > 2
Valor No. 2: > 5
*** Ejecución Finalizada. ***

```

Como se aprecia, en la ejecución 1 el resultado coincide con la prueba de escritorio: la suma es igual a 7. En la ejecución 2 no hay suma, dado que 2 es menor que 5, y no se realiza ningún proceso ni salida de datos, lo cual indica que el algoritmo es correcto.

- Realizar un algoritmo que lea o capture dos valores. Si el valor 1 es menor o igual al valor 2, hacer la suma; de lo contrario, hacer la diferencia.

Análisis del problema

Tabla 2.2: Identificación de variables Ejercicio2C2

Tipo	Variable	Descripción
Real	val1	Valor 1
Real	val2	Valor 2
Real	res	Resultado de la operación matemática

Datos de entrada: val1, val2

Procesos parciales:

$res \leftarrow val1 + val2$

$res \leftarrow val1 - val2$

Procesos totales: N/A

Datos de salida parciales: res

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio2C2

```
//Definición o declaración de las variables
Definir val1, val2, res Como Real

//Datos de entrada
Escribir Sin Saltar "Valor No. 1: "
Leer val1
Escribir Sin Saltar "Valor No. 2: "
Leer val2

//Procesos parciales
Si val1 <= val2 Entonces
    res ← val1 + val2
SiNo
    res ← val1 - val2
FinSi

//Datos de salida parciales
Escribir "Resultado = ", Res
```

FinAlgoritmo

Prueba de escritorio

Val1:	5	2
Val2:	2	5
Suma:	3	7

Al ejecutar el algoritmo, se obtiene el siguiente resultado (**figuras 2.3 y 2.4**).

Figura 2.3: Ejecución 1 Ejercicio2C2

```

*** Ejecución Iniciada. ***
Valor No. 1: > 5
Valor No. 2: > 2
Resultado = 3
*** Ejecución Finalizada. ***

```

Figura 2.4: Ejecución 2 Ejercicio2C2

```

*** Ejecución Iniciada. ***
Valor No. 1: > 2
Valor No. 2: > 5
Resultado = 7
*** Ejecución Finalizada. ***

```

En la ejecución 1, como 5 es mayor que 2, se realiza la resta entre 5 y 2, y el resultado es 3. En la ejecución 2, como 2 es menor que 5, se realiza la suma de los valores, y 7 es el resultado, lo cual coincide con la prueba de escritorio.

- Realizar un algoritmo que lea o capture dos valores. Si el valor 1 es menor que el valor 2, hacer la suma; si el valor 1 es mayor que el valor 2, hacer la diferencia; si son iguales, hacer la multiplicación.

Análisis del problema

Tabla 2.3: Identificación de variables Ejercicio3C2

Tipo	Variable	Descripción
Real	val1	Valor 1
Real	val2	Valor 2
Real	res	Resultado de la operación matemática

Datos de entrada: val1, val2

Procesos parciales:

$res \leftarrow val1 + val2$

$res \leftarrow val1 - val2$

$res \leftarrow val1 * val2$

Procesos totales: N/A

Datos de salida parciales: res

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio3C2

```
//Definición o declaración de las variables
Definir val1, val2, res Como Real

//Datos de entrada
Escribir Sin Saltar "Valor No. 1: "
Leer val1
Escribir Sin Saltar "Valor No. 2: "
Leer val2

//Procesos parciales
Si val1 < val2 Entonces
    res ← val1 + val2
SiNo
    Si val > val2 Entonces
        res ← val1 - val2
    SiNo
        res ← val1 * val2
    FinSi
FinSi

//Datos de salida parciales
Escribir "Resultado = ", Res
```

FinAlgoritmo

Observaciones

Nótese que primero se valida si el valor 1 es menor que el valor 2 (Si $val1 < val2$ Entonces), y se realiza la suma; si no se cumple la condición, entonces se valida si el valor 1 es mayor que el valor 2 (Si $val1 > val2$ Entonces), y se realiza la diferencia entre el valor 1 y el valor 2; si no se cumple, se pasa directamente a realizar la multiplicación del valor 1 por el valor 2. No es necesario hacer la pregunta Si $val1 = val2$ porque en la primera condición (Si $val1 < val2$ Entonces) se está negando que $val1$ es menor que el valor 2 parte SiNo, y en la segunda condición (Si $val1 > val2$ Entonces) se está negando que el valor 1 es mayor que el valor 2 parte SiNo; por tanto, si el valor 1 no es mayor ni menor que el valor 2, entonces son iguales.

Prueba de escritorio

Val1:	2	5	10
Val2:	5	2	10
res:	7	3	100

Al ejecutar el algoritmo, se obtiene el siguiente resultado (**figuras 2.5, 2.6 y 2.7**).

Figura 2.5: Ejecución 1 Ejercicio3C2

```
*** Ejecución Iniciada. ***  
Valor No. 1: > 2  
Valor No. 2: > 5  
Resultado = 7  
*** Ejecución Finalizada. ***
```

Figura 2.6: Ejecución 2 Ejercicio3C2

```
*** Ejecución Iniciada. ***  
Valor No. 1: > 5  
Valor No. 2: > 2  
Resultado = 3  
*** Ejecución Finalizada. ***
```

Figura 2.7: Ejecución 3 Ejercicio3C2

```
*** Ejecución Iniciada. ***  
Valor No. 1: > 10  
Valor No. 2: > 10  
Resultado = 100  
*** Ejecución Finalizada. ***
```

En la ejecución 1, como 2 es menor que 5, se realiza la suma de $2 + 5$ y el resultado es 7; en la ejecución 2, como 5 es mayor que 2, se realiza la diferencia $5 - 2$, y el resultado es 3; en la ejecución 3, como 10 es igual a 10, se realiza la multiplicación de $10 * 10$, y el resultado es 100, lo cual coincide con la prueba de escritorio.

4. A los empleados de la compañía ABC les otorgan una sola vez al año una bonificación de acuerdo con su salario básico y los años de antigüedad en la organización según la siguiente información:

Tiempo en años	Porcentaje
Menos de 5 años	5 % del salario básico
5 años o más y menos de 10 años	10 % del salario básico
10 años o más y menos de 15 años	15 % del salario básico
15 años o más y menos de 20 años	20 % del salario básico
20 años o más y menos de 25 años	25 % del salario básico
25 años o más y menos de 30 años	35 % del salario básico
30 años o más	50 % del salario básico

Realizar un algoritmo para determinar la bonificación que recibe un empleado.

Análisis del problema

Tabla 2.4: Identificación de variables Ejercicio4C2

Tipo	Variable	Descripción
Real	salBas	Salario básico
Entero	tieSer	Tiempo de servicio en años
Real	porBon	Porcentaje de bonificación
Real	valBon	Valor de la bonificación

Datos de entrada: salBas, tieSer

Procesos parciales:

$porBon \leftarrow [5 \mid 10 \mid 15 \mid 20 \mid 25 \mid 35 \mid 50]$

$valBon \leftarrow (sueBas * porBon) / 100$

Procesos totales: N/A

Datos de salida parciales: porBon, valBon

Datos de salida totales: N/A

Observaciones

El proceso $porBon \leftarrow [5 \mid 10 \mid 15 \mid 20 \mid 25 \mid 35 \mid 50]$ se lee como porBon se le asigna 5 o 10 o 15 o 20 o 25 o 35 o 50; para su codificación, en el algoritmo se realiza con un condicional SI en este caso, ya que su valor está determinado por el tiempo de servicio del empleado.

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio4C2

```
//Definición o declaración de las variables
Definir salBas, porBon, valBon Como Real
Definir tieSer Como Entero
```


Al ejecutar el algoritmo, se obtiene el siguiente resultado (figuras 2.8, 2.9 y 2.10).

Figura 2.8: Ejecución 1 Ejercicio4C2

```
*** Ejecución Iniciada. ***
Salario básico           : > 1000
Tiempo de servicios en años: > 10
Porcentaje de bonificación: 15
Valor de la bonificación  : 150
*** Ejecución Finalizada. ***
```

Figura 2.9: Ejecución 2 Ejercicio4C2

```
*** Ejecución Iniciada. ***
Salario básico           : > 5000
Tiempo de servicios en años: > 25
Porcentaje de bonificación: 35
Valor de la bonificación  : 1750
*** Ejecución Finalizada. ***
```

Figura 2.10: Ejecución 3 Ejercicio4C2

```
*** Ejecución Iniciada. ***
Salario básico           : > 2250000
Tiempo de servicios en años: > 18
Porcentaje de bonificación: 20
Valor de la bonificación  : 450000
*** Ejecución Finalizada. ***
```

5. Realizar un algoritmo que lea un número entero y determine si es par o impar.

Análisis del problema

Tabla 2.5: Identificación de variables Ejercicio5C2

Tipo	Variable	Descripción
Entero	nro	Número entero
Entero	res	Resultado operación par e impar
Cadena	msg	Mensaje

Datos de entrada: nro

Procesos parciales:

$res \leftarrow nro \text{ MOD } 2$

$msg \leftarrow [“es par” \mid “es impar”]$

Procesos totales: N/A

Datos de salida parciales: msg

Datos de salida totales: N/A

Observaciones

El operador MOD significa módulo o resto de la división; por ejemplo, $x \leftarrow 5 \text{ MOD } 2$: se tiene que x es igual a 1, al dividir $5 / 2 = 2$ y el residuo es 1, porque $2 * 2 = 4$ y $5 - 4 = 1$. Otro ejemplo: $10 \text{ MOD } 2$ da como resultado 0 porque $10 / 2 = 5$, por lo que $5 * 2 = 10$ y $10 - 10 = 0$. También se puede usar el símbolo % para el módulo o resto de la división.

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio5C2

```
//Definición o declaración de las variables
Definir nro, res Como Entero
Definir msg Como Cadena

//Datos de entrada
Escribir Sin Saltar "Ingrese un valor entero: "
Leer nro

//Procesos parciales
res ← nro MOD 2
Si res = 0 Entonces
    msg ← "es par"
SiNo
    msg ← "es impar"
FinSi

//Datos de salida parciales
Escribir "El número ", nro, msg
```

FinAlgoritmo

Prueba de escritorio

Nro	5	10
res	$5 \text{ MOD } 2 = 1$	$10 \text{ MOD } 2 = 0$
msg	“ es impar”	“ es par”

Al ejecutar el algoritmo, se obtiene el siguiente resultado (**figuras 2.11 y 2.12**).

Figura 2.11: Ejecución 1 Ejercicio5C2

```

*** Ejecución Iniciada. ***
Ingrese un valor entero: > 5
El número 5 es impar
*** Ejecución Finalizada. ***

```

Figura 2.12: Ejecución 2 Ejercicio5C2

```

*** Ejecución Iniciada. ***
Ingrese un valor entero: > 10
El número 10 es par
*** Ejecución Finalizada. ***

```

6. Un vendedor recibe un sueldo básico más una comisión del 10 % si su venta es menor que 100000 pesos o del 15 % si su venta es mayor o igual a 100000 pesos. El vendedor desea saber cuánto dinero obtendrá por concepto de comisión y su sueldo.

Análisis del problema

Tabla 2.6: Identificación de variables Ejercicio6C2

Tipo	Variable	Descripción
Real	sueBas	Sueldo básico
Real	porCom	Porcentaje de comisión
Real	valCom	Valor de la comisión
Real	valVen	Valor de la venta
Real	sueNet	Sueldo neto

Datos de entrada: sueBas, valVen

Procesos parciales:

$porCom \leftarrow [10 \mid 15]$

$valCom \leftarrow (valVen * porCom)/100$

$sueNet \leftarrow sueBas + valCom$

Procesos totales: N/A

Datos de salida parciales: porCom, valCom, sueNet

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio6C2

```
//Definición o declaración de las variables
Definir sueBas, porCom, valCom, valVen, sueNet Como Real

//Datos de entrada
Escribir Sin Saltar "Sueldo básico: "
Leer sueBas
Escribir Sin Saltar "Valor venta : "
Leer valVen

//Procesos parciales
Si valVen < 100000 Entonces
    porCom ← 10
SiNo
    porCom ← 15
FinSi

valCom ← (valVen * porCom) / 100
sueNet ← sueBas + valCom

//Datos de salida parciales
Escribir "Porcentaje de comisión: ", porCom
Escribir "Valor comisión : ", valCom
Escribir "Sueldo neto : ", sueNet
```

FinAlgoritmo

Prueba de escritorio

sueBas	1500000	1500000
valVen	90000	150000
porCom	15	20
valCom	$(90000 * 15) / 100 = 13500$	$(150000 * 20) / 100 = 30000$
sueNet	$1500000 + 13500 = 1513500$	$1500000 + 30000 = 1530000$

Al ejecutar el algoritmo, se obtiene el siguiente resultado (**figuras 2.13 y 2.14**).

Figura 2.13: Ejecución 1 Ejercicio6C2

```
*** Ejecución Iniciada. ***
Sueldo básico: > 1500000
Valor venta : > 90000
Porcentaje de comisión: 15
Valor comisión : 13500
Sueldo neto : 1513500
*** Ejecución Finalizada. ***
```

Figura 2.14: Ejecución 2 Ejercicio6C2

```

*** Ejecución Iniciada. ***
Sueldo básico: > 1500000
Valor venta : > 150000
Porcentaje de comisión: 20
Valor comisión : 30000
Sueldo neto : 1530000
*** Ejecución Finalizada. ***

```

7. Un almacén les hace descuento a sus clientes de acuerdo con la siguiente información:

Compras mayores o iguales a 100000 y menores de 200000 tienen descuento del 10 %.
 Compras mayores o iguales a 200000 y menores de 300000 tienen descuento del 15 %.
 Compras mayores o iguales a 300000 y menores de 400000 tienen descuento del 20 %.
 Compras mayores o iguales a 400000 y menores de 500000 tienen descuento del 25 %.
 Compras mayores o iguales a 500000 tienen descuento del 30 %

Realizar un algoritmo para determinar el valor que un cliente debe pagar por su compra.

Análisis del problema

Tabla 2.7: Identificación de variables Ejercicio7C2

Tipo	Variable	Descripción
Real	valCom	Valor de la compra
Real	porDes	Porcentaje de descuento
Real	valDes	Valor descontado
Real	valPag	Valor por pagar

Datos de entrada: valCom

Procesos parciales:

$porDes \leftarrow [0 \mid 10 \mid 15 \mid 20 \mid 25 \mid 30]$

$valDes \leftarrow (valCom * porDes) / 100$

$valPag \leftarrow valCom - valDes$

Procesos totales: N/A

Datos de salida parciales: porDes, valDes, valPag

Datos de salida totales: N/A

Observaciones

Nótese que porDes puede tomar el valor de 0, aunque en el enunciado del problema no está específicamente indicado: se puede deducir que para las ventas menores de 100000 pesos no hay descuento según la información que se presenta; es decir, si las compras mayores o iguales a 100000 y menores de 200000 tienen descuento del 10 %, entonces las compras inferiores a 100000 no lo tienen (el porcentaje de descuento es 0).

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio7C2

```
//Definición o declaración de las variables
Definir valCom, porDes, valDes, valPag Como Real

//Datos de entrada
Escribir Sin Saltar "Valor de la compra: "
Leer valCom

//Procesos parciales
Si valCom < 100000 Entonces
    porDes ← 0
SiNo
    Si valCom < 200000 Entonces
        porDes ← 10
    SiNo
        Si valCom < 300000 Entonces
            porDes ← 15
        SiNo
            Si valCom < 400000 Entonces
                porDes ← 20
            SiNo
                Si ValCom < 500000 Entonces
                    porDes ← 25
                SiNo
                    porDes ← 30
            FinSi
        FinSi
    FinSi
FinSi

valDes ← (valCom * porDes) / 100
valPag ← valCom - valDes

//Datos de salida parciales
```

```
    Escribir "Porcentaje de descuento: ", porDes
    Escribir "Valor descontado : ", valDes
    Escribir "Valor a pagar : ", valPag
```

```
FinAlgoritmo
```

Prueba de escritorio

valCom	250000	80000	650000
porDes	15	0	30
valDes	37500	0	195000
valPag	212500	80000	455000

Al ejecutar el algoritmo, se obtiene el siguiente resultado (figuras 2.15, 2.16 y 2.17).

Figura 2.15: Ejecución 1 Ejercicio7C2

```
*** Ejecución Iniciada. ***
Valor de la compra: > 250000
Porcentaje de descuento: 15
Valor descontado      : 37500
Valor a pagar         : 212500
*** Ejecución Finalizada. ***
```

Figura 2.16: Ejecución 2 Ejercicio7C2

```
*** Ejecución Iniciada. ***
Valor de la compra: > 80000
Porcentaje de descuento: 0
Valor descontado      : 0
Valor a pagar         : 80000
*** Ejecución Finalizada. ***
```

Figura 2.17: Ejecución 3 Ejercicio7C2

```
*** Ejecución Iniciada. ***
Valor de la compra: > 650000
Porcentaje de descuento: 30
Valor descontado      : 195000
Valor a pagar         : 455000
*** Ejecución Finalizada. ***
```

8. Realizar un algoritmo que lea dos valores. De acuerdo con el menú de opciones siguiente, ejecutar la operación indicada:

Prueba de escritorio

1. Sumar
2. Restar
3. Multiplicar
4. Dividir
5. Potencia

Análisis del problema

Tabla 2.8: Identificación de variables Ejercicio8C2

Tipo	Variable	Descripción
Real	val1	Valor 1
Real	val2	Valor 2
Real	res	Resultado de la operación
Entero	opc	Opción seleccionada por el usuario

Datos de entrada: val1, val2, opc

Procesos parciales:

$res \leftarrow [val1 + val2 \mid val1 - val2 \mid val1 * val2 \mid val1/val2 \mid val1 \wedge val2]$

Procesos totales: N/A

Datos de salida parciales:

Res, ["ERROR. No se puede dividir entre 0" | "La opción seleccionada no es válida"]

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Este algoritmo se puede codificar de dos formas: la primera usando el condicional SI y la segunda con el condicional SEGÚN.

Primera forma

Algoritmo Ejercicio8.1C2

```
//Definición o declaración de las variables
Definir val1, val2, res Como Real
Definir opc Como Entero
```

```
//Datos de entrada
Escribir Sin Saltar "Valor 1: "
Leer val1
Escribir Sin Saltar "Valor 2: "
Leer val2
Escribir ""
Escribir "MENÚ DE OPCIONES"
Escribir ""
Escribir "1. Suma"
Escribir "2. Resta"
Escribir "3. Multiplicar"
Escribir "4. Dividir"
Escribir "5. Potencia"
Escribir Sin Saltar "Seleccione una opción: "
Leer opc

//Procesos parciales y salidas parciales
Si opc = 1 Entoncess
    res ← val1 + val2
    Escribir val1, " + ", val2, " = ", res
SiNo
    Si opc = 2 Entonces
        Res ← val1 - val2
        Escribir val1, " - ", val2, " = ", res
    SiNo
        Si opc = 3 Entonces
            res ← val1 * val2
            Escribir val1, " * ", val2, " = ", res
        SiNo
            Si opc = 4 Entonces
                Si val2 ≠ 0 Entonces
                    res ← val1 / val2
                    Escribir val1, " / ", val2, " = ", res
                SiNo
                    Escribir "ERROR. No se puede dividir entre 0"
            FinSi
        SiNo
            Si opc = 5 Entonces
                res ← val1 ^ val2
                Escribir val1, " ^ ", val2, " = ", res
            SiNo
                Escribir "La opción seleccionada no es válida"
            FinSi
        FinSi
    FinSi
FinSi
```

```
FinSi
FinSi
```

```
FinAlgoritmo
```

Segunda forma

```
Algoritmo Ejercicio8_2C2
```

```
//Definición o declaración de las variables
Definir val1, val2, res Como Real
Definir opc Como Entero

//Datos de entrada
Escribir Sin Saltar "Valor 1: "
Leer val1
Escribir Sin Saltar "Valor 2: "
Leer val2
Escribir ""
Escribir "MENÚ DE OPCIONES"
Escribir ""
Escribir "1. Suma"
Escribir "2. Resta"
Escribir "3. Multiplicar"
Escribir "4. Dividir"
Escribir "5. Potencia"
Escribir Sin Saltar "Seleccione una opción: "
Leer opc

//Procesos parciales y salidas parciales
Segun opc Hacer
  1:
    res ← val1 + val2
    Escribir val1, " + ", val2, " = ", res
  2:
    Res ← val1 - val2
    Escribir val1, " - ", val2, " = ", res
  3:
    res ← val1 * val2
    Escribir val1, " * ", val2, " = ", res
  4:
    Si val2 <> 0 Entonces
      res ← val1 / val2
      Escribir val1, " / ", val2, " = ", res
    SiNo
      Escribir "ERROR. No se puede dividir entre 0"
FinSi
```

```
5:
    res ← val1 ^ val2
    Escribir val1, " ^ ", val2, " = ", res
De Otro Modo:
    Escribir "La opción seleccionada no es válida"
FinSegun
```

FinAlgoritmo

Prueba de escritorio

Primera forma

val1	5	10	20	15
val2	2	4	5	0
opc	1	3	2	4
res	7	40	15	ERROR. No se puede dividir entre 0

Segunda forma

val1	5	10	20	15
val2	2	4	5	0
opc	1	3	2	4
res	7	40	15	ERROR. No se puede dividir entre 0

Independientemente de cómo se codifique el algoritmo, sea con SI o con SEGÚN, los resultados serán los mismos.

2.3 Ejercicios propuestos

1. Calcular la utilidad que un trabajador recibe en el reparto anual de utilidades si se le asignan como un porcentaje de su salario mensual que depende de su antigüedad en la empresa:

Tiempo	Porcentaje
Menos de 1 año	5 % del salario
1 año o más y menos de 5 años	10 % del salario
5 años o más y menos de 10 años	15 % del salario
10 años o más y menos de 15 años	20 % del salario
15 años o más y menos de 20 años	30 % del salario
20 años o más	50 % del salario

2. Una ONG ofrece un subsidio de estudio a sus afiliados de acuerdo con la siguiente información:

Estrato	Porcentaje de subsidio
0	100
1	90
2	80
3	70
4	40
5	20
6	0

Determinar el valor del subsidio otorgado por la ONG a un afiliado y el valor que este debe pagar por su matrícula.

3. Realizar un algoritmo que determine el tipo de triángulo según sus lados (equilátero, isósceles, escaleno).
4. Leer tres números enteros y determinar el mayor, el menor y el del medio.
5. Una persona realiza una compra por N valor en un establecimiento. Tiene las siguientes formas de pago:

Forma de pago	Porcentaje de descuento o financiamiento
Contado	Tiene un descuento del 20 %
Crédito a 15 días	Tiene un incremento del 10 % por financiación
Crédito a 30 días	Tiene un incremento del 15 % por financiación
Crédito a 60 días	Tiene un incremento del 20 % por financiación
Crédito a 90 días	Tiene un incremento del 30 % por financiación

Realizar un algoritmo que permita determinar el valor por pagar para el cliente según la forma de pago seleccionada. Se le debe indicar el porcentaje de descuento y el valor descontado, el porcentaje de financiación y el valor de incremento por financiamiento, además del neto que ha de pagar por su compra.

2.4 Ciclos o bucles

Los ciclos o bucles permiten repetir un grupo de sentencias o instrucciones un número determinado de veces, o hasta que se cumpla una condición; en combinación con los condicionales sirven para crear algoritmos bien estructurados que dan solución a problemas de forma eficiente y eficaz.

Hasta el momento se habían desarrollado algoritmos secuenciales, es decir, aquellos cuyas líneas de código se ejecutan secuencialmente desde el inicio hasta llegar al fin del algoritmo; con el uso de los ciclos esta secuencia cambia, gracias a lo cual el flujo de ejecución regresa a una línea determinada del código para ejecutar las líneas de código que formen parte del cuerpo de un determinado ciclo.

En el ejercicio 4 de este capítulo se había calculado la bonificación que recibe un empleado de acuerdo con su salario básico y con el tiempo de servicio en la organización; en el ejercicio 6 se había calculado la comisión que un empleado obtenía por sus ventas y el salario neto; y en el ejercicio 7 se había determinado el valor que un cliente debe pagar por su compra luego de que se le hiciera un descuento según la cuantía de la compra. Estos ejemplos calculan o determinan la bonificación de un solo empleado, la comisión que un solo empleado recibe por su venta y el valor que un solo cliente debe pagar por su compra. Generalmente se requiere procesar más de un empleado o cliente sin necesidad de volver a ejecutar el algoritmo o programa; es aquí donde los ciclos juegan un papel importante, ya que permiten llevar a cabo estos procesos repetitivos. Existen tres tipos de ciclos:

Ciclo REPETIR

Permite repetir un grupo o número de líneas de código hasta que se cumpla una determinada condición. Tiene la siguiente sintaxis:

Repetir

```
Acción 1
Acción 2
Acción 3
...
Acción N
```

Hasta Que <Condición>

Ciclo MIENTRAS

Permite repetir un grupo o número de líneas de código mientras se cumpla una determinada condición. Tiene la siguiente sintaxis:

Mientras <Condición>**Hacer**

```
Acción 1
Acción 2
Acción 3
```

```

...
Acción N

```

FinMientras

La diferencia entre un ciclo REPETIR y un ciclo MIENTRAS es que en el primero se ejecutan las líneas de código o sentencias al menos una o más veces, debido a que valida la condición al final; en cambio, el segundo valida que la condición se cumpla al principio, lo cual hace que las líneas de código o sentencias se puedan ejecutar cero o más veces

Ciclo PARA

Permite repetir un grupo o número de líneas de código una cantidad determinada de veces. Tiene la siguiente sintaxis:

```

Para <Variable> ← <Valor inicial> Hasta <Valor final> Con Paso <Incremento |
Decremento> Hacer
    Acción 1
    Acción 2
    Acción 3
    ...
    Acción N

```

FinPara

¿Cuándo utilizar uno u otro ciclo?

Al momento de utilizar cualquiera de los tres, se debe tener en cuenta:

- Se conoce el número de veces que debe repetir un grupo o líneas de código; por ejemplo, 10 veces se hará uso del ciclo PARA.
- Se deben ejecutar al menos una vez el grupo o líneas de código: se hará uso del ciclo REPETIR.
- Se debe validar que una condición se cumpla inicialmente para ejecutar un grupo o líneas de código: se hará uso del ciclo MIENTRAS.

A partir del uso de los ciclos surgen dos nuevos conceptos en el mundo del desarrollo de *software*, ambos de gran utilidad: *contadores y acumuladores*.

2.5 Contadores

Es una variable cuyo valor se incrementa sucesivamente de uno en uno. Debe ser inicializada en cero y tiene la siguiente estructura:

$$C \leftarrow C + 1$$

2.6 Acumuladores

Es una variable que acumula sucesivamente valores diferentes. Debe ser inicializada en cero y tiene la siguiente estructura:

$$A \leftarrow A + Valor$$

2.7 Ejemplos resueltos

1. Realizar un algoritmo que permita visualizar la tabla de multiplicar del ocho.

Análisis del problema

Tabla 2.9: Identificación de variables Ejercicio9C2

Tipo	Variable	Descripción
Entero	mdo	Multiplicando
Entero	mdor	Multiplicador
Entero	prod	Producto

Datos de entrada: N/A

Procesos parciales:

$$mdo \leftarrow 8$$

$$mdor \leftarrow \{1, 9\}$$

$$prod \leftarrow mdo * mdor$$

Procesos totales: N/A

Datos de salida parciales: mdo, mdor, prod

Datos de salida totales: N/A

Observaciones

$mdor \leftarrow \{1, 9\}$ significa que la variable mdor va a tomar los valores de 1, 2, 3, 4, 5, 6, 7, 8 y 9 de forma consecutiva.

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio91C2

```

//Definición o declaración de las variables
Definir mdo, mdor, prod Como Entero

//Inicializar variables
mdo ← 8

//Procesos parciales
Para mdor ← 1 Hasta 9 Con Paso 1 Hacer
    prod ← mdo * mdor

    //Datos de salida parciales
    Escribir mdo, " * ", mdor, " = ", prod
FinPara

```

FinAlgoritmo

Observaciones

En la línea de código **Para mdor ← 1 Hasta 9 Con Paso 1 Hacer** se tiene, entonces, que la variable mdor comienza en 1, y se incrementa en cada pasada en 1 (Con Paso 1) hasta llegar al valor 9; por lo tanto, toma los valores de 1, 2, 3, 4, 5, 6, 7, 8, y 9, y por cada pasada se realiza el proceso $\text{prod} \leftarrow \text{mdo} * \text{mdor}$ y se visualiza información (Escribir mdo, “ * ”, mdor, “ = ”, prod); al llegar al FinPara, retorna al inicio del Para, y se repiten las líneas de código entre el Para y FinPara hasta que mdor llegue a 9, lo que permite ver la tabla de multiplicar del 8, como se aprecia en la siguiente ejecución (figura 2.18). Se puede omitir la parte **Con Paso** porque, al no indicarla, de forma automática se aplica el incremento de la variable mdor en 1. Dicha parte resulta exigida cuando el incremento es de 2 o más.

Figura 2.18: Ejecución Ejercicio9_1C2 ciclo PARA

