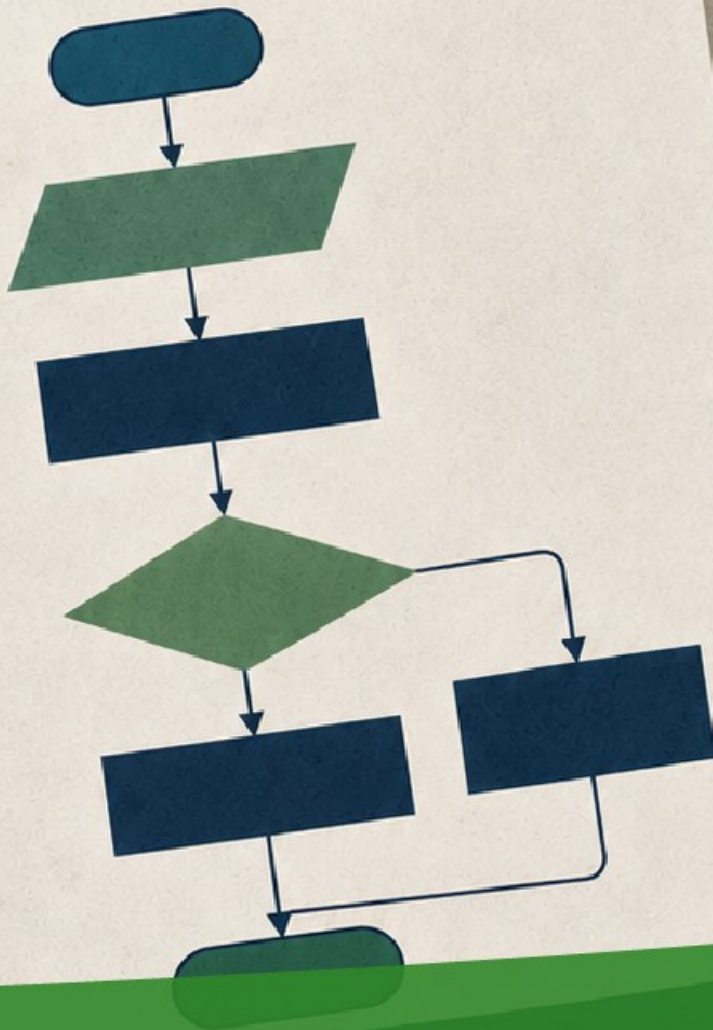


Fundamentos de algoritmos

Diagramas de Flujo y Pseudocódigos



PSEUDOCÓDIGO

Algoritmo Ejemplo

Definir n , i , suma Como Entero

suma $\leftarrow 0$

Escribir "Ingrese n :"

Leer n

Para $i \leftarrow 1$ Hasta n Hacer

 suma $\leftarrow$ suma + i

FinPara

Escribir "La suma es: ", suma

FinAlgoritmo

Winther Molina Loo

```
01 | if (condicion) {  
02 |     resultado = calcular();  
03 | } else {  
04 |     resultado = alternativo();  
05 | }  
06 | mostrar(resultado);
```


Ediciones
Ulearn

Fundamentos de algoritmos

Diagramas de Flujo y Pseudocódigos

Ing. Winther Abel Molina Loor, Mg.





Texto arbitrado bajo la modalidad doble par ciego

Universidad Laica Eloy Alfaro de Manabí

Ciudadela universitaria vía circunvalación (Manta)

www.uleam.edu.ec

Dr. Marcos Zambrano Zambrano, PhD.

Rector

Dra. Jackeline Terranova Ruiz, PhD.

Vicerrectora Académica

Lcdo. Kléver Delgado Reyes, Mg.

Dirección de Investigación, Publicaciones y Servicios Bibliográficos

Fundamentos de algoritmos Diagramas de Flujo y Pseudocódigos

Ing. Winther Abel Molina Loor, Mg.

Edición: Primera. Mayo de 2026. Publicación digital

ISBN: 978-9942-681-96-6

Prohibida su venta

Trabajo de edición y revisión de texto: Mg. Alexis Cuzme Espinales

Diagramación, edición de estilo y diseño de portada: Mg. José Márquez Rodríguez

Una producción de la Universidad Laica Eloy Alfaro de Manabí, registrada en la Cámara Ecuatoriana del Libro.

Sitio Web: uleam.edu.ec

Correo institucional: diist@uleam.edu.ec

Teléfonos: 2 623 026 Ext. 255

Todos los derechos reservados ©2026

Prohibida su venta

Dedicatoria

A mi amada esposa,

Mónica

A mis hijos,

Karen Betsabé

Crhis Josué

Winther Joel

A mis nietos,

Ian Alessandro

Haziel Alejandro

Zaid Alexander

Índice General

Capítulo I. Fundamentos conceptuales de los algoritmos y la programación.....	14
1.1 La computadora como sistema de procesamiento de información.....	15
1.2 El Software.....	15
1.2.1 Características del Software	16
1.2.2 Objetivos básicos del Software	17
1.3 El Hardware	17
1.3.1 Componentes básicos del hardware	17
1.4 Comparación resumida hardware vs. software.....	18
1.5 La Programación y Programa.....	19
1.5.1 La Programación	19
1.5.2 Programa	19
1.5.2.1 Características esenciales de un programa	19
1.5.2.2 Relación Programa vs. Software	20
1.6 El Desarrollador y el Programador.....	20
1.6.1 El programador	20
1.6.2 El Desarrollador	21
1.6.3 Programador versus Desarrollador	22
1.7 Lenguajes de Programación	22
1.7.1 Componentes fundamentales de un Lenguaje de Programación	23
1.8 Paradigmas de la Programación.....	24
1.9 Áreas de la Programación.....	25
1.10 Dato e Información	27
1.10.1 El Dato.....	27
1.10.2 La Información	27
1.10.3 ¿Por qué es importante la Información?	28
1.11 La Lógica.....	29
1.12 El Razonamiento	30
1.12.1 El Razonamiento Lógico-Matemático.....	32
1.12.2 El Razonamiento en el ámbito de la programación	35

1.13 Metodologías Cognitivas para la resolución de problemas.....	36
1.13.1 Razonamiento Analógico.....	36
1.13.2 Por Descubrimiento	37
1.13.3 Pensamiento Lateral	37
1.14 El problema.- Definición	38
1.14.1 Generalidades de un problema	39
1.14.2 Características de un Problema	39
1.14.3 Elementos Comunes de un Problema	40
1.14.3.1 El Enunciado	40
1.14.3.2 El Procedimiento	41
1.14.3.3 El Resultado	41
1.15 Los Algoritmos.....	46
1.15.1 Concepto de Algoritmo y su evolución	46
1.15.2 Importancia del Algoritmo en la Informática	47
1.15.3 Características de los Algoritmos	48
1.15.4 Estructura de un Algoritmo	49
1.15.5 Tipos de Algoritmos	51
1.15.5.1 Algoritmos Cualitativos	51
1.15.5.2 Algoritmos Cuantitativos.....	53
1.15.6 Formas de Representar un Algoritmo.....	57
1.15.6.1 Representación en Lenguaje Natural	57
1.15.6.2 Diagrama de Flujo	58
1.15.6.3 Pseudocódigo.....	58
1.15.6.4 Representación en Código (programa en un lenguaje de programación)	59
Capítulo II. Representación formal de algoritmos.....	60
2.1 Diagrama de Flujo	61
2.2 Símbolos básicos del diagrama de flujo.....	61
2.3 Reglas para elaborar diagramas de flujo	62
2.4 Primer acercamiento a los diagramas de flujo.....	63
2.5 Elementos básicos de los algoritmos en diagramas de flujo.....	64
2.5.1 Dato y Tipos de Datos.....	65
2.5.1.1 Dato	65
2.5.1.2 Tipos de Datos Básicos (datos primitivos)	65
2.5.1.3 Características y propiedades de los Tipos de Datos Primitivos	66

2.5.2 Almacenamiento de información: Variables	68
2.5.2.1 Definición	68
2.5.2.2 Características principales de una variable	69
2.5.2.3 Clasificación de las Variables según el Tipo de Información	69
2.5.2.3.1 Variables Numéricas	69
2.5.2.3.2 Variables Alfanuméricas	70
2.5.2.3.3 Variables Lógicas	70
2.5.2.4 Asignación de nombres a las variables	70
2.5.2.5 Cómo Declarar Variables en un Diagrama de Flujo.....	72
2.5.2.6 Paradigma de Programación a utilizar en los Diagramas de Flujo	74
2.5.3 Las Constantes	81
2.5.3.1 Definición	81
2.5.3.2 Constantes Numéricas	81
2.5.3.3 Constantes Alfanuméricas	82
2.5.3.4 Constante Lógica	83
2.5.4 Las Expresiones	84
2.5.4.1 Expresiones formadas por Procesos Aritméticos.....	85
2.5.4.2 Expresión formada solo por Variable.....	86
2.5.4.3 Expresión formada solo por Constante.....	87
2.5.5 Los Operadores.....	87
2.5.5.1 Los Operadores Aritméticos	88
2.5.5.1.1 Jerarquía de los Operadores Aritméticos	89
2.5.5.1.2 Consideraciones en una Expresión Aritmética	89
2.5.5.1.3 Operador Aritmético MOD (Módulo o Residuo)	91
2.5.6 La Prueba de Escritorio.....	93
2.5.6.1 La importancia de la Prueba de Escritorio	93
2.5.6.2 Pasos para realizar una prueba de escritorio.....	94
2.5.7 El Pseudocódigo	98
2.5.7.1 Características de un Pseudocódigo	98
2.5.7.2 Estructura de un Pseudocódigo.....	98
2.5.7.3 Instrucciones formales y básicas de un Pseudocódigo	99
2.5.7.3.1 Para comentarios	100
2.5.7.3.2 Para Definir variables y sus Tipos de Datos.....	100
2.5.7.3.3 Para Ingresar Datos	101
2.5.7.3.4 Para Mostrar Datos o Información	101
2.5.7.3.5 Para hacer Cálculos Aritméticos	101
2.5.7.3.6 Para realizar Condiciones	104

2.5.7.3.7 Para realizar Bucles o Repeticiones	104
2.5.7.3.8 La Indentación	105
2.5.8 Estructuras de Control	105
2.5.8.1 Estructura Secuencial	106
2.5.8.1.1 Consideraciones al utilizar una estructura secuencial	106
2.5.8.2 Estructuras Condicionales	110
2.5.8.2.1 Introducción y Definición	110
2.5.8.2.2 Operadores de Relación	111
2.5.8.2.3 Tipos de Estructuras Condicionales	112
2.5.8.2.3.1 Estructuras Condicionales Simples	112
2.5.8.2.3.2 Estructuras Condicionales Dobles	120
2.5.8.2.3.3 Estructuras Condicionales Anidadas	126
2.5.8.2.3.4 Estructuras Condicionales Dobles - Compuestas	135
2.5.8.2.3.5 Operadores Lógicos	136
2.5.8.2.3.5.1 Operador lógico OR	136
2.5.8.2.3.5.2 Operador lógico AND	143
2.5.8.2.3.5.3 Operador lógico NOT	150
2.5.8.2.3.5 Estructuras Condicionales Múltiples o Selectivas	152
2.5.8.2.3.6 Consideraciones al utilizar condiciones Múltiples o Selectivas	153
2.5.8.3 Estructuras Repetitivas o de Iteración	161
2.5.8.3.1 Tipos de Estructuras repetitivas en programación	162
2.5.8.3.1.1 Estructura Repetitiva Para / Desde (For)	162
2.5.8.3.1.2 Contadores y Acumuladores	176
2.5.8.3.1.2.1 Variable de Tipo Contador	176
2.5.8.3.1.2.2 Variables Tipo Acumulador	182
2.5.8.3.1.3 La Validación de Datos	192
2.5.8.3.1.4 Importancia de la Validación de Datos	192
2.5.8.3.1.5 Controles de Validación de Datos	193
2.5.8.3.1.6 Tipos de Mensajes en la Validación de Datos	194
2.5.8.3.1.7 Estructura Repetitiva Mientras (While)	195
2.5.8.3.1.8 Estructura Repetitiva Repetir/Hasta (Repeat/Until)	202
2.5.8.3.1.9 Bucles Anidados con Estructuras Repetitivas	210
2.5.8.3.1.10 Esquema de las Estructuras Repetitivas Anidadas	211
Capítulo III. Manipulación de Cadenas	219
3.1 Cadenas	220
3.1.1 Definición	220