```

*** Ejecución Iniciada. ***
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
*** Ejecución Finalizada. ***

```

Este mismo algoritmo se puede desarrollar con los ciclos REPETIR y MIENTRAS, así:

Con el ciclo REPETIR

Algoritmo Ejercicio92C2

```
//Definición o declaración de las variables
Definir mdo, mdor, prod Como Entero

//Inicializar variables
mdo ← 8
mdor ← 0

//Procesos parciales
Repetir
    mdor ← mdor + 1 //Es un contador, en este caso tiene la forma
                    //C ← C + 1
    prod ← mdo * mdor

    //Datos de salida parciales
    Escribir mdo, " * ", mdor, " = ", prod
Hasta Que mdor = 9
```

FinAlgoritmo

Figura 2.19: Ejecución Ejercicio9_2C2 ciclo REPETIR

```
*** Ejecución Iniciada. ***
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
*** Ejecución Finalizada. ***
```

Con el ciclo MIENTRAS

Algoritmo Ejercicio93C2

```
//Definición o declaración de las variables
Definir mdo, mdor, prod Como Entero

//Inicializar variables
mdo ← 8
mdor ← 0
```

```

//Procesos parciales
Mientras mdor < 9 Hacer
    mdor ← mdor + 1 //Es un contador, en este caso tiene la forma
                    //C ← C + 1
    prod ← mdo * mdor

//Datos de salida parciales
Escribir mdo, " * ", mdor, " = ", prod
FinMientras

FinAlgoritmo

```

Figura 2.20: Ejecución Ejercicio9_3C2 ciclo MIENTRAS

```

*** Ejecución Iniciada. ***
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
*** Ejecución Finalizada. ***

```

De los tres algoritmos, el óptimo es el desarrollado con el ciclo PARA porque tiene menos líneas de código y el incremento de la variable mdor se realiza automáticamente.

- Realizar un algoritmo que imprima los números pares comprendidos entre el 1 y el 20.

Análisis del problema

Tabla 2.10: Identificación de variables Ejercicio10C2

Tipo	Variable	Descripción
Entero	nroPar	Número par

Datos de entrada: N/A

Procesos parciales: N/A

Procesos totales: N/A

Datos de salida parciales: nroPar

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio10C2

```
//Definición o declaración de las variables
Definir nroPar Como Entero

Para nroPar ← 2 Hasta 20 Con Paso 2 Hacer
    Escribir nroPar
FinPara
```

FinAlgoritmo

Al ejecutar el algoritmo, se obtiene el siguiente resultado (figura 2.21).

Figura 2.21: Ejecución Ejercicio10C2

```
*** Ejecución Iniciada. ***
2
4
6
8
10
12
14
16
18
20
*** Ejecución Finalizada. ***
```

3. Realizar un algoritmo para generar las tablas de multiplicar del 1 al 10.

Análisis del problema

Tabla 2.11: Identificación de variables Ejercicio11C2

Tipo	Variable	Descripción
Entero	mdo	Multiplicando
Entero	mdor	Multiplicador
Entero	prod	Producto

Datos de entrada: N/A

Procesos parciales:

$prod \leftarrow mdo * mdor$

Procesos totales: N/A

Datos de salida parciales: mdo, mdor, prod

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio11C2

```
//Definición o declaración de las variables
Definir mdo, mdor, prod Como Entero

Para mdo ← 1 Hasta 10 Hacer

    Para mdor ← 1 Hasta 9 Hacer

        prod ← mdo * mdor
        Escribir mdo, " * ", mdor, " = ", prod

    FinPara
    Escribir ""

FinPara

FinAlgoritmo
```

Observaciones

Como se aprecia, se hace uso de dos ciclos PARA: el primero se repite 10 veces y el segundo se repite nueve veces por cada iteración del primero, gracias a lo cual se generan las tablas de multiplicar del 1 al 10. Se explica así:

La variable mdo comienza en 1, mdor comienza en 1; por lo tanto, $prod = 1 * 1 = 1$; al llegar al **FinPara** retorna a la línea de código **Para mdor ← 1 Hasta 9 Hacer**, por lo que mdor se incrementa en 1, tomando ahora el valor de 2, y $prod = 1 * 2 = 2$; este proceso se repite hasta que mdor llegue a 9; en este momento, al regresar a la línea de código **Para mdor ← 1 Hasta 9 Hacer**, ya $mdor = 9$ se salta a la línea de código debajo de **FinPara**, ejecutándose la línea de código Escribir "", lo que permite dejar una línea en blanco entre cada tabla. Al llegar a la línea de código **FinPara** se retorna a la línea **Para mdo ← 1 Hasta 10 Hacer**, por lo que la variable mdo se incrementa en 1, tomando el valor de 2, y mdor se inicia nuevamente en 1 hasta llegar a 9 nuevamente, de manera que se genera la tabla de multiplicar del 2. Todo este proceso se repite hasta que mdo sea igual a 10 y mdor sea igual a 9: en ese momento se habrán producido las tablas de multiplicar del 1 al 10, y finaliza la ejecución del algoritmo.

Al ejecutar el algoritmo, se obtiene el siguiente resultado (**figura 2.22**).

Figura 2.22: Ejecución Ejercicio11C2

1 * 1 = 1	2 * 1 = 2	3 * 1 = 3	4 * 1 = 4	5 * 1 = 5
1 * 2 = 2	2 * 2 = 4	3 * 2 = 6	4 * 2 = 8	5 * 2 = 10
1 * 3 = 3	2 * 3 = 6	3 * 3 = 9	4 * 3 = 12	5 * 3 = 15
1 * 4 = 4	2 * 4 = 8	3 * 4 = 12	4 * 4 = 16	5 * 4 = 20
1 * 5 = 5	2 * 5 = 10	3 * 5 = 15	4 * 5 = 20	5 * 5 = 25
1 * 6 = 6	2 * 6 = 12	3 * 6 = 18	4 * 6 = 24	5 * 6 = 30
1 * 7 = 7	2 * 7 = 14	3 * 7 = 21	4 * 7 = 28	5 * 7 = 35
1 * 8 = 8	2 * 8 = 16	3 * 8 = 24	4 * 8 = 32	5 * 8 = 40
1 * 9 = 9	2 * 9 = 18	3 * 9 = 27	4 * 9 = 36	5 * 9 = 45
6 * 1 = 6	7 * 1 = 7	8 * 1 = 8	9 * 1 = 9	10 * 1 = 10
6 * 2 = 12	7 * 2 = 14	8 * 2 = 16	9 * 2 = 18	10 * 2 = 20
6 * 3 = 18	7 * 3 = 21	8 * 3 = 24	9 * 3 = 27	10 * 3 = 30
6 * 4 = 24	7 * 4 = 28	8 * 4 = 32	9 * 4 = 36	10 * 4 = 40
6 * 5 = 30	7 * 5 = 35	8 * 5 = 40	9 * 5 = 45	10 * 5 = 50
6 * 6 = 36	7 * 6 = 42	8 * 6 = 48	9 * 6 = 54	10 * 6 = 60
6 * 7 = 42	7 * 7 = 49	8 * 7 = 56	9 * 7 = 63	10 * 7 = 70
6 * 8 = 48	7 * 8 = 56	8 * 8 = 64	9 * 8 = 72	10 * 8 = 80
6 * 9 = 54	7 * 9 = 63	8 * 9 = 72	9 * 9 = 81	10 * 9 = 90

12. A los empleados de la compañía ABC les otorgan una sola vez al año una bonificación de acuerdo con su salario básico y los años de antigüedad en la organización según la siguiente información:

Tiempo en años**Porcentaje**

Menos de 5 año

5 % del salario básico

5 años o más y menos de 10 años

10 % del salario básico

10 años o más y menos de 15 años

15 % del salario básico

15 años o más y menos de 20 años

20 % del salario básico

20 años o más y menos de 25 años

25 % del salario básico

25 años o más y menos de 30 años

35 % del salario básico

30 años o más

50 % del salario básico

Realizar un algoritmo para determinar la bonificación que recibe un empleado; además, se desea conocer el número de empleados de cada rango de tiempo, el porcentaje que representa cada uno respecto al total de empleados y el total pagado en bonificaciones.

Análisis del problema

En este caso, como se van a procesar varios empleados, es necesario identificar a cada uno por lo menos con su número de identificación, apellidos y nombres.

Tabla 2.12: Identificación de variables Ejercicio12C2

Tipo	Variable	Descripción
Cadena	ideEmp	Identificación del empleado
Cadena	apeEmp	Apellidos del empleado
Cadena	nomEmp	Nombres del empleado
Real	salBas	Salario básico
Entero	tieSer	Tiempo de servicios en años
Real	porBon	Porcentaje de bonificación
Real	valBon	Valor de la bonificación
Entero	conEmp1	Empleados con menos de 5 años
Entero	conEmp2	Empleados con 5 años o más y menos de 10
Entero	conEmp3	Empleados con 10 años o más y menos de 15
Entero	conEmp4	Empleados con 15 años o más y menos de 20
Entero	conEmp5	Empleados con 20 años o más y menos de 25
Entero	conEmp6	Empleados con 25 años o más y menos de 30
Entero	conEmp7	Empleados con 30 años o más
Entero	empTot	Empleados totales
Real	porEmp1	Porcentaje de empleados con menos de 5 años
Real	porEmp2	Porcentaje de empleados con 5 años o más y menos de 10
Real	porEmp3	Porcentaje de empleados con 10 años o más y menos de 15
Real	porEmp4	Porcentaje de empleados con 15 años o más y menos de 20
Real	porEmp5	Porcentaje de empleados con 20 años o más y menos de 25
Real	porEmp6	Porcentaje de empleados con 25 años o más y menos de 30
Real	porEmp7	Porcentaje de empleados con 30 años o más
Real	bonTot	Bonificaciones totales pagadas a los empleados
Carácter	opc	Opción para continuar o no ingresando datos

Datos de entrada: ideEmp, apeEmp, nomEmp, salBas, tieSer, opc

Procesos parciales:

$porBon \leftarrow [5 \mid 10 \mid 15 \mid 20 \mid 25 \mid 35 \mid 50]$

$valBon \leftarrow (sueBas * porBon)/100$

```
bonTot ← 0
bonTot ← bonTot + valBon //Proceso tipo acumulador
conEmp1 ← 0
conEmp1 ← conEmp1 + 1 //Proceso tipo contador
conEmp2 ← 0
conEmp2 ← conEmp2 + 1 //Proceso tipo contador
conEmp3 ← 0
conEmp3 ← conEmp3 + 1 //Proceso tipo contador
conEmp4 ← 0
conEmp4 ← conEmp4 + 1 //Proceso tipo contador
conEmp5 ← 0
conEmp5 ← conEmp5 + 1 //Proceso tipo contador
conEmp6 ← 0
conEmp6 ← conEmp6 + 1 //Proceso tipo contador
conEmp7 ← 0
conEmp7 ← conEmp7 + 1 //Proceso tipo contador
```

Procesos totales:

```
empTot ← conEmp1 + conEmp2 + conEmp3 + conEmp4 + conEmp5
        + conEmp6 + conEmp7
porEmp1 ← (conEmp1 * 100) / empTot
porEmp2 ← (conEmp2 * 100) / empTot
porEmp3 ← (conEmp3 * 100) / empTot
porEmp4 ← (conEmp4 * 100) / empTot
porEmp5 ← (conEmp5 * 100) / empTot
porEmp6 ← (conEmp6 * 100) / empTot
porEmp7 ← (conEmp7 * 100) / empTot
```

Datos de salida parciales: ideEmp, nomEmp, apeEmp, porBon, valBon

Datos de salida totales: conEmp1, conEmp2, conEmp3, conEmp4, conEmp5, conEmp6, conEmp7, empTot, porEmp1, porEmp2, porEmp3, porEmp4, porEmp5, porEmp6, porEmp7.

Observaciones

Como regla general, aquellas variables que almacenan datos de tipo numérico, como es el caso de la identificación del empleado, pero que no se requiera hacer cálculos matemáticos, se definen con un tipo de dato CADENA, por cuanto ocupa menos espacio de memoria y de almacenamiento físico en disco que un dato de

tipo entero. Otros ejemplos son el número de teléfono fijo, el número de teléfono móvil, entre otros.

Este ejercicio tiene procesos parciales y totales, salidas parciales y totales. Se recuerda que los procesos parciales son todos aquellos cálculos que se deben realizar por cada carga o entrada de datos; para el ejemplo en desarrollo se tiene, entonces, que a cada empleado que se ingresa (datos de entrada) se le debe determinar el porcentaje de bonificación de acuerdo con los años de servicio en la organización; con este dato se calcula la bonificación, la cual se acumula en la bonificación total, que es un proceso de tipo acumulador, por cuanto la variable `bonTot` debe ir acumulando todos los valores resultado del cálculo de la bonificación de cada empleado. Igualmente, se dan varios procesos de tipo contador, porque se debe ir contando cada empleado de acuerdo con el tiempo de servicio.

En cuanto a los procesos totales, se tiene que los empleados totales corresponden a la sumatoria de todos los empleados de cada uno de los rangos de tiempo indicados en el enunciado del problema. Estos datos solo se conocen cuando se haya finalizado el ingreso de datos (datos de entrada) al algoritmo. Como se conocen el número de empleados de cada rango de tiempo y el total de empleados, entonces se puede determinar el porcentaje para cada uno de los rangos de tiempo (estos procesos se realizan una sola vez).

Desarrollo del algoritmo en PSeInt

Para desarrollar este algoritmo se deben seguir estos pasos:

1. Definir o declarar las variables.
2. Inicializar las variables de tipo contador y acumulador.
3. Hacer uso de un ciclo REPETIR para poder ingresar y procesar los datos de varios empleados en una misma ejecución.
4. Ingresar los datos de entrada.
5. Validar con el condicional SI el tiempo de servicio para asignar el respectivo porcentaje e incrementar la variable de tipo contador.
6. Realizar los procesos para calcular y acumular la bonificación de cada empleado ingresado.
7. Visualizar los datos de salida parciales.

8. Hacer la pregunta para indicarle al usuario si desea ingresar más datos y capturar su respuesta.
9. Realizar los procesos totales.
10. Mostrar los datos de salida totales.

Algoritmo Ejercicio12C2

```
//1. Definición o declaración de las variables
Definir ideEmp, apeEmp, nomEmp Como Cadena
Definir salBas, porBon, valBon, bonTot, porEmp1, porEmp2, porEmp3,
    porEmp4, porEmp5, porEmp6, porEmp7 Como Real
Definir tieSer, conEmp1, conEmp2, conEmp3, conEmp4, conEmp5, conEmp6,
    conEmp7, empTot Como Entero
Definir opc Como Caracter

//2. Inicialización de variables
bonTot ← 0
conEmp1 ← 0
conEmp2 ← 0
conEmp3 ← 0
conEmp4 ← 0
conEmp5 ← 0
conEmp6 ← 0
conEmp7 ← 0

//3. Ciclo repetir
Repetir

    //4. Datos de entrada
    Borrar Pantalla
    Escribir Sin Saltar "Identificación del empleado: "
    Leer ideEmp
    Escribir Sin Saltar "Apellidos del empleado : "
    Leer apeEmp
    Escribir Sin Saltar "Nombres del empleado : "
    Leer nomEmp
    Escribir Sin Saltar "Salario básico : "
    Leer salBas
    Escribir Sin Saltar "Años de servicio : "
    Leer tieSer

//5. Validación del tiempo de servicio
//6. Procesos parciales
Si tieSer < 5 Entonces
    porBon ← 5
    conEmp1 ← conEmp1 + 1
```

```

SiNo      Si tieSer < 10 Entonces
          porBon ← 10
          conEmp2 ← conEmp2 + 1
SiNo
          Si tieSer < 15 Entonces
            porBon ← 15
            conEmp3 ← conEmp3 + 1
SiNo
          Si tieSer < 20 Entonces
            porBon ← 20
            conEmp4 ← conEmp4 + 1
SiNo
          Si tieSer < 25 Entonces
            porBon ← 25
            conEmp5 ← conEmp5 + 1
SiNo
          Si tieSer < 30 Entonces
            porBon ← 35
            conEmp6 ← conEmp6 + 1
SiNo
          porBon ← 50
          conEmp7 ← conEmp7 + 1
          FinSi
        FinSi
      FinSi
    FinSi
  FinSi
FinSi

valBon ← (salBas * porBon) / 100
bonTot ← bonTot + valBon

//7. Datos de salida parciales
Escribir "Identificación del empleado: ", ideEmp
Escribir "Nombres del empleado :", nomEmp
Escribir "Apellidos del empleado : ", apeEmp
Escribir "Porcentaje de bonificación : ", porBon
Escribir "Valor bonificación : ", valBon

//8. Pregunta al usuario
Escribir Sin Saltar "¿Desea ingresar nuevos datos [S/N]? "
Leer opc

Hasta Que Mayusculas(opc) = "N"

//9. Procesos totales
empTot ← conEmp1 + conEmp2 + conEmp3 + conEmp4 + conEmp5 + conEmp6 +
conEmp7

```

```
porEmp1 ← (conEmp1 * 100) / empTot
porEmp2 ← (conEmp2 * 100) / empTot
porEmp3 ← (conEmp3 * 100) / empTot
porEmp4 ← (conEmp4 * 100) / empTot
porEmp5 ← (conEmp5 * 100) / empTot
porEmp6 ← (conEmp6 * 100) / empTot
porEmp7 ← (conEmp7 * 100) / empTot

//10. Datos de salida totales
Borrar Pantalla
Escribir "Empleados con menos de 5 años : ", conEmp1, " ",
        Redon(porEmp1 * 100) / 100, "%"
Escribir "Empleados con 5 años o más y menos de 10 : ", conEmp2, " ",
        Redon(porEmp2 * 100) / 100, "%"
Escribir "Empleados con 10 años o más y menos de 15: ", conEmp3, " ",
        Redon(porEmp3 * 100) / 100, "%"
Escribir "Empleados con 15 años o más y menos de 20: ", conEmp4, " ",
        Redon(porEmp4 * 100) / 100, "%"
Escribir "Empleados con 20 años o más y menos de 25: ", conEmp5, " ",
        Redon(porEmp5 * 100) / 100, "%"
Escribir "Empleados con 25 años o más y menos de 30: ", conEmp6, " ",
        Redon(porEmp6 * 100) / 100, "%"
Escribir "Empleados con 30 años o más : ", conEmp7, " ",
        Redon(porEmp7 * 100) / 100, "%"
Escribir "Total empleados : ", empTot
Escribir "Bonificaciones total pagadas : ", bonTot
```

FinAlgoritmo

Observaciones

En la definición o declaración de las variables se adicionó `opc`, que se usa para capturar la respuesta del usuario a fin de saber si va a seguir ingresando datos o no.

Se inicializaron en 0 todas las variables de tipo contador y acumulador. Se utilizó un ciclo REPETIR para que se puedan ingresar los datos de varios empleados en una sola ejecución, siempre y cuando se responda con S a la pregunta **¿Desea ingresar nuevos datos [S/N]?**

Se aplicó el condicional SI para validar el tiempo de servicio del empleado y asignar el porcentaje de bonificación e incrementar en 1 cada variable de tipo contador. Nótese que no es necesario preguntar si el tiempo de servicio es mayor o igual a

5 y menor o igual a 10 (Si tieSer \geq 5) Y (tieSer \leq 10), debido a que, cuando se preguntó Si tieSer $<$ 5 en la parte del SiNo, se está negando; es decir, no es menor que 5, lo que indica que es igual o mayor que 5. Esto mismo sucede con los demás condicionales.

4. Un almacén hace descuento a sus clientes de acuerdo con la siguiente información:

Compras mayores o iguales a 100000 y menores de 200000 tienen descuento del 10%.
 Compras mayores o iguales a 200000 y menores de 300000 tienen descuento del 15%.
 Compras mayores o iguales a 300000 y menores de 400000 tienen descuento del 20%.
 Compras mayores o iguales a 400000 y menores de 500000 tienen descuento del 25%.
 Compras mayores o iguales a 500000 tienen descuento del 30%.

Realizar un algoritmo para determinar el valor que un cliente debe pagar por su compra; adicionalmente, se desea conocer cuántos clientes obtuvieron descuentos del 10%, 15%, 20%, 25% y 30%, y cuántos no tuvieron descuento; asimismo, el monto total de las compras sin descuento y el monto total de las compras con descuento.

Análisis del problema

Como en el caso anterior, aquí es necesario procesar varios clientes, por lo que es importante identificar a cada uno al menos con su número de identificación, apellidos y nombres.

Tabla 2.13: Identificación de variables Ejercicio13C2

Tipo	Variable	Descripción
Cadena	ideCli	Identificación del cliente
Cadena	apeCli	Apellidos del cliente
Cadena	nomCli	Nombres del cliente
Real	valCom	Valor de la compra
Real	porDes	Porcentaje de descuento
Real	valDes	Valor descontado
Real	valPag	Valor por pagar
Entero	conCli10	Contador para cliente con descuento del 10%
Entero	conCli15	Contador para cliente con descuento del 15%

Tipo	Variable	Descripción
Entero	conCli20	Contador para cliente con descuento del 20 %
Entero	conCli25	Contador para cliente con descuento del 25 %
Entero	conCli30	Contador para cliente con descuento del 30 %
Entero	conCli0	Contador para cliente sin descuentos
Real	totConDes	Total de las compras con descuentos
Real	totSinDes	Total de las compras sin descuentos
Carácter	opc	Opción para continuar o no ingresando datos

Datos de entrada: ideCli, apeCli, nomCli, valCom, opc.

Procesos parciales:

$porDes \leftarrow [0 \mid 10 \mid 15 \mid 20 \mid 25 \mid 30]$

$valDes \leftarrow (valCom * porDes) / 100$

$valPag \leftarrow valCom - valDes$

$totConDes \leftarrow 0$

$totConDes \leftarrow totConDes + valPag$

$totSinDes \leftarrow 0$

$totSinDes \leftarrow totSinDes + valCom$

$conCli10 \leftarrow 0$

$conCli10 \leftarrow conCli10 + 1$

$conCli15 \leftarrow 0$

$conCli15 \leftarrow conCli15 + 1$

$conCli20 \leftarrow 0$

$conCli20 \leftarrow conCli20 + 1$

$conCli25 \leftarrow 0$

$conCli25 \leftarrow conCli25 + 1$

$conCli30 \leftarrow 0$

$conCli30 \leftarrow conCli30 + 1$

$conCli0 \leftarrow 0$

$conCli0 \leftarrow conCli0 + 1$

Procesos totales: N/A

Datos de salida parciales: ideCli, nomCli, apeCli, porDes, valDes, valPag

Datos de salida totales: conCli10, conCli15, conCli20, conCli25, conCli30, conCli0, totConDes, totSinDes

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio13C2

```

//1. Definición o declaración de las variables
Definir ideCli, apeCli, nomCli Como Cadena
Definir valCom, porDes, valDes, valPag Como Real
Definir conCli10, conCli15, conCli20, conCli25, conCli30, conCli0,
    totConDes, totSinDes Como Entero
Definir opc Como Caracter

//2. Inicialización de variables
conCli10 ← 0
conCli15 ← 0
conCli20 ← 0
conCli25 ← 0
conCli30 ← 0
conCli0 ← 0
totConDes ← 0
totSinDes ← 0

//3. Ciclo repetir
Repetir

    //4. Datos de entrada
    Borrar Pantalla
    Escribir Sin Saltar "Identificación del cliente: "
    Leer ideCli
    Escribir Sin Saltar "Apellidos del cliente : "
    Leer apeCli
    Escribir Sin Saltar "Nombres del cliente : "
    Leer nomCli
    Escribir Sin Saltar "Valor de la compra : "
    Leer valCom

//5. Validación valor compra
//6. Procesos parciales
Si valCom < 100000 Entonces
    porDes ← 0
    conCli0 ← conCli0 + 1
SiNo
    Si valCom < 200000 Entonces
        porDes ← 10
        conCli10 ← conCli10 + 1
    SiNo
        Si valCom < 300000 Entonces
            porDes ← 15
            conCli15 ← conCli15 + 1
        SiNo

```

```
        Si valCom < 400000 Entonces
            porDes ← 20
            conCli20 ← conCli20 + 1
        SiNo
            Si valCom < 500000 Entonces
                porDes ← 25
                conCli25 ← conCli25 + 1
            SiNo
                porDes ← 30
                conCli30 ← conCli30 + 1
            FinSi
        FinSi
    FinSi
    FinSi
    FinSi

    valDes ← (valCom * porDes) / 100
    valPag ← valCom - valDes
    totConDes ← totConDes + valPag
    totSinDes ← totSinDes + valCom

    //7. Datos de salida parciales
    Escribir "Identificación del cliente: ", ideCli
    Escribir "Apellidos del cliente : ", apeCli
    Escribir "Nombres del cliente : ", nomCli
    Escribir "Porcentaje de descuento : ", porDes
    Escribir "Valor descontado : ", valDes
    Escribir "Valor a pagar : ", valPag

    //8. Pregunta al usuario
    Escribir sin saltar "¿Desea ingresar un nuevo cliente [S/N]? "
    Leer opc

    Hasta Que Mayusculas(opc) = "N"

    //9. Datos de salida totales
    Borrar Pantalla
    Escribir "Clientes sin descuentos : ", conCli0
    Escribir "Clientes con descuentos del 10% : ", conCli10
    Escribir "Clientes con descuentos del 15% : ", conCli15
    Escribir "Clientes con descuentos del 20% : ", conCli20
    Escribir "Clientes con descuentos del 25% : ", conCli25
    Escribir "Clientes con descuentos del 30% : ", conCli30
    Escribir "Total de las compras sin descuento : ", totSinDes
    Escribir "Total de las compras con descuento: ", totConDes
```

FinAlgoritmo

5. Realizar un algoritmo que lea dos valores. De acuerdo con el menú de opciones siguiente, desarrollar la operación indicada:

1. Sumar
2. Restar
3. Multiplicar
4. Dividir
5. Potencia

Diseñar el algoritmo de tal manera que el usuario pueda hacer varias operaciones con los valores ingresados y que pueda ingresar otros nuevos.

Análisis del problema

Tabla 2.14: Identificación de variables Ejercicio14C2

Tipo	Variable	Descripción
Real	val1	Valor 1
Real	val2	Valor 2
Real	res	Resultado de la operación
Entero	opc	Opción seleccionada por el usuario
Carácter	sig1	Opción para realizar otra operación
Carácter	sig2	Opción para ingresar nuevos valores

Datos de entrada: val1, val2, opc, sig1, sig2.

Procesos parciales:

$res \leftarrow [val1 + val2 \mid val1 - val2 \mid val1 * val2 \mid val1/val2 \mid val1^{val2}]$

Procesos totales: N/A

Datos de salida parciales:

Res, ["ERROR. No se puede dividir entre 0" | "La opción seleccionada no es válida"]

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Algoritmo Ejercicio14C2

```
//Definición o declaración de las variables
Definir val1, val2, res Como Real
```

Definir opc Como Entero

Definir sig1, sig2 Como Caracter

Repetir

//Datos de entrada

Borrar Pantalla

Escribir Sin Saltar "Valor 1: "

Leer val1

Escribir Sin Saltar "Valor 2: "

Leer val2

Repetir

Escribir ""

Escribir "MENÚ DE OPCIONES"

Escribir ""

Escribir "1. Suma"

Escribir "2. Resta"

Escribir "3. Multiplicar"

Escribir "4. Dividir"

Escribir "5. Potencia"

Escribir Sin Saltar "Seleccione una opción: "

Leer opc

//Procesos parciales y salida parciales

Segun opc Hacer

1: res ← val1 + val2

Escribir val1, " + ", val2, " = ", res

2: Res ← val1 - val2

Escribir val1, " - ", val2, " = ", res

3: res ← val1 * val2

Escribir val1, " * ", val2, " = ", res

4: Si val2 <> 0 Entonces

res ← val1 / val2

Escribir val1, " / ", val2, " = ", Redon(res*100)/ 100

SiNo

Escribir "ERROR. No se puede dividir entre 0"

FinSi

5: res ← val1 ^ val2

Escribir val1, " ^ ", val2, " = ", res

De Otro Modo:

Escribir "La opción seleccionada no es válida"

FinSegun

Escribir ""

Escribir Sin Saltar "¿Desea realizar una nueva operación [S/N]? "

Leer sig1

```
Hasta Que Mayusculas(sig1) = "N"