3.1.2 Diferencia entre Carácter y Cadena	220
3.2 Importancia de las Cadenas en Programación	221
3.2.1 En el ingreso, almacenamiento y presentación de datos	221
3.2.2 En la validación de datos	221
3.2.3 En el procesamiento de datos	221
3.2.4 En el tratamiento de cadenas en los Lenguajes de Programación	222
3.3 Códigos ASCII	222
3.4 Operaciones Fundamentales con Cadenas.....	223
3.4.1 Operaciones con Cadenas: Comparación	224
3.4.2 Operaciones con Cadenas: Longitud o Tamaño	225
3.4.3 Operaciones con Cadenas: Concatenación	228
3.4.4 Operaciones con Cadenas: Subcadenas	233
3.4.4.1 Función para obtener caracteres: SubCadena	233
3.4.5. Otras Funciones para Manipular Cadenas	245
3.4.5.1 Función CHR o caracter	245
3.4.5.2 Función ASC o ASCII	245
3.4.5.3 Función UPPER o Mayusculas	248
3.4.5.4 Función LOWER o Minusculas	248
3.4.5.5 Función VAL o ConvertirANumero	249
3.4.5.6 Función STR o ConvertirATexto	250
Capítulo IV. Modularización y uso de Funciones	255
4.1 Modularización	256
4.1.1 Ventajas de la Modularización	256
4.2 Diseño Top-Down.....	257
4.3 Elementos del Diseño Modular	258
4.3.1 Módulo Principal	258
4.3.2 Módulos o Submódulos	259
4.3.2.1 Consideraciones al utilizar los Módulos o Submódulos	260
4.3.2.2 Llamada a Ejecución de un Módulo o Submódulo	261
4.3.3 Ambientes de Trabajo en el Diseño Modular	266
4.3.4 Tipos de Variables Según su Ambiente de Trabajo	266
4.3.4.1 Variables Globales	267
4.3.4.2 Variables Locales	268

4.4 Parametrización	270
4.4.1 Parámetros	270
4.4.1.1 Clasificación de los Parámetros Según su Función en la Llamada al Módulo	270
4.4.1.1.1 Parámetros Actuales o Argumentos	271
4.4.1.1.2 Parámetros Formales.....	271
4.4.1.2 Clasificación de los Parámetros Según la forma de Transmisión de Datos al Módulo	274
4.4.1.2.1 Parámetros por Valor	275
4.4.1.2.2 Parámetros por Referencia	276
4.5 Funciones.....	279
4.5.1 Clasificación de las Funciones	279
4.5.2 Funciones Predefinidas en los Lenguajes de Programación	279
4.5.2.1 Funciones numéricas	280
4.5.2.2 Funciones para uso de Cadenas.....	280
4.5.3 Beneficios de las Funciones Predefinidas en los Lenguajes de Programación	280
4.5.4 Funciones Definidas o Creadas por el Desarrollador	281
4.5.4.1 Beneficios de las Funciones Creadas por el Desarrollador	283
Capítulo V. Vectores y Matrices: Introducción a los Arreglos.....	292
5.1 Introducción a los Arreglos	293
5.2 Tipos de arreglos	295
5.2.1 Arreglos Unidimensionales o Vectores	295
5.2.1.1 Creación de Arreglos Unidimensionales o Vectores	296
5.2.1.2 ¿Cómo Trabajar con los Elementos de un Vector?.....	297
5.2.1.3 Consideraciones al Utilizar Vectores.....	299
5.2.2 Arreglos Bidimensionales (Matrices)	312
5.2.2.1 Creación de Arreglos Bidimensionales o Matrices	312
5.2.2.2 Consideraciones al Utilizar Matrices	314
Bibliografía.....	323
Autor	324

Prefacio

Aprender a programar ha sido, uno de los desafíos más significativos para los estudiantes que inician su formación en carreras universitarias relacionadas con la informática. La abstracción requerida para transformar un problema real en una serie de instrucciones lógicas suele ser un obstáculo, ya que no se trata únicamente de memorizar instrucciones de un lenguaje de programación, sino en el reto de aplicar una forma distinta de pensar: lógica, estructurada y orientada a la resolución de problemas. Este proceso, en muchos casos, resulta abstracto y difícil de asimilar en las primeras etapas del aprendizaje.

Como consecuencia de esta dificultad, es común observar que numerosos estudiantes enfrentan obstáculos significativos en asignaturas relacionadas con algoritmos y programación. Esta situación se refleja lamentablemente en bajas calificaciones, otros reprueban repetidamente, y no son pocos los que, ante la frustración, optan por abandonar sus estudios ya que sienten que la lógica de la programación es inalcanzable. Esta realidad, presente en múltiples contextos educativos, evidencia la necesidad de contar con recursos didácticos más claros, accesibles y efectivos.

Este libro surge precisamente como una respuesta a esa necesidad. Está dirigido a personas que se inician en el aprendizaje de la programación y que requieren una base sólida para comprender sus fundamentos. Parte de una premisa esencial: antes de escribir código, es imprescindible aprender a pensar en términos de algoritmos. Es en esta etapa donde se construye el verdadero cimiento del desarrollo del pensamiento lógico y algorítmico.

El contenido de esta obra se fundamenta principalmente en la experiencia docente del autor a lo largo de muchos años, en los cuales ha sido testigo directo de las dificultades, avances y necesidades de los estudiantes en su proceso de formación. A partir de esta experiencia, se ha estructurado y diseñado un enfoque didáctico que busca facilitar la comprensión de conceptos abstractos mediante una progresión ordenada, práctica y contextualizada para los estudiantes de primeros niveles.

Con el propósito de facilitar el aprendizaje, el texto ha sido redactado en un lenguaje claro y accesible, eliminando las barreras de la terminología excesivamente técnica para que cualquier persona, sin importar su nivel previo, pueda comprender

los cimientos de la programación. Por lo tanto, para guiar este aprendizaje, el contenido del libro está organizado en los cinco siguientes capítulos, diseñados para llevar al lector, desde un nivel básico hasta un nivel progresivamente más avanzado:

Capítulo 1: Fundamentos conceptuales de los algoritmos y la programación. Aquí se abordan los fundamentos conceptuales de los algoritmos y la programación, estableciendo las bases necesarias para comprender el resto del contenido.

Capítulo 2: Representación formal de algoritmos. En este capítulo se desarrolla la capacidad de analizar y expresar soluciones algorítmicas de manera estructurada mediante diagramas de flujo y pseudocódigos como herramientas formales de representación. Para ello, se estudia y se aplican los componentes básicos de los algoritmos, entre ellos las estructuras de control (secuenciales, condicionales y repetitivas), con el propósito de potenciar el pensamiento lógico y algorítmico en la resolución de problemas.

Capítulo 3: Manipulación de cadenas. En el desarrollo y utilización de programas, la información que se utiliza no siempre se limita a valores numéricos. Por lo tanto, en este capítulo se desarrolla la manipulación de cadenas, un aspecto crucial en el tratamiento de texto y datos alfanuméricos.

Capítulo 4: Modularización y uso de Funciones. Se introduce la importancia del concepto de modularización como una estrategia de aprendizaje para estructurar y organizar problemas en partes más pequeñas y manejables que faciliten su comprensión, solución y desarrollo, mediante el uso de módulos y funciones.

Capítulo 5: Vectores y Matrices: Introducción a los Arreglos. Finalmente, se aborda el almacenamiento y gestión de conjuntos de datos, mediante vectores y matrices, sentando las bases para el manejo de estructuras de datos más complejas.

El fundamento pedagógico de este texto se basa en el “Constructivismo y las Metodologías Activas”, por lo que la filosofía de este libro es “aprender haciendo”, ya que la teoría por sí sola no es suficiente. Por lo tanto, desde el primer capítulo se incorporan una amplia variedad de problemas resueltos que permiten al lector observar la aplicación práctica de la teoría. De igual manera, se incluyen numerosos problemas propuestos a lo largo de todos los capítulos, diseñados para fomentar la práctica en el contexto de la didáctica de la programación, según el ambiente de aprendizaje que se aplique. Estos ejercicios han sido organizados de manera

progresiva, bajo un criterio de “complejidad incremental” avanzando desde situaciones simples hasta problemas de mayor complejidad.

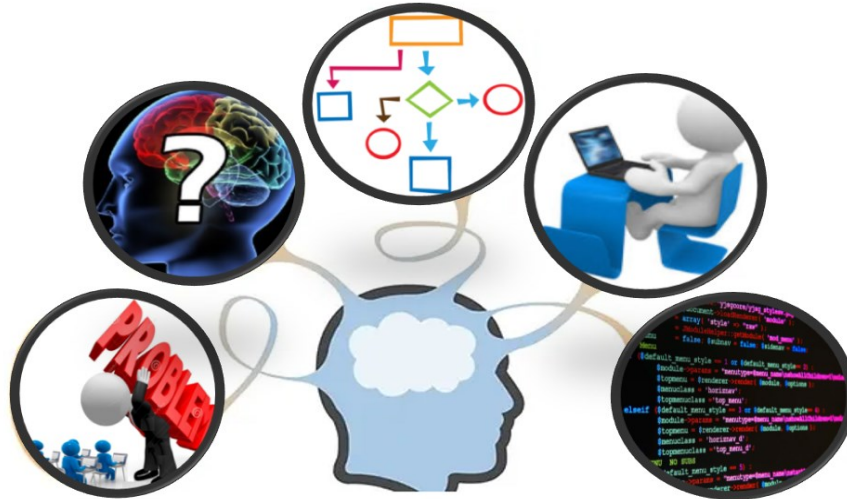
Un elemento distintivo que el autor ha implementado en este texto es el uso de una etiqueta gráfica de “importante”, la cual aparece en todos los capítulos, para destacar aspectos claves de cada tema o para proporcionar aclaraciones analíticas en los diagramas de flujo y pseudocódigos ya elaborados. Este recurso busca orientar la atención, asegurando que del lector no pase por alto detalles hacia aquellos “temas o puntos” que requieren especial comprensión.

Finalmente, es importante señalar que este libro es el resultado de un proceso investigativo titulado “La Didáctica de la Programación en la creación de Algoritmos para el fortalecimiento de la Lógica en Carreras Informáticas. Caso de estudio: Carreras de Tecnologías de Información y Software de la ULEAM”. Dentro de los objetivos de dicha investigación se planteó la elaboración de un material didáctico que contribuyera al proceso de enseñanza y aprendizaje de algoritmos, con un enfoque práctico centrado en la resolución de problemas.

Mi deseo es que esta obra sirva como una guía y apoyo de estudio, para quienes enfrentan el desafío de aprender a programar, brindándoles las herramientas básicas necesarias para convertir la dificultad inicial en dominio técnico, mediante el desarrollo del pensamiento lógico y algorítmico y a futuro consolidarse, como profesionales competentes en el área de las tecnologías de la información.