Escribir ""
Escribir Sin Saltar "¿Desea ingresar nuevos valores [S/N]? "
Leer sig2

Hasta Que Mayusculas(sig2) = "N"
```

FinAlgoritmo

6. Se requiere un algoritmo que sirva para calcular la nota definitiva de los estudiantes, teniendo en cuenta que un mismo estudiante puede cursar varias materias y que cada una tiene tres notas (30 %, 30 % y 40 %, respectivamente). Adicionalmente, se desea conocer el promedio de cada estudiante, el promedio del grupo y el estudiante con mejor promedio.

Análisis del problema

Tabla 2.15: Identificación de variables Ejercicio15C2

Tipo	Variable	Descripción
Cadena	ideEst	Identificación del estudiante
Cadena	apeEst	Apellidos del estudiante
Cadena	nomEst	Nombres del estudiante
Cadena	codMat	Código de la materia
Cadena	nomMat	Nombre de la materia
Real	not1	Nota 1
Real	not2	Nota 2
Real	not3	Nota 3
Real	notDef	Nota definitiva
Entero	conMat	Contador de materia para un estudiante
Real	sumNotDef	Sumatoria de notas definitivas de un estudiante
Real	proEst	Promedio del estudiante
Entero	conEst	Contador de estudiantes
Real	sumNotPro	Sumatoria de las notas promedio de todos los estudiantes
Real	proGru	Promedio del grupo
Real	proEstMay	Promedio mayor

Tipo	Variable	Descripción
Cadena	ideEstMay	Identificación del estudiante con mejor promedio
Cadena	apeEstMay	Apellidos del estudiante con mejor promedio
Cadena	nomEstMay	Nombres del estudiante con mejor promedio
Carácter	opc	Opción para continuar ingresando más materias
Carácter	seg	Opción para ingresar un nuevo estudiante
Cadena	ideEstMay	Identificación del estudiante con mejor promedio
Cadena	apeEstMay	Apellidos del estudiante con mejor promedio
Cadena	nomEstMay	Nombres del estudiante con mejor promedio
Carácter	opc	Opción para continuar ingresando más materias
Carácter	seg	Opción para ingresar un nuevo estudiante

Datos de entrada: ideEst, apeEst, nomEst, not1, not2, not3, opc, seg.

Procesos parciales:

$$notDef \leftarrow ((not1 * 30)/100) + ((not2 * 30)/100) + ((not3 * 40)/100)$$

$$conMat \leftarrow 0$$

$$conMat \leftarrow conMat + 1$$

$$sumNotDef \leftarrow 0$$

$$sumNotDef \leftarrow sumNotDef + notDef$$

$$proEst \leftarrow sumNotDef / conMat$$

$$conEst \leftarrow 0$$

$$conEst \leftarrow conEst + 1$$

$$sumNotPro \leftarrow 0$$

$$sumNotPro \leftarrow sumNotPro + proEst$$

$$ideEstMay \leftarrow ""$$

$$nomEstMay \leftarrow ""$$

$$apeEstMay \leftarrow ""$$

$$proEstMay \leftarrow 0$$

$$ideEstMay \leftarrow ideEst$$

$$nomEstMay \leftarrow nomEst$$

$$apeEstMay \leftarrow apeEst$$

$$proEstMay \leftarrow proEst$$

Procesos totales:

$$proGru \leftarrow sumNotPro / conEst$$

Datos de salida parciales: notDef, proEst

Datos de salida totales: conEst, proGru, ideEstMay, nomEstMay, apeEstMay, proEstMay

Desarrollo del algoritmo en PSeInt

Los pasos que se deben seguir para codificar el algoritmo son:

1. Definir o declarar las variables.
2. Inicializar las variables globales o totales.
3. Ciclo para ingresar nuevo estudiante.
4. Capturar o leer los datos del estudiante.
5. Inicializar el contador de materias y sumatoria de notas definitivas en cero.
6. Ciclo para ingresar las materias y notas del estudiante.
7. Capturar o leer las materias con sus respectivas notas.
8. Calcular la nota definitiva de la materia ingresada, contarla y acumularla.
9. Visualizar la nota definitiva de la materia ingresada y preguntar si se desea ingresar una nueva materia.
10. Calcular el promedio del estudiante, acumularlo y contar al estudiante.
11. Determinar el estudiante con mejor promedio.
12. Visualizar el promedio del estudiante y preguntar para ingresar uno nuevo.
13. Calcular el promedio del grupo.
14. Visualizar el número de estudiantes, promedio del grupo y el estudiante con mejor promedio.

Algoritmo Ejercicio15C2

```
//Definición o declaración de variables
Definir ideEst, nomEst, apeEst, ideEstMay, nomEstMay, apeEstMay, codMat
    Como Cadena
Definir opc, seg Como Caracter
Definir not1, not2, not3, notDef, sumNotDef, proEst, sumNotPro, proGru,
    proEstMay Como Real
Definir conMat, conEst Como Entero

//Inicializar variables
conEst ← 0
sumNotPro ← 0
ideEstMay ← ""
```

```
nomEstMay ← ""
apeEstMay ← ""
proEstMay ← 0

//Ciclo para ingresar nuevo estudiante
Repetir

    //Capturar los datos del estudiante
    Borrar Pantalla
    Escribir Sin Saltar "Identificación del estudiante : "
    Leer ideEst
    Escribir Sin Saltar "Apellidos del estudiante : "
    Leer apeEst
    Escribir Sin Saltar "Nombres del estudiante : "
    Leer nomEst

    //Inicializar el contador de materias y sumatoria de notas
    //definitivas en cero
    conMat ← 0
    sumNotDef ← 0

    //Ciclo para ingresar las materias y notas de un estudiante
    Repetir

        //Capturar o leer las materias con sus respectivas notas
        Escribir Sin Saltar "Código de la materia : "
        Leer codMat
        Escribir Sin Saltar "Nota 1 30% : "
        Leer not1
        Escribir Sin Saltar "Nota 2 30% : "
        Leer not2
        Escribir Sin Saltar "Nota 3 40% : "
        Leer not3

        //Calcular nota definitiva, contarla y acumularla
        notDef ← ((not1 * 30/100) + ((not2 * 30)/100)+ (not3 *
            40) / 100)
        conMat ← conMat + 1
        sumNotDef ← sumNotDef + notDef

        //Visualizar la nota definitiva de la materia ingresada
        Escribir "Nota definitiva: ", Redon(notDef * 100)/100
        Escribir Sin Saltar "¿Ingresa una nueva materia [S/N]? "
        Leer opc

Hasta Que Mayusculas(opc) = "N"

//Calcular el promedio del estudiante, acumularlo y contar el
//estudiante
```

```

proEst ← sumNotDef / conMat
sumNotPro ← sumNotPro + proEst
conEst ← conEst + 1

//Determinar el estudiante con mejor promedio
Si proEst > proEstMay Entonces
    ideEstMay ← ideEst
    apeEstMay ← apeEst
    nomEstMay ← nomEst
    proEstMay ← proEst

FinSi

//Visualizar el promedio del estudiante, preguntar para ingresar
//uno nuevo
Escribir ""
Escribir "Promedio del estudiante: ", Redon(proEst * 100)/100
Escribir ""
Escribir Sin Saltar "¿Desea ingresar un nuevo estudiante [S/N]? "
Leer seg

Hasta Que Mayusculas(seg) = "N"

//Calcular el promedio del grupo
proGru ← sumNotPro / conEst

//Visualizar el número de estudiante, promedio del grupo y estudiante
//con mejor promedio
Escribir ""
Escribir "Número de estudiantes: ", conEst
Escribir "Promedio del grupo : ", Redon(proGru * 100) / 100
Escribir ""
Escribir "Estudiante con mejor promedio"
Escribir ideEstMay, " ", apeEstMay, " ", nomEstMay, " ",
    Redon(proEstMay * 100) / 100

FinAlgoritmo

```

2.8 Ejercicios propuestos

1. Realizar un algoritmo que permita ingresar N números enteros y determinar el promedio de estos; además, se desea conocer cuántos son pares y cuántos impares.
2. Realizar un algoritmo para ingresar N números enteros, y determinar cuántos de estos son primos y cuántos no.

3. Una entidad otorga préstamos a sus clientes a una tasa de interés N mensual. Se requiere un algoritmo que determine el valor que debe pagar mensualmente un cliente por su préstamo; además, se desea conocer el número de clientes a quienes les otorgó préstamos, el total de dinero prestado, el total de dinero en intereses recibido en un mes y el total de capital luego de cancelados todos los intereses de los préstamos en un mes.
4. Una tienda está interesada en desarrollar un algoritmo que le permita registrar los datos del cliente, la fecha de compra, la descripción de cada uno de los productos adquiridos, la cantidad, el valor unitario y el porcentaje del IVA. Se desea conocer el valor total de la compra (un cliente puede comprar uno o más productos en una misma compra), además de cuántos clientes se atendieron y el valor total de las compras.
5. Los empleados de una fábrica trabajan en dos turnos: diurno y nocturno. Se desea calcular el jornal diario de acuerdo con los siguientes puntos:

La tarifa de las horas diurnas es de 5 dólares.

La tarifa de las horas nocturnas es de 8 dólares.

En caso de ser domingo, la tarifa se incrementará en 2 dólares para el turno diurno y en 3 dólares para el nocturno. Además, se desea conocer el total pagado a los empleados y el número de empleados procesados.

Capítulo 3

PROGRAMACIÓN MODULAR



Los algoritmos descritos hasta el momento se han desarrollado de forma secuencial, es decir, todo el código se escribe entre las sentencias **Algoritmo** y **FinAlgoritmo**. Así que se efectúa toda la lógica del problema en un solo bloque de código (instrucciones). Pero, cuando el problema es complejo y requiere de muchas líneas de código, se torna inmanejable, difícil de leer y de mantener. Con el fin de solucionar este tipo de inconvenientes, surge el concepto de la **MODULARIDAD**, cuyo fin es dividir el problema en pequeños problemas o módulos conocidos como funciones y procedimientos, los cuales se diseñan para hacer solo una tarea específica.

EcuRed (2019) define la *programación modular* como “un paradigma de programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable” (párr. 3); Barber y Ferrís (s. f.) la definen como “la descomposición de un programa en trozos más pequeños denominados módulos o subprogramas, en el que cada uno de ellos se encargará de llevar a cabo una tarea concreta y bien definida, y se agrupará según su funcionalidad” (párr. 1).

Algunas de las ventajas de la programación modular, según Nieva (2016), son:

- Reducir la complejidad del problema.
- Reducir el tamaño del problema.
- Favorecer el entendimiento del problema.
- Facilitar la cooperación entre programadores.
- Reutilizar código.
- Facilitar la lectura del código.
- Plantear una solución completa del problema, para luego profundizar en los detalles.
- La depuración es más fácil de realizar, ya que primero se corrigen errores en los módulos de nivel inferior (párr. 3).

Así que esta técnica facilita la resolución de problemas complejos al escribir algoritmos y programas más entendibles y mantenibles, al permitir la reutilización de código; es decir, se puede invocar una función o procedimiento las veces que sean necesarias con diferentes parámetros, gracias a lo cual se ahorran muchas líneas de código, debido a que se escribe una función o procedimiento una sola vez.

Al momento de realizar un algoritmo bajo el paradigma de la programación modular, se deben tener en cuenta las siguientes características:

- Una función solo hace una única actividad, tarea o proceso.
- Los datos de entrada se capturan a través de funciones para verificar que solo se permita el ingreso de datos (valores) válidos y no se acepten datos en blanco.
- Solo se hace uso de parámetros por referencias en los casos en que son estrictamente necesarios, y se les da prioridad a los parámetros por valor (el uso de los parámetros por referencia y por valor se explica más adelante).
- En el algoritmo principal no se realizan validaciones con el condicional SI ni con SEGÚN, sino a través de funciones.
- En el algoritmo principal no hay procesos (fórmulas matemáticas): estos se desarrollan a través de funciones.
- Las funciones se deben diseñar lo más generales posible, para ser utilizadas en un algoritmo varias veces (reutilización de código) mediante el llamado con diferentes parámetros.
- La salida de datos se hace a través de funciones.

3.1 Funciones

Lemonaki (2021) indica que “una función no es más que un bloque de código aislado que lleva a cabo una tarea específica” (párr. 3). Para la Universidad Nacional del Rosario (s. f.), “una función es un subalgoritmo que recibiendo o no datos devuelve un único resultado” (p. 3). Por su parte, Manuais (2020) la define como “la forma de agrupar expresiones y sentencias (algoritmos) que realicen determinadas acciones, pero que éstas solo se ejecuten cuando son llamadas” (párr. 1).

De acuerdo con las anteriores definiciones, una función puede tener cero, uno o más argumentos, con los cuales ejecuta determinado proceso para devolver un único resultado al algoritmo principal. Tiene la siguiente sintaxis:

```

Funcion variableDeRetorno ← NombreFuncion(argumento1, argumento2, ... argumento N)
    Acción 1
    Acción 2
    ...
    Acción N
FinFuncion

```

3.2 Procedimientos

Un procedimiento se diferencia de una función en que carece de variable de retorno (no devuelve ningún valor). Puede tener cero, uno o más argumentos, con los cuales realiza determinados procesos o acciones. Tiene la siguiente sintaxis:

```
Funcion NombreFuncion(argumento1, argumento2, ... Argumento N)
    Acción 1
    Acción 2
    ...
    Acción N
FinFuncion
```

3.3 Parámetros

Cuando desde el algoritmo principal se llama a una función o procedimiento, se establece una correspondencia entre los parámetros actuales (los valores enviados desde el algoritmo principal) y los formales (los parámetros que reciben los valores enviados desde el algoritmo principal en la función o procedimiento) en el mismo orden en que fueron enviados.

En pseudocódigo, como en los lenguajes de programación, se dispone de dos tipos de parámetros: por valor y por referencia.

Parámetros por valor

Los parámetros formales reciben una copia de los valores de los parámetros actuales; por tanto, los cambios que se produzcan en dichos parámetros por efecto de los procesos que se ejecuten en una función o procedimiento no afectan a los valores de los parámetros actuales. Para indicar que un parámetro es por valor se utiliza la siguiente sintaxis:

```
Funcion Nombre_Funcion(argumento 1 Por Valor, argumento 2 Por Valor, ...
                        argumento N Por Valor)
```

Se puede omitir **Por Valor**, entendiéndose que, si no se especifica, se asume como un parámetro por valor:

```
Funcion Nombre_Funcion(argumento 1, argumento 2, ... argumento N)
```


Parámetros por referencia

Los parámetros formales reciben la dirección de memoria de los parámetros actuales; por tanto, los cambios que se produzcan en los parámetros formales por efecto de los procesos que se ejecuten en una función o procedimiento afectan a los valores de los parámetros actuales; es decir, los cambios aplicados en las funciones y procedimientos a los formales se ven reflejados en los actuales.

Para indicar que un parámetro es por referencia, se utiliza la siguiente sintaxis:

```
Funcion Nombre_Funcion(argumento 1 Por Referencia, argumento 2 Por Referencia, ...
                        argumento N Por Referencia)
```

En este caso sí es obligatorio indicar **Por Referencia**.

3.4 Variables globales y locales

Una variable es global cuando se declara o define en el algoritmo principal y hay acceso a ella en todo el algoritmo. Por su parte, las variables locales son las definidas o declaradas en las funciones o procedimientos; estas solo permanecen en memoria mientras se está en una determinada función o procedimiento, y, al salir de estos, se eliminan de la memoria.

3.5 Ejemplos resueltos

1. En este primer ejercicio se va a explicar el concepto de los *parámetros por valor y por referencia* de forma práctica. Se van a leer dos valores enteros y se creará un procedimiento que intercambie dichos valores; por ejemplo, si $a = 10$ y $b = 5$, entonces en el procedimiento se van a intercambiar asignándole a la variable **a** el valor de la variable **b** y a la variable **b** el valor de la variable **a**: $a = 5$ y $b = 10$.

Análisis del problema

Tabla 3.1: Identificación de variables Ejercicio1C3

Tipo	Variable	Descripción
Entero	val1	Valor 1
Entero	val2	Valor 2
Entero	aux	Auxiliar

Datos de entrada: val1, val2

Procesos parciales:

$aux \leftarrow val1$

$val1 \leftarrow val2$

$val2 \leftarrow aux$

Procesos totales: N/A

Datos de salida parciales: val1, val2

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Se creará una función para leer o ingresar un valor entero, una para valores reales, una para datos de tipo cadena y una para datos de tipo carácter, de tal forma que se seguirán usando en el resto de los algoritmos sin tener que volver a escribirlas; asimismo, una función para intercambiar los valores y una para mostrar información.

```
//Algoritmo principal
Algoritmo Ejercicio1C3

    //Definir o declarar las variables
    Definir val1, val2, aux Como Enteros

    //Datos de entrada
    val1 ← LeerValorEntero("Digite valor 1: ", 1, 100)
    val2 ← LeerValorEntero("Digite valor 2: ", 1, 100)

    //Visualizar los datos ingresados
    MostrarDatos("EN EL ALGORITMO PRINCIPAL ANTES DE LLAMAR A LA
                FUNCIÓN IntercambiarValores", "Valor 1 = ", "Valor 2 = ",
                val1, val2)

    //Llamar o invocar a la función IntercambiarValores
    IntercambiarValores(val1, val2)

    //Visualizar los datos ingresados
    MostrarDatos("EN EL ALGORITMO PRINCIPAL DESPUÉS DE LLAMAR A LA
                FUNCIÓN IntercambiarValores", "Valor 1 = ", "Valor 2 = ",
                val1, val2)

FinAlgoritmo

//Funciones para datos de entrada

//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs)
    Definir val Como Entero
```

```

Repetir
    Escribir ""
    Escribir Sin saltar msg
    Leer Val
    Si (Val < vi) O (Val > vs) Entonces
        Escribir ""
        Escribir "*** ERROR *** debe ser >= ", vi, " y <= ", vs
        Escribir "Presione una tecla para continuar..."
        Esperar Tecla

    FinSi
Hasta Que (Val >= vi) & (Val <= vs)
FinFuncion

//Función para leer o capturar un valor real
Funcion val ← LeerValorReal(msg, vi, vs)

Definir val Como Real
Repetir
    Escribir ""
    Escribir Sin saltar msg
    Leer Val
    Si (Val < vi) O (Val > vs) Entonces
        Escribir ""
        Escribir "*** ERROR *** debe ser >= ", vi, " y <= ", vs
        Escribir "Presione una tecla para continuar..."
        Esperar Tecla

    FinSi
Hasta Que (Val >= vi) & (Val <= vs)
FinFuncion

//Función para leer una cadena como nombres, direcciones, etc.
Funcion cad ← LeerCadena(msg)

Definir cad Como Cadena
Repetir
    Escribir ""
    Escribir Sin Saltar msg
    Leer cad
    Si cad = Entonces
        Escribir ""
        Escribir "*** ERROR *** no se permiten datos en blanco..."
        Escribir "Presione una tecla para continuar..."
        Esperar Tecla

```

```
        FinSi

        Hasta Que cad <> ""

FinFuncion

//Función para leer un carácter como S, N, F, M, etc.
Funcion car ← LeerCaracter(msg, c1, c2)
    Definir car Como Caracter
    Repetir
        Escribir ""
        Escribir Sin Saltar msg
        Leer car
        Si (Mayusculas(car) <> Mayusculas(c1)) & (Mayusculas(car) <>
            Mayusculas(c2)) Entonces
            Escribir ""
            Escribir "*** ERROR *** debe ingresar ", Mayusculas(c1), "
                o ", Mayusculas(c2), "...
            Escribir "Presione una tecla para continuar..."
            Esperar Tecla
        FinSi

        Hasta Que (Mayusculas(car) = Mayusculas(c1)) | (Mayusculas(car) =
            Mayusculas(c2))

FinFuncion

//Funciones adicionales requeridas según los procesos

Funcion IntercambiarValores(a Por Valor, b Por Valor) //Función para
intercambiar los valores de A y B
    Definir aux Como Entero
    MostrarDatos("EN LA FUNCIÓN IntercambiarValores ANTES DEL
        INTERCAMBIO ", "Valor de a = ", "Valor de b = ", a, b)
    aux ← a
    a ← b
    b ← aux
    MostrarDatos("EN LA FUNCIÓN IntercambiarValores DESPUÉS DEL
        INTERCAMBIO ", "Valor de a = ", "Valor de b = ", a, b)

FinFuncion

Funcion MostrarDatos(msg1, msg2, msg3, a, b) //Función para mostrar el
contenido de las variables A y B
    Escribir ""
    Escribir msg1
    Escribir msg2, a
    Escribir msg3, b
FinFuncion
```

Explicación

Se abre PSeInt y se le da el nombre al algoritmo del Ejercicio1C3. Queda como:

```
Algoritmo Ejercicio1C3
```

```
FinAlgoritmo
```

Este es el algoritmo principal, debajo del cual se crean las funciones para los datos de entrada y las funciones adicionales requeridas.

Funcion val ← **LeerValorEntero(msg, vi, vs)**. En esta primera línea están la palabra reservada **Funcion**, la variable de retorno **val**, la asignación ←, el nombre de la función **LeerValorEntero** y, entre paréntesis, los argumentos **msg**, **vi**, **vs**.

msg es un mensaje que se le muestra al usuario para indicarle lo que debe hacer; por ejemplo: “Digite valor 1: ” (nótese que debe ir entre comillas dobles).

vi es el valor inferior o menor que puede digitar el usuario; por ejemplo, 1.

vs es el valor superior o mayor que puede digitar el usuario; por ejemplo, 100.

Los valores **vi** y **vs** pueden ser cualquier valor entero: la condición es que **vi** debe ser menor que **vs**.

Como en el algoritmo principal, en las funciones y procedimientos se deben definir o declarar las variables requeridas; en este caso, **Definir val Como Entero**, y seguidamente se hace uso de un ciclo **REPETIR... Hasta que**. El ciclo permite repetir la entrada del valor entero hasta que sea mayor o igual al valor inferior y menor o igual al valor superior.

Escribir "". Esta línea de código deja una línea en blanco.

Escribir Sin Saltar msg visualiza el mensaje enviado desde el algoritmo principal; para el ejemplo, **Digite valor 1:**

Leer val permite que el usuario ingrese un valor.

Si (val < vi) O (val > vs) Entonces. Este condicional valida que el valor ingresado por el usuario no sea menor que el valor inferior ni mayor que el valor superior; para el ejemplo, que no sea menor que 1 ni mayor que 100.

Escribir “***** ERROR *** debe ser \geq ”, **vi**, “ **y \leq ”, **vs**. Mensaje de alerta de error en caso de que el usuario haya digitado un valor menor que el valor inferior o mayor que el valor superior; para el ejemplo se visualiza el mensaje:****

```
“*** ERROR *** debe ser  $\geq$  ”1 y  $\leq$  100
```

Escribir “**Presione una tecla para continuar...**”. Se visualiza el mensaje entre comillas.

Esperar Tecla. Se realiza una pausa hasta que el usuario presione cualquier tecla.

FinSi cierra el condicional.

Hasta Que (**val \geq vi**) & (**val \leq vs**). Condición del ciclo REPETIR que valida el valor ingresado. Si es menor que 1 o mayor que 100, retorna a la línea de código siguiente a la sentencia **repetir**; en este caso, Escribir “ ”. Si no se ejecuta la línea de código **FinFuncion**, retorna el flujo de ejecución al algoritmo principal, a la línea de código siguiente a la línea que invocó a la función.

Funcion val \leftarrow **LeerValorReal(msg, vi, vs)** tiene exactamente la misma estructura, solo que el dato ingresado es de tipo real.

Funcion cad \leftarrow **LeerCadena(msg)** tiene la misma estructura, y solo valida que el dato ingresado no esté en blanco; se valida en el condicional:

Si cad = "" Entonces

Funcion car \leftarrow **leerCaracter(msg, c1, c2)** tiene la misma estructura, pero esta función valida que el usuario solo pueda ingresar dos caracteres; por ejemplo, S o N, F o M, o cualesquier otros dos que se requieran, como P y Q.

Dentro de las funciones adicionales están:

Funcion IntercambiarValores(a Por Valor, b Por Valor). Como se aprecia, esta función no tiene variable de retorno, y recibe dos parámetros por valor en las variables **a** y **b**.

Definir aux Como Entero declara o define la variable local **aux** de tipo entero.

MostrarDatos(“**EN LA FUNCIÓN IntercambiarValores ANTES DEL INTERCAMBIO**”, “**Valor de a =** ”, “**Valor de b =** ”, **a, b**). En es-

ta línea de código se invoca a la función **MostrarDatos**, y se le pasan como parámetros los mensajes “EN LA FUNCIÓN IntercambiarValores ANTES DEL INTERCAMBIO”, “Valor de a = ”, “Valor de b = ”, y las variables **a** y **b**.

aux ← **a**. Se le asigna a la variable **aux** el valor o contenido de la variable **a**.

a ← **b**. A la variable **a** se le asigna el valor o contenido de la variable **b**.

b ← **aux**. A la variable **b** se le asigna el valor o contenido de la variable **aux**.

MostrarDatos(“EN LA FUNCIÓN IntercambiarValores DESPUÉS DEL INTERCAMBIO ”, “Valor de a = ”, “Valor de b = ”, **a**, **b**). Se invoca nuevamente a la función **MostrarDatos** con los parámetros “EN LA FUNCIÓN IntercambiarValores DESPUÉS DEL INTERCAMBIO ”, “Valor de a = ”, “Valor de b = ”, **a**, **b**.

FinFuncion sentencia que declara el fin de una función.

Funcion MostrarDatos(msg1, msg2, msg3, a, b). Función para visualizar información. Recibe tres parámetros de tipo cadena en las variables **msg1**, **msg2**, **msg3**, y dos valores en las variables **a** y **b**, respectivamente.

Escribir "" deja una línea en blanco al momento de ejecutar el algoritmo.

Escribir msg1 visualiza el contenido de **msg1**.

Escribir msg2, a visualiza el contenido de **msg2** y **a**.

Escribir msg3, b visualiza el contenido de **msg3** y **b**.

FinFuncion sentencia que declara el fin de una función.

En el algoritmo principal se tiene:

Algoritmo Ejercicio1C3, sentencia que da inicio al algoritmo principal con el nombre de **Ejercicio1C3**.

Definir val1, val2 Como Enteros. Se declaran o definen las variables **val1** y **val2** de tipo entero.

val1 ← **LeerValorEntero**(“Digite valor 1: ”, **1**, **100**). Se invoca a la función **LeerValorEntero** y se le pasa como parámetro el mensaje “Digite valor 1: ”, el

valor 1 como límite inferior y 100 como límite superior; es decir, el usuario debe ingresar un valor comprendido entre 1 y 100. De no ser así, se genera un mensaje de error hasta que se ingrese un valor válido, y el valor ingresado se almacena en la variable de retorno `val1`.

`val2 ← LeerValorEntero(“Digite valor 2: ”, 1, 100)`. Tiene el mismo funcionamiento de la anterior, y el valor ingresado se almacena en la variable de retorno `val2`.

`MostrarDatos(“EN EL ALGORITMO PRINCIPAL ANTES DE LLAMAR A LA FUNCIÓN IntercambiarValores”, “Valor 1 = ”, “Valor 2 = ”, val1, val2)`. Se invoca a la función `MostrarDatos`, y se le envían como parámetros los mensajes entre comillas “EN EL ALGORITMO PRINCIPAL ANTES DE LLAMAR A LA FUNCIÓN IntercambiarValores”, “Valor 1 = ”, “Valor 2 = ”, y los valores almacenados en las variables `val1`, `val2`.

`IntercambiarValores(val1, val2)`. Se invoca a la función `IntercambiarValores` y se le pasan como parámetros los valores contenidos en las variables `val1` y `val2`, correspondientes a los valores digitados por el usuario.

`MostrarDatos(“EN EL ALGORITMO PRINCIPAL DESPUÉS DE LLAMAR A LA FUNCIÓN IntercambiarValores”, “Valor 1 = ”, “Valor 2 = ”, val1, val2)`. Nuevamente se invoca a la función `MostrarDatos`, ahora con los parámetros “EN EL ALGORITMO PRINCIPAL DESPUÉS DE LLAMAR A LA FUNCIÓN IntercambiarValores”, “Valor 1 = ”, “Valor 2 = ”, `val1`, `val2`.

FinAlgoritmo sentencia que determina el final del algoritmo.

Al ejecutar el algoritmo, se visualiza lo siguiente en pantalla (**figura 3.1**).

Figura 3.1: Inicio ejecución del Ejercicio1C3