Bienvenidos al aprendizaje significativo de los algoritmos y la programación.

Capítulo I. Fundamentos conceptuales de los algoritmos y la programación



Objetivo General del Capítulo

Establecer las bases para comprender los fundamentos conceptuales de los algoritmos y la programación, abordando desde los conceptos básicos de software hasta la formulación de algoritmos, con el propósito de iniciar al estudiante en el análisis de problemas y en desarrollo del pensamiento algorítmico.

1.1 La computadora como sistema de procesamiento de información

Antes de abordar los elementos que conforman un sistema informático, es importante comprender qué se entiende por computadora en el contexto de la programación. Aunque existen muchas definiciones, en forma general, **una computadora** es un sistema electrónico (máquina) capaz de recibir datos, procesarlos mediante un conjunto de instrucciones y generar resultados. Estas instrucciones son escritas por el ser humano y organizadas de manera lógica a través de algoritmos.

Desde esta perspectiva, la computadora no funciona por sí sola, sino que requiere de programas que le indiquen **qué hacer** y **cómo hacerlo**. Dichos programas están contruidos a partir de algoritmos, los cuales constituyen la base para la resolución de problemas mediante el uso de la tecnología.

En este contexto, es necesario comprender que una computadora está formada por dos componentes fundamentales que trabajan de manera conjunta: el software y el hardware.

A continuación, se abordará en primer lugar el estudio del software, para posteriormente analizar el hardware como parte integral del funcionamiento de la computadora.

1.2 El Software

El software es el término que se utiliza para denominar los componentes lógicos o no físicos de un sistema informático. Los sistemas informáticos se constituyen de un componente físico (Hardware) y un componente lógico (Software), que es en el cual nos vamos a enfocar en este capítulo. Estos componentes lógicos están formados por instrucciones lógicas y detalladas, que permiten realizar diferentes tareas a través de los componentes físicos (Hardware) en un sistema informático (figura 1). Imagine que se desea cocinar una pizza: Se tiene harina, huevos, pasta de tomate, pepperoni y queso. Estos elementos, por sí solos, no hacen una pizza. Sin embargo, una secuencia de pasos lógicos o instrucciones son los que harían posible la ejecución del plato mencionado. Estas instrucciones o procesos lógicos son lo que conoceríamos como Software.

Figura 1

Interacción entre Software y Hardware



Nota. Fuente: Elaboración propia basada en imágenes obtenidas de diversas fuentes disponibles en la Web.



El **Software** es un término informático procedente del idioma inglés, que es aceptado por la RAE y que no posee una traducción que se ajuste al español.

1.2.1 Características del Software

El software es un “*producto intangible*”, que por su naturaleza tiene varias características propias y únicas que lo diferencian de otros “productos físicos” de las ingenierías y tecnologías:

- Se “desarrolla o construye, no se fabrica”. Por lo que se trata de un producto lógico, “sin existencia física”, pero que es utilizable en una máquina y/o dispositivo.
- El software se crea en base a la “lógica” para resolver un “problema” y satisfacer alguna “necesidad” de los usuarios.
- El software está formado por “estructuras lógicas, algoritmos y datos”, no por componentes físicos. Por lo tanto, su construcción depende del razonamiento lógico.
- El software “No se degrada con el uso, ni se desgasta con el tiempo”, que sin embargo si se produce en el soporte físico (hardware) que lo contiene.
- El software “es modificable y evolutivo” por lo que puede ser actualizado y adaptado, según las necesidades y entornos para su uso.
- El software puede tener “altos niveles de complejidad” debido a su estructura interna y ello dependerá de su propósito de creación y de los elementos internos con que fue desarrollado.

- El software es “propenso a errores lógicos”, por lo que debe cumplir con estándares mínimos de “calidad” para su funcionamiento. Por eso requiere pruebas de V&V (Verificación y Validación).
- El mantenimiento del software suele ser más complicado que el del hardware. Si una pieza del hardware se deteriora se sustituye, pero un “fallo” en el software implica un “error en su desarrollo y/o diseño” o en su implementación.
- Para construir un software, se pueden reutilizar “fragmentos ya desarrollados” (reutilización de código).

1.2.2 Objetivos básicos del Software

El software debe cumplir con las siguientes propiedades básicas:

- **Funcionalidad:** Que cumpla su objetivo.
- **Usabilidad:** Que su uso sea fácil de entender.
- **Seguridad:** Que su ejecución no implique un riesgo para el equipo del usuario.
- **Flexibilidad:** Que pueda ser modificado para diferentes condiciones en su uso.
- **Portabilidad:** Que pueda ser utilizado en diferentes equipos o hardware.

1.3 El Hardware

El hardware corresponde al conjunto de componentes físicos y tangibles que conforman un sistema informático (o de cómputo). Incluye todos los dispositivos electrónicos, mecánicos y electromagnéticos que permiten la ejecución del software (programas) y el procesamiento de datos.

1.3.1 Componentes básicos del hardware

Los componentes básicos del hardware constituyen la estructura fundamental de todo sistema informático (o computacional). Cada uno de estos elementos cumple una función específica dentro del proceso de entrada, procesamiento, almacenamiento y salida de la información. Su correcta interacción permite la ejecución eficiente de los programas y el funcionamiento integral del computador y otros dispositivos computacionales. Sus componentes principales son:

- **Unidad Central de Procesamiento (CPU):** Es el “cerebro” de la computadora. Ejecuta instrucciones, realiza cálculos y controla el funcionamiento general.
- **Memoria principal (RAM):** Almacena temporalmente los datos y programas que están en ejecución.
- **Dispositivos de almacenamiento:** Permiten guardar información de forma permanente, como discos duros, unidades SSD y memorias externas.
- **Dispositivos de entrada:** Permiten ingresar datos al sistema, como teclado, ratón, micrófono y cámara.
- **Dispositivos de salida:** Presentan la información procesada, como monitor, impresora y altavoces.

1.4 Comparación resumida hardware vs. software

A continuación, la tabla 1 presenta una comparación resumida entre hardware y software con la finalidad de destacar sus principales diferencias y facilitar la comprensión de sus características fundamentales.

Tabla 1

Comparación resumida entre Software y Hardware

Hardware	Software
Físico	Lógico
Tangible	Intangible
Se desgasta físico con el uso	No se desgasta por el uso
Vida útil limitada	Evolutivo
Reemplazable	Actualizable

Nota. Fuente: Elaboración propia



Software sin hardware “no puede ejecutarse” y Hardware sin **software** “no es funcional”.

1.5 La Programación y Programa

1.5.1 La Programación

La programación es un proceso lógico, mediante la cual se diseñan y construyen programas informáticos que permitan resolver problemas y automatizar tareas. Para programar se requiere de la persona principalmente:

- Capacidad de análisis y habilidades lógicas para resolver problemas mediante la programación.
- Conocimientos de lenguajes de programación (uno o varios, tales como C++, Java, Python, PHP, etc.) y herramientas de desarrollo que le permitan escribir programas.
- Demás competencias establecidas en el perfil de su formación.



La programación convierte las ideas y los algoritmos en soluciones ejecutables en una computadora

1.5.2 Programa

Un programa es un conjunto ordenado y estructurado de instrucciones, escritas en un lenguaje de programación, que al ser interpretadas y ejecutadas por una computadora y/o dispositivo, ejecutan un algoritmo para resolver un problema o realizar una tarea específica.

Para comprender correctamente el proceso de programación, es fundamental establecer la relación existente entre el algoritmo y el programa. Mientras el **algoritmo define qué** se debe hacer (solución lógica), el **programa especifica cómo** debe hacerlo la computadora.

1.5.2.1 Características esenciales de un programa

- Está escrito en un lenguaje de programación.
- Traduce un algoritmo a código ejecutable.
- Puede ser procesado por la computadora.
- Permite automatizar procesos.
- Produce resultados verificables.

1.5.2.2 Relación Programa vs. Software

Aunque los términos programa y software suelen utilizarse comúnmente en forma semejante, desde el punto de vista técnico no son equivalentes. Un programa constituye una unidad funcional específica, mientras que el software engloba a conjunto de programas, datos y configuraciones que permiten el funcionamiento integral de un sistema computacional. Para entenderlo mejor, veamos el siguiente ejemplo: Windows tiene muchos programa internos.

- Programas: El explorador de archivos - El Navegador Microsoft Edge
- Software: El sistema operativo Windows



El **programa** es la unidad básica de ejecución y el **software** es el sistema completo de funcionamiento.

1.6 El Desarrollador y el Programador

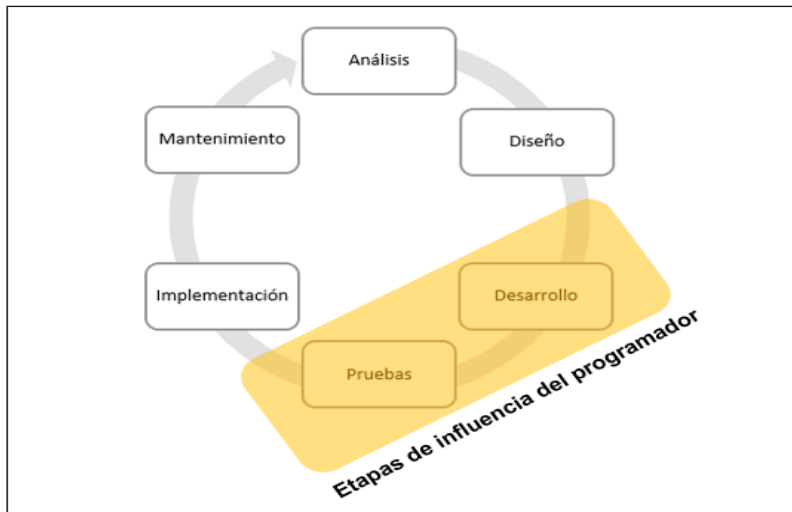
1.6.1 El programador

El programador se enfoca principalmente en la parte técnica y detallada de la creación y depuración de programas para soluciones informáticas específicas (figura 2). Su tarea principal es traducir los requisitos y diseños en instrucciones de código escrito en un lenguaje de programación, que una computadora (u otro dispositivo) puede ejecutar. Este profesional tiene las siguientes características:

- Formación técnica (o empírica) o algo entre medio.
- Conocimiento profundo en lenguajes de programación y herramientas tecnológicas.
- Habilidades técnicas en escribir, depurar y solución de problemas de código.
- Suele trabajar con algoritmos, estructuras de datos y lógica de programación.
- Puede no participar en etapas como el análisis de requisitos o el diseño del sistema.
- A nivel de autonomía, sigue las instrucciones y directrices detalladas dadas por otros.

Figura 2

Ciclo de Vida del Desarrollo del Software



Nota. Fuente: Elaboración propia.

1.6.2 El Desarrollador

El desarrollador de software es el profesional, encargado de transformar las necesidades de un usuario en soluciones informáticas funcionales. Generalmente tienen un enfoque más amplio (integral) y estratégico que abarca todo el ciclo de vida del desarrollo de software (figura 2). No solo programa, sino que también participa en varias de las fases de este ciclo, que incluye el análisis de requisitos, diseño, desarrollo, pruebas, implementación y mantenimiento del software. Este profesional informático tiene las siguientes características:

- Formación en ingeniería de software.
- Liderazgo y conocimiento en gestión de proyecto, lenguajes de programación y herramientas tecnológicas.
- Capacidad para visualizar el proyecto final y cómo las diferentes piezas del software interactúan entre sí y cómo resolver problemas reales.
- Aplicación de metodologías y estándares para el desarrollo de software en trabajo colaborativo con equipos multidisciplinarios.
- A nivel de autonomía, puede tomar decisiones de alto nivel sobre arquitectura, herramientas y tecnologías a usar.