```
*** Ejecución Iniciada. ***
```

```
Digite valor 1: >|
```

El algoritmo espera un valor entre 1 y 100, porque así está especificado en la línea de código `val1 ← LeerValorEntero(“Digite valor 1: ”, 1, 100)`. Si el usuario ingresa, por ejemplo, el valor 0, se verá el siguiente mensaje en pantalla (**figura 3.2**):

Figura 3.2: Validación en tiempo de ejecución del valor 1 Ejercicio1C3 para que no sea menor que 1

```
*** Ejecución Iniciada. ***
Digite valor 1: > 0
*** ERROR *** debe ser >= 1 y <= 100
Presione una tecla para continuar...
```

La ejecución no continúa hasta que el usuario presione una tecla; y al presionarla se visualiza la siguiente información (**figura 3.3**).

Figura 3.3: Validación en tiempo de ejecución del valor 1 Ejercicio1C3

```
*** Ejecución Iniciada. ***
Digite valor 1: > 0
*** ERROR *** debe ser >= 1 y <= 100
Presione una tecla para continuar...
Digite valor 1: >
```

Está pidiendo ingresar nuevamente el valor 1. Ahora, si se ingresa, por ejemplo, 120, se visualizará en pantalla lo siguiente (**figura 3.4**):

Figura 3.4: Validación en tiempo de ejecución del valor 1 Ejercicio1C3 para que no sea mayor que 100

```
*** Ejecución Iniciada. ***
Digite valor 1: > 0
*** ERROR *** debe ser >= 1 y <= 100
Presione una tecla para continuar...
Digite valor 1: > 120
*** ERROR *** debe ser >= 1 y <= 100
Presione una tecla para continuar...
```

No permite que se avance en la ejecución hasta tanto no se ingrese un valor en el rango 1-100 (por ejemplo, 5). Inmediatamente solicita que se ingrese el valor 2, el cual debe cumplir con el mismo requisito, es decir, estar en el rango 1-100, porque

así está indicado en la línea de código `val2 ← LeerValorEntero("Digite valor 2: ", 1, 100)`. Por ejemplo, si fuera 10, se vería lo siguiente en pantalla (**figura 3.5**):

Figura 3.5: Validación en tiempo de ejecución del valor 1 Ejercicio1C3 ingreso de los valores 1 y 2 en el rango 1-100

```

*** Ejecución Iniciada. ***

Digite valor 1: > 0

*** ERROR *** debe ser >= 1 y <= 100
Presione una tecla para continuar...

Digite valor 1: > 120

*** ERROR *** debe ser >= 1 y <= 100
Presione una tecla para continuar...

Digite valor 1: > 5

Digite valor 2: > 10

EN EL ALGORITMO PRINCIPAL ANTES DE LLAMAR A LA FUNCIÓN IntercambiarValores 1
Valor 1 = 5
Valor 2 = 10

EN LA FUNCIÓN IntercambiarValores ANTES DEL INTERCAMBIO 2
Valor de a = 5
Valor de b = 10

EN LA FUNCIÓN IntercambiarValores DESPUÉS DEL INTERCAMBIO 3
Valor de a = 10
Valor de b = 5

EN EL ALGORITMO PRINCIPAL DESPUÉS DE LLAMAR A LA FUNCIÓN IntercambiarValores 4
Valor 1 = 5
Valor 2 = 10
*** Ejecución Finalizada. ***

```

En 1 se visualiza el contenido de las variables `val1` y `val2`, que tienen los valores 5 y 10, respectivamente, correspondientes estos a los ingresados por el usuario antes de invocar o llamar a la función **IntercambiarValores**.

En 2 **a** tiene el valor de 5 y **b** el valor de 10, que corresponden, a su vez, a los valores de las variables `val1` y `val2`, respectivamente; estos fueron asignados desde el algoritmo principal en la línea de código **IntercambiarValores(val1, val2)**

y recibidos en la Funcion **IntercambiarValores(a Por Valor, b Por Valor)**. Como se da una correspondencia entre los parámetros actuales y los formales, entonces a **val1** lo recibe **a** y a **val2** lo recibe **b**.

En 3 **a** tiene el valor 10 y **b** el valor 5, es decir, se han intercambiado los valores de las variables **a** y **b**; esto se da en la **Funcion IntercambiarValores**, en las líneas de código:

```
aux ← a
a ← b
b ← aux
```

Finalmente, en 4 se aprecia que val1 es igual a 5 y val2 es igual a 10, es decir, tienen los mismos valores ingresados por el usuario. Esto se da porque los parámetros indicados en la **Funcion IntercambiarValores** están declarados por valor, por lo cual los cambios aplicados en esta función no afectan a los valores de las variables val1 y val2, debido a que se pasa una copia del contenido de estas variables.

Si se desea que se modifiquen los valores de las variables **val1** y **val2** por la **Funcion IntercambiarValores**, entonces se deben modificar los parámetros por valor a por referencia (Ejercicio2C3). La función queda así:

```
Funcion IntercambiarValores(a Por Referencia, b Por Referencia) //Función para
intercambiar los valores de a y b
```

```
    Definir aux Como Entero
    MostrarDatos("EN LA FUNCIÓN IntercambiarValores ANTES DEL
                INTERCAMBIO ", "Valor de a = ", "Valor de b = ", a, b)
    aux ← a
    a ← b
    b ← aux
    MostrarDatos("EN LA FUNCIÓN IntercambiarValores DESPUÉS DEL
                INTERCAMBIO ", "Valor de a = ", "Valor de b = ", a, b)
```

```
FinFuncion
```

Al ejecutar el algoritmo, ahora se tiene el siguiente resultado (**figura 3.6**):

Figura 3.6: Ejecución del Ejercicio2C3

```

*** Ejecución Iniciada. ***

Digite valor 1: > 5

Digite valor 2: > 10

EN EL ALGORITMO PRINCIPAL ANTES DE LLAMAR A LA FUNCIÓN IntercambiarValores
Valor 1 = 5
Valor 2 = 10
1

EN LA FUNCIÓN IntercambiarValores ANTES DEL INTERCAMBIO
Valor de a = 5
Valor de b = 10
2

EN LA FUNCIÓN IntercambiarValores DESPUÉS DEL INTERCAMBIO
Valor de a = 10
Valor de b = 5
3

EN EL ALGORITMO PRINCIPAL DESPUÉS DE LLAMAR A LA FUNCIÓN IntercambiarValores
Valor 1 = 10
Valor 2 = 5
4

*** Ejecución Finalizada. ***

```

Como se puede observar, en 3 **a** tiene el valor de 10 y **b** el valor de 5 (están intercambiados). En 4 sucede exactamente lo mismo: **val1** ahora tiene el valor 10 y **val2** el valor 5, y esto es posible porque los parámetros en la **Funcion IntercambiarValores** están por referencia, lo cual indica que reciben una dirección de memoria de los parámetros actuales. Así, los cambios que se produzcan en la función **IntercambiarValores** afectan directamente a los parámetros actuales **val1** y **val2**.

¿Qué sucede si en la **Funcion IntercambiarValores** se coloca **a** por valor y **b** por referencia?

En este caso, como **a** está por valor, no afecta al parámetro actual **val1**, y queda con el valor de 5. En cambio, como **b** está por referencia, afecta directamente al parámetro actual **val2**, que queda también con el valor 5. La función **IntercambiarValores** ahora queda como:

```

Funcion IntercambiarValores(a Por Valor, b Por Referencia) //Función para
intercambiar los valores de a y b

```

```

    Definir aux Como Entero

```

```

MostrarDatos("EN LA FUNCIÓN IntercambiarValores ANTES DEL
              INTERCAMBIO ", "Valor de a = ", "Valor de b = ", a, b)
aux ← a
a ← b
b ← aux
MostrarDatos("EN LA FUNCIÓN IntercambiarValores DESPUÉS DEL
              INTERCAMBIO ", "Valor de a = ", "Valor de b = ", a, b)
FinFuncion

```

Al ejecutar el algoritmo Ejercicio3C3, se tiene el siguiente resultado (figura 3.7):

Figura 3.7: Ejecución del Ejercicio3C3

```

*** Ejecución Iniciada. ***

Digite valor 1: > 5

Digite valor 2: > 10

EN EL ALGORITMO PRINCIPAL ANTES DE LLAMAR A LA FUNCIÓN IntercambiarValores
Valor 1 = 5
Valor 2 = 10

EN LA FUNCIÓN IntercambiarValores ANTES DEL INTERCAMBIO
Valor de a = 5
Valor de b = 10

EN LA FUNCIÓN IntercambiarValores DESPUÉS DEL INTERCAMBIO
Valor de a = 10
Valor de b = 5

EN EL ALGORITMO PRINCIPAL DESPUÉS DE LLAMAR A LA FUNCIÓN IntercambiarValores
Valor 1 = 5
Valor 2 = 5
*** Ejecución Finalizada. ***

```

Como se observa en la parte final de la ejecución, tanto val1 como val2 tienen valor 5.

2. Realizar un algoritmo que lea dos valores. De acuerdo con el menú de opciones siguiente, llevar a cabo la operación indicada:
 1. Sumar
 2. Restar
 3. Multiplicar
 4. Dividir
 5. Potencia

Diseñar el algoritmo de tal manera que el usuario pueda hacer varias operaciones con los valores ingresados; también, ingresar nuevos valores. Aplicar el paradigma de programación modular.

Análisis del problema

Este ejercicio se había desarrollado en el capítulo II de forma secuencial: ahora se desarrollará de manera modular. El análisis es el mismo.

Tabla 3.2: Identificación de variables Ejercicio4C3

Tipo	Variable	Descripción
Real	val1	Valor 1
Real	val2	Valor 2
Real	res	Resultado de la operación
Entero	opc	Opción seleccionada por el usuario
Carácter	seg1	Opción para realizar otra operación
Carácter	seg2	Opción para ingresar nuevos valores

Datos de entrada: val1, val2, opc, seg1, seg2

Procesos parciales:

$res \leftarrow [val1 + val2 \mid val1 - val2 \mid val1 * val2 \mid val1 / val2 \mid val1 \wedge val2]$

Procesos totales: N/A

Datos de salida parciales:

Res, ["ERROR. No se puede dividir entre 0" | "La opción seleccionada no es válida"]

Datos de salida totales: N/A

Desarrollo del algoritmo en PSeInt

Se usarán las mismas funciones desarrolladas en el ejercicio anterior para la entrada de datos (LeerValorEntero, LeerValorReal, LeerCadena, LeerCaracter), teniendo en cuenta que se debe realizar la suma, la resta, la multiplicación, la división y la potencia. Entonces, se tiene que desarrollar una función para cada uno de estos procesos: una para visualizar la información, otra, muy conveniente, para el menú de opciones y una más para validar la operación que se realizará según la selección del usuario.

```

//Algoritmo principal
Algoritmo Ejercicio4C3

    //Definir o declarar las variables
    Definir val1, val2, res Como Real
    Definir opc Como Entero
    Definir seg1, seg2 Como Caracter

    Repetir

        Borrar Pantalla
        val1 ← LeerValorReal("Digite valor 1: ", 0, 100)
        val2 ← LeerValorReal("Digite valor 2: ", 0, 100)

        Repetir

            MenuOperaciones()
            opc ← LeerValorEntero("Seleccione una operación: ", 1, 5)
            res ← ValidarOperacion(opc, val1, val2)
            MostrarDatos(val1, val2, res, opc)
            seg1 ← leerCaracter("¿Desea realizar una nueva operación
                               [S/N]? ", "S", "N")

            Hasta Que Mayusculas(Seg1) = "N"

            seg2 ← leerCaracter("¿Desea ingresar nuevos valores [S/N]? ", "S",
                               "N")

            Hasta Que Mayusculas(seg2) = "N"

    FinAlgoritmo

//Funciones para datos de entrada

//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer o capturar un valor real
Funcion val ← LeerValorReal(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer una cadena como nombres, direcciones, etc.
Funcion cad ← LeerCadena(msg) (ver Ejercicio1C3)

//Función para leer un carácter como S, N, F, M, etc.
Funcion car ← leerCaracter(msg, c1, c2) (ver Ejercicio1C3)

//Funciones adicionales requeridas según los Procesos

//Función para crear el menú de operaciones
Funcion MenuOperaciones()
    Borrar Pantalla
    Escribir "1. Sumar"

```

```
    Escribir "2. Restar"
    Escribir "3. Multiplicar"
    Escribir "4. Dividir"
    Escribir "5. Potencia"
FinFuncion

// Función para sumar dos valores
Funcion res ← Sumar(val1, val2)
    Definir res Como Real
    res ← val1 + val2
FinFuncion

//Función para restar valores
Funcion res ← Restar(val1, val2)
    Definir res Como Real
    res ← val1 - val2
FinFuncion

//Función para multiplicar dos valores
Funcion res ← Multiplicar(val1, val2)
    Definir res Como Real
    res ← val1 * val2
FinFuncion

//Función para dividir dos valores
Funcion res ← Dividir(val1, val2)
    Definir res Como Real

    Si val2 = 0 Entonces
        res ← -0.999
    SiNo
        res ← val1 / val2
    FinSi
FinFuncion

//Función para potencia de dos números
Funcion res ← Potencia(val1, val2)
    Definir res Como Real
    res ← val1 ^ val2
FinFuncion

//Función para validar la opción seleccionada por el usuario y realizar la
//operación indicada
Funcion res ← ValidarOperacion(opc, val1, val2)
    Definir res Como Real
    Segun opc Hacer
```



```

1: res ← Sumar(val1, val2)
2: res ← Restar(val1, val2)
3: res ← Multiplicar(val1, val2)
4: res ← Dividir(val1, val2)
5: res ← Potencia(val1, val2)

```

FinSegun

FinFuncion

//Función para mostrar los resultados de las operaciones

Funcion MostrarDatos(val1, val2, res, opc)

Segun opc Hacer

```

1: Escribir val1, " + ", val2, " = ", res
2: Escribir val1, " - ", val2, " = ", res
3: Escribir val1, " * ", val2, " = ", res
4: Si res = -0.999 Entonces
    Escribir "ERROR. No se puede dividir entre 0"
  SiNo
    Escribir val1, " / ", val2, " = ", Redon(res * 100)/100
  FinSi
5: Escribir val1 " ^ ", val2, " = ", Redon(res * 100)/100

```

FinSegun

FinFuncion

Observaciones

Los valores de las variables val1 y val2 van a estar entre 0 y 100, dadas las siguientes líneas de código:

```
val1 ← LeerValorReal("Digite valor 1: ", 0, 100)
```

```
val2 ← LeerValorReal("Digite valor 2: ", 0, 100)
```

Como el menú de operaciones tiene cinco opciones, entonces la variable opc va a tener un valor entre 1 y 5, por la línea de código siguiente:

```
opc ← LeerValorEntero("Seleccione una operación: ", 1, 5)
```

Las variables seg1 y seg2 van a tomar el carácter S o N, dadas las siguientes líneas de código:

```
seg1 ← leerCaracter("¿Desea realizar una nueva operación [S/N]? ", "S", "N")
```

```
seg2 ← leerCaracter("¿Desea ingresar nuevos valores [S/N]? ", "S", "N")
```

En **Funcion** `res ← Dividir(val1, val2)` se valida que si la variable `val2 = 0`, entonces a la variable `res` se le asigna el valor de `-0.999`; a este se le conoce como centinela o bandera, y es asignado directamente por el programador (puede ser cualquier otro valor). Este es usado en la **Funcion** `MostrarDatos(val1, val2, res, opc)` para validar que si `res = -0.999`, se visualiza el mensaje **ERROR**. No se puede dividir entre 0; de lo contrario, se muestra el `val1` dividido entre el valor `2` igual al resultado.

Nótese que cada una de las funciones solo realiza una tarea, actividad o proceso específico, cumpliendo así con una de las características de la programación modular.

Al ejecutar el algoritmo, se tienen los siguientes resultados (**figuras 3.8 y 3.9**).

Figura 3.8: Ingresar valor 1 y 2 Ejercicio4C3

```
Digite valor 1: > 5
Digite valor 2: > 2
```

Figura 3.9: Seleccionar la operación por realizar Ejercicio4C3

```
1. Sumar
2. Restar
3. Multiplicar
4. Dividir
5. Potencia

Seleccione una operación: > 2
```

Se visualiza el resultado de la operación seleccionada (**figura 3.10**).

Figura 3.10: Resultado operación seleccionada Ejercicio4C3

```
5 - 2 = 3
```

Responder la pregunta **si se desea realizar una nueva operación**. Si la respuesta es S, entonces regresa a **figura 3.9**; si no, pasa a responder la pregunta **si desea ingresar nuevos valores**, y, si la respuesta es S, regresa a **figura 3.8**; de lo contrario, finaliza la ejecución.

3. A los empleados de la compañía ABC les otorgan una sola vez al año una bonificación de acuerdo con su salario básico y los años de antigüedad en la organización según la siguiente información:

Tiempo en años	Porcentaje
Menos de 5 años	5 % del salario básico
5 años o más y menos de 10 años	10 % del salario básico
10 años o más y menos de 15 años	15 % del salario básico
15 años o más y menos de 20 años	20 % del salario básico
20 años o más y menos de 25 años	25 % del salario básico
25 años o más y menos de 30 años	35 % del salario básico
30 años o más	50 % del salario básico

Realizar un algoritmo modular para determinar la bonificación que recibe un empleado; además, se desea conocer el número de empleados de cada rango de tiempo, el porcentaje que representa cada uno respecto al total de empleados y el total pagado en bonificaciones.

Análisis del problema

Este ejercicio fue desarrollado en el capítulo II de manera secuencial: ahora se desarrolla de forma modular. El análisis del problema es el mismo.

Tabla 3.3: Identificación de variables Ejercicio5C3

Tipo	Variable	Descripción
Cadena	ideEmp	Identificación del empleado
Cadena	apeEmp	Apellidos del empleado
Cadena	nomEmp	Nombres del empleado
Real	salBas	Salario básico
Entero	tieSer	Tiempo de servicio en años
Real	porBon	Porcentaje de bonificación
Real	valBon	Valor de la bonificación
Entero	conEmp1	Empleados con menos de 5 años
Entero	conEmp2	Empleados con 5 años o más y menos de 10 años
Entero	conEmp3	Empleados con 10 años o más y menos de 15 años
Entero	conEmp4	Empleados con 15 años o más y menos de 20 años

Tipo	Variable	Descripción
Entero	conEmp5	Empleados con 20 años o más y menos de 25 años
Entero	conEmp6	Empleados con 25 años o más y menos de 30 años
Entero	conEmp7	Empleados con 30 años o más
Entero	empTot	Empleados totales
Real	porEmp1	Porcentaje de empleados con menos de 5 años
Real	porEmp2	Porcentaje de empleados con 5 años o más y menos de 10 años
Real	porEmp3	Porcentaje de empleados con 10 años o más y menos de 15 años
Real	porEmp4	Porcentaje de empleados con 15 años o más y menos de 20 años
Real	porEmp5	Porcentaje de empleados con 20 años o más y menos de 25 años
Real	porEmp6	Porcentaje de empleados con 25 años o más y menos de 30 años
Real	porEmp7	Porcentaje de empleados con 30 años o más
Real	bonTot	Bonificaciones totales pagadas a los empleados
Carácter	opc	Opción para continuar o no ingresando datos

Datos de entrada: ideEmp, apeEmp, nomEmp, salBas, tieSer, opc

Procesos parciales:

$porBon \leftarrow [5 \mid 10 \mid 15 \mid 20 \mid 25 \mid 35 \mid 50]$

$valBon \leftarrow (sueBas * porBon)/100$

$bonTot \leftarrow 0$

$bonTot \leftarrow bonTot + valBon // \text{Proceso tipo acumulador}$

$conEmp1 \leftarrow 0$

$conEmp1 \leftarrow conEmp1 + 1 // \text{Proceso tipo contador}$

$conEmp2 \leftarrow 0$

$conEmp2 \leftarrow conEmp2 + 1 // \text{Proceso tipo contador}$

$conEmp3 \leftarrow 0$

$conEmp3 \leftarrow conEmp3 + 1 // \text{Proceso tipo contador}$

$conEmp4 \leftarrow 0$

$conEmp4 \leftarrow conEmp1 + 1 // \text{Proceso tipo contador}$

$conEmp5 \leftarrow 0$

$conEmp5 \leftarrow conEmp5 + 1 // \text{Proceso tipo contador}$

$conEmp6 \leftarrow 0$

$conEmp6 \leftarrow conEmp6 + 1 // \text{Proceso tipo contador}$

$conEmp7 \leftarrow 0$

$conEmp7 \leftarrow conEmp7 + 1 // \text{Proceso tipo contador}$

Procesos totales:

$$\begin{aligned} empTot \leftarrow & conEmp1 + conEmp2 + conEmp3 + conEmp4 + conEmp5 \\ & + conEmp6 + conEmp7 \end{aligned}$$

$$porEmp1 \leftarrow (conEmp1 * 100) / empTot$$

$$porEmp2 \leftarrow (conEmp2 * 100) / empTot$$

$$porEmp3 \leftarrow (conEmp3 * 100) / empTot$$

$$porEmp4 \leftarrow (conEmp4 * 100) / empTot$$

$$porEmp5 \leftarrow (conEmp5 * 100) / empTot$$

$$porEmp6 \leftarrow (conEmp6 * 100) / empTot$$

$$porEmp7 \leftarrow (conEmp7 * 100) / empTot$$

Datos de salida parciales: porBon, valBon

Datos de salida totales: conEmp1, conEmp2, conEmp3, conEmp4, conEmp5, conEmp6, conEmp7, empTot, porEmp1, porEmp2, porEmp3, porEmp4, porEmp5, porEmp6, porEmp7

Desarrollo del algoritmo en PSeInt

Lo primero es analizar los procesos parciales y totales para determinar las funciones que se deben realizar; para ello se tiene:

- a. Las variables tipo contador y acumulador deben ser inicializadas en 0 y, si existen constantes, también se inicializan. Se creará la Funcion Inicializar, de tipo procedimiento, con todos sus parámetros por referencia.
- b. $porBon \leftarrow [5 \mid 10 \mid 15 \mid 20 \mid 25 \mid 35 \mid 50]$. Se requiere una función que determine el porcentaje de bonificación. Se creará la **Funcion Porcentaje**.
- c. Los procesos:

$$valBon \leftarrow (sueBas * porBon) / 100$$

$$porEmp1 \leftarrow (conEmp1 * 100) / empTot$$

$$porEmp2 \leftarrow (conEmp2 * 100) / empTot$$

$$porEmp3 \leftarrow (conEmp3 * 100) / empTot$$

$$porEmp4 \leftarrow (conEmp4 * 100) / empTot$$

$$porEmp5 \leftarrow (conEmp5 * 100) / empTot$$

$$porEmp6 \leftarrow (conEmp6 * 100) / empTot$$

$$porEmp7 \leftarrow (conEmp7 * 100) / empTot$$

Todas las variables que forman parte de estos procesos son de tipo real.

Todos realizan la multiplicación de un valor por otro y lo dividen entre otro valor; por lo tanto, se requiere solo una función con manejo de diferentes parámetros. Se creará la **Funcion MultiplicarDividir**.

d. Los procesos:

```
bonTot ← bonTot + valBon
conEmp1 ← conEmp1 + 1
conEmp2 ← conEmp2 + 1
conEmp3 ← conEmp3 + 1
conEmp4 ← conEmp4 + 1
conEmp5 ← conEmp5 + 1
conEmp6 ← conEmp6 + 1
conEmp7 ← conEmp7 + 1
bonTot ← bonTot + valBon
```

Los anteriores procesos tienen la misma estructura, y suman dos valores; pero las variables **bonTot** y **valBon** son de tipo real y las demás son enteras, por lo que se requiere de dos funciones: una para sumar dos valores reales y otra para sumar dos valores enteros. Se crearán las funciones **Sumar2ValoresEnteros** y **Sumar2ValoresReales**.

Solo queda el proceso:

```
vempTot ← conEmp1 + conEmp2 + conEmp3 + conEmp4 + conEmp5
          + conEmp6 + conEmp7
```

Suma siete valores enteros, por lo que se debe crear una función que realice esta tarea: **Sumar7Valores**.

- e. Como se debe contar el número de empleados según los años de servicios, se realiza la función **ContarEmpleados**.
- f. Como se tienen datos de salida parciales y totales, entonces se realizan dos funciones, una para visualizar los datos parciales y otra para los datos totales: **DatosSalidaParciales** y **DatosSalidaTotales**.
- g. Para los datos de entrada se tienen las funciones que se realizaron en los ejercicios anteriores: **LeerValorEntero**, **LeerValorReal**, **LeerCadena** y **leerCaracter**.

```

//Algoritmo principal
Algoritmo Ejercicio5C3

    //Definir o declarar las variables
    Definir ideEmp, apeEmp, nomEmp Como Cadena
    Definir salBas, porBon, valBon, porEmp1, porEmp2, porEmp3, porEmp4,
        porEmp5,
        porEmp6, porEmp7, bonTot Como Real
    Definir tieSer, conEmp1, conEmp2, conEmp3, conEmp4, conEmp5, conEmp6,
        conEmp7, empTot Como Entero
    Definir opc Como Carácter

    //Inicializar variables y constantes
    Inicializar(bonTot, conEmp1, conEmp2, conEmp3, conEmp4, conEmp5,
        conEmp6, conEmp7)

    Repetir

        Borrar Pantalla

        //Datos de entrada
        ideEmp ← LeerCadena("Identificación del empleado: ")
        apeEmp ← LeerCadena("Apellidos del empleado : ")
        nomEmp ← LeerCadena("Nombres del empleado : ")
        salBas ← LeerValorReal("Salario básico : ", 1000000, 20000000)
        tieSer ← LeerValorEntero("Años de servicios : ", 0, 50)

        //Procesos parciales
        porBon ← Porcentaje(salBas, tieSer)
        valBon ← MultiplicarDividir(salBas, porBon, 100)
        bonTot ← Sumar2ValoresReales(bonTot, valBon)
        ContarEmpleados(conEmp1, conEmp2, conEmp3, conEmp4, conEmp5,
            conEmp6, conEmp7, porBon)

        //Datos de salida parciales
        DatosSalidaParciales(porBon, valBon)

        opc ← LeerCaracter("¿Desea ingresar un nuevo empleado [S/N]? ",
            "S", "N")

    Hasta Que Mayusculas(opc) = "N"

    //Procesos totales
    empTot ← Sumar7Valores(conEmp1, conEmp2, conEmp3, conEmp4, conEmp5,
        conEmp6, conEmp7)
    porEmp1 ← MultiplicarDividir(conEmp1, 100, empTot)
    porEmp2 ← MultiplicarDividir(conEmp2, 100, empTot)
    porEmp3 ← MultiplicarDividir(conEmp3, 100, empTot)
    porEmp4 ← MultiplicarDividir(conEmp4, 100, empTot)
    porEmp5 ← MultiplicarDividir(conEmp5, 100, empTot)

```

```
porEmp6 ← MultiplicarDividir(conEmp6, 100, empTot)
porEmp7 ← MultiplicarDividir(conEmp7, 100, empTot)

//Datos de salida totales
DatosSalidaTotales(conEmp1, conEmp2, conEmp3, conEmp4, conEmp5,
                    conEmp6, conEmp7, empTot, porEmp1, porEmp2,
                    porEmp3, porEmp4, porEmp5, porEmp6, porEmp7,
                    bonTot)

FinAlgoritmo

//Funciones para datos de entrada

//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer o capturar un valor real
Funcion val ← LeerValorReal(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer una cadena como nombres, direcciones, etc.
Funcion cad ← LeerCadena(msg) (ver Ejercicio1C3)

//Función para leer un carácter como S, N, F, M, etc.
Funcion car ← leerCaracter(msg, c1, c2) (ver Ejercicio1C3)

//Funciones adicionales requeridas según los Procesos

Funcion Inicializar(bonTot Por Referencia, conEmp1 Por Referencia,
                    conEmp2 Por Referencia, conEmp3 Por Referencia, conEmp4 Por
                    Referencia, conEmp5 Por Referencia, conEmp6 Por Referencia,
                    conEmp7 Por Referencia)

    bonTot ← 0
    conEmp1 ← 0
    conEmp2 ← 0
    conEmp3 ← 0
    conEmp4 ← 0
    conEmp5 ← 0
    conEmp6 ← 0
    conEmp7 ← 0

FinFuncion

Funcion porc ← Porcentaje(salBas, tieSer)

    Definir porc Como Real

    Si tieSer < 5 Entonces
        porc ← 5
    SiNo
        Si tieSer < 10 Entonces
            porc ← 10
        SiNo
            Si tieSer < 15 Entonces
```



```

        porc ← 15
    SiNo
        Si tieSer < 20 Entonces
            porc ← 20
        SiNo
            Si tieSer < 25 Entonces
                porc ← 25
            SiNo
                Si tieSer < 30 Entonces
                    porc ← 35
                SiNo
                    porc ← 50
            FinSi
        FinSi
    FinSi
FinSi
FinSi
FinSi
FinSi
FinFuncion