En el lenguaje cotidiano, al desarrollador suele denominarse también programador; sin embargo, el término desarrollador resulta más amplio y preciso desde el punto de vista académico y profesional.

1.6.3 Programador versus Desarrollador

Aunque "desarrollador de software" y "programador de software" suelen usarse a menudo como sinónimos, no significan exactamente lo mismo, especialmente en el ámbito de la tecnología, aunque si están muy relacionados. En la práctica cotidiana, especialmente en empresas pequeñas o proyectos ágiles, una misma persona puede actuar como programador y desarrollador al mismo tiempo, por lo que muchas veces los términos se usan indistintamente (tabla 2).

Tabla 2

Características de la participación en ciclo de vida del desarrollador y el programador

Características	Desarrollador	Programador
Participa en el análisis de requisitos	✓	✗
Usa metodologías ágiles	✓	✗ / ✓
Diseña arquitectura del sistema	✓	✗
Escribe código	✓	✓
Participa en Pruebas y Mantenimiento	✓	✗ / ✓
Implementación	✓	✗
Toma decisiones técnicas	✓	✗
Enfoque	Integral	Técnico

Nota. Fuente: Propia del autor a partir de diversas fuentes consultadas en línea



Todo **desarrollador** debe saber programar, pero no todo **programador** necesariamente desarrolla software de forma integral

1.7 Lenguajes de Programación

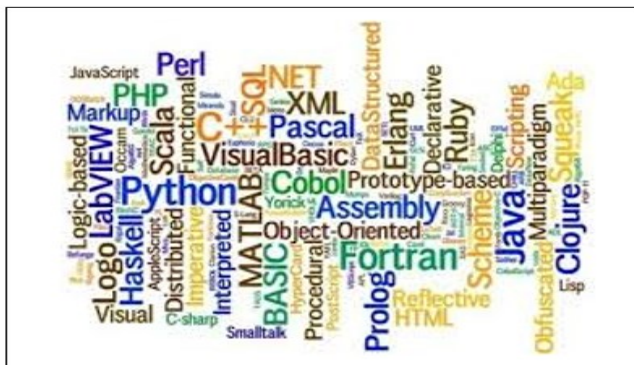
Así como los seres humanos utilizan idiomas para comunicarse, los desarrolladores/programadores emplean lenguajes de programación para expresar instrucciones que la computadora puede interpretar y ejecutar. Un Lenguaje de

Programación es un *idioma artificial*, formado por un conjunto de símbolos y reglas sintácticas y semánticas, que permite a un desarrollador expresar algoritmos, transformándolos en instrucciones comprensibles para una computadora, con el fin de desarrollar programas que resuelvan problemas específicos.

En el área informática, existen múltiples lenguajes de programación (figura 3), cada uno diseñado con propósitos específicos, niveles de complejidad distintos y enfoques particulares, como el desarrollo web, aplicaciones móviles, software de escritorio, inteligencia artificial, videojuegos o sistemas embebidos. El desarrollador, selecciona el lenguaje de programación en base a las características y particularidades del software que va construir

Figura 3

Lenguajes de Programación, clásicos y contemporáneos



Nota. Fuente: Imagen disponible en la web.

1.7.1 Componentes fundamentales de un Lenguaje de Programación

Todo lenguaje de programación está compuesto por un conjunto de componentes que permiten escribir las instrucciones comprensibles para la computadora. Con los siguientes elementos se construyen los programas y cómo se interpretan sus órdenes.

- **Sintaxis:** Reglas para escribir bien el código o programa.
- **Semántica:** Significado de las instrucciones.
- **Palabras reservadas:** Instrucciones y comandos propios del lenguaje.
- **Variables:** Espacios donde se guarda y se manipula la información.
- **Tipos de datos:** Definen la clase de información que se usa (números, letras, verdadero o falso).

- **Operadores:** Símbolos para hacer operaciones: matemáticas, lógicas y de comparación.
- **Estructuras de control:** Permiten tomar decisiones y repetir acciones (condicionales y bucles).
- **Funciones:** bloques de instrucciones que realizan una tarea específica.



Los Lenguajes de Programación son el corazón de la Ciencias Informáticas y son fundamentales para la innovación tecnológica

1.8 Paradigmas de la Programación

Un paradigma es un término que hace referencia a una “**teoría**”, que se toma como “**modelo o ejemplo**” para resolver “**algo**” y que tiene consenso y aceptación en el seno de una comunidad.

En el mundo de la informática, los paradigmas de la programación representan un “modelo” o “estilo” de programación para la construcción de un programa informático, determinando así su estructura. La figura 4, muestra los principales paradigmas de la programación:

Figura 4

Paradigmas de la Programación



Nota. Fuente: Elaboración propia basada en diversas fuentes de la web.

- **Paradigma Imperativo:** Este paradigma es de los más conocidos en el mundo de la informática y es el paradigma en el se va a enfocar este libro. Su importancia radica, porque permite dar instrucciones claras y ordenadas lógicamente a la computadora, indicando “**paso a paso qué debe hacer**” para resolver un problema. Este paradigma, es la base de muchos lenguajes populares como C, C++, Java y Python, lo que lo convierte en una excelente

forma de iniciar el aprendizaje de la programación. Responde a la pregunta: ***“¿Cómo se hace?”***.

- **Paradigma Funcional:** Este paradigma basa su trabajo a partir de “funciones matemáticas”. Evita modificar datos directamente y busca programas más claros, seguros y fáciles de mantener. Responde a la pregunta: ***¿Qué se debe calcular?***
- **Paradigma Lógico:** Se fundamenta en “reglas y hechos lógicos”. El desarrollador define condiciones y la computadora deduce las respuestas automáticamente. Responde a la pregunta: ***¿Qué se quiere resolver?***
- **Paradigma Orientado a Objetos:** Este paradigma se basa en objetos que representan elementos del mundo real. Los programas se centran en los objetos, sus características y sus acciones. Facilita la organización, reutilización y mantenimiento del código. Responde a la pregunta: ***¿Quién lo hace y con qué?***



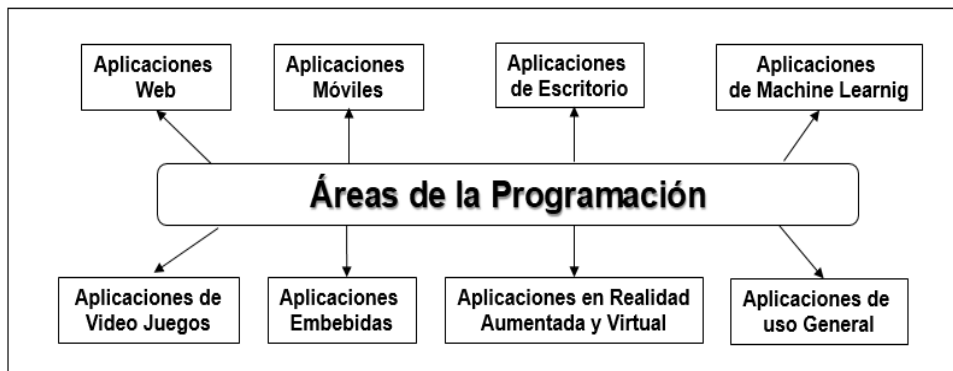
- Diferentes Paradigmas **resultan** en diferentes estilos de programación y en diferentes formas de pensar la solución de problemas.
- Un lenguaje de programación **puede soportar** diferentes paradigmas

1.9 Áreas de la Programación

En el contexto actual, la programación no solo se concibe como una habilidad técnica, sino como una forma de pensamiento estructurado que permite resolver problemas, automatizar tareas y construir soluciones digitales. Como cualquier carrera profesional, el desarrollo de software tiene una amplia gama de áreas que constantemente evolucionan y requieren de nuevas necesidades, por lo que la demanda de desarrolladores se relaciona con la transformación digital de empresas e instituciones, que requieren soluciones a medida para automatizar sus procesos. En este sentido, la programación se orienta a las siguientes áreas (figura 5) en el desarrollo de software.

Figura 5

Áreas de la Programación



Nota. Fuente: Elaboración propia

- **Desarrollo Web:** Esta área está basada en todo aquello a lo que el usuario accede a través del Internet. Desde páginas web hasta navegadores.
- **Desarrollo móvil:** Tiene su enfoque en el desarrollo de aplicaciones o herramientas para dispositivos electrónicos móviles como smartphones, relojes inteligentes o tablets.
- **Machine Learning:** El machine learning es el área de la programación donde, a partir de herramientas automatizadas y grandes suministros de volúmenes de información, se desarrollan programas informáticos complejos que aprenden a partir de datos y mejoran su rendimiento sin ser programados explícitamente.
- **Programación Embebida:** Se basa en programas sencillos que se incorporan a una placa electrónica y cumplen funciones básicas. Desde electrodomésticos hasta máquinas de ejercicio.
- **Realidad Aumentada:** Esta área está enfocada en la programación que conecta entornos virtuales a la realidad física, por medios físicos como visores o componentes conectados al cerebro de una persona (Neuralink).
- **Videojuegos:** Esta área se enfoca en el desarrollo de aplicaciones virtuales con fines de entretenimiento.
- **Desarrollo de Software de Escritorio:** Se enfoca en el desarrollo de herramientas o aplicaciones para computadoras ejecutables sin internet. Esta área está desapareciendo en la actualidad, ya que la industria se está desplazando al sector web y móvil.

- **Programación de Sistemas:** Esta rama se dedica a la construcción de lenguajes de programación y utilidades para sistemas operativos.



Todas las áreas de programación tienen la misma raíz en común...
El Pensamiento Lógico y Algorítmico

1.10 Dato e Información

1.10.1 El Dato

(Del Latín “datum” que significa “dado”, “proporcionado”, “hecho”)

Representa “algo” que puede ser una “cantidad, hecho o acontecimiento sin analizar”, que por sí solo tiene poco significado, pero cuando se “recopilan, organizan y se procesan con otros datos en un contexto”, se convierten en información relevante dentro de ese contexto. Por ejemplo, analicemos la siguiente lista:

153 - 170 - 168 - 146 - 174

Esta lista contiene números que pueden significar cualquier cosa o también no significar absolutamente nada. Más adelante transformaremos estos datos en información.

Las principales características de los datos son:

- Un dato es considerado la “Unidad Mínima de Información”.
- Cuando los datos se agrupan y se procesan, ofrecen un significado que se convierte en información.
- Muchas veces los datos generan incertidumbre.
- Pueden ser expresados y registrados de cualquier forma: Letras, números, sonidos, dibujos, señales, símbolos con sentido, etc.
- Los datos pueden ser cualitativos (descriptivos: Ej. “Azul”) o cuantitativos (numéricos: Ej. 15)

1.10.2 La Información

(Del Latín “in-formare” ...**in** = al interior... **formare** = dar forma o estructurar)

La información constituye el conjunto de datos que, ordenados y procesados bajo una interpretación o contexto, resultan útiles a un receptor para llegar al

conocimiento de un tema específico. Ejemplo: En la definición de **dato**, se mostró una lista de números: ahora, observe la tabla 3 con los mismos números: En esta tabla, se agregan nuevos datos y se acaba de dar un contexto y orden a números que, hasta hace un momento, no tenían sentido concreto: ahora estos datos representan la información de la altura en centímetros de una familia.

Tabla 3

Ejemplo de datos e información

Altura de miembros de una familia en orden descendente (cm)	
Pedro	174
María	170
Juan	168
José	153
Karen	146

Nota. Fuente: Elaboración propia

Entre las principales características de la Información se tienen:

- La información debe ser “obtenida” de los datos.
- Tiene carácter subjetivo, porque depende del significado del observador: (Depende del significado que se dé a los datos y ... De la información previa que se tiene).
- Se puede presentar y almacenar en diferentes formatos: Texto, imagen, sonido o video.
- Puede ser transmitida por diferentes medios: Oral, escrita, visual, digital, señales y símbolos, por radio (por ondas), mediante la luz (comunicación óptica).
- La información debe tener: relevancia, exactitud, oportuna, completa, accesible, confidencial, etc.

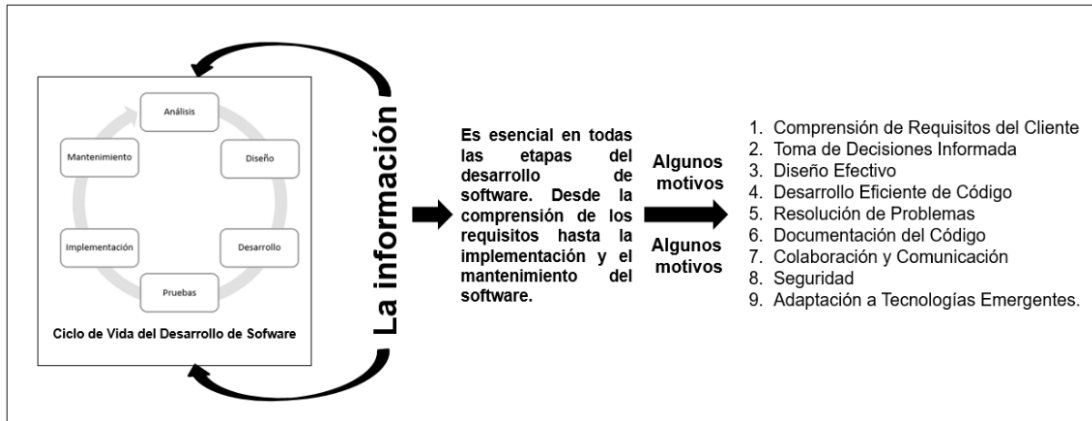
1.10.3 ¿Por qué es importante la Información?

La información es fundamental para un desarrollador / programador, ya que impacta en su capacidad de crear software de alta calidad y cumplir con los objetivos

de su proyecto. La figura 6, sintetiza esta afirmación, considerando el ciclo de vida del software.

Figura 6

La información en el Ciclo de Vida del Software



Nota. Fuente: Elaboración propia



Después del capital humano, la información es a menudo considerada el siguiente activo más importante de una empresa.

1.11 La Lógica

El propósito de este texto no es tratar y definir la lógica como una disciplina filosófica, sino como un elemento central del pensamiento algorítmico y básico para la introducción a la programación. Por lo tanto, en un sentido “habitual y ordinario”, se puede definir a la lógica como el: Razonamiento, en el que las ideas o la sucesión de los hechos se desarrollan de forma coherente. Es decir, el razonamiento correcto, que pone orden nuestros pensamientos, las palabras que los expresan y los hechos que realizamos.

Por lo expuesto, la “lógica” se refiere a lo que es congruente, ordenado, bien estructurado. Lo “ilógico” es lo mismo que incongruente, desordenado, incoherente. Esto se aplica tanto a las personas como a las situaciones y a los pensamientos.

Tres ejemplos de **LÓGICA** en la vida cotidiana:

- a) Siempre es preferible tomar el camino más corto para ir a un lugar, porque se tardará menos tiempo en llegar.

- b) Siempre es conveniente consultar con un médico antes de tomar un medicamento; de lo contrario, un paciente puede empeorar su estado de salud.
- c) Si está nublado, entonces puede llover, por lo tanto, lo sensato es llevar un paraguas.

Ejemplos **ILÓGICOS** en la vida cotidiana:

- a) Matricularse en la Universidad en una carrera y no asistir a clase.
- b) Ir a la Panadería y preguntar si hacen Panes.



Las **personas a diario** usamos la **lógica** a pesar de no pensar que es algo lógico.

Actividades sugeridas

- a) Plantear dos (2) ejemplos de **su vida diaria**, acerca de situaciones LÓGICAS.
- b) Plantear dos (2) ejemplos de **su vida diaria**, acerca de situaciones ILÓGICAS.

1.12 El Razonamiento

El razonamiento es la facultad humana que permite procesar información, establecer relaciones entre ideas, y establecer conclusiones a partir de premisas, observaciones, experiencias o conocimientos previos. Es decir, es la capacidad del ser humano para enfrentar mentalmente un problema o situación aplicando la lógica y la experiencia a su resolución y/o entendimiento. Por lo tanto, es un proceso cognitivo que pasa de los “conocimientos existentes” a “nuevas ideas y conclusiones”. El razonamiento está formado por:

- a) **Premisas:** Son los datos que se tienen previamente y que constituyen el punto de partida del razonamiento.
- b) **Deducción / Inducción:** Ambos permiten analizar los datos, establecer relaciones lógicas y llegar a conclusiones válidas, aunque lo hacen de manera diferente.

b.1 Deducción: Se parte de las premisas., por lo tanto, “La veracidad de las premisas garantiza la veracidad de la conclusión”. Va de “lo general a lo particular”.

b.2 Inducción: Se parte de las premisas, por lo tanto, “La veracidad de las premisas no garantiza la veracidad de la conclusión, solo establece que es “probable”. Va de “lo particular a lo general”.

a) **Conclusión:** Es el resultado y la finalidad del razonamiento.

Ejemplos de Razonamiento Deductivo:

- a) Todos los seres humanos respiran (premisa 1), María es humano (premisa 2); por lo tanto, María respira (conclusión).
- b) En los días soleados la ropa se seca más rápido (premisa 1), hoy está soleado (premisa 2); por lo tanto, la ropa se secará más rápido (conclusión).
- c) Las estrellas están constantemente en combustión (premisa 1), el Sol es una estrella (premisa 2); por lo tanto, el Sol está constantemente en combustión (conclusión).
- d) Todos los números mayores que 10 son de dos cifras o más (premisa 1), 18 es mayor que 10 (premisa 2), por lo tanto 18 tiene dos cifras (conclusión).

Ejemplos de Razonamiento Inductivo:

- a) En la asignatura de Algoritmos, Pedro estudió y aprobó (premisa 1), María estudió y aprobó (premisa 2), Karen estudió y aprobó (premisa 3); por lo tanto, al estudiar se aumenta la probabilidad de aprobar (conclusión).
- b) En las playas de Ecuador la marea baja cada 12 horas (premisa 1), en las playas de México la marea baja cada 12 horas (premisa 2), en las playas de Tailandia la marea baja cada 12 horas (premisa 3); por lo tanto, en todas las playas la marea baja cada 12 horas (conclusión).
- c) Mi casa tiene pisos de mármol y siempre está fresca (premisa 1), la casa de mi vecino tiene pisos de mármol y siempre está fresca (premisa 2); por lo tanto, las casas que tienen pisos de mármol siempre están frescas (conclusión).
- d) $2 + 2 = 4$ $4 + 2 = 6$ $6 + 2 = 8$

Premisas: Al sumar 2 a un número par se obtiene otro número par.

Conclusión: Sumar 2 a cualquier número par da como resultado otro número par.



RAZONAR: Es **deducir / inducir** una o varias **conclusiones** a partir de ciertos datos (**premisas**) que ya se poseían previamente.

Actividades sugeridas

- a) Explica con tus propias palabras la diferencia entre razonamiento inductivo y deductivo.
- b) Plantear dos ejercicios de lógica de la **vida diaria** con razonamiento **DEDUCTIVO**.
- c) Plantear dos ejercicios de lógica de la **vida diaria** con razonamiento **INDUCTIVO**.
- d) Plantear dos ejercicios por cada razonamiento **DEDUCTIVO** e **INDUCTIVO**, usando valores numéricos.

1.12.1 El Razonamiento Lógico-Matemático

El razonamiento lógico-matemático es la capacidad de analizar, relacionar y organizar información numérica mediante reglas lógicas, con el fin de identificar patrones, establecer relaciones y resolver problemas de manera estructurada.

En este apartado, el propósito de utilizar el razonamiento lógico-matemático, se basa principalmente en la resolución de ejercicios con series, progresiones numéricas y matrices, es para preparar al estudiante a que aprenda con estos ejercicios a observar, comparar, identificar patrones, formular reglas y verificar resultados, procesos fundamentales del razonamiento lógico.

Actividades con ejercicio propuestos

Práctica: Ejercicios con series numéricas

1. Completa la secuencia: 2, 4, 6, 8, __
2. Completa la secuencia: 9, 12, 15, 18, __
3. Completa la secuencia: 10, 7, 4, 1, __
4. Completa la secuencia: 1, 3, 5, 7, __, __
5. Completa la secuencia: 100, 90, 80, 70, __, __
6. Completa la secuencia: 5, 10, 15, 20, __, __
7. Completa la secuencia: 20, 17, 14, 11, __
8. Completa la secuencia: 3, 6, 9, 15, 24, __
9. Completa la secuencia: 14, 9, 4, -1, __
10. Completa la secuencia: 7, 14, 28, 56, __
11. Completa la secuencia: 37, 38, 40, 41, 43, 44, __

12. Completa la secuencia: 18, 19, 15, 16, 12, 13, __
13. Completa la secuencia: 10, 20, 40, 70, __, __
14. Completa la secuencia: 3, 6, 9, 15, 24, __
15. Completa la secuencia: 48, 8, 35, 7, 24, 6, 21, __
16. Completa la secuencia: 3, 6, 12, 24, __, __
17. Completa la secuencia: 1, 4, 9, 16, __, __
18. Completa la secuencia: 2, 6, 12, 20, __, __
19. Completa la secuencia: 1, 1, 2, 3, 5, 8, __, __
20. Completa la secuencia: 81, 27, 9, 3, __, __
21. Completa la secuencia: 2, 6, 12, 20, 30, __, __
22. Completa la secuencia: 1, 2, 6, 24, __, __
23. Completa la secuencia: 3, 9, 27, 81, __, __
24. Completa la secuencia: 5, 10, 20, 40, 80, __, __
25. Completa la secuencia: 1, 4, 10, 20, 35, __, __

Actividades con ejercicios propuestos

Práctica: Ejercicios con Matrices numéricas, gráficas y otros

Ejercicio 1:

Los números de la columna central están relacionados con los números de la izquierda y la derecha.

¿Por qué?.....

3	42	8
5	51	3
8	61	2
7	53	5

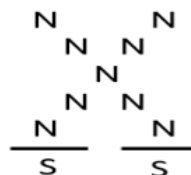
Ejercicio 2:

En la siguiente matriz identifique la relación en los números de cada fila y de acuerdo a ello, sustituya la interrogante por el número que debería ir en su lugar:

12	10	4
87	16	22
169	81	?

Ejercicio 3:

Ubicar los números del 1 al 9 (ambos incluidos) en forma de una "X", de tal manera que sumen los mismo.



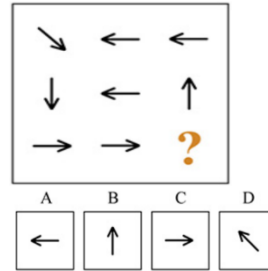
Ejercicio 4:

En la siguiente matriz identifique el número que no tiene relación con los demás.