Funcion res ← MultiplicarDividir(val1, val2, val3)
    Definir res Como Real
    res ← (val1 * val2) / val3
FinFuncion

Funcion res ← Sumar2ValoresEnteros(val1, val2)
    Definir res Como Entero
    res ← val1 + val2
FinFuncion

Funcion res ← Sumar2ValoresReales(val1, val2)
    Definir res Como Real
    res ← val1 + val2
FinFuncion

Funcion res ← Sumar7Valores(val1, val2, val3, val4, val5, val6, val7)
    Definir res Como Entero
    res ← val1 + val2 + val3 + val4 + val5 + val6 + val7
FinFuncion

Funcion ContarEmpleados(conEmp1 Por Referencia, conEmp2 Por Referencia,
                        conEmp3 Por Referencia, conEmp4 Por Referencia,
                        conEmp5 Por Referencia, conEmp6 Por Referencia,
                        conEmp7 Por Referencia, porBon)

Segun porBon Hacer
    5: conEmp1 ← Sumar2ValoresEnteros(conEmp1, 1)
    10: conEmp2 ← Sumar2ValoresEnteros(conEmp2, 1)

```

```
15: conEmp3 ← Sumar2ValoresEnteros(conEmp3, 1)
20: conEmp4 ← Sumar2ValoresEnteros(conEmp4, 1)
25: conEmp5 ← Sumar2ValoresEnteros(conEmp5, 1)
35: conEmp6 ← Sumar2ValoresEnteros(conEmp6, 1)
50: conEmp7 ← Sumar2ValoresEnteros(conEmp7, 1)
```

FinSegun

FinFuncion

Funcion DatosSalidaParciales(porBon, valBon)

```
    Escribir ""
    Escribir "Porcentaje de bonificación: ", Redon(porBon * 100) / 100
    Escribir ""
    Escribir "Valor bonificación : ", Redon(valBon * 100) / 100
```

FinFuncion

Funcion DatosSalidaTotales(conEmp1, conEmp2, conEmp3, conEmp4, conEmp5,
 conEmp6, conEmp7, empTot, porEmp1, porEmp2,
 porEmp3, porEmp4, porEmp5, porEmp6, porEmp7,
 bonTot)

Borrar Pantalla

```
    Escribir "Empleados con menos de 5 años : ", conEmp1, " ",
            Redon(porEmp1 * 100) / 100, "%"
    Escribir "Empleados con 5 años o más y menos de 10 : ", conEmp2, " ",
            Redon(porEmp2 * 100) / 100, "%"
    Escribir "Empleados con 10 años o más y menos de 15 : ", conEmp3, " ",
            Redon(porEmp3 * 100) / 100, "%"
    Escribir "Empleados con 15 años o más y menos de 20 : ", conEmp4, " ",
            Redon(porEmp4 * 100) / 100, "%"
    Escribir "Empleados con 20 años o más y menos de 25 : ", conEmp5, " ",
            Redon(porEmp5 * 100) / 100, "%"
    Escribir "Empleados con 25 años o más y menos de 30 : ", conEmp6, " ",
            Redon(porEmp6 * 100) / 100, "%"
    Escribir "Empleados con 30 años o más : ", conEmp7, " ",
            Redon(porEmp7 * 100) / 100, "%"
    Escribir "Total de empleados : ", empTot
    Escribir "Total bonificaciones pagadas : ", bonTot
```

FinFuncion

4. Un almacén hace descuento a sus clientes de acuerdo con la siguiente información:

Compras mayores o iguales a 100000 y menores de 200000 tienen descuento del 10%.
Compras mayores o iguales a 200000 y menores de 300000 tienen descuento del 15%.
Compras mayores o iguales a 300000 y menores de 400000 tienen descuento del 20%

Compras mayores o iguales a 400000 y menores de 500000 tienen descuento del 25 %.
Compras mayores o iguales a 500000 tienen descuento del 30 %.

Realizar un algoritmo para determinar el valor que un cliente debe pagar por su compra, adicional se desea conocer cuantos clientes obtuvieron descuentos del 10, 15, 20, 25 y 30 por ciento y los clientes que no tuvieron descuento, el monto total de las compras sin descuento, el monto total de las compras con descuento

Análisis del problema

Como en el caso anterior, aquí es necesario procesar varios clientes, por lo que es importante identificar a cada uno con al menos el número de identificación, apellidos y nombres.

Tabla 3.4: Identificación de variables Ejercicio6C3

Tipo	Variable	Descripción
Cadena	ideCli	Identificación del cliente
Cadena	apeCli	Apellidos del cliente
Cadena	nomCli	Nombres del cliente
Real	valCom	Valor de la compra
Real	porDes	Porcentaje de descuento
Real	valDes	Valor descontado
Real	valPag	Valor por pagar
Entero	conCli10	Contador para cliente con descuento del 10 %
Entero	conCli15	Contador para cliente con descuento del 15 %
Entero	conCli20	Contador para cliente con descuento del 20 %
Entero	conCli25	Contador para cliente con descuento del 25 %
Entero	conCli30	Contador para cliente con descuento del 30 %
Entero	conCli0	Contador para cliente sin descuentos
Real	totConDes	Total de las compras con descuentos
Real	totSinDes	Total de las compras sin descuentos
Carácter	opc	Opción para continuar o no ingresando datos

Datos de entrada: ideCli, apeCli, nomCli, valCom, opc

Procesos parciales:

$porDes \leftarrow [0 \mid 10 \mid 15 \mid 20 \mid 25 \mid 30]$

$valDes \leftarrow (valCom * porDes)/100$

```
valPag ← valCom - valDes
totConDes ← 0
totConDes ← totConDes + valPag
totSinDes ← 0
totSinDes ← totSinDes + valCom
conCli10 ← 0
conCli10 ← conCli10 + 1
conCli15 ← 0
conCli15 ← conCli15 + 1
conCli20 ← 0
conCli20 ← conCli20 + 1
conCli25 ← 0
conCli25 ← conCli25 + 1
conCli30 ← 0
conCli30 ← conCli30 + 1
conCli0 ← 0
conCli0 ← conCli0 + 1
```

Procesos totales: N/A

Datos de salida parciales: porDes, valDes, valPag

Datos de salida totales: conCli10, conCli15, conCli20, conCli25, conCli30, conCli0, totConDes, totSinDes

Desarrollo del algoritmo en PSeInt

Lo primero es analizar los procesos parciales y totales para determinar las funciones que se deben realizar; para ello se tiene:

- a. Las variables tipo contador y acumulador deben ser inicializadas en 0 y, si existen constantes, también se inicializan. Se creará la función inicializar de tipo procedimiento con todos sus parámetros por referencia.

Para el proceso $porDes \leftarrow [0 \mid 10 \mid 15 \mid 20 \mid 25 \mid 30]$ se requiere una función que determine el porcentaje por descontar. Se creará la función **Porcentaje**.

Para el proceso $valDes \leftarrow (valCom * porDes)/100$ se creará la función **MultiplicarDividir**.

Para el proceso $valPag \leftarrow valCom - valDes$ se creará la función **Restar2Valores**.

b. Para los procesos:

```

totConDes ← totConDes + valPag
totSinDes ← totSinDes + valCom
ConCli10 ← conCli10 + 1
ConCli15 ← conCli15 + 1
ConCli20 ← conCli20 + 1
ConCli25 ← conCli25 + 1
ConCli30 ← conCli30 + 1
ConCli0 ← conCli0 + 1

```

Todos tienen la misma estructura. Las variables **totConDes** y **totSinDes** son de tipo real, pero las demás son de tipo entero; por lo tanto, se requieren dos funciones: **Sumar2ValoresReales** y **Sumar2ValoresEnteros**.

- c. Como se deben contar los clientes según el porcentaje de descuento obtenido, se realiza la función **ContarClientes**.
- d. En los datos de entrada se usarán las mismas funciones para la entrada de datos (**LeerValorEntero**, **LeerValorReal**, **LeerCadena**, **LeerCaracter**).
- e. Para los datos de salida parciales y totales se crearán las funciones **DatosSalidaParciales** y **DatosSalidaTotales**.

```

//Algoritmo principal
Algoritmo Ejercicio6C3

    //Definir o declarar las variables
    Definir ideCli, nomCli, apeCli Como Cadena
    Definir valCom, porDes, valDes, valPag, totConDes, totSinDes Como
        Real
    Definir conCli10, conCli15, conCli20, conCli25, conCli30, conCli0
        Como Entero
    Definir opc Como Caracter

    //Inicializar variables y constantes
    Inicializar(conCli10, conCli15, conCli20, conCli25, conCli30, conCli0,
        totConDes, totSinDes)

    Repetir
        Borrar Pantalla

        //Datos de entrada
        ideCli ← LeerCadena("Identificación del cliente: ")
        apeCli ← LeerCadena("Apellidos del cliente : ")
        nomCli ← LeerCadena("Nombres del cliente : ")

```

```
    valCom ← LeerValorReal("Valor compra  : ", 1000, 90000000)

    //Procesos parciales
    porDes ← Porcentaje(valCom)
    valDes ← MultiplicarDividir(valCom, porDes, 100)
    valPag ← Restar2Valores(valCom, valDes)
    totConDes ← Sumar2ValoresReales(totConDes, valPag)
    totSinDes ← Sumar2ValoresReales(totSinDes, valCom)
    ContarClientes(conCli0, conCli10, conCli15, conCli20, conCli25,
                  conCli30, porDes)

    //Datos de salida parciales
    DatosSalidaParciales(porDes, valDes, valPag)

    opc ← leerCaracter("Desea ingresar a otro cliente [S/N]? ",
                      "S", "N")

    Hasta Que Mayusculas(opc) = "N"

    //Datos de salida totales
    DatosSalidaTotales(conCli0, conCli10, conCli15, conCli20, conCli25,
                      conCli30, totConDes, totSinDes)

FinAlgoritmo

//Funciones para datos de entrada

//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer o capturar un valor real
Funcion val ← LeerValorReal(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer una cadena como nombres, direcciones, etc.
Funcion cad ← LeerCadena(msg) (ver Ejercicio1C3)

//Función para leer un carácter como S, N, F, M, etc.
Funcion car ← leerCaracter(msg, c1, c2) (ver Ejercicio1C3)

//Funciones adicionales requeridas según los procesos

//Función para inicializar variables tipo contador y acumulador
Funcion Inicializar(conCli10 Por Referencia, conCli15 Por Referencia,
                  conCli20 Por Referencia, conCli25 Por Referencia,
                  conCli30 Por Referencia, conCli0 Por Referencia,
                  totConDes PorReferencia, totSinDes Por Referencia)

    conCli10 ← 0
    conCli15 ← 0
    conCli20 ← 0
    conCli25 ← 0
```

```
conCli30 ← 0
conCli0 ← 0
totSinDes ← 0
totConDes ← 0
```

FinFuncion

Funcion porc ← Porcentaje(valCom)

```
  Definir porc Como Entero
```

```
  Si valCom < 100000 Entonces
```

```
    porc ← 0
```

```
  SiNo
```

```
    Si valCom < 200000 Entonces
```

```
      porc ← 10
```

```
    SiNo
```

```
      Si valCom < 300000 Entonces
```

```
        porc ← 15
```

```
      SiNo
```

```
        Si valCom < 400000 Entonces
```

```
          porc ← 20
```

```
        SiNo
```

```
          Si valCom < 500000 Entonces
```

```
            porc ← 25
```

```
          SiNo
```

```
            porc ← 30
```

```
          FinSi
```

```
        FinSi
```

```
      FinSi
```

```
    FinSi
```

```
  FinSi
```

FinFuncion

Funcion res ← MultiplicarDividir(val1, val2, val3)

```
  Definir res Como Real
```

```
  res ← (val1 * val2) / val3
```

FinFuncion

Funcion res ← Restar2Valores(val1, val2)

```
  Definir res Como Real
```

```
  res ← val1 - val2
```

FinFuncion

Funcion res ← Sumar2ValoresEnteros(val1, val2)

```
  Definir res Como Entero
```

```
  res ← val1 + val2
```

FinFuncion

```
Funcion res ← Sumar2ValoresReales(val1, val2)
```

```
    Definir res Como Real
```

```
    res ← val1 + val2
```

```
FinFuncion
```

```
Funcion ContarClientes(conCli0 Por Referencia, conCli10 Por Referencia,  
    conCli15 Por Referencia, conCli20 Por Referencia,  
    conCli25 Por Referencia, conCli30 Por Referencia,  
    porDes)
```

```
    Segun porDes Hacer
```

```
        0: conCli0 ← Sumar2ValoresEnteros(conCli0, 1)
```

```
        10: conCli10 ← Sumar2ValoresEnteros(conCli10, 1)
```

```
        15: conCli15 ← Sumar2ValoresEnteros(conCli15, 1)
```

```
        20: conCli20 ← Sumar2ValoresEnteros(conCli20, 1)
```

```
        25: conCli25 ← Sumar2ValoresEnteros(conCli25, 1)
```

```
        30: conCli30 ← Sumar2ValoresEnteros(conCli30, 1)
```

```
    FinSegun
```

```
FinFuncion
```

```
Funcion DatosSalidaParciales(porDes, valDes, valPag)
```

```
    Escribir ""
```

```
    Escribir "Porcentaje de descuento: ", Redon(porDes * 100) / 100
```

```
    Escribir ""
```

```
    Escribir "Valor descontado : ", Redon(valDes * 100) / 100
```

```
    Escribir ""
```

```
    Escribir "Valor para pagar : ", Redon(valPag * 100) / 100
```

```
FinFuncion
```

```
Funcion DatosSalidaTotales(conCli0, conCli10, conCli15, conCli20, conCli25,  
    conCli30, totConDes, totSinDes)
```

```
    Escribir ""
```

```
    Escribir "Clientes sin descuentos : ", conCli0
```

```
    Escribir "Clientes con 10% de descuentos: ", conCli10
```

```
    Escribir "Clientes con 15% de descuentos: ", conCli15
```

```
    Escribir "Clientes con 20% de descuentos: ", conCli20
```

```
    Escribir "Clientes con 25% de descuentos: ", conCli25
```

```
    Escribir "Clientes con 30% de descuentos: ", conCli30
```

```
    Escribir "Total compras con descuentos : ", totConDes
```

```
    Escribir "Total compras sin descuentos : ", totSinDes
```

```
FinFuncion
```


3.6 Ejercicios propuestos

Para los siguientes ejercicios, realizar el análisis del problema y desarrollar el algoritmo aplicando programación modular.

1. Realizar un algoritmo que permita ingresar N números enteros y determinar el promedio de estos; además, se desea conocer cuántos son pares y cuántos impares.
2. Realizar un algoritmo para ingresar N números enteros, y determinar cuántos son primos y cuántos no.
3. Una entidad les ofrece préstamos a sus clientes a una tasa de interés N mensual. Se requiere realizar un algoritmo que determine el valor que debe pagar mensualmente un cliente por su préstamo; además, se desea conocer el número de clientes a los cuales se les realizó préstamos, el total de dinero prestado, el total de dinero en intereses recibido en un mes y el total de capital luego de cancelados todos los intereses de cada uno de los préstamos en dicho periodo.
4. Una tienda está interesada en desarrollar un algoritmo que le permita registrar los datos del cliente, la fecha de compra, la descripción de cada uno de los productos adquiridos por él, la cantidad, el valor unitario y el porcentaje de IVA por un determinado artículo. Se desea conocer el valor total de la compra (un mismo cliente puede comprar uno o más productos en una misma compra), además de cuántos clientes se atendieron en un día y el valor total de las compras.
5. Los empleados de una fábrica trabajan en dos turnos: diurno y nocturno. Se desea calcular el jornal diario de acuerdo con los siguientes puntos:

La tarifa de las horas diurnas es de 5 dólares.

La tarifa de las horas nocturnas es de 8 dólares.

En caso de ser domingo, la tarifa se incrementará en 2 dólares para el turno diurno y en 3 dólares para el nocturno. Además, se desea conocer el total pagado a los empleados y el número de empleados procesados.

6. Se requiere un algoritmo que sirva para calcular la nota definitiva de los estudiantes, teniendo en cuenta que un mismo estudiante puede cursar varias materias y que cada una tiene tres notas (30 %, 30 % y 40 %, respectivamente). Adicionalmente, se desea conocer el promedio de cada estudiante, el promedio del grupo y el estudiante con mejor promedio.

Capítulo 4

ARREGLOS



En los capítulos anteriores se ha trabajado con los tipos de datos entero, real, cadena, carácter y lógico, conocidos como simples o primitivos. Frecuentemente, en el desarrollo de *software* se debe procesar una serie de datos o grupos de valores de forma simultánea, lo cual no es posible con datos simples, o se hace muy complejo con estos. Para facilitar el procesamiento de estas series o grupos surge el concepto de *arreglos*.

Un arreglo es una secuencia de posiciones de la memoria central a la que se puede acceder directamente, que contiene datos del mismo tipo y pueden ser seleccionados individualmente mediante el uso de subíndices (Joyanes Aguilar, 2020, p. 255).

Un arreglo puede definirse como un grupo o una colección finita, homogénea y ordenada de elementos (EcuRed, 2023, párr. 1).

Los arreglos son estructuras de datos homogéneas (todos sus datos son del mismo tipo) que permiten almacenar un determinado número de datos bajo un mismo identificador, para luego referirse a los mismos utilizando uno o más subíndices (Novara, 2003c, párr. 1).

Un arreglo es una estructura de datos con elementos homogéneos, del mismo tipo, numérico o alfanumérico, reconocidos por un nombre en común (Muñoz, 2010, párr. 3).

Entonces, un *arreglo* es un conjunto finito de elementos de un mismo tipo, asociados bajo un solo nombre y a los cuales se puede acceder a través de un subíndice.

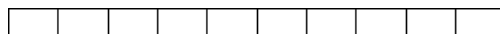
Un *subíndice* es una variable de tipo entero que ayuda a identificar la posición que ocupa un elemento específico dentro del arreglo.

Hay dos tipos de arreglos: los vectores y las matrices.

4.1 Vector

Es un arreglo de una sola dimensión, el cual puede ser representado gráficamente como se aprecia en la **figura 4.1**.

Figura 4.1: Representación gráfica de un vector



Se le debe asignar un nombre que identifique a todos sus elementos, por ejemplo, **sueBas** (sueldo básico), de forma gráfica, como se aprecia en la **figura 4.2**.

Figura 4.2: Asignación de un nombre a un vector

sueBas

--	--	--	--	--	--	--	--	--	--

Cada recuadro representa una posición en el vector. El vector **sueBas** tiene 10 elementos: el primero es la posición 0, el segundo es la posición 1, y así sucesivamente. Entonces, el vector **sueBas** tiene las posiciones 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, respectivamente (**figura 4.3**).

Figura 4.3: Representación gráfica de posiciones de los elementos en un vector

sueBas

0	1	2	3	4	5	6	7	8	9

Para usar el vector **sueBas** se debe, primero, definir o declarar la variable y, posteriormente, indicar la dimensión, es decir, el número de elementos que va a contener. Por ejemplo, se desea almacenar los sueldos básicos de 10 empleados; en este caso, el tipo de dato es real y la dimensión es de 10, así:

Algoritmo Ejemplo_Vector

```
Definir sueBas Como Real
Dimension sueBas[10]
```

FinAlgoritmo

Se le puede asignar un sueldo a cada posición del vector (**figura 4.4**).

Figura 4.4: Representación gráfica de asignación de valores a un vector

sueBas

0	1	2	3	4	5	6	7	8	9
1200,0	1500,0	1800,0	2000,0	1750,0	1950,0	1870,0	1350,0	1480,0	1670,0

Para acceder a cada uno de los elementos del vector **sueBas**, se requiere de una variable llamada *subíndice de tipo entero*, que permite tener acceso a cada uno de los elementos del vector; por ejemplo:

Si $I = 5$, ¿qué valor tiene `sueBas[I]`? De acuerdo con la imagen, en la posición 5 el vector `sueBas` tiene el valor de 1950.0.

Si $I = 2$, ¿qué valor tiene `sueBas[I]`?, `sueBas[I] = 1800.0`

Si $I = 7$, ¿qué valor tiene `sueBas[I]`?, `sueBas[I] = 1350.0`

Para asignarle valor a un vector, se usa la sentencia **Leer** indicando el nombre del vector y, entre corchetes, el subíndice que denota la posición donde se va a almacenar

el valor. Para asignarle valores al vector `sueBas`, se puede recurrir a un ciclo **PARA** con la variable `i`, que sería el subíndice en una función; por ejemplo:

```
Para i ← 0 Hasta 9 Hacer
    Leer sueBas[i]
FinPara
```

De esta forma se le asignan valores al vector `sueBas` iniciando en la posición 0 y hasta la 9. Pero para hacerlo más comprensible al usuario, se puede acompañar de un mensaje indicativo:

```
Para i ← 0 Hasta 9 Hacer
    Escribir Sin Saltar \Ingrese el sueldo no. ", i
    Leer sueBas[i]
FinPara
```

También se puede recorrer el vector para visualizar su contenido a través de la sentencia **Escribir**; por ejemplo:

```
Para i ← 0 Hasta 9 Hacer
    Escribir sueBas[i]
FinPara
```

1. A continuación, se realiza un algoritmo para crear el vector **sueBas** de cinco elementos, asignarle valores y visualizar su contenido. El algoritmo se realiza aplicando el concepto de *modularidad*, es decir, se va a crear una función para asignarle los valores al vector y otra para visualizar su contenido. Como observación general, cuando se pasa un vector como parámetro a una función, pasa como parámetro por referencia, pero no hay que especificarlo.

Los pasos para realizar el algoritmo son:

- a. Definir las variables requeridas, en este caso **sueBas**, para el nombre del vector de tipo real.
- b. Dimensionar el vector.
- c. Crear la función para llenar o asignarle valor al vector.
- d. Crear la función para visualizar su contenido.
- e. Llamar o invocar desde el algoritmo principal las funciones para asignarle valores al vector y visualizar su contenido.

Algoritmo Ejercicio1C4

```
//Definir o declarar la variable
Definir sueBas Como Real

//Dimensionar el vector
Dimension sueBas[5]

//Asignarle valores al vector
LlenarVector(sueBas)

//visualizar su contenido
MostrarVector(sueBas)
```

FinAlgoritmo**Funcion LlenarVector(A)**

```
Definir i Como Entero
Para i ← 0 Hasta 4 Hacer
    Escribir Sin Saltar "Sueldo básico en ", i, ": "
    Leer A[i]
FinPara
```

FinFuncion**Funcion MostrarVector(A)**

```
Definir i Como Entero
Borrar Pantalla
Escribir "El contenido del vector es:"
Para i ← 0 Hasta 4 Hacer
    Escribir "Sueldo básico en ", i, ": ", A[i]
FinPara
```

FinFuncion

Al ejecutar el algoritmo, si el usuario ingresa los valores 1000, 2000, 3000, 5000, 4000, obtiene el resultado de la **figura 4.5**.

Figura 4.5: Ejecución Ejercicio1C4

```
*** Ejecución Iniciada. ***
Sueldo básico en 0: > 1000
Sueldo básico en 1: > 2000
Sueldo básico en 2: > 3000
Sueldo básico en 3: > 5000
Sueldo básico en 4: > 4000
El contenido del vector es:
Sueldo básico en 0: 1000
Sueldo básico en 1: 2000
Sueldo básico en 2: 3000
Sueldo básico en 3: 5000
Sueldo básico en 4: 4000
*** Ejecución Finalizada. ***
```

Es posible hacer uso de las funciones para entrada de datos vistas en el capítulo anterior con el propósito de validar los diversos valores asignados a un vector según su tipo de datos; por ejemplo, asignarle sueldo que no sea inferior a 1000 ni superior a 10000 al vector **sueBas**; en este caso, el algoritmo queda como sigue:

Algoritmo Ejercicio2C4

```
//Definir o declarar la variable
Definir sueBas Como Real

//Dimensionar el vector
Dimension sueBas[5]

//Asignarle valor al vector
LlenarVector(sueBas)

//visualizar su contenido
MostrarVector(sueBas)
```

FinAlgoritmo

```
//Funciones para datos de entrada

//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer o capturar un valor real
Funcion val ← LeerValorReal(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer una cadena como nombres, direcciones, etc.
Funcion cad ← LeerCadena(msg) (ver Ejercicio1C3)

//Función para leer un carácter como S, N, F, M, etc.
Funcion car ← leerCaracter(msg, c1, c2) (ver Ejercicio1C3)

//Funciones adicionales

//Función para llenar el vector o asignarle valores
Funcion LlenarVector(A) (ver Ejercicio1C4)
Funcion MostrarVector(A) (ver Ejercicio1C4)
```

Observaciones

Nótese que en la función **LlenarVector(A)**, en el cuerpo del ciclo **Para – Fin-Para**, se invoca la función **LlenarVector** desde la línea de código:

```
A[i] ← LeerValorReal(“Sueldo básico ”, 1000, 10000)
```

Esto permite validar que el sueldo ingresado no sea inferior a 1000 ni superior a 10000.

Al ejecutar el algoritmo e ingresar los sueldos 8000, 5000, 4000, 2000, 7000, se obtiene la **figura 4.6**.

Figura 4.6: Ejecución Ejercicio2C4

```

*** Ejecución Iniciada. ***

Sueldo básico > 8000

Sueldo básico > 5000

Sueldo básico > 4000

Sueldo básico > 2000

Sueldo básico > 7000

El contenido del vector es:
Sueldo básico en 0: 8000
Sueldo básico en 1: 5000
Sueldo básico en 2: 4000
Sueldo básico en 3: 2000
Sueldo básico en 4: 7000
*** Ejecución Finalizada. ***

```

Es posible trabajar al mismo tiempo con dos o más vectores de forma simultánea para almacenar datos de distintos tipos y ejecutar algún proceso con ellos.

2. Realizar un algoritmo para almacenar en vectores los datos de los estudiantes del curso de Lógica de Programación, así como sus notas, y determinar la nota definitiva teniendo en cuenta los siguientes ítems de calificación:

Primer parcial: 20 %

Segundo parcial: 20 %

Trabajo práctico: 35 %

Parcial final: 25 %

Se desea conocer el promedio del curso y cuántas mujeres y hombres hay en el grupo.

Análisis del problema

Tabla 4.1: Identificación de variables Ejercicio3C4

Tipo	Variable	Descripción
Cadena	ideEst	Identificación del estudiante
Cadena	apeEst	Apellidos del estudiante

Tipo	Variable	Descripción
Cadena	nomEst	Nombres del estudiante
Carácter	sexEst	Sexo del estudiante
Real	priPar	Primer parcial
Real	segPar	Segundo parcial
Real	traPra	Trabajo práctico
Real	parFin	Parcial final
Real	notDef	Nota definitiva
Real	proGru	Promedio del grupo
Entero	nroEst	Número de estudiantes del curso
Real	sumNot	Sumatoria de notas definitivas
Entero	nroMuj	Número de mujeres
Entero	nroHom	Número de hombres

Datos de entrada: ideEst, apeEst, nomEst, priPar, segPar, traPra, parFin, nroEst

Procesos parciales:

$$\begin{aligned} notDef \leftarrow & ((priPar * 20)/100) + ((segPar * 20)/100) \\ & + ((traPra * 35)/100) + ((parFin * 25)/100) \end{aligned}$$

$$sumNot \leftarrow 0$$

$$sumNot \leftarrow sumNot + notDef$$

$$nroMuj \leftarrow 0$$

$$nroMuj \leftarrow nroMuj + 1$$

$$nroHom \leftarrow 0$$

$$nroHom \leftarrow nroHom + 1$$

Procesos totales:

$$nroEst \leftarrow nroMuj + nroHom$$

$$proGru \leftarrow sumNot/nroEst$$

Datos de salida parciales: notDef

Datos de salida totales: nroEst, proGrup, nroMuj, nroHom

Desarrollo del algoritmo en PSeInt

Los pasos para seguir son:

- a. Definir o declarar las variables requeridas.

- b. Solicitar el número de estudiantes por procesar.
- c. Dimensionar los distintos vectores de acuerdo con el número de estudiantes del curso.
- d. Inicializar las variables de tipo contador y acumulador.
- e. Asignarles valores a los vectores (datos de entrada).
- f. Calcular la nota definitiva de cada estudiante.
- g. Determinar el número de mujeres y hombres del curso.
- h. Calcular el promedio del grupo.
- i. Visualizar la información.

Algoritmo Ejercicio3C4

```
//Definir o declarar la variable
Definir ideEst, apeEst, nomEst Como Cadena
Definir sexEst Como Caracter
Definir priPar, segPar, traPra, parFin, notDef, proGru Como Real
Definir nroEst, nroMuj, nroHom Como Entero