3	33	15	36
27	12	34	6
21	9	18	30

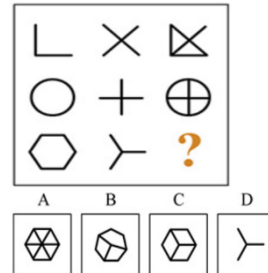
Ejercicio 5:

En la siguiente matriz, sustituya el interrogante por la flecha que debería ir en su lugar:



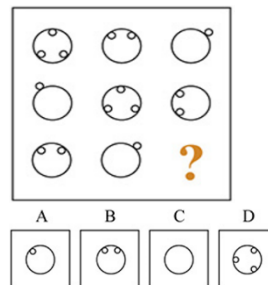
Ejercicio 6:

En la siguiente matriz, sustituya el interrogante por la figura que debería ir en su lugar:



Ejercicio 7:

En la siguiente matriz, sustituya el interrogante por la figura que debería ir en su lugar:



Ejercicio 8.

En una reunión familiar se encuentran las siguientes relaciones de parentescos entre los presentes: Padre, Madre, hijo, hija, tío, tía, hermano, hermana, primo, prima, sobrino y sobrina. Sin embargo, sólo había 4 personas. ¿Cómo puede ser posible?



Ejercicio 9.

Estoy buscando un número de dos cifras que al multiplicarlas me dé 6, siendo el número que busco menor que 20 y mayor que 10.

¿Cuál es el número?

—

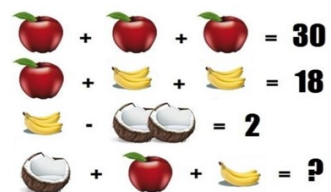
Ejercicio 10.

Darío, que es mayor que Daniel en 1 año menos un día, nació el 1 de enero de 2010. ¿Qué día nació Daniel?

- a) 1 de diciembre de 2011
- b) 31 de diciembre de 2008
- c) 31 de diciembre de 2009
- d) de enero de 2011
- e) 31 de diciembre de 2010
- f) enero de 2009

Ejercicio 11.

Observe detenidamente las figuras y reemplace el signo “?” por el valor correspondiente.



Ejercicio 12.

Observe detenidamente las figuras y reemplace el signo “?” por el valor correspondiente.

$$\begin{array}{rclcl} \text{🍏} & + & \text{🍏} & + & \text{🍏} & = & 30 \\ \text{🍏} & + & \text{🍌} & + & \text{🍌} & = & 18 \\ \text{🍌} & + & \text{🥥} & + & \text{🥥} & = & 6 \\ \text{🍌} & - & \text{🥥} & = & ? \end{array}$$

1.12.2 El Razonamiento en el ámbito de la programación

Programar no consiste únicamente en escribir instrucciones para crear un programa, sino en pensar de forma lógica y ordenada, analizando los problemas, estructurar soluciones lógicas y construir algoritmos eficientes que luego mediante un lenguaje de programación son transformados en instrucciones que ejecuta una computadora (figura 7). En este contexto, el razonamiento adquiere una importancia central en el aprendizaje de los algoritmos y la programación, al permitir comprender la estructura de los problemas en forma lógica y construir acciones coherentes mediante procedimientos claros, que luego se traducen en algoritmos y programas funcionales. Por lo tanto, el razonamiento es la base del pensamiento computacional y algorítmico en donde el desarrollador debe:

- Analizar y organizar la información del enunciado del problema que se plantea de una forma lógica.
- Razonar sobre el análisis, diseño y la eficiencia de una solución algorítmica de un problema.
- Automatizar soluciones haciendo uso de algoritmos.

Figura 7

El razonamiento en el ámbito informático



Nota. Fuente: Elaboración propia



La Lógica Computacional Orientada a la Programación, hace referencia al **razonamiento enfocado en comprender, analizar y generar posibles soluciones** a problemas mediante programas computacionales.

1.13 Metodologías Cognitivas para la resolución de problemas

Los estudiantes pueden encontrar diversas maneras de abordar problemas y plantear soluciones, al mismo tiempo que desarrollan habilidades. En este contexto, existen varias estrategias para solucionar problemas que son estrategias mentales que permiten analizar, comprender y solucionar situaciones de manera lógica, creativa y eficiente. Estas metodologías fortalecen el razonamiento y preparan al estudiante para el pensamiento algorítmico y la programación.

1.13.1 Razonamiento Analógico

Se apoya en el establecimiento de una analogía entre una situación que resulte familiar y la situación problema. Consiste en buscar semejanzas con otros problemas (aprendizaje significativo). Es decir, resolver un problema nuevo usando la estructura de uno ya conocido.

Este método es fundamental en los algoritmos y la programación, porque:

- Reutilizamos patrones
- Reutilizamos soluciones previas
- Usamos esquemas ya probados

Ejemplo 1: Completa la secuencia

2, 4, 6, 8, __

Análisis: Observamos que hay un patrón... a cada número + 2

Respuesta: **10**

Entonces.... Si planteamos el siguiente ejemplo:

Ejemplo 2: Completa la secuencia

3, 6, 9, 12, __

Análisis: Observamos que al igual que el ejercicio anterior, hay un patrón de resolución que se repite, cambiando únicamente los valores, en este caso + 3:

Respuesta: **15**



Este método es base de la **abstracción y reutilización algorítmica**.

1.13.2 Por Descubrimiento

Se basa en la exploración, el análisis y la experimentación, mediante la cual el estudiante construye activamente el conocimiento. Por lo tanto, el nuevo conocimiento se adquiere a través de los propios esfuerzos del alumno al resolver un problema, con la exploración, prueba, error y ajuste progresivo.

Este método es fundamental en los algoritmos y la programación, porque:

- Permite la corrección de los algoritmos y la depuración de programas
- Permite realizar pruebas a las soluciones planteadas
- Permite la eficiencia y optimización de los algoritmos

Ejemplo 1: Complete la secuencia

1, 4, 9, 16, 25, __

- a) Observación: Cada número parece ser el resultado de elevar al cuadrado los números consecutivos:

$1^2, 2^2, 3^2, 4^2, 5^2$

- b) Descubrimiento:

$6^2 = 36$

Solución: **36**

Ejemplo 2: Complete

1, 211, 3111, 41111, 511111, _____

- a) Observación: No es una secuencia aritmética
- b) Descubrimiento: Cada término describe cuantas veces se repite el número 1

Solución: **6111111**



Este método es base para la **experimentación algorítmica**.

1.13.3 Pensamiento Lateral

Requiere que se exploren y consideren la mayor cantidad posible de alternativas para solucionar un problema. Permite que el estudiante explore (escuche y acepte puntos de vista diferentes); libere (escapar a ideas preestablecidas); y contrarreste la rigidez (ver las cosas desde diferentes ángulos). Por lo tanto, se estimula la

innovación, la flexibilidad mental y la capacidad de encontrar múltiples alternativas para un mismo problema.

Este método es fundamental en los algoritmos y la programación, porque:

- Fomenta la creatividad y flexibilidad mental
- Permite la eficiencia y optimización de los algoritmos
- Desarrolla un pensamiento computacional y algorítmico más avanzado

Ejemplo 1: Coloca tres números 3 para obtener el resultado 30.

Solución: $28 + 2 = 30$

Pero.....Alguien rompe la idea de usar solo sumas, y lo resuelve así..

Solución: $33 - 3 = 30$

Ejemplo 2: Calcular el cubo del número 5.

Solución: $5 * 5 * 5 = 125$ (multiplicación sucesiva)

Pero.....Alguien rompe la idea de usar multiplicaciones, y lo resuelve así..

Solución: $5^3 = 125$ (utiliza potencia)



Este método es base para el diseño creativo de algoritmos, encontrando soluciones **ingeniosas e inesperadas**.

1.14 El problema.- Definición

En el ámbito de los algoritmos y la programación, el término *problema* ocupa un lugar central. Un problema en términos generales puede entenderse como “una cuestión, situación o circunstancia que plantea una dificultad o desafío y que requiere de una solución”. Para hallar esta solución, se requiere analizar la situación, identificar sus elementos, establecer relaciones lógicas entre ellos y diseñar una estrategia sistemática que conduzca al resultado deseado.

Desde una perspectiva de aprendizaje de la programación, los problemas constituyen herramientas fundamentales para el desarrollo del pensamiento lógico, crítico y creativo del estudiante. Aprender a reconocer, interpretar y plantear adecuadamente los problemas representa el punto de partida para permitir su tratamiento algorítmico. En este contexto, un problema se caracteriza por tres

componentes fundamentales: entrada (datos), proceso (algoritmo) y salida (resultado).



Los problemas son parte de la vida de todos. Nos guste o no, tenemos que tomar decisiones y las que tomemos deben resolver los problemas que enfrentamos.

1.14.1 Generalidades de un problema

Un problema tiene aspectos generales que permiten comprender su naturaleza, su contexto y su propósito, facilitando su análisis y la formulación de estrategias adecuadas para su resolución.

- Todo problema surge a partir de una necesidad, dificultad o situación que requiere solución inmediata.
- Los problemas tienen diferentes magnitudes y complejidades. Desde sencillos a complicados en su solución.
- Puede presentarse en diversos contextos, de ahí que existen variados tipos que son específicos de diversas áreas donde se desarrollan: Problemas matemáticos, físicos, químicos, económicos, sociales, educativos, personales, etc.
- Permite la transferencia de conocimientos entre distintas áreas del saber.
- Favorece el aprendizaje activo, al promover la participación directa del estudiante.
- Constituye la base del aprendizaje basado en problemas, especialmente en la programación.

1.14.2 Características de un Problema

Un problema tiene un conjunto de elementos esenciales que definen su estructura y orientan el proceso de resolución:

- **Meta:** Existe una situación que se quiere resolver.
- **Enunciado:** Descripción clara para poder ser comprendido y resuelto.
- **Estado inicial:** Es el punto de partida dentro del enunciado, con lo cual se inicia el proceso de resolución.
- **Datos:** Tiene información relevante que se utiliza para encontrar la solución.

- **Restricciones:** Puede incluir límites o condiciones que deben respetarse durante el proceso de resolución.
- **Necesidad de razonamiento:** Requiere la aplicación del razonamiento lógico para comprenderlo, analizarlo y resolverlo.
- **Múltiples soluciones:** Un mismo problema puede tener varias soluciones.
- **Verificabilidad:** La solución obtenida puede comprobarse y validarse.
- **Modelabilidad:** Puede representarse mediante diferentes esquemas.



Los problemas exigen conocimientos, habilidades y actitudes, pero, sobre todo capacidad creativa e inventiva de los estudiantes.

1.14.3 Elementos Comunes de un Problema

Independientemente de la naturaleza, tamaño o grado de complejidad, todo problema está compuesto por tres elementos fundamentales: el enunciado, el procedimiento y el resultado. Estos elementos constituyen la estructura básica para la comprensión, el análisis lógico y la formulación de soluciones sistemáticas.

1.14.3.1 El Enunciado

El enunciado consiste en la descripción clara, precisa y sin ambigüedades, de la tarea o situación que requiere de una solución. Proporciona los elementos necesarios para entender el problema, estableciendo los datos disponibles (la entrada de datos/información), las restricciones, así como la(s) salida(s) o resultado(s) que se esperan. Por lo tanto, su función principal es explicar el contexto del problema, identificando correctamente qué se pide, qué información se tiene y qué restricciones existen. El enunciado de se puede plantear con variedad de formas: literal, gráfica, uso de fórmulas, entre otros.

Para comprender un problema a partir del enunciado se recomienda:

- Leer el enunciado del problema varias veces.
- Aclarar lo que se va a resolver.
- Precisar el resultado que se desea lograr.
- Identificar los datos como insumos necesarios para resolver el problema
- Organizar los datos, eliminando información innecesaria o irrelevante.



No se puede resolver un problema si no se comprende el enunciado.

1.14.3.2 El Procedimiento

El procedimiento consiste en analizar el problema, para poder crear una estrategia de solución mediante un conjunto de pasos lógicamente estructurados, partiendo de los datos proporcionados en el enunciado hasta llegar a los resultados. En el contexto de la programación y la resolución de problemas informáticos, los procedimientos se pueden traducir en lo que conocerá como **Algoritmo**.