//Obtener el número de estudiantes del curso Lógica de Programación
nroEst ← LeerValorEntero("Número de estudiantes: ", 1, 50)

//Dimensionar los vectores
Dimension ideEst[nroEst], apeEst[nroEst], nomEst[nroEst], sexEst[nroEst]
Dimension priPar[nroEst], segPar[nroEst], traPra[nroEst], parFin[nroEst],
        notDef[nroEst]

//Inicializar variables de tipo contador y acumulador
Inicializar(nroMuj, nroHom)

//Asignarles valor a los vectores
LlenarVectores(ideEst, apeEst, nomEst, sexEst, priPar, segPar, traPra,
        parFin, nroEst)

//Calcular la nota definitiva de cada estudiante
CalcularNotaDefinitiva(priPar, segPar, traPra, parFin, notDef, nroEst)

//Determinar el número de mujeres y hombres del curso Lógica de
//Programación
ContarMujeresHombres(sexEst, nroMuj, nroHom, nroEst)

//Determinar el promedio del grupo
proGru ← CalcularPromedioGrupo(notDef, nroEst)
```

```

//visualizar la información
MostrarVectores(ideEst, apeEst, nomEst, sexEst, priPar, segPar, traPra,
                parFin, notDef, nroMuj, nroHom, proGru, nroEst)

FinAlgoritmo

//Funciones para datos de entrada
//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer o capturar un valor real
Funcion val ← LeerValorReal(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer una cadena como nombres, direcciones, etc.
Funcion cad ← LeerCadena(msg) (ver Ejercicio1C3)

//Función para leer un carácter como S, N, F, M, etc.
Funcion car ← leerCaracter(msg, c1, c2) (ver Ejercicio1C3)

//Funciones adicionales

//Inicializar variables de tipo contador y acumulador
Funcion Inicializar(nroMuj Por Referencia, nroHom Por Referencia)
    nroMuj ← 0
    nroHom ← 0
FinFuncion

//Función para llenar los vectores o asignarles valores (datos de entrada)
Funcion LlenarVectores(ideEst, apeEst, nomEst, sexEst, priPar, segPar,
                    traPra, parFin, nroEst)

    Definir i Como Entero

    Para i ← 0 Hasta nroEst -1 Hacer
        Borrar Pantalla
        Escribir "Estudiante nro. ", i+1
        ideEst[i] ← LeerCadena("Identificación del estudiante: ")
        apeEst[i] ← LeerCadena("Apellidos del estudiante : ")
        nomEst[i] ← LeerCadena("Nombres del estudiante : ")
        sexEst[i] ← leerCaracter("Sexo del estudiante : ", "F", "M")
        priPar[i] ← LeerValorReal("Primer parcial : ", 0, 5)
        segPar[i] ← LeerValorReal("Segundo parcial : ", 0, 5)
        traPra[i] ← LeerValorReal("Trabajo práctico : ", 0, 5)
        parFin[i] ← LeerValorReal("Parcial final : ", 0, 5)
    FinPara
FinFuncion

//Función para calcular la nota definitiva de cada estudiante
Funcion CalcularNotaDefinitiva(priPar, segPar, traPra, parFin, notDef, nroEst)

```

```
    Definir i Como Entero

    Para i ← 0 Hasta nroEst -1 Hacer
        notDef[i] ← ((priPar[i] * 20) / 100) + ((segPar[i] * 20) / 100) +
                    ((traPra[i] * 35) / 100) + ((parFin[i] * 25) / 100)
    FinPara
FinFuncion

//Función para sumar dos valores
Funcion res ← Sumar2Valores(val1, val2)
    Definir res Como Entero

    res ← val1 + val2
FinFuncion

//Función para determinar el número de mujeres y hombres del curso Lógica de
//Programación
Funcion ContarMujeresHombres(sexEst, nroMuj Por Referencia, nroHom Por
                             Referencia, nroEst)
    Definir i Como Entero

    Para i ← 0 Hasta nroEst -1 Hacer

        Si Mayusculas(sexEst[I]) = "F" Entonces
            nroMuj ← Sumar2Valores(nroMuj, 1)
        SiNo
            nroHom ← Sumar2Valores(nroHom, 1)
        FinSi
    FinPara
FinFuncion

//Función para calcular el promedio del grupo
Funcion proG ← CalcularPromedioGrupo(notDef, nroEst)
    Definir proG, sumNot Como Real
    Definir i Como Entero

    sumNot ← 0

    Para i ← 0 Hasta nroEst -1 Hacer
        sumNot ← sumNot + notDef[I]
    FinPara

    proG ← sumNot/nroEst
FinFuncion

//Función para visualizar la información
Funcion MostrarVectores(ideEst, apeEst, nomEst, sexEst, priPar, segPar,
                        traPra, parFin, notDef, nroMuj, nroHom,
```

```

        proGru, nroEst)

Definir I Como Entero

Borrar Pantalla
Escribir "*** PLANILLA DE CALIFICACIONES ***"
Escribir ""
Escribir " IDE APE NOM SEX PP SP PRA PF NF"

Para i ← 0 Hasta nroEst -1 Hacer
    Escribir " ",ideEst[i], " ", apeEst[i], " ", nomEst[i], " ",
            sexEst[i], " ", priPar[i], " ", segPar[i], " ",
            traPra[i], " ", parFin[i], " ", notDef[i]
FinPara

Escribir ""
Escribir " Promedio del grupo: ", Redon(proGru * 100) / 100
Escribir " Número de mujeres : ", nroMuj
Escribir " Número de hombres : ", nroHom

FinFuncion

```

Observaciones

En este algoritmo se han dimensionado los vectores de forma dinámica para que en tiempo de ejecución se les asigne el tamaño o número de elementos que van a contener. Así, en cada ejecución se pueden asignar distintos tamaños o dimensiones a los vectores, lo cual facilita el que se ajusten a cualquier número de estudiantes, sin tener que modificar ninguna línea de código.

```
nroEst ← LeerValorEntero("Número de estudiantes: ", 1, 50)
```

```
Dimension ideEst[nroEst], apeEst[nroEst], nomEst[nroEst], sexEst[nroEst]
Dimension priPar[nroEst], segPar[nroEst], traPra[nroEst], parFin[nroEst],
        notDef[nroEst]
```

Otro aspecto para considerar es que los ciclos **PARA** van de 0 hasta nroEst -1 (Para i ← 0 Hasta nroEst -1 Hacer); y esto es así porque la primera posición de un vector es la 0, por lo cual, si se dice que el curso tiene 10 estudiantes, entonces las posiciones de los vectores son 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9.

Cuando se utilizan vectores, se ingresan todos los datos de entrada requeridos y, posteriormente, se ejecutan los diferentes procesos, ya que los datos permanecen en memoria mientras se esté ejecutando el algoritmo. En cambio, en los ejercicios de los capítulos anteriores, se debía ir procesando cada entrada

de datos porque se usaban variables simples. Es esta una de las ventajas de trabajar con arreglos.

Se hace una sola visualización de información, incluyendo los datos de entrada, para tener una vista completa de toda la información; es decir, tanto los datos de entrada como los de salida en una sola vista. Gracias a esto el usuario puede observar un panorama general de la información disponible, tal y como lo muestran las **figuras 4.7, 4.8, 4.9, 4.10, 4.11, 4.12 y 4.13**.

Figura 4.7: Inicio ejecución Ejercicio3C4

```
*** Ejecución Iniciada. ***  
  
Número de estudiantes: > 5
```

Figura 4.8: Ingreso de los datos del estudiante 1

```
Estudiante nro. 1  
  
Identificación del estudiante: > 111  
  
Apellidos del estudiante      : > AAA  
  
Nombres del estudiante       : > BBB  
  
Sexo del estudiante          : > F  
  
Primer parcial               : > 4  
  
Segundo parcial              : > 5  
  
Trabajo práctico             : > 3  
  
Parcial final                 : > 3
```

Figura 4.9: Ingreso de los datos del estudiante 2

```
Estudiante nro. 2  
  
Identificación del estudiante: > 222  
  
Apellidos del estudiante      : > CCC  
  
Nombres del estudiante       : > DDD  
  
Sexo del estudiante          : > F  
  
Primer parcial               : > 5  
  
Segundo parcial              : > 2  
  
Trabajo práctico             : > 4  
  
Parcial final                 : > 3
```

Figura 4.10: Ingreso de los datos del estudiante 3

```
Estudiante nro. 3
Identificación del estudiante: > 333
Apellidos del estudiante      : > EEE
Nombres del estudiante       : > FFF
Sexo del estudiante          : > M
Primer parcial               : > 5
Segundo parcial              : > 2
Trabajo práctico             : > 3
Parcial final                : > 4
```

Figura 4.11: Ingreso de los datos del estudiante 4

```
Estudiante nro. 4
Identificación del estudiante: > 444
Apellidos del estudiante      : > III
Nombres del estudiante       : > JJJ
Sexo del estudiante          : > M
Primer parcial               : > 4
Segundo parcial              : > 5
Trabajo práctico             : > 3
Parcial final                : > 4
```

Figura 4.12: Ingreso de los datos del estudiante 5

```
Estudiante nro. 5
Identificación del estudiante: > 555
Apellidos del estudiante      : > PPP
Nombres del estudiante       : > KKK
Sexo del estudiante          : > M
Primer parcial               : > 3
Segundo parcial              : > 4
Trabajo práctico             : > 5
Parcial final                : > 4
```

Figura 4.13: Datos de salida Ejercicio3C4

```

*** PLANILLA DE CALIFICACIONES ***

IDE  APE  NOM  SEX  PP  SP  PRA  PF  NF
111  AAA  BBB  F    4   5   3   3  3.6
222  CCC  DDD  F    5   2   4   3  3.55
333  EEE  FFF  M    5   2   3   4  3.45
444  III  JJJ  M    4   5   3   4  3.85
555  PPP  KKK  M    3   4   5   4  4.15

Promedio del grupo: 3.72
Número de mujeres : 2
Número de hombres : 3
*** Ejecución Finalizada. ***

```

Se sugiere, a manera de práctica, desarrollar los demás ejercicios del capítulo 3 con manejo de vectores y aplicando programación modular.

Se ha visto ya cómo trabajar con distintos vectores a la vez, con un proceso de asignación de valores con datos de entrada a varios vectores. También, la asignación de valores mediante procesos (cálculo matemático) al vector **notDef**, cuyo resultado implicó manipular distintos vectores al mismo tiempo: $(\text{notDef}[i] \leftarrow ((\text{priPar}[i] * 20) / 100) + ((\text{segPar}[i] * 20) / 100) + ((\text{traPra}[i] * 35) / 100) + (\text{parFin}[i] * 25) / 100)$; y el recorrido simultáneo de los diferentes vectores para visualizar información. Pero hay otro tipo de operaciones que se pueden realizar con un vector, como, por ejemplo, agregarle o insertarle un elemento a un vector, siempre y cuando no se haya alcanzado el número máximo de elementos de este, o suprimir un elemento, ordenar un vector y hacer una búsqueda de un dato en particular.

4.2 Insertar un elemento en un vector

Para insertar un elemento en un vector se debe verificar que no se haya alcanzado el número máximo de elementos que puede contener (tal máximo es indicado al momento de dimensionarlo). Hay tres posibilidades para insertar un elemento en un vector: al principio, en una posición específica o al final.

3. Para el ejemplo se va a crear un vector de N elementos de tipo entero, donde N se inicializa en 10, aunque puede ser cualquier valor. Va a tener un menú de opciones para llenar el vector, insertar un elemento al final, insertar un elemento al

principio, insertar un elemento en una posición específica, visualizar el contenido del vector y salir. Se usan las mismas funciones de entrada de datos explicadas en el capítulo III.

Para **insertar un elemento al final** del vector se debe tener en cuenta que no se haya alcanzado el número máximo de elementos que puede contener. En este ejemplo son 10, en cuyo caso, si se llega a la posición (subíndice) 9, se habrá llegado al límite y no será posible seguir insertando más elementos (**figura 4.14**).

Figura 4.14: Representación gráfica de un vector de 10 elementos lleno

	0	1	2	3	4	5	6	7	8	9
vecEnt	50	70	90	10	15	25	65	85	45	80

Si, por ejemplo, hay cinco elementos, entonces se puede insertar la posición 5, el valor 25 (**figura 4.15**).

Figura 4.15: Vector con cinco elementos, pero con capacidad para almacenar diez

	0	1	2	3	4	5	6	7	8	9
vecEnt	50	70	90	10	15					

Por lo que el valor 25 se inserta en la posición 5 (**figura 4.16**).

Figura 4.16: Inserción de un elemento al final

	0	1	2	3	4	5	6	7	8	9
vecEnt	50	70	90	10	15	25				

Para **insertar un elemento al principio del vector**, es decir, posición 0, se deben tener en cuenta varios aspectos:

- Validar que no se haya alcanzado el número máximo de elementos que puede almacenar el vector.
- Validar que al menos haya uno o más elementos en el vector, cada uno de los cuales se debe correr una posición hacia adelante; por ejemplo, si hay tres elementos, como en la **figura 4.17**.

Figura 4.17: Vector con tres elementos, pero con capacidad para almacenar diez

	0	1	2	3	4	5	6	7	8	9
vecEnt	50	70	90							

El 90 pasa a la posición 3, el 70 a la 2 y el 50 a la 1, en tanto la posición 0 queda libre para insertar el elemento (**figura 4.18**).

Figura 4.18: Desplazamiento de los elementos de un vector una posición a la derecha

	0	1	2	3	4	5	6	7	8	9
vecEnt		50	70	90						

- Insertar el nuevo elemento en la posición 0: el 40, por ejemplo (**figura 4.19**).

Figura 4.19: Inserción de un elemento al principio de un vector

	0	1	2	3	4	5	6	7	8	9
vecEnt		50	70	90						

- Otra posibilidad es que el vector no contenga elementos: en ese caso, se inserta directamente la posición 0.

Para **insertar en una posición específica del vector**, ha de tenerse en cuenta:

- Validar que no se haya alcanzado el número máximo de elementos que puede almacenar el vector.
- Validar si la posición de inserción es la 0, en cuyo caso se sigue el procedimiento descrito para insertar al principio.
- Otra alternativa es validar que la posición de inserción sea la posición que ocupa el último elemento en el vector, en cuyo caso se corre una hacia adelante y se inserta el elemento en la indicada (**figura 4.20**).

Figura 4.20: Vector con cuatro elementos, pero con capacidad para almacenar diez

	0	1	2	3	4	5	6	7	8	9
vecEnt	40	50	70	90						

Se desea insertar el valor 30 en la posición 3, en cuyo caso el 90 pasa a la posición 4 y el 30 queda en la 3 (**figura 4.21**).

Figura 4.21: Inserción de un elemento en una posición específica del vector

	0	1	2	3	4	5	6	7	8	9
vecEnt	40	50	70	30	90					

- Otra alternativa es que no sea ni la primera ni la posición que ocupa el último elemento en el vector, en cuyo caso se debe correr cada elemento una posición hacia adelante hasta aquella donde se va a insertar (**figura 4.22**).

Figura 4.22: Vector con cinco elementos, pero con capacidad para almacenar diez

	0	1	2	3	4	5	6	7	8	9
vecEnt	40	50	70	30	90					

Se desea insertar el valor 80 en la posición 2; entonces, el 90 pasa a la 5, el 30 a la 4 y el 70 a la 3 (**figura 4.23**).

Figura 4.23: Desplazar los elementos de un vector una posición a la derecha a partir de una posición específica

	0	1	2	3	4	5	6	7	8	9
vecEnt	40	50		70	30	90				

Seguidamente se inserta el valor 80 en la posición 2 (**figura 4.24**).

Figura 4.24: Insertar un elemento en una posición específica del vector

	0	1	2	3	4	5	6	7	8	9
vecEnt	40	50	80	70	30	90				

A continuación, presentamos el algoritmo:

Algoritmo Ejercicio4C4

```

Definir vecEnt, tope, pos, opcMen, n Como Entero

//Dimensionar el vector
n ← 10 //Número máximo de elementos del vector vecEnt
Dimension vecEnt[n]

//Cuando se ejecuta tope, se inicia en 0 porque no hay elementos en el
//vector
tope ← 0

Repetir
  Menu() //Visualiza el menú de opciones
  //El usuario selecciona una opción entre 1 y 5
  opcMen ← LeerValorEntero("Seleccione una opción: ", 0, 5)

  //Se valida la opción seleccionada por el usuario
  Segun opcMen Hacer
    1: tope ← LlenarVector(vecEnt, n, tope)

```

```
        2: tope ← InsertarFinal(vecEnt, tope, n)
        3: tope ← InsertarPrincipio(vecEnt, tope, n)
        4: tope ← InsertarPosicion(vecEnt, tope, n)
        5: MostrarVector(vecEnt, tope)

    FinSegun

Hasta Que opcMen = 0

FinAlgoritmo

//Funciones para datos de entrada

//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer o capturar un valor real
Funcion val ← LeerValorReal(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer una cadena como nombres, direcciones, etc.
Funcion cad ← LeerCadena(msg) (ver Ejercicio1C3)

//Función para leer un carácter como S, N, F, M, etc.
Funcion car ← leerCaracter(msg, c1, c2) (ver Ejercicio1C3)

//Funciones adicionales

//Función para visualizar el menú de opciones
Funcion Menu()
    Borrar Pantalla
    Escribir "*** MENÚ DE OPCIONES ***"
    Escribir ""
    Escribir "1. Llenar el vector"
    Escribir "2. Insertar un elemento al final"
    Escribir "3. Insertar un elemento al principio"
    Escribir "4. Insertar un elemento en una posición específica"
    Escribir "5. Mostrar vector"
    Escribir "0. Salir"
FinFuncion

//Función para asignarle valores al vector
Funcion tope ← LlenarVector(A, n, ind)
    //i es el subíndice para las posiciones en el vector
    Definir i, tope Como Entero
    Definir cad Como Cadena
    Definir opc Como Caracter
    Definir sw Como Logico

    Si (ind = 0) y A[ind] = 0 Entonces
        i ← 0
        sw ← Verdadero
    Mientras sw
```

```

//Se le asigna a la variable cad el mensaje Valor
//concatenado o unido con lo que vale i + 1
//ConvertirATexto convierte una expresión a cadena o texto
//Concatenar para unir 2 cadenas en una sola
cad ← Concatenar("Valor ", ConvertirATexto(i+1))

A[i] ← LeerValorEntero(cad, 1, 100)
opc ← leerCaracter("¿Ingresa un nuevo valor [S/N]? ", "S",
                  "N")

//Se valida que el usuario haya respondido con S y que no se
//haya alcanzado el número máximo de elementos n-1, es decir,
//10 - 1 = 9, recordando que el vector puede contener hasta
//10 elementos iniciando en la posición 0
Si (Mayusculas(opc) = \S") Y (i < n-1) Entonces
    i ← i + 1
SiNo
    tope ← i
    sw ← Falso
    Si (i = n-1) Y Mayusculas(opc) = \S" Entonces
        Mensaje2()
    FinSi
FinSi
FinMientras
SiNo
    Mensaje1()
    tope ← ind
FinSi
FinFuncion

//Función para visualizar el contenido del vector
Funcion MostrarVector(A, tope)
    Definir i Como Entero

    Borrar Pantalla
    Si (tope = 0) Y (A[tope] = 0) Entonces
        Mensaje3()
    SiNo

        Escribir "El contenido del vector es:"

        Para i ← 0 Hasta tope Hacer
            Escribir "A[" , i , "]" = " , A[i]
        FinPara

        Escribir ""
        Escribir "Presione una tecla para continuar"

```

Esperar Tecla

FinSi

FinFuncion

```
//Función para insertar un elemento al final del vector
//A es el vector, i recibe tope, la posición del último elemento
//n es la dimensión del vector, es decir, el tamaño o número de elementos que
//puede contener
```

```
Funcion tam ← InsertarFinal(A, i, n)
```

```
Definir tam Como Entero
```

```
Definir cad Como Cadena
```

```
//Se valida que no se haya alcanzado el número máximo de elementos
```

```
Si i < n-1 Entonces
```

```
    //Se valida que haya 1 0 más elementos en el vector
```

```
    Si A[i] > 0 Entonces
```

```
        i ← i+1
```

```
    FinSi
```

```
    cad ← Concatenar("Valor ", ConvertirATexto(i+1))
```

```
    A[i] ← LeerValorEntero(cad, 1, 100)
```

```
SiNo
```

```
    //Visualizar un mensaje de alerta donde se indica que se alcanzó el
```

```
    //número máximo de elementos permitidos
```

```
    Mensaje2()
```

```
FinSi
```

```
tam ← i
```

FinFuncion

```
//Función para insertar un elemento al principio del vector
```

```
Funcion tam ← InsertarPrincipio(A, i, n)
```

```
Definir tam, j Como Entero
```

```
j ← i //Se guarda la posición del último elemento en j
```

```
//Se valida que no se haya alcanzado el número máximo de elementos
```

```
Si i < n-1 Entonces
```

```
    //Se valida que haya 1 0 más elementos en el vector para
```

```
    //incrementar i en 1
```

```
    Si A[i] > 0 Entonces
```

```
        //Se corre cada elemento del vector una posición hacia
```

```
        //adelante para dejar la posición 0 libre
```

```
        Mientras i >= 0 Hacer
```

```
            A[i+1] ← A[i]
```

```
            i ← i -1
```

```
        FinMientras
```

```

//Se inserta el elemento en la posición 0
A[i+1] ← LeerValorEntero("Digite el valor a insertar: ", 1, 100)
tam ← j+1 //Se guarda la posición del último elemento

SiNo
//Se inserta el elemento en la posición i = 0
A[i] ← LeerValorEntero("Digite el valor a insertar: ", 1,100)
tam ← i
FinSi

SiNo
Mensaje2()
tam ← i
FinSi

FinFuncion

//Función para insertar un elemento en una posición específica del vector
Funcion tam ← InsertarPosicion(A, i, n)
Definir tam, j, pos Como Entero

j ← i //Se guarda la posición del último elemento n j
Si i = n-1 Entonces
Mensaje2()
SiNo
//Se captura la posición para insertar un nuevo elemento
pos ← LeerValorEntero("Posición para insertar el elemento: ", 0, i)

Si (pos = 0) Entonces
tam ← InsertarPrincipio(A, i, n)
SiNo
//Se valida que la posición p sea igual a i, es decir, al
//tope, el último elemento se corre una posición adelante
//se inserta el elemento nuevo en la posición i, se
//incrementa el tamaño (tope) en 1
Si (pos = i) Entonces
A[i+1] ← A[i]
A[i] ← LeerValorEntero("Digite el valor a insertar: ",
1, 100)

tam ← i+1
SiNo
//Se corre cada elemento del vector una posición hacia
//adelante hasta la posición a insertar
//iniciando del último elemento hacia atrás
Mientras i >= pos Hacer
A[i+1] ← A[i]
i ← i - 1
FinMientras

```

```
        //Se inserta el elemento en la posición p
        A[pos] ← LeerValorEntero("Digite el valor a insertar:",
                                1, 100)
        //Se incrementa en 1 el tamaño del vector por el nuevo
        //elemento insertado
        tam ← j+1
    FinSi
FinSi
FinSi
FinFuncion

//Mensaje para indicarle al usuario que el vector ya contiene elementos
Funcion Mensaje1()
    Escribir ""
    Escribir "El vector ya contiene elementos"
    Escribir "Presione una tecla para continuar..."
    Esperar Tecla
FinFuncion

//Función para indicarle al usuario que alcanzó el número máximo de elementos permitidos
en el vector
Funcion Mensaje2()
    Escribir ""
    Escribir "*** ERROR *** se alcanzó el número máximo de elementos
                permitidos"
    Escribir "Presione una tecla para continuar..."
    Esperar Tecla
FinFuncion

Funcion Mensaje3()
    Escribir ""
    Escribir "El vector no contiene elementos"
    Escribir "Presione una tecla para continuar..."
    Esperar Tecla
FinFuncion
```

4.3 Eliminar un elemento de un vector

Para eliminar un elemento en un vector hay tres posibilidades: eliminar el último elemento, eliminar el primer elemento o eliminar un elemento en una posición específica.

4. Para el ejemplo se va a crear un vector de N elementos de tipo entero, donde N se inicializa en 10, aunque puede ser cualquier valor. Va a tener un menú de opciones para llenar el vector, eliminar un elemento al final, eliminar un elemento al principio, eliminar un elemento en una posición específica, visualizar el contenido del vector y salir. Se usan las mismas funciones de entrada de datos explicadas en el capítulo III.

Para **eliminar un elemento al final del vector** se debe verificar que no esté vacío y que solo tenga un elemento (**figura 4.25**).

Figura 4.25: Vector con un elemento, pero con capacidad para diez

	0	1	2	3	4	5	6	7	8	9
sueBas	10									

Se le asigna 0 en la posición 0 y al tope (**figura 4.26**).

Figura 4.26: Eliminación de un elemento al final del vector con un solo elemento

	0	1	2	3	4	5	6	7	8	9
sueBas	0									

Si tiene más de un elemento, al último se le asigna 0 y se decrementa el tope en 1 (**figura 4.27**).

Figura 4.27: Vector con seis elementos, pero con capacidad para diez

	0	1	2	3	4	5	6	7	8	9
vecEnt	40	50	80	70	30	90				

A la posición 5 se le asigna 0 y tope queda en 4 (**figura 4.28**).

Figura 4.28: Eliminación del último elemento de un vector con dos o más elementos

	0	1	2	3	4	5	6	7	8	9
vecEnt	40	50	80	70	30	0				

Para **eliminar un elemento al principio del vector** se valida que no esté vacío; si el vector tiene un solo elemento, se sigue el mismo procedimiento descrito anteriormente.

Si el vector tiene dos o más elementos, se corre cada elemento una posición hacia atrás desde la posición 1 hasta la última posición que contenga elementos en el vector (**figura 4.29**).

Figura 4.29: Vector con seis elementos, pero con capacidad para diez

	0	1	2	3	4	5	6	7	8	9
vecEnt	40	50	80	70	30	90				

En el ejemplo, el 50 pasa a la posición 0, el 80 a la 1, el 70 a la 2, el 30 a la 3 y el 90 a la 4; finalmente, a la posición que ocupaba el último elemento se le asigna 0, y se decrementa el tope en 1 (**figura 4.30**).

Figura 4.30: Eliminación del primer elemento de un vector con dos o más elementos

	0	1	2	3	4	5	6	7	8	9
vecEnt	50	80	70	30	90	0				

Para **eliminar un elemento en una posición específica del vector** se debe validar que no está vacío; si la posición por eliminar es la 0, se sigue el procedimiento para eliminar al principio del vector; si la posición es igual al tope (último elemento en el vector), se sigue el mismo procedimiento para eliminar el último elemento del vector; si no es ni el primero ni el último elemento del vector, se corre una posición a la izquierda cada elemento que esté a la derecha de la posición que se va a eliminar. Por ejemplo, si se tiene un vector con cinco elementos y se desea eliminar el elemento de la posición 2, entonces el elemento de la 3 pasa a la 2, el elemento de la 4 pasa a la 3 y en la 4 se asigna 0, y se decrementa el tope en 1 (**figuras 4.31 y 4.32**).

Figura 4.31: Vector con cinco elementos, pero con capacidad para diez

	0	1	2	3	4	5	6	7	8	9
vecEnt	50	80	70	30	90					

Para el ejemplo se tienen cinco elementos y se desea eliminar el de la posición 2, por lo cual el 30 pasa a la 2 y el 90 a la 3; finalmente, se asigna 0 al vector en la posición tope y se decrementa el tope en 1 (**figura 4.32**).

Figura 4.32: Eliminación del último elemento de un vector

	0	1	2	3	4	5	6	7	8	9
vecEnt	50	80	30	90	0					

A continuación, el algoritmo:

Algoritmo Ejercicio5C4

```

//Definir o declarar las variables
//vecEnt vector de tipo entero
//tope va a contener la posición del último elemento insertado en el
//vector
//pos se va a usar para indicar la posición en la cual se desea insertar
//un elemento en el vector
//opcMen va a contener la opción seleccionada por el usuario según menú
//de opciones
//n constante que contiene el tamaño del vector, para el ejemplo, 10,
//puede ser cualquier otro valor
Definir vecEnt, tope, pos, opcMen, n Como Entero

//Dimensionar el vector
n ← 10 //Número máximo de elementos del vector vecEnt
Dimension vecEnt[n]

//Cuando se ejecuta tope inicia en 0 porque no hay elementos en el
//vector
tope ← 0

Repetir
  Menu() //Visualiza el menú de opciones
  //El usuario selecciona una opción entre 1 y 5
  opcMen ← LeerValorEntero("Seleccione una opción: ", 0, 5)

  Segun opcMen Hacer
    1: tope ← LlenarVector(vecEnt, n, tope)
    2: tope ← EliminarFinal(vecEnt, tope)
    3: tope ← EliminarPrincipio(vecEnt, tope)
    4: tope ← EliminarPosicion(vecEnt, tope)
    5: MostrarVector(vecEnt, tope)

  FinSegun

Hasta Que opcMen = 0
FinAlgoritmo

//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer o capturar un valor real
Funcion val ← LeerValorReal(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer una cadena como nombres, direcciones, etc.
Funcion cad ← LeerCadena(msg) (ver Ejercicio1C3)

```

```
//Función para leer un carácter como S, N, F, M, etc.
Funcion car ← leerCaracter(msg, c1, c2) (ver Ejercicio1C3)

//Funciones adicionales

//Función para visualizar el menú de opciones
Funcion Menu()
    Borrar Pantalla
    Escribir "*** MENÚ DE OPCIONES ***"
    Escribir ""
    Escribir "1. Llenar el vector"
    Escribir "2. Eliminar un elemento al final"
    Escribir "3. Eliminar un elemento al principio"
    Escribir "4. Eliminar un elemento en una posición específica"
    Escribir "5. Mostrar vector"
    Escribir "0. Salir"
FinFuncion

//Función para asignarle valores al vector
Funcion tope ← LlenarVector(A, n, ind) (ver Ejercicio4C4)

//Función para visualizar el contenido del vector
Funcion MostrarVector(A, tope) (ver Ejercicio4C4)

//Función para eliminar un elemento al final del vector
//A es el vector
//i recibe tope, la posición del último elemento
Funcion tam ← EliminarFinal(A, i)
    Definir tam Como Entero

    Si (i = 0) Y (A[i] = 0) Entonces //Se valida que el vector no esté vacío
        Mensaje3()
    Sino
        Si (i = 0) Y (A[i] > 0) Entonces
            A[i] ← 0
            tam ← i
        SiNo
            A[i] ← 0
            tam ← i-1
        FinSi
    Mensaje2()
    FinSi
FinFuncion

//Función para eliminar un elemento al principio del vector
Funcion tam ← EliminarPrincipio(A, i)
    Definir tam, j Como Entero
```

```

j ← 0
Si (i = 0) Y (A[i] = 0) Entonces //Se valida que el vector no esté vacío
    Mensaje3()
SiNo
    Si (i = 0) Y (A[i] > 0) Entonces //Se valida que el vector solo
        //tenga un elemento
            A[i] ← 0
            tam ← i
        SiNo
            //Se corre cada elemento una posición hacia atrás desde 1
            //hasta i
            Mientras j < i Hacer
                A[j] ← A[j+1]
                j ← j + 1
            FinMientras
            A[i] ← 0
            tam ← i-1
        FinSi
    Mensaje2()
FinSi
FinFuncion

//Función para eliminar un elemento en una posición específica del vector
Funcion tam ← EliminarPosicion(A, i)
    Definir tam, j, pos Como Entero

    j ← i //Se guarda la posición del último elemento n j

    Si (i = 0) Y (A[i] = 0) Entonces //Se valida que el vector no esté vacío
        Mensaje3()
    SiNo
        //Se captura la posición para eliminar un nuevo elemento
        pos ← LeerValorEntero("Posición para eliminar un elemento: ",0,i)

        Si (pos = 0) Entonces
            tam ← EliminarPrincipio(A, i)
        SiNo
            //Se valida que la posición p sea igual a i, se elimina al
            //final del vector
            Si pos = i Entonces
                tam ← EliminarFinal(A, i)
            SiNo
                //Se corre cada elemento a partir de pos + 1 una
                //posición atrás
                hasta posición I

```

```
        //se asigna 0 al vector en i, se decrementa en 1 i
        Mientras pos < i Hacer
            A[pos] ← A[pos + 1]
            pos ← pos + 1
        FinMientras
        A[i] ← 0
        //Se decrementa en 1 el tamaño del vector por el nuevo
        //elemento eliminado
        tam ← i-1
        Mensaje2()
    FinSi
FinSi
FinSi
FinFuncion

//Mensaje para indicarle al usuario que el vector ya contiene elementos
Funcion Mensaje1()
    Escribir ""
    Escribir "El vector ya contiene elementos"
    Escribir "Presione una tecla para continuar..."
    Esperar Tecla
FinFuncion

//Función para indicarle al usuario que se eliminó correctamente el elemento
del vector
Funcion Mensaje2()
    Escribir ""
    Escribir "Elemento eliminado correctamente"
    Escribir "Presione una tecla para continuar..."
    Esperar Tecla
FinFuncion

Funcion Mensaje3()
    Escribir ""
    Escribir "El vector no contiene elementos"
    Escribir "Presione una tecla para continuar..."
    Esperar Tecla
FinFuncion

Funcion Mensaje4()
    Escribir ""
    Escribir "*** ERROR *** se alcanzó el número máximo de elementos
        permitidos"
    Escribir "Presione una tecla para continuar..."
    Esperar Tecla
FinFuncion
```

4.4 Métodos de ordenamiento

Estos métodos permiten ordenar los elementos de un vector de forma ascendente o descendente, gracias a lo cual la información queda organizada de acuerdo con algún criterio. Existen varios métodos o algoritmos para ordenar un vector, entre ellos:

- Burbuja
- Burbuja mejorado
- Selección
- Inserción
- Shell
- Quicksort

Hay mucha literatura que explica en detalle el funcionamiento de cada uno de estos, como el libro *Fundamentos de programación, algoritmos, estructuras de datos y objetos*, de Luis Joyanes Aguilar (p. 365), o el sitio web TutosPOO¹. A continuación, presentamos los algoritmos de los métodos relacionados.

Ordenamiento por burbuja

Compara cada elemento del vector con el elemento siguiente realizando el intercambio de estos, si no están ordenados. Para un vector de cinco elementos, se llevan a cabo las siguientes comparaciones intercambiando los elementos según sea necesario.

Posición 0 con 1
Posición 1 con 2
Posición 2 con 3
Posición 3 con 4
Posición 0 con 1
Posición 1 con 2
Posición 2 con 3
Posición 0 con 1
Posición 1 con 2
Posición 0 con 1

Al finalizar las comparaciones, el vector queda totalmente ordenado. Para un vector de cinco elementos con los valores 50, 30, 10, 40, 20, se hacen los siguientes intercambios (**figura 4.33**).

¹ <https://tutospoo.jimdofree.com/tutoriales-java/m%C3%A9todos-de-ordenaci%C3%B3n/>

Figura 4.33: Proceso de ordenación de un vector de cinco elementos por burbuja

VECTOR ORIGINAL						
50	30	10	40	20	Compara 50 con 30, hay intercambio	posición (0,1)
30	50	10	40	20	Compara 30 con 10, hay intercambio	posición (0,2)
10	50	30	40	20	Compara 10 con 40, no hay intercambio	posición (0,3)
10	50	30	40	20	Compara 10 con 20, no hay intercambio	posición (0,4)
10	50	30	40	20	Compara 50 con 30, hay intercambio	posición (1,2)
10	30	50	40	20	Compara 30 con 40, no hay intercambio	posición (1,3)
10	30	50	40	20	Compara 30 con 20, hay intercambio	posición (1,4)
10	20	50	40	30	Compara 50 con 40, hay intercambio	posición (2,3)
10	20	40	50	30	Compara 40 con 30, hay intercambio	posición (2,4)
10	20	30	50	40	Compara 50 con 40, hay intercambio	posición (3,4)
10	20	30	40	50	Vector ordenado	

El algoritmo es el siguiente:

```
//Ordenamiento por burbuja
```

```
Algoritmo Ejercicio6C4
```

```
    Definir vecEnt, n Como Entero
```

```
    //Dimensionar el vector
```

```
    n ← LeerValorEntero("Indique el número de elementos del vector: ", 5, 100)
```

```
    Dimension vecEnt[n]
```

```
    LlenarVector(vecEnt, n)
```

```
    MostrarVector(vecEnt, n, "Vector original")
```

```
    Burbuja(vecEnt, n)
```

```
    MostrarVector(vecEnt, n, "Vector ordenado")
```

```
FinAlgoritmo
```

```
//Función para leer o capturar un valor entero
```

```
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)
```

```
Funcion LlenarVector(A, n)
```

```
    Definir i Como Entero
```

```
    Para i ← 0 Hasta n-1 Hacer
```

```
        Escribir Sin Saltar "Valor no. ", i+1, ": "
```

```
        Leer A[i]
```

```
    FinPara
```

```
FinFuncion
```

```
Funcion MostrarVector(A, n, msg)
```



```

Definir i Como Entero
Borrar Pantalla
Escribir msg
Escribir ""
Para i ← 0 Hasta n-1 Hacer
    Escribir "vecEnt[" , i, "] = " , A[i]
FinPara
Escribir ""
Escribir "Presione una tecla para continuar..."
Esperar Tecla

```

FinFuncion

Funcion Burbuja(A, n)

```

Definir i, j, aux Como Entero

Para i ← 0 Hasta n-2 Hacer
    Para j ← i+1 Hasta n-1 Hacer
        Si A[i] > A[j] Entonces
            aux ← A[i]
            A[i] ← A[j]
            A[j] ← aux
        FinSi
    FinPara
FinPara

```

FinFuncion

Ordenamiento por burbuja mejorado

Es similar al anterior, pero, al finalizar cada iteración, el elemento mayor queda ubicado en la última posición, mientras los demás elementos menores van descendiendo. Para un vector de cinco elementos, se realizan las siguientes comparaciones intercambiando los elementos según sea necesario.

```

Posición 0 con 1
Posición 1 con 2
Posición 2 con 3
Posición 3 con 4
Posición 0 con 1
Posición 1 con 2
Posición 2 con 3
Posición 0 con 1
Posición 1 con 2
Posición 0 con 1

```

Al finalizar las comparaciones, el vector queda totalmente ordenado. Para un vector de cinco elementos con los valores 50, 30, 10, 40, 20, se realizan los siguientes intercambios (**figura 4.34**).

Figura 4.34: Proceso de ordenación de un vector de cinco elementos por burbuja mejorado

VECTOR ORIGINAL						
50	30	10	40	20	Compara 50 con 30, hay intercambia	posición (0,1)
30	50	10	40	20	Compara 50 con 10, hay intercambia	posición (1,2)
30	10	50	40	20	Compara 50 con 40, hay intercambia	posición (2,3)
30	10	40	50	20	Compara 50 con 20, hay intercambia	posición (3,4)
30	10	40	20	50	Compara 30 con 10, hay intercambia	posición (0,1)
10	30	40	20	50	Compara 30 con 40, no hay intercambio	posición (1,2)
10	30	40	20	50	Compara 40 con 20, hay intercambio	posición (2,3)
10	30	20	40	50	Compara 10 con 30, no hay intercambio	posición (0,1)
10	30	20	40	50	Compara 30 con 20, hay intercambio	posición (1,2)
10	20	30	40	50	Compara 10 con 20, no hay intercambia	posición (0,1)
10	20	30	40	50	Vector ordenado	

El algoritmo es el siguiente:

```
//Ordenamiento por burbuja mejorado
Algoritmo Ejercicio7C4
    Definir vecEnt, n Como Entero

    //Dimensionar el vector
    n ← LeerValorEntero(Índique el número de elementos del vector: ", 5, 100)
    Dimension vecEnt[n]

    LlenarVector(vecEnt, n)
    MostrarVector(vecEnt, n, "Vector original")
    BurbujaMejorado(vecEnt, n)
    MostrarVector(vecEnt, n, "Vector ordenado")

FinAlgoritmo

//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

Funcion LlenarVector(A, n) (ver Ejercicio6C4)

Funcion MostrarVector(A, n, msg) (ver Ejercicio6C4)

Funcion BurbujaMejorado(A, n)
    Definir i, j, aux Como Entero
    Definir sw Como Logico
```

```

Para i ← n-1 Hasta 1 Con Paso -1 Hacer
  Para j ← 0 Hasta i-1 Hacer
    Si A[j] > A[j+1] Entonces
      aux ← A[j]
      A[j] ← A[j+1]
      A[j+1] ← aux
    FinSi
  FinPara
FinPara

```

FinFuncion

Ordenamiento por selección

Este método recorre todo el vector desde la posición 0 buscando el menor elemento de todos. Cuando termina el recorrido, hace el intercambio por el elemento de la primera posición, y se repite el proceso sucesivamente con los demás elementos del vector.

Al finalizar las comparaciones e intercambios, el vector queda totalmente ordenado. Para un vector de cinco elementos con los valores 50, 30, 10, 40, 20, se realizan las siguientes comparaciones e intercambios (**figura 4.35**):

Figura 4.35: Proceso de ordenación de un vector de cinco elementos por selección

VECTOR ORIGINAL							
50	30	10	40	20	Compara 30 con 50	posición (1,0)	posicion menor 1
					Compara 10 con 30	posición (2,1)	posicion menor 2
					Compara 40 con 10	posición (3,2)	posicion menor 2
					Compara 20 con 10	posición (4,2)	posicion menor 2
10	30	50	40	20	Intercambio	posición (0,2)	
					Compara 50 con 30	posición (2,1)	posicion menor 1
					Compara 40 con 30	posición (3,1)	posicion menor 1
					Compara 20 con 30	posición (4,1)	posicion menor 4
10	20	50	40	30	Intercambio	posición (1,4)	
					Compara 40 con 50	posición (3,2)	posicion menor 3
					Compara 30 con 40	posición (4,3)	posicion menor 4
10	20	30	40	50	Intercambio	posición (2,4)	
					Compara 50 con 40	posición (4,3)	posicion menor 3
10	20	30	40	50	No hay intercsmbio, vector ordenado		

El algoritmo es el siguiente:

```
//Ordenamiento por selección
Algoritmo Ejercicio8C4
    Definir vecEnt, n Como Entero

    //Dimensionar el vector
    n ← LeerValorEntero("Indique el número de elementos del vector: ", 5, 100)
    Dimension vecEnt[n]

    LlenarVector(vecEnt, n)
    MostrarVector(vecEnt, n, "Vector original")
    Seleccion(vecEnt, n)
    MostrarVector(vecEnt, n, "Vector ordenado")
FinAlgoritmo

//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

Funcion LlenarVector(A, n) (ver Ejercicio6C4)

Funcion MostrarVector(A, n, msg) (ver Ejercicio6C4)

Funcion Seleccion(A, n)
    Definir i, j, aux, posMen Como Entero
    Para i ← 0 Hasta n-2 Hacer
        posMen ← i
        Para j ← i+1 Hasta n-1 Hacer
            Si A[j] < A[posMen] Entonces
                posMen ← j
            FinSi
        FinPara
        Si posMen <> i Entonces
            aux ← A[i]
            A[i] ← A[posMen]
            A[posMen] ← aux
        FinSi
    FinPara
FinFuncion
```

Ordenamiento por inserción

Este método ordena los primeros dos elementos del vector; seguidamente, toma el tercer elemento y lo ordena en la posición correspondiente, y continúa sucesivamente este proceso con todos los demás elementos. Para un vector de cinco elementos con los valores 50, 30, 10, 40, 20, se realiza los siguientes intercambios (figura 4.36).

Figura 4.36: Proceso de ordenación de un vector de cinco elementos por inserción

VECTOR ORIGINAL						
50	30	10	40	20	Compara 30 con 50, hay intercambia	posición (1,0)
30	50	10	40	20	Compara 10 con 50, hay intercambia	posición (2,1)
30	10	50	40	20	Compara 10 con 30, hay intercambia	posición (1,0)
10	30	50	40	20	Compara 40 con 50, hay intercambia	posición (3,2)
10	30	40	50	20	Compara 20 con 50, hay intercambia	posición (4,3)
10	30	40	20	50	Compara 20 con 40, hay intercambia	posición (3,2)
10	30	20	40	50	Compara 20 con 30, hay intercambia	posición (2,1)
10	20	30	40	50	Vector ordenado	

El algoritmo es el siguiente:

```
//Ordenamiento por inserción
Algoritmo Ejercicio9C4
    Definir vecEnt, n Como Entero

    //Dimensionar el vector
    n ← LeerValorEntero(Índique el número de elementos del vector: ", 5, 100)
    Dimension vecEnt[n]

    LlenarVector(vecEnt, n)
    MostrarVector(vecEnt, n, "Vector original")
    Insercion(vecEnt, n)
    MostrarVector(vecEnt, n, "Vector ordenado")
FinAlgoritmo

//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

Funcion LlenarVector(A, n) (ver Ejercicio6C4)

Funcion MostrarVector(A, n, msg) (ver Ejercicio6C4)

Funcion Insercion(A, n)
    Definir i, j, aux Como Entero
    Para i ← 1 Hasta n-1 Hacer
        j ← i
        Mientras (j>0) Y (A[j] < A[j-1]) Hacer
            aux ← A[j]
            A[j] ← A[j-1]
            A[j-1] ← aux
            j ← j -1
        FinMientras
    FinPara
FinFuncion
```

Ordenamiento Shell

Este método ordena insertando subconjuntos del vector, separados entre sí, iguales a la mitad del tamaño del vector, es decir, va organizando subvectores de tamaño $n/2$ e intercambia valores que disminuyen rápidamente. Los intercambios que se producen, para un vector de cinco elementos con los valores 50, 30, 10, 40, 20, están en la figura 4.37.

En este método se hace uso de la función $\text{Trunc}(x)$, parte entera de X ; por ejemplo, si se tiene a $j = \text{Trunc}(5/2)$, $a = 2$, porque $5/2 = 2.5$; pero, como Trunc retorna la parte entera, entonces devuelve 2.

Figura 4.37: Proceso de ordenación de un vector de cinco elementos por Shell

VECTOR ORIGINAL								
50	30	10	40	20	Compara 50 con 10, hay intercambio	posición (0,2)	Salto	2
10	30	50	40	20	Compara 50 con 20, hay intercambio	posición (2,4)	Salto	2
10	30	20	40	50	Compara 30 con 20, hay intercambio	posición (1,2)	Salto	1
10	20	30	40	50	Vector ordenado			

El algoritmo es el siguiente:

```
//Ordenamiento por Shell
Algoritmo Ejercicio10C4
    Definir vecEnt, n Como Entero

    //Dimensionar el vector
    n ← LeerValorEntero("Indique el número de elementos del vector: ", 5, 100)
    Dimension vecEnt[n]

    LlenarVector(vecEnt, n)
    MostrarVector(vecEnt, n, "Vector original")
    Shell(vecEnt, n)
    MostrarVector(vecEnt, n, "Vector ordenado")
FinAlgoritmo

//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

Funcion LlenarVector(A, n) (ver Ejercicio6C4)

Funcion MostrarVector(A, n, msg) (ver Ejercicio6C4)

Funcion Shell(A, n)
    Definir i, j, cont, sal, v, m Como Entero
    sal ← Trunc(n/2)
```

```

Mientras sal >= 1 Hacer
  Para i ← 0 Hasta sal-1 Hacer
    Para j ← sal+i Hasta n-1 Con Paso sal Hacer
      v ← A[j]
      m ← j-sal
      Mientras (m>=0) Y (A[m] > v) Hacer
        A[m+sal] ← A[m]
        m ← m - sal
      FinMientras
      A[m+sal] ← v
    FinPara
  FinPara
  sal ← Trunc(sal/2)
FinMientras

```

FinFuncion

Ordenamiento Quicksort

Este método hace uso de la **recursividad**, muy importante en el ámbito del desarrollo del software, y se da cuando una función se llama a sí misma. Debe existir una condición de terminación o finalización, porque, de lo contrario, se ejecutará indefinidamente y, entonces, bloqueará el computador. Un algoritmo sencillo para ejemplificar la recursividad es el cálculo de una factorial. Matemáticamente se sabe que el factorial de 5 es $5*4*3*2*1 = 120$; también se puede expresar como $1*2*3*4*5 = 120$ aplicando la propiedad conmutativa de la multiplicación (el orden de los factores no altera el producto).

Inicialmente se desarrolla el algoritmo de forma iterativa:

Algoritmo Ejercicio11C4

```

Definir nro, res, i Como Entero

nro ← 5
res ← FactorialIterativo(nro)
MostrarFactorial(nro, res)

```

FinAlgoritmo

Funcion res ← FactorialIterativo(nro)

```

Definir res, i Como Entero
res ← 1
Para i ← 1 Hasta nro Hacer
  res ← res * i
FinPara

```

FinFuncion

```
Funcion MostrarFactorial(nro, res)
    Escribir "Factorial de ", nro, "es: ",res
FinFuncion
```

Como se puede apreciar en la función **FactorialIterativo**, hay un ciclo **PARA** que va de **1** hasta **nro** (en este caso, 5); realizando sucesivamente el cálculo de $res \leftarrow res * i$, al final del ciclo se retorna el valor de la variable *res* al algoritmo principal invocando seguidamente la función **MostrarFactorial** en la línea de código `MostrarFactorial(nro, res)`, y se visualiza, así, el resultado de **120**.

El algoritmo recursivo es el siguiente:

```
Algoritmo Ejercicio12C4
    Definir nro, res Como Entero

    nro  $\leftarrow$  5
    res  $\leftarrow$  FactorialRecursivo(nro)
    MostrarFactorial(nro, res)
FinAlgoritmo
```

```
Funcion res  $\leftarrow$  FactorialRecursivo(nro)
    Definir res Como Entero
    Si nro = 0 Entonces
        res  $\leftarrow$  1
    SiNo
        Res  $\leftarrow$  nro * FactorialRecursivo(nro - 1)
    FinSi
FinFuncion
```

```
Funcion MostrarFactorial(nro, res)
    Escribir "Factorial de ", nro, "es: ",res
FinFuncion
```

La diferencia con el anterior es que en la función **FactorialRecursivo** no se hace uso del ciclo **PARA**, y tiene esta condición: cuando $nro = 0$, finaliza la ejecución, o, de lo contrario, se invoca sucesivamente a sí misma en la línea de código:

$$Res \leftarrow nro * \text{FactorialRecursivo}(nro - 1)$$

Es, entonces, en esta línea de código donde se da la recursividad. Al ejecutar ambos algoritmos se obtiene el mismo resultado (**figuras 4.38 y 4.39**).

Figura 4.38: Ejecución factorial iterativo

```

*** Ejecución Iniciada. ***
Factorial de 5 es: 120
*** Ejecución Finalizada. ***

```

Figura 4.39: Ejecución factorial recursivo

```

*** Ejecución Iniciada. ***
Factorial de 5 es: 120
*** Ejecución Finalizada. ***

```

Explicación funcionamiento de la recursividad

En el algoritmo principal se invoca la función FactorialRecursivo, así:

$\text{res} \leftarrow \text{FactorialRecursivo}(\text{nro})$, con $\text{nro} = 5$

Ya en la función, se valida si $\text{nro} = 0$, se le asigna 1 a res

Si $\text{nro} = 0$ Entonces

$\text{res} \leftarrow 1$

Allí finaliza el llamado recursivo de la función, pero, como **nro** no es igual a 0, inicialmente vale 5, entonces se ejecuta la parte del **SiNo**.

$\text{Res} \leftarrow \text{nro} * \text{FactorialRecursivo}(\text{nro} - 1)$

Esto indica $5! = 5 * 4!$

Como no se conoce factorial de 4, entonces queda indicado; nuevamente se valida **Si nro = 0**, y, como **nro = 5** inicialmente, y en el llamado recursivo se indica $n-1$, entonces ahora vale 4.

$\text{Res} \leftarrow \text{nro} * \text{FactorialRecursivo}(\text{nro} - 1)$

$4! = 4 * 3!$

Este proceso se repite sucesivamente hasta que **nro = 0**, y queda lo siguiente:

$5! = 5 * 4!$

$4! = 4 * 3!$

$3! = 3 * 2!$

$2! = 2 * 1!$

$1! = 1 * 0!$

Como **nro** es igual a **0**, entonces $\text{res} = 1$; en este punto finaliza el llamado recursivo de la función y se resuelve cada uno de los factoriales indicados anteriormente, del último al primero, así:

Como $0! = 1$, dado por la condición

Si $\text{nro} = 0$ Entonces

$\text{res} \leftarrow 1$

se reemplaza $0!$ por 1 en:

$1! = 1 * 0! = 1 * 1 = 1$, luego se reemplaza $1!$ por 1 en

$2! = 2 * 1! = 2 * 1 = 2$, continuando el mismo proceso hasta terminar así:

$3! = 3 * 2! = 3 * 2 = 6$

$4! = 4 * 3! = 4 * 6 = 24$

$5! = 5 * 4! = 5 * 24 = 120$

120 es el valor que finalmente se retorna al algoritmo principal en la variable **res**.

El método de ordenamiento **Quicksort** elige un elemento del vector como referente, el cual toma el nombre de *pivote*; a continuación, se reordenan los demás elementos del vector y quedan, así, los valores inferiores al pivote a la izquierda y los mayores a la derecha. Se invoca a sí mismo (recursividad) para ordenar los elementos tanto de la izquierda como de la derecha. Entonces, en un vector de cinco elementos con los valores $50, 30, 10, 40, 20$, se realizan los siguientes intercambios (**figura 4.40**):

Figura 4.40: Proceso de ordenación de un vector de cinco elementos por Quicksort

VECTOR ORIGINAL							
50	30	10	40	20	Compara 50 con 20, hay intercambia	posición (0,4)	pivote 50
20	30	10	40	50	Compara 30 con 10, hay intercambia	posición (1,2)	pivote 20
20	10	30	40	50	Compara 20 con 10, hay intercambia	posición (0,1)	pivote 20
10	20	30	40	50	Vector ordenado		

El algoritmo recursivo es el siguiente:

```
//Ordenamiento QuickSort
```

```
Algoritmo Ejercicio13C4
```

```
Definir vecEnt, n Como Entero
```

```
//Dimensionar el vector
```

```
n ← LeerValorEntero(Índique el número de elementos del vector: ", 5, 100)
```

```
Dimension vecEnt[n]
```

```

    LlenarVector(vecEnt, n)
    MostrarVector(vecEnt, n, "Vector original")
    QuickSort(vecEnt, 0, n-1)
    MostrarVector(vecEnt, n, "Vector ordenado")
FinAlgoritmo

//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

Funcion LlenarVector(A, n) (ver Ejercicio6C4)

Funcion MostrarVector(A, n, msg) (ver Ejercicio6C4)

Funcion QuickSort(A, inicio, final)
    Definir pivote, izq, der, aux, i Como Entero
    Si inicio ≤ final Entonces
        pivote ← A[inicio]
        izq ← inicio +1
        der ← final
        Mientras izq ≤ der Hacer

            Mientras (izq ≤ final) Y (A[izq] < pivote) Hacer
                izq ← izq + 1
            FinMientras

            Mientras (der > inicio) Y (A[der] ≥ pivote) Hacer
                izq ← izq + 1
                der ← der -1
            FinMientras

            Si izq < der Entonces
                aux ← A[izq]
                A[izq] ← A[der]
                A[der] ← aux
            FinSi

        FinMientras

    Si der > inicio Entonces
        aux ← A[inicio]
        A[inicio] ← A[der]
        A[der] ← aux
    FinSi

    //Si inicio ≥ final Entonces
    //    Escribir "Vector ordenado... FIN"
    //    Esperar tecla
    //SiNo
        QuickSort(A, inicio, der-1)

```

```
        QuickSort(A, der+1, final)
    //FinSi
FinSi
FinFuncion
```

4.5 Métodos de búsqueda

Una de las funciones más importantes de un sistema de información es la consulta o búsqueda según algún criterio. En un vector se pueden hacer básicamente dos tipos de búsqueda: secuencial y binaria.

Búsqueda secuencial

Como su nombre lo indica, se realiza una búsqueda iniciando por el primer elemento del vector hasta localizar el elemento buscado, o hasta llegar al último elemento. Se deben considerar los siguientes criterios:

- Validar que el vector no esté vacío.
- Hacer uso de una bandera de tipo lógico para controlar la búsqueda y el ciclo.
- Si el valor que se va a buscar no está en el vector, se devuelve el valor -1; de lo contrario, se devuelve la posición del elemento en el vector.
- Validar el valor retornado: si es -1 , visualizar el mensaje “El valor **X**, no se encuentra en el vector”; si es diferente de -1 , visualizar el mensaje “El valor **X**, se encuentra en la posición **res**, donde **X** es el valor buscado y **res** es la posición que ocupa en el vector.

A continuación, el algoritmo:

```
//Búsqueda secuencial
Algoritmo Ejercicio14C4

    Definir vecEnt, n, pos, nro Como Entero
    Definir opc Como Caracter
    //Dimensionar el vector
    n ← LeerValorEntero("Indique el número de elementos del vector: ", 5, 100)

    Dimension vecEnt[n]
    LlenarVector(vecEnt, n)
    Repetir
        nro ← LeerValorEntero("Valor a buscar: ", 1, 100)
        pos ← BusquedaSecuencial(vecEnt, n, nro)
```