Un eficiente procedimiento de solución debe proporcionar una respuesta correcta al problema, debiendo ser claro y fácil de entender. Este enfoque sistemático es fundamental en la resolución de los problemas en la programación y en otras áreas del saber. Para trazar o ejecutar un plan dentro del procedimiento:

- Pregúntese: ¿He resuelto un problema similar?
- Pregúntese: ¿Qué relación tienen los datos identificados del enunciado entre sí?
- Busque una solución basada en sus **CONOCIMIENTOS** y pregúntese: ¿Hay alguna fórmula para llegar a la solución?
- Busque una solución basada en la **LÓGICA**, aplique su imaginación y sea creativo.
- Defina y ejecute las operaciones y decisiones necesarias a efectuar según la solución.



Es posible que después de este análisis, tenga más de un procedimiento de solución.

1.14.3.3 El Resultado

En el contexto de la resolución de problemas, el resultado es la solución o respuesta obtenida después de aplicar un procedimiento o estrategia de solución de un problema. Es decir, que los resultados representan la solución final obtenida al aplicar el procedimiento sobre los datos del problema. Dado que el resultado es la

manifestación observable de la ejecución de un procedimiento, se debe revisar y/o evaluar la solución y asegurarse que cumple con lo solicitado en el enunciado. Un resultado adecuado debe cumplir con tres condiciones fundamentales:

- **Exactitud:** Debe ser correcto desde el punto de vista lógico y matemático.
- **Claridad:** Debe presentarse de forma comprensible.
- **Validación:** Debe poder comprobarse mediante pruebas o verificación.

En el contexto de la programación, el resultado puede ser la salida de un programa después de procesar ciertos datos de entrada. De esta manera, el resultado ayuda a evaluar la efectividad de las acciones realizadas (el algoritmo). Al obtener un resultado se recomienda preguntarse:

- ¿Es la solución correcta del problema?
- ¿La respuesta satisface lo solicitado en el problema?
- ¿Existe un procedimiento más sencillo para llegar a este resultado?



En programación, los resultados corresponden a la salida del algoritmo, es decir, a la información producida por el programa una vez ejecutadas todas las instrucciones.

Actividades con ejercicios resueltos

Ejemplo 1:

Enunciado: Convierta 50 kilogramos a libras.

Comprender que se busca convertir una cantidad de kilogramos a libras.

Procedimiento de solución (Trazar y Ejecutar un plan):

- a) Trazar el Plan (Planear): Buscar la solución basada en el **conocimiento**
 - (1 kg = 2.20462 lb)
- b) Ejecutar el Plan (Resolver): Aplicar la relación de conversión para convertir kilogramos a libras.
 - Multiplicar 50 kg por la tasa de conversión: $50 \text{ kg} * 2.20462 \text{ lb}$.

El resultado: Hay que confirmar que la respuesta obtenida mediante la conversión (110.231 libras) esté realizada correctamente.

Ejemplo 2:

Enunciado: Hallar el área de un círculo con radio 5 cm.

Comprender el problema: Entender que se busca calcular el área de un círculo dado su radio.

Procedimiento de solución (Trazar y Ejecutar un plan):

- a) Trazar el Plan (Planear): Buscar la solución basada en el conocimiento
 - $(A = \pi * r^2, \text{ donde } r \text{ es el radio}).$
- b) Ejecutar el Plan (Resolver): Sustituir $r = 5 \text{ cm}$ y $\pi = 3.1416$. Por lo tanto en la fórmula: $A = 3.1416 * 5 \text{ cm}^2$.

El resultado: Verificar que la respuesta (78.54) tenga sentido en términos de unidades y valor numérico.

Ejemplo 3:

Enunciado: Encuentre la suma de los primeros 10 números enteros.

Comprender que se busca la suma de los primeros números enteros (1 al 10) en una secuencia.

Procedimiento de solución (Trazar y Ejecutar un plan):

- a) Trazar el Plan (Planear): Buscar la solución basada en el **conocimiento**.
 - $\text{Suma} = (n/2) * (\text{primer término} + \text{último término}).$
- b) Ejecutar el Plan (Resolver): Sustituir $n = 10$, primer término = 1, último término = 10 en la fórmula.
 - $10 / 2 * (1 + 10) = 5 * 11 = 55$

El resultado:

Confirmar que la respuesta obtenida mediante la conversión (55) esté realizada correctamente.

Ejemplo 4:

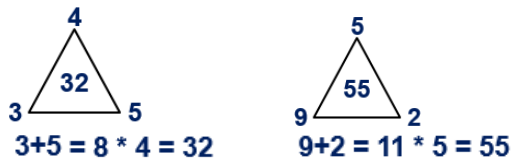
Enunciado: Encuentre la relación que existe entre los números en cada triángulo y complete.



Procedimiento de solución (Trazar y Ejecutar un plan):

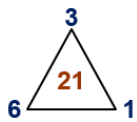
- a) Trazar el Plan (Planear): Buscar la solución **basada en la IMAGINACIÓN y CREATIVIDAD**

¿Cuál es la relación?



- b) Ejecutar el Plan (Resolver): $6+1 = 7 * 3 = 21$

El resultado:



Ejercicios propuestos

Para cada uno de los siguientes ejercicios identifique: El enunciado, diseñe el procedimiento lógico o matemático y obtenga el resultado. Use la siguiente forma por cada problema:

Enunciado : _____

Procedimiento: _____

Resultado : _____

Ejercicio 1: Una tienda ofrece un descuento del 20 % sobre un producto que cuesta \$50. ¿Cuál es el precio final?

Ejercicio 2: Un depósito contiene 120 litros de agua. Se extraen 15 litros cada día. ¿Cuántos litros quedarán después de 4 días?

Ejercicio 3: Un número aumentado en su mitad es igual a 30. ¿Cuál es ese número?

Ejercicio 4: La suma de tres números consecutivos es 72. ¿Cuáles son esos números?

Ejercicio 5: Un trabajador gana \$480 en 5 días. ¿Cuánto ganará en 15 días si mantiene el mismo ritmo de trabajo?

Problema 6: En la empresa INEPACA una máquina produce 120 latas de atún en 4 horas, ¿cuántas latas producirá en 1 día?

Ejercicio 7: En un curso de Algoritmos hay 36 estudiantes. Si el 25 % son mujeres, ¿cuántos hombres hay?

Ejercicio 8: Un número multiplicado por 4, menos 7, da como resultado 45. ¿Cuál es ese número?

Ejercicio 9: Un estudiante de la ULEAM gasta \$25 (movilización y alimentación) en 1 semana. ¿Cuánto gastará en un semestre con el mismo ritmo de gasto?

Ejercicio 10: El automóvil de su casa consume 1 galón de combustible por cada 40 km. ¿Cuántos galones consumirá en un viaje de Manta a Guayaquil, cuya distancia promedio es de 200 km?

Ejercicio 11: Un estudiante de la asignatura de Algoritmos, resuelve 5 ejercicios en 30 minutos. ¿Cuántos ejercicios podrá resolver en 3 clases a la semana, considerando que cada clase es de 2 horas?

Ejercicio 12: Un computadora descarga 600 MB en 3 minutos. ¿Cuántos MB descargará en 1 hora?

Ejercicio 13: Una impresora imprime 480 páginas en 8 minutos. ¿Cuánto minutos tardará en imprimir un libro digital de 600 páginas?

Ejercicio 14: Pedro colecciona números naturales, cuyo dígito de las unidades es la suma de los otros números. Por ejemplo, él coleccionó 42107, pues: $4 + 2 + 1 + 0 = 7$. En la colección de Pedro, hay números de 5 dígitos y cuyo dígito de las unidades es 8. Por lo expuesto, se solicita determinar cuáles son esos números.

Ejercicio 15: Cuadro Mágico Reversible

Rellena todos los cuadros de la siguiente matriz con las cifras 1 al 16 de forma secuencial:

	1	2	3	4
→				
→				
→				

Siga las siguientes instrucciones:

- Selecciona un número cualquiera de la matriz y tacha los demás números que estén en la misma fila y columna que el número señalado. Un número que ha sido ya tachado no puede ya ser seleccionado.
- Repite esta operación cuatro veces (a la cuarta ocasión sólo puedes elegir un único número, pues todos los demás ya han sido elegidos o tachados).

¿Qué valor da la suma de los números seleccionados?

a) 24 b) 15 c) 34 d) 50

1.15 Los Algoritmos

1.15.1 Concepto de Algoritmo y su evolución

El término Algoritmo deriva de la palabra árabe “*Al Kwarizmi*”, nombre del padre de la algoritmia: *Abu Jafar Muhammed ibn Musä Al-Kwarizmi* (780 DC), quien fue un matemático y astrónomo persa (figura 8) y representa el inicio histórico del pensamiento algorítmico. Su apellido fue traducido al latín como *Algorismus* y posteriormente pasó al español como *Algoritmo*.

En términos generales, un algoritmo “**es un conjunto finito de pasos, acciones u instrucciones lógicas, que siguiendo un orden específico, conducen a la**

solución de un problema determinado o a ejecutar una tarea”. Esta definición ha evolucionado desde su aplicación matemática hacia un uso transversal en la informática, la educación y otras ciencias aplicadas.

Figura 8

Ilustración artística de Al-Juarismi (Abu Jafar Muhammad ibn Musa al-Khwarizmi)



Nota. Fuente: Wikimedia Commons

1.15.2 Importancia del Algoritmo en la Informática

Los algoritmos son fundamentales para el desarrollo tecnológico informático y se los considera la **columna vertebral** de esta disciplina, pues, permiten el desarrollo y funcionamiento de cualquier sistema computacional por lo que están presente en múltiples aspectos de la vida cotidiana. Su importancia radica en que toda aplicación informática está basada en uno o varios algoritmos que definen su funcionalidad, calidad de este software, su eficiencia y su sostenibilidad. Actualmente su trascendencia se ha incrementado con la llegada del Big Data y la Inteligencia Artificial.

A continuación, se describe en forma general los aspectos clave que resaltan su relevancia:

- Los algoritmos son el fundamento o base de la programación.
- Los algoritmos son herramientas sistemáticas para resolver problemas.
- Los algoritmos son la base de la automatización.

- Los algoritmos son la base de la Inteligencia Artificial (IA) y el Aprendizaje Automático (ML).
- Los algoritmos son elementales en áreas como: buscadores web, sistemas operativos, bases de datos, entre otras.
- Los algoritmos determinan el desarrollo y la eficiencia computacional.
- Los algoritmos impulsan la innovación tecnológica.

1.15.3 Características de los Algoritmos

Un algoritmo debe cumplir con las siguientes características:

- **Finitud:** Establece que la secuencia de pasos es finita. Es decir, que esta secuencia lógica tiene inicio y fin. No importa que tan larga sea la lista de pasos, deberá iniciar en un momento y deberá terminar en otro.
- **Claridad:** Sus instrucciones deben ser comprensibles, directas y fáciles de seguir.
- **Precisión:** Cada paso debe estar claramente definido, sin ambigüedades ni interpretaciones múltiples.
- **Orden lógico:** Los pasos o acciones del algoritmo deben estar organizados en un orden lógico de realización, por lo que cada paso depende del anterior.
- **Entrada y Salida definidas:** Todo algoritmo toma alguna forma de entrada con datos iniciales, necesarios para iniciar el proceso y generar una salida (resultados). Estas salidas corresponden a uno o varios resultados que representan la solución al problema. Las entradas y salidas deben estar claramente definidas para que el algoritmo pueda cumplir su función.
- **Efectividad:** Cada operación debe ser realizable de manera concreta y eficiente, por lo que debe resolver un problema en poco tiempo y con los recursos necesarios.
- **Determinístico:** Implica que, si se probó el algoritmo con ciertas entradas, lo lógico sería esperar que la siguiente prueba, con las mismas entradas, generen el mismo resultado. Es decir, ante las mismas entradas, el algoritmo debe producir siempre la misma salida.
- **Óptimos:** Deben cumplir con el objetivo, para el cual fueron desarrollados.