```

        ResultadoBusqueda(nro, pos)
        MostrarVector(vecEnt, n)
        opc ← LeerCaracter("¿Realiza otra búsqueda [S/N]? ", "S", "N")
        Hasta Que Mayusculas(opc) = "N"
FinAlgoritmo

//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

//Función para leer un carácter como S, N, F, M, etc.
Funcion car ← leerCaracter(msg, c1, c2) (ver Ejercicio1C3)

Funcion LlenarVector(A, n) (ver Ejercicio6C4)

Funcion MostrarVector(A, n, msg) (ver Ejercicio6C4)

Funcion Res ← BusquedaSecuencial(A, n, val)
    Definir sw Como logico
    Definir i, res Como Entero

    sw ← Verdadero

    Si A[0] = 0 Entonces
        sw ← Falso
    SiNo
        i ← 0
        Mientras (i < n) Y sw Hacer
            Si A[i] = val Entonces
                sw ← Falso
            SiNo
                i ← i + 1
            FinSi
        FinMientras

        Si sw Entonces
            res ← -1
        SiNo
            res ← i
        FinSi
    FinSi
FinFuncion

Funcion ResultadoBusqueda(nro, res)
    Borrar Pantalla
    Si res = -1 Entonces
        Escribir .51 valor ", nro, "no se encuentra en el vector"
    SiNo
        Escribir .51 valor ", nro, "se encuentra en la posición ", res

```

```

FinSi
Escribir ""
Escribir "Presione una tecla para continuar..."
Esperar Tecla

```

FinFuncion

Búsqueda binaria

Cuando la cantidad de información es considerable, el método de búsqueda secuencial no es eficiente ni eficaz. En tal sentido, se recomienda una búsqueda binaria, que consiste en hacer divisiones sucesivas del vector en mitades hasta hallar el elemento deseado. Se deben considerar los siguientes criterios:

- El vector debe estar ordenado.
- Se examina primero el elemento central del vector.
- Si coincide con el valor buscado, finaliza la búsqueda.
- De lo contrario, se determina si está a la izquierda o a la derecha del valor central.
- A continuación, se examina el elemento central del subvector, y se repite este proceso hasta localizar el valor deseado.

Si se tiene un vector de 11 elementos ordenados (**figura 4.41**):

Figura 4.41: Vector de 11 elementos lleno y ordenado

	0	1	2	3	4	5	6	7	8	9	10
vecEnt	5	10	15	20	25	30	35	40	45	50	55

Se desea localizar el valor 10; en ese caso se determina el valor central, que corresponde al 30 (**figura 4.42**).

Figura 4.42: Elemento central del vector

	0	1	2	3	4	5	6	7	8	9	10
vecEnt	5	10	15	20	25	30	35	40	45	50	55

Como 10 es menor que 30, entonces se sabe que está a la izquierda; el valor central del subvector ahora es 15 (**figura 4.43**).

Figura 4.43: Elemento central subvector izquierdo

vecEnt	5	10	15	20	25	30	35	40	45	50	55
--------	---	----	----	----	----	----	----	----	----	----	----

Como 10 es menor que 15, el valor que se va a buscar nuevamente está a la izquierda del valor central, del subvector: ahora es 10 (**figura 4.44**).

Figura 4.44: Elemento central del subvector anterior

vecEnt	5	10	15	20	25	30	35	40	45	50	55
--------	---	----	----	----	----	----	----	----	----	----	----

El valor central es igual al valor buscado. Finaliza la búsqueda de forma exitosa.

Si se busca un valor que no existe, el procedimiento es el siguiente: se toma el mismo vector inicial, y el valor que se va a buscar es 42 si se tiene un vector de 11 elementos ordenados (**figura 4.45**).

Figura 4.45: Vector de 11 elementos ordenado

	0	1	2	3	4	5	6	7	8	9	10
vecEnt	5	10	15	20	25	30	35	40	45	50	55

Si se desea localizar el valor 42, se determina el valor central, que corresponde a 30 (**figura 4.46**).

Figura 4.46: Elemento central del vector

	0	1	2	3	4	5	6	7	8	9	10
vecEnt	5	10	15	20	25	30	35	40	45	50	55

Como 42 es mayor que 30, entonces se sabe que está a la derecha; el valor central del subvector ahora es 45 (**figura 4.47**).

Figura 4.47: Elemento central subvector derecho

vecEnt	5	10	15	20	25	30	35	40	45	50	55
--------	---	----	----	----	----	----	----	----	----	----	----

Como 45 es mayor que 42, entonces se sabe que está a la izquierda; el valor central del subvector ahora es 40 (**figura 4.48**).

Figura 4.48: Elemento central del subvector a la izquierda

vecEnt	5	10	15	20	25	30	35	40	45	50	55
---------------	---	----	----	----	----	----	----	----	----	----	----

El valor central 40 es menor que el valor buscado 42, por lo que la búsqueda finaliza de forma no exitosa; es decir, no está el valor 42 en el vector.

A continuación, el algoritmo:

```
//Búsqueda binaria
```

```
Algoritmo Ejercicio15C4
```

```
    Definir vecEnt, n, pos, nro Como Entero
```

```
    Definir opc Como Caracter
```

```
    //Dimensionar el vector
```

```
    n ← LeerValorEntero("Indique el número de elementos del vector: ", 5,100)
```

```
    Dimension vecEnt[n]
```

```
    LlenarVector(vecEnt, n)
```

```
    Ordenar(vecEnt, n)
```

```
    MostrarVector(vecEnt, n)
```

```
    Repetir
```

```
        nro ← LeerValorEntero("Valor a buscar: ", 1, 100)
```

```
        pos ← BusquedaBinaria(vecEnt, n-1, nro)
```

```
        ResultadoBusqueda(nro, pos)
```

```
        MostrarVector(vecEnt, n)
```

```
        opc ← LeerCaracter("-¿Realiza otra búsqueda [S/N]? ", "S", "N")
```

```
    Hasta Que Mayusculas(opc) = "N"
```

```
FinAlgoritmo
```

```
//Función para leer o capturar un valor entero
```

```
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)
```

```
//Función para leer un carácter como S, N, F, M, etc.
```

```
Funcion car ← leerCaracter(msg, c1, c2) (ver Ejercicio1C3)
```

```
Funcion LlenarVector(A, n) (ver Ejercicio6C4)
```

```
Funcion MostrarVector(A, n, msg) (ver Ejercicio6C4)
```

```
Funcion Ordenar(A, n)
```

```
    Definir i, j, aux Como Entero
```

```
    Para i ← 0 Hasta n-2 Hacer
```

```
        Para j ← i+1 Hasta n-1 Hacer
```

```
            Si A[i] > A[j] Entonces
```

```
                aux ← A[i]
```



```

                A[i] ← A[j]
                A[j] ← aux
            FinSi
        FinPara
    FinPara
FinFuncion

Funcion Res ← BusquedaBinaria(A, n, val)

    Definir sw Como logico
    Definir inicial, final, centro, res Como Entero

    inicial ← 0
    final ← n
    sw ← Verdadero

    Mientras (inicial ≤ final) Y sw Hacer
        centro ← Redon((inicial + final)/2)
        Escribir "Centro = ", centro
        Escribir "A[centro] = ", A[centro]
        Si A[centro] = val Entonces
            sw ← Falso
        SiNo
            Si A[centro] > val Entonces
                final ← centro -1
            SiNo
                inicial ← centro +1
            FinSi
        FinSi
    FinMientras

    Si sw Entonces
        res ← -1
    SiNo
        res ← centro
    FinSi
FinFuncion

Funcion ResultadoBusqueda(nro, res)

    Borrar Pantalla
    Si res = -1 Entonces
        Escribir "El valor ", nro, "no se encuentra en el vector"
    SiNo
        Escribir "El valor ", nro, "se encuentra en la posición ", res
    FinSi
    Escribir ""

```

```

Escribir "Presione una tecla para continuar..."
Esperar Tecla

```

```
FinFuncion
```

4.6 Matriz

Es un arreglo de dos dimensiones, el cual puede ser representado gráficamente como se observa en la **figura 4.49**.

Figura 4.49: Representación gráfica de una matriz

Se le debe asignar un nombre que identifique a todos sus elementos; por ejemplo, `matEnt`. Cada recuadro representa una posición en la matriz, por lo que, entonces, `matEnt` tiene 16 elementos, compuestos por cuatro filas y cuatro columnas (**figura 4.50**).

Figura 4.50: Representación gráfica de la asignación de un nombre a la matriz

	C1	C2	C3	C4	
					F1
					F2
					F3
					F4

Donde F1, F2, F3, F4 representan las filas y C1, C2, C3, C4 las columnas. `matEnt` es el nombre correspondiente a una matriz de enteros. Para manipular una matriz se requieren dos subíndices: uno para las filas F y otro para las columnas C, aunque pueden ser I y J, por ejemplo, o cualesquiera otras letras. Las filas se inician en 0, al igual que las columnas; por lo tanto, la matriz `matEnt` tiene las filas 0, 1, 2, 3 y las columnas 0, 1, 2, 3 (**figura 4.51**).

Figura 4.51: Representación gráfica de las filas y columnas de una matriz

	0	1	2	3	
					0
					1
					2
					3

El recorrido de una matriz se puede hacer por filas o por columnas; en ambos casos se requieren dos ciclos que permitan desplazarse entre filas y columnas, o entre columnas y filas.

Para recorrer la matriz del ejemplo por filas:

```

Para f ← 0 Hasta 3 Hacer
    Para c ← 0 Hasta 3 Hacer
        Escribir Sin Saltar A[f,c], " "
    FinPara
    Escribir ""
FinPara

```

Para recorrer la matriz del ejemplo por columnas:

```

Para c ← 0 Hasta 3 Hacer
    Para f ← 0 Hasta 3 Hacer
        Escribir Sin Saltar A[c,f], " "
    FinPara
    Escribir ""
FinPara

```

Cuando el número de filas es igual al número de columnas, se dice que la matriz es cuadrada (**figura 4.52**), en cuyo caso tiene diagonal principal y secundaria.

Figura 4.52: Representación gráfica de una matriz cuadrada de 4 por 4

		0	1	2	3	
matEnt	10	11	12	13	0	
	14	15	16	17	1	
	18	19	20	21	2	
	22	23	24	25	3	

Los elementos de la diagonal principal son los resaltados en azul (**figura 4.53**).

Figura 4.53: Elementos de la diagonal principal de una matriz de 4 por 4

		0	1	2	3	
matEnt	10	11	12	13	0	
	14	15	16	17	1	
	18	19	20	21	2	
	22	23	24	25	3	

Que corresponden a los elementos (0,0), (1,1), (2,2), (3,3). Los elementos que la componen tienen igual número de filas y columnas.

Los elementos de la diagonal secundaria son los resaltados en rojo (figura 4.54).

Figura 4.54: Elementos de la diagonal secundaria de una matriz de 4 por 4

	0	1	2	3	
matEnt	10	11	12	13	0
	14	15	16	17	1
	18	19	20	21	2
	22	23	24	25	3

Que corresponden a los elementos (0,3), (1,2), (2,1), (3,0). Las filas van aumentando (0, 1, 2, 3), mientras que las columnas disminuyen (3, 2, 1, 0).

A continuación, el algoritmo para crear una matriz de 4 x 4. Se ingresan los valores del ejemplo para llenarla, se visualiza su contenido por filas y por columnas y se muestran los elementos de la diagonal principal y de la secundaria. Para ello, entonces, se procede a definir la variable **matEnt** de tipo entero, se dimensiona la matriz con la sentencia `Dimension matEnt[4,4]`, y se crean las funciones **LlenarMatriz**, **MostrarMatrizPorFila**, **MostrarMatrizPorColumnas**, **MostrarDiagonalPrincipal** y **MostrarDiagonalSecundaria**, las cuales son invocadas desde el algoritmo principal.

Algoritmo Ejercicio16C4

```
//Definir o declarar la matriz
Definir matEnt Como Entero

//Dimensionar la matriz
Dimension matEnt [4,4]

//Asignarle valor a la matriz
LlenarMatriz(matEnt)

Borrar Pantalla

//visualizar su contenido
MostrarMatrizPorFila(matEnt)
MostrarMatrizPorColumnas(matEnt)
MostrarDiagonalPrincipal(matEnt)
MostrarDiagonalSecundaria(matEnt)
```

FinAlgoritmo

Funcion LlenarMatriz(A)

```
Definir f, c Como Entero
Para f ← 0 Hasta 3 Hacer
```

```
        Para c ← 0 Hasta 3 Hacer
            Escribir Sin Saltar "Valor para ", f, ", ", c, ": "
            Leer A[f,c]
        FinPara
    FinPara
FinFuncion
```

Funcion MostrarMatrizPorFila(A)

```
    Definir f, c Como Entero
    Escribir "El contenido de la matriz es:"
    Escribir ""
    Escribir "Por filas"
    Escribir ""
    Para f ← 0 Hasta 3 Hacer
        Para c ← 0 Hasta 3 Hacer
            Escribir Sin Saltar A[f,c], " "
        FinPara
        Escribir ""
    FinPara
```

FinFuncion

Funcion MostrarMatrizPorColumnas(A)

```
    Definir f, c Como Entero
    Escribir ""
    Escribir "Por columnas"
    Escribir ""
    Para c ← 0 Hasta 3 Hacer
        Para f ← 0 Hasta 3 Hacer
            Escribir Sin Saltar A[c,f], " "
        FinPara
        Escribir ""
    FinPara
```

FinFuncion

Funcion MostrarDiagonalPrincipal(A)

```
    Definir f Como Entero
    Escribir ""
    Escribir "Elementos diagonal principal"
    Escribir ""
    Para f ← 0 Hasta 3 Hacer
        Escribir A[f,f]
    FinPara
```

FinFuncion

Funcion MostrarDiagonalSecundaria(A)

```
Definir f, c Como Entero  
  
c ← 3  
Escribir ""  
Escribir "Elementos diagonal secundaria"  
Escribir ""  
Para f ← 0 Hasta 3 Hacer  
    Escribir A[f,c]  
    c ← c -1  
FinPara
```

FinFuncion

Al ejecutar el algoritmo e ingresar los datos del ejemplo, se visualiza en pantalla lo siguiente (**figura 4.55**).

Figura 4.55: Ejecución Ejercicio16C4

El contenido de la matriz es:

Por filas

10	11	12	13
14	15	16	17
18	19	20	21
22	23	24	25

Por columnas

10	11	12	13
14	15	16	17
18	19	20	21
22	23	24	25

Elementos diagonal principal

10
15
20
25

Elementos diagonal secundaria

13
16

Al igual que los vectores, las matrices también se pueden dimensionar en tiempo de ejecución indicando el número de filas y columnas que se deseen. También se pueden usar las funciones para entrada de datos con el propósito de verificarlos, garantizando que solo ingrese información de forma válida.

A continuación, un ejemplo para crear una matriz de N filas por M columnas. Se van a llenar con valores enteros comprendidos entre 1 y 100; luego se desea crear un vector con los elementos pares y otro con los elementos impares de la matriz, además de calcular el promedio de los números pares e impares, y, finalmente, visualizar el contenido de la matriz, de los dos vectores y el promedio de los pares e impares.

Para desarrollar este algoritmo se deben tener en cuenta los siguientes aspectos:

- Se debe capturar o leer el número de filas N y el número de columnas M para dimensionar la matriz y los dos vectores en tiempo de ejecución.
- Los vectores van a tener una dimensión de N*M porque se puede dar el caso de que todos los elementos ingresados a la matriz sean pares o impares.
- Se deben usar dos variables para almacenar la posición del último elemento en el vector par e impar; aquellas se inicializarán en -1 porque, al insertar un elemento en los vectores, es necesario incrementar en 1, por lo cual $-1 + 1 = 0$, primera posición en un vector.
- Se usará la función **LeerValorEntero**, con la cual se ha venido trabajando desde el capítulo III.
- Se creará la función **LlenarMatriz** para asignarle valores, a fin de visualizar la posición de la matriz en la cual se va a asignar un valor; se usa el mensaje **valor para A[0,0]**, **valor para A[0,1]**, **valor para A[0,2]**, etc. Se utilizan las funciones **Concatenar** y **ConvertirATexto**, explicadas anteriormente.
- Para saber si un número es par, se usa el operador **Mod**, visto en el capítulo I, en combinación con el condicional **SI (Si (A[f,c] Mod 2) = 0 Entonces)**; si el resultado es 0, es par; de lo contrario, es impar.
- Para calcular el promedio de los números pares e impares se deben recorrer los dos vectores por separado, a fin de obtener la sumatoria de sus valores, para, posteriormente, dividirla entre el número de elementos de cada uno de los vectores, representados en las variables **topePar + 1** y **topeImp + 1**. Esto porque, si hay,

por ejemplo, cinco elementos en el vector par, este tiene las posiciones 0, 1, 2, 3, 4, que serían el valor de **topePar**; como hay cinco elementos, se debe sumar 1.

- Finalmente, se crea una función para mostrar el contenido de la matriz, una para los contenidos de los vectores y una más para los promedios.

El algoritmo es el siguiente:

Algoritmo Ejercicio17C4

```
//Definir o declarar la matriz, los vectores y las variables requeridas
Definir matEnt, vecPar, vecImp, topePar, topeImp, sumaPar, sumaImp, n, m
    Como Entero
Definir proPar, proImp Como Reales

//Dimensionar la matriz y los vectores
n ← LeerValorEntero("Indique el número de filas de la matriz : ", 2, 5)
m ← LeerValorEntero("Indique el número de columnas de la matriz: ", 2, 5)

Dimension matEnt[n,m]
Dimension vecPar[n*m]
Dimension vecImp[n*m]

//topePar para guardar el número de elementos pares
//topeImp para guardar el número de elementos impares
//Se inicializan en -1 para que al sumarles 1 quede en 0 primera posición
//de un vector
topePar ← -1
topeImp ← -1

LlenarMatriz(matEnt, n, m)
ObtenerVectores(matEnt, vecPar, vecImp, n, m, topePar, topeImp)
proPar ← CalcularPromedio(vecPar, topePar)
proImp ← CalcularPromedio(vecImp, topeImp)
MostrarMatriz(matEnt, n, m)
MostrarVector(vecPar, topePar, "Vector par")
MostrarVector(vecImp, topeImp, "Vector impar")
MostrarPromedios(proPar, proImp)
```

FinAlgoritmo

```
//Función para leer o capturar un valor entero
Funcion val ← LeerValorEntero(msg, vi, vs) (ver Ejercicio1C3)

//Función para asignarle valores a la matriz
Funcion LlenarMatriz(A, n, m)
```



```

Definir f, c Como Enteros
Definir cad, cad1, cad2 Como Cadena

Para f ← 0 Hasta n-1 Hacer
    Para c ← 0 Hasta m-1 Hacer
        cad1 ← Concatenar("valor para A[",ConvertirATexto(f))
        cad2 ← Concatenar(", ", ConvertirATexto(c))
        cad ← Concatenar(cad1, cad2)
        Cad ← Concatenar(cad, "] ")
        A[f,c] ← LeerValorEntero(cad, 1, 100)
    FinPara
FinPara

FinFuncion

//Función para obtener los vectores con los valores pares e impares
Funcion ObtenerVectores(A, Vp, Vi, n, m, tp Por Referencia, ti Por Referencia)

    Definir f, c Como Enteros
    Para f ← 0 Hasta n-1 Hacer
        Para c ← 0 Hasta m-1 Hacer
            Si (A[f,c] Mod 2) = 0 Entonces
                tp ← tp + 1
                Vp[tp] ← A[f,c]
            SiNo
                ti ← ti + 1
                Vi[ti] ← A[f,c]
            FinSi
        FinPara
    FinPara

FinFuncion

//Función para calcular el promedio de los valores de un vector
Funcion res ← CalcularPromedio(A, tope)

    Definir i, sum Como Entero
    Definir res Como Real

    Si A[0] > 0 Entonces
        sum ← 0

        Para i ← 0 Hasta tope Hacer
            sum ← sum + A[i]
        FinPara

        res ← sum / (tope+1)

```

```
        SiNo
            res ← 0
        FinSi
FinFuncion

//Función para visualizar el contenido de la matriz
Funcion MostrarMatriz(A, n, m)
    Definir f, c Como Entero
    Borrar Pantalla
    Escribir ""
    Escribir "El contenido de la matriz es:"
    Escribir ""
    Para f ← 0 Hasta n-1 Hacer
        Para c ← 0 Hasta m-1 Hacer
            Escribir Sin Saltar A[f,c], " "
        FinPara
    Escribir ""
    FinPara
FinFuncion

//Función para visualizar el contenido de un vector
Funcion MostrarVector(A, tope, msg)
    Definir i Como Entero
    Escribir ""
    Si A[0] > 0 Entonces
        Escribir msg
        Para i ← 0 Hasta tope Hacer
            Escribir Sin Saltar A[i], " "
        FinPara
    Escribir ""
    FinSi
FinFuncion

//Función para visualizar los promedios
Funcion MostrarPromedios(p1, p2)
    Escribir ""
    Escribir "Promedio pares : ", Redon(p1 * 100) / 100
    Escribir "Promedio impares: ", Redon(p2 * 100) / 100
FinFuncion
```

Al ejecutar el algoritmo para crear una matriz de $3 * 4$, se obtiene lo siguiente (figura 4.56):

Figura 4.56: Ejecución Ejercicio17C4

```
*** Ejecución Iniciada. ***

Indique el número de filas de la
Indique el número de columnas de
valor para A[0,0] > 10
valor para A[0,1] > 15
valor para A[0,2] > 23
valor para A[0,3] > 45
valor para A[1,0] > 76
valor para A[1,1] > 89
valor para A[1,2] > 18
valor para A[1,3] > 36
valor para A[2,0] > 44
valor para A[2,1] > 65
valor para A[2,2] > 28
valor para A[2,3] > 54

El contenido de la matriz es:

10 15 23 45
76 89 18 36
44 65 28 54

Vector par
10 76 18 36 44 28 54

Vector impar
15 23 45 89 65

Pormedio pares : 38
Pormedio impares: 47.4
*** Ejecución Finalizada. ***
```

4.7 Ejercicios propuestos

1. Realizar un algoritmo para ingresar N valores enteros, determinar cuántos son primos y almacenarlos en un segundo vector; finalmente, determinar el promedio del vector inicial y el promedio del vector con los valores primos, y visualizar el contenido de los dos vectores y sus promedios.
2. Realizar un algoritmo para crear dos vectores de tipo real de N elementos cada uno, y obtener un tercer vector igual a la sumatoria de los dos anteriores, es decir, $V3[0] \leftarrow V1[0] + V2[0]$, $V3[1] \leftarrow V1[1] + V2[1]$, y así sucesivamente hasta la posición N .
3. Realizar un algoritmo para crear dos vectores de tipo entero y almacenar en cada uno de ellos los valores 1, 2, 3, 4, 5, 6, 7, 8, 9 mediante un menú que contenga las opciones **tablas de sumar**, **tablas de restar**, **tablas de multiplicar** y **tablas de dividir**. Además, obtener un tercer vector con el resultado de la operación seleccionada por el usuario; se deben visualizar las tablas según la operación seleccionada y permitir la ejecución de varias operaciones con una misma carga de datos.
4. Realizar un algoritmo para crear una matriz de $N \times M$ de tipo entero, además de determinar el valor mayor y menor almacenado en la matriz y las posiciones que ocupan estos.
5. Realizar un algoritmo para crear una matriz de $N \times N$, visualizar los elementos de la diagonal principal y secundaria, y determinar el promedio de sus elementos.
6. Una entidad hace préstamos a sus clientes a una tasa de interés N mensual. Se requiere un algoritmo que determine el valor que debe pagar un cliente por su préstamo mensualmente; además, se desea conocer el número de clientes que se le realizó préstamos, el total de dinero prestado, el total de dinero en intereses recibido en un mes y el total de capital luego de cancelados todos los intereses de cada uno de los préstamos en un mes; asimismo, almacenar en vectores los datos para cada cliente y visualizar sus contenidos a fin de generar un informe general.
7. Los empleados de una fábrica trabajan en dos turnos: diurno y nocturno. Se desea calcular el jornal diario de acuerdo con los siguientes puntos:

La tarifa de las horas diurnas es de 5 dólares.

La tarifa de las horas nocturnas es de 8 dólares.

En caso de ser domingo, la tarifa se incrementará en 2 dólares en turno diurno y 3 dólares en turno nocturno. Además, se desea conocer el total pagado a los empleados y el número de empleados procesados, así como almacenar los datos en vectores y visualizar el informe completo.

Referencias

Barber, F. y Ferrís, R. (s. f.). Tema 5: Subprogramas, programación modular [PDF]. http://informatica.uv.es/iiguia/AED/oldwww/2004_05/AED.Tema.05.pdf

EcuRed. (2019). Programación modular. https://www.ecured.cu/Programaci%C3%B3n_Modular

EcuRed. (2023). Arreglos (Informática). [https://www.ecured.cu/Arreglos_\(Inform%C3%A1tica\)](https://www.ecured.cu/Arreglos_(Inform%C3%A1tica))

GameDevTraum. (2023). ¿Qué es el pseudocódigo en programación? - Teoría de programación. <https://gamedevtraum.com/es/programacion-informatica/teoria-de-programacion/que-es-el-pseudocodigo-en-programacion/>

Geomatemáticas. (2005). Estructuras condicionales. <https://sites.google.com/site/geomatematicasyalgoritmos/estructuras/2-estructuras-condicionales>

Joyanes Aguilar, L. (2020). Fundamentos de programación, algoritmos, estructuras de datos y objetos. McGraw-Hill.

Juganaru Mathieu, M. (2015). Introducción a la programación. Grupo Editorial Patria.

Lemonaki, D. (2021). Ejemplos de funciones en Python: Cómo declarar y llamar con parámetros. <https://www.freecodecamp.org/espanol/news/ejemplos-de-funciones-de-python-como-declarar-y-llamar-con-parametros/>

Mancilla Herrera, A., Ebratt Gómez, R. y Capacho Portilla, J. (2015). Diseño y construcción de algoritmos. Editorial Universidad de Norte.

- Manuais. (2020). Python - Funciones definidas por el usuario. https://manuais.iessanclemente.net/index.php/Python_-_Funciones_definidas_por_el_usuario
- Moreno, E. (2012). Grafos: fundamentos y algoritmos. Editorial Ebooks Patagonia. Muñoz, J. D. (2010). Estructuras de datos: arreglos. <https://plataforma.josedomingo.org/pledin/cursos/programacion/curso/u26/>
- Murillo Escobar, A. (2014). Concepto de algoritmo [blog]. <https://andreiitam23.wordpress.com/2014/09/01/94/>
- Nieva, G. (2016). Programación modular. <https://dcodingames.com/programacion-modular-p1/>
- Novara, P. (2003a). ¿Qué es PSeInt? <https://pseint.sourceforge.net/>
- Novara, P. (2003b). ¿Para qué sirve PSeInt? <https://pseint.sourceforge.net/slide/pseint.html>
- Novara, P. (2003c). Arreglos. Obtenido de la ayuda de PSeInt.
- Oviedo Regino, E. M. (2012). Lógica de programación. Ecoediciones.
- Robledano, A. (2019). Qué es pseudocódigo. <https://openwebinars.net/blog/que-es-pseudocodigo/#:~:text=El%20pseudoc%C3%B3digo%20es%20una%20forma,a%20un%20lenguaje%20de%20programaci%C3%B3n.>
- Universidad Nacional del Rosario (s. f.). Programación modular: Subalgoritmos - funciones y procedimientos [PDF]. https://usuarios.fceia.unr.edu.ar/~sorribas/info1_notas_de_clase_3.pdf

Máximo Miguel Arteaga Martínez

Ingeniero informático de la Fundación Universitaria Católica del Norte, con estudios de máster en Derecho de Internet y de las Nuevas Tecnologías de las Comunicaciones del Instituto Europeo Campus Stellae de Santiago de Compostela (España), y maestría en Gestión de Tecnología de la Información de la Universidad Cooperativa de Colombia, sede Bucaramanga. Cuenta con 15 años de experiencia como profesor universitario en el área de Ingeniería de Sistemas e Informática, en modalidades presencial, distancia y virtual.



Este libro se terminó de editar en diciembre de 2023
La fuente tipográfica empleada es: Times New Roman
12 puntos en texto corrido.

El libro es producto de la experiencia de 32 años en docencia técnica, tecnológica y universitaria, su contenido se enfoca en el desarrollo de la lógica de programación base fundamental para estudiantes universitarios, tecnólogos, técnicos, para la educación y el desarrollo humano que incluyan cursos o asignaturas relacionadas con lógica de programación y en general para cualquier persona que desee incursionar en el mundo del desarrollo de software.