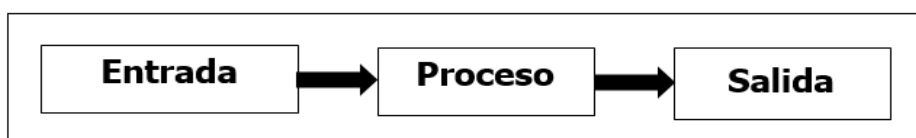
1.15.4 Estructura de un Algoritmo

La estructura de un algoritmo hace referencia a la organización de sus componentes fundamentales, los cuales permiten describir como se disponen lógicamente sus acciones para la resolución de un problema. Todo algoritmo se compone, de forma general, de tres partes básicas (figura 9), independientemente de su tamaño o tipo:

- a) **Entrada (Input):** Son los datos y/o información de partida del algoritmo y que requiere el proceso para ejecutar las operaciones. Tiene las siguientes características:
 - Son indispensables para iniciar el proceso, y deben ser identificados en el enunciado.
 - Pueden ser valores numéricos, textos o de otros tipos según el problema.
- b) **Proceso (Processing):** Corresponde al conjunto de todas las operaciones a realizar, a partir de los datos de entrada para dar solución al problema. Tiene las siguientes características:
 - Representa el eje central del algoritmo.
 - Define las acciones del procedimiento paso a paso en forma lógica.
 - Está formado por operaciones matemáticas, lógicas y de comparaciones.
- c) **Salida (Output):** Son los resultados obtenidos de la ejecución del proceso. Tiene las siguientes características:
 - Representa la solución del problema.
 - Puede ser uno o varios valores, dependiendo del problema
 - Debe ser claro y verificable para determinar si el algoritmo cumple con el objetivo, para el cual fue desarrollado.

Figura 9

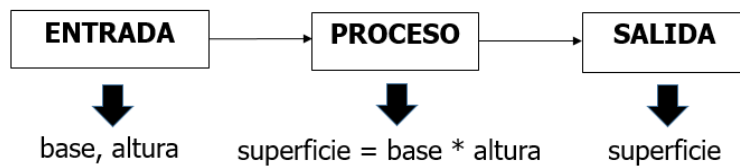
Elementos que conforman la estructura de algoritmo



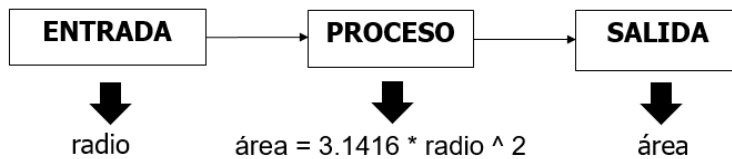
Nota. Fuente: Elaboración propia

Actividades con ejercicios resueltos

Ejemplo 1: Algoritmo que dado la base y altura de un rectángulo calcule la superficie.



Ejemplo 2: Algoritmo que, dado el radio de un círculo, permita calcular su área.



Ejemplo 3: Imagine que desea hacer unas papas fritas para su familia: las papas fritas no nacen solas de la tierra, se necesitan una serie de pasos secuenciales para llegar de la papa cruda hasta el plato final en su mesa. Para hacerlas, necesitamos:

Entrada: Papas, Aceite, Sal y Agua

Proceso (Pasos):

1. Lavar las papas con abundante agua.
2. Pelar las papas.
3. Cortar las papas en tiras uniformes.
4. Colocar aceite en una sartén o freidora.
5. Calentar el aceite hasta alcanzar una temperatura adecuada.
6. Introducir cuidadosamente las papas en el aceite caliente.
7. Freír las papas hasta que estén doradas y crujientes.
8. Retirar las papas del aceite.
9. Agregar sal al gusto.
10. Servir.

Salida: Papas fritas listas para consumir.

1.15.5 Tipos de Algoritmos

Desde un enfoque formativo e introductorio, los algoritmos se pueden clasificar en dos categorías principales en función de la naturaleza de los datos con los que trabajan y el tipo de problemas que resuelven:

- Algoritmos Cualitativos
- Algoritmos Cuantitativos

Esta clasificación permite comprender la diversidad de problemas que pueden resolverse mediante procedimientos lógicos y constituye un primer acercamiento al pensamiento algorítmico.

1.15.5.1 Algoritmos Cualitativos

Se trata de algoritmos que, en su resolución, no intervienen cálculos matemáticos. Requieren de un conjunto finito de pasos y acciones lógicas que se describen por medio de palabras (lenguaje natural) para obtener la solución al problema. Tienen las siguientes características:

- Por lo general modelan las actividades en la vida cotidiana de las personas.
- Difícilmente podrá ser ejecutado por una computadora.
- En sus acciones no existen cálculos matemáticos.
- Emplea un lenguaje natural (palabras) para describir las acciones y cada paso del algoritmo se enumera.
- Se diseñan para ser entendidos y ejecutados por cualquier persona.
- En un proceso de formación, facilitan la comprensión inicial del concepto de algoritmo, fortalece el desarrollo del pensamiento lógico, permiten entender la lógica secuencial y sirven como base para el aprendizaje algorítmico.



Las personas en su vida diaria, **resuelven problemas** sin darse cuenta de que están siguiendo una metodología para su solución.

Actividades con ejercicios resueltos

Ejemplo 1: Redactar un correo electrónico a un profesor desde su computadora

1. Encender la computadora.
2. Iniciar sesión en Outlook.

3. Ingresar la cuenta de correo institucional.
4. Ingresar la clave.
5. Seleccionar Nuevo Correo.
6. Ingresar el correo del profesor.
7. Ingresar el asunto del correo.
8. Redactar el correo que se va a enviar.
9. Enviar el correo.

Ejemplo 2: Cambiar una bombilla fundida

1. Ubicar una escalera debajo de la bombilla fundida
2. Tomar una bombilla nueva
3. Subir por la escalera
4. Girar la bombilla fundida hacia la izquierda hasta soltarla
5. Enroscar la bombilla nueva en el plafón hasta apretarla
6. Bajar de la escalera

Ejemplo 3: Cambiar una llanta pinchada

1. Traer gata, llave en cruz y llanta de repuesto.
2. Aflojar pernos de la llanta pinchada.
3. Ubicar la gata en un lugar seguro y levantar el coche.
4. Sacar los “pernos” de la llanta pinchada.
5. Sacar la llanta pinchada.
6. Poner la llanta de repuesto.
7. Poner los pernos.
8. Bajar la gata.
9. Apretar los pernos.
10. Sacamos la gata.
11. Guardar llanta pinchada y demás herramientas.
12. Proceso finalizado.

Ejercicios propuestos

Bloque 1: (Comprensión de secuencia y orden lógico)

Ejercicio 1: Solicitar una carrera en un taxi mediante la aplicación “in drive”.

Ejercicio 2: Hacer una llamada por WhatsApp en su celular.

Ejercicio 3: En diez (10) pasos, ir de su casa a la universidad.

Ejercicio 4: Consultar el significado de una palabra en el diccionario.

Ejercicio 5: Retirar dinero de un cajero automático, mediante tarjeta.

Ejercicio 6: Justificar una inasistencia a clases con el Director de Carrera.

Ejercicio 7: Ver una película en un canal de televisión.

Ejercicio 8: Entrar a una sesión en ZOOM para una clase virtual en su computadora.

Ejercicio 9: Crear una cuenta de correo electrónico en Gmail.

Ejercicio 10: En seis (6) pasos, hacer una compra de un producto en “Tutti”.

Ejercicio 11: Revisar el perfil de Facebook en su celular.

Ejercicio 12: En diez (10) pasos, ir al Mall a ver una película en el cine

Ejercicio 13: Escuchar música en Spotify en su celular

Ejercicio 14: Lavar ropa en una lavadora (10 pasos)

Ejercicio 15: Encontrar un libro específico en una biblioteca.

Bloque 2: (Entrada – Proceso – Salida)

Ejercicio 16: Preparar una taza de café de forma manual.

Ejercicio 17: Preparar una taza de “Té de bolsita”

Ejercicio 18: Hacer una compra por internet de un producto en “Amazon”.

Ejercicio 19: Recargar la batería a un teléfono celular.

Ejercicio 20: Insertar una tarjeta SIM a un celular.

1.15.5.2 Algoritmos Cuantitativos

Son aquellos que resuelven problemas que dependen del uso de datos numéricos y de operaciones matemáticas, realizando un conjunto finito de secuencias lógicas para obtener resultados cuantificables y verificables. Tienen las siguientes características:

- En sus acciones utilizan datos numéricos y pueden incluir operaciones matemáticas, lógicas y de comparación dependiendo del problema.
- Las operaciones de sus acciones se ejecutan siguiendo un orden lógico bien definido, donde cada paso depende del anterior.
- Producen como salida resultados numéricos, que pueden ser cuantificables y verificables.

- Pueden ser traducibles a un lenguaje de programación y ser ejecutados por una computadora u otro dispositivo.
- Se pueden emplear diferentes “notaciones” para describir su enunciado.
- Se pueden emplear diferentes formas para “describir o representar” el paso a paso del algoritmo (pseudocódigos, diagramas y programas).
- En un proceso de formación, fortalecen el desarrollo del pensamiento lógico y computacional y sirven como base para el aprendizaje algorítmico y la programación.

Actividades con ejercicios resueltos

Ejemplo 1: Algoritmo que, dados tres números, calcular y dar conocer su promedio.

1. Leer los tres números: N1, N2, N3
2. Sumar: Los tres números
3. Promedio: La suma dividida para 3
4. Mostrar: El Promedio calculado
5. Proceso finalizado

Ejemplo: Asignamos cualquier valor a N1, N2 y N3

N1 = 18

N2 = 20

N3 = 10

Suma = $18 + 20 + 10 = 48$

Promedio = $48 / 3$

Resultado: 18

Ejemplo 2: Diseñe un algoritmo para calcular y mostrar el cuadrado de número.

1. Leer el número
2. Cuadrado: Multiplicar el número por sí mismo.
3. Mostrar: El cuadrado calculado
4. Proceso finalizado

Ejemplo: Asignamos cualquier valor a número

Número = 5

Cuadrado = $5 * 5$

Resultado: 25

Ejemplo 3: Algoritmo para calcular y mostrar el área de un triángulo, conociendo su base y altura. **Fórmula:** $\text{área} = (\text{base} * \text{altura}) / 2$

1. Leer base
2. Leer altura
3. Multiplicar base por altura
4. Dividir el resultado entre 2
5. Mostrar: el área calculada
6. Proceso finalizado

Ejemplo: Asignamos cualquier valor a la base y a la altura

base = 20

altura = 5

$\text{área} = (20 * 5) / 2 = 100 / 2 = 50$

Resultado: 50

Ejemplo 4: Algoritmo para calcular cuantos metros hay en una cantidad de kilómetros dados. **Observación:** 1 kilómetro es igual a 1000 metros.

1. Leer el número de kilómetros
2. Multiplicar kilómetros por 1000 metros
3. Mostrar: el número de metros
4. Proceso finalizado

Ejemplo: Asignamos cualquier valor a kilómetros

kilómetros = 8

$\text{metros} = 8 * 1000 = 8000$

Resultado: 8000

Ejemplo 5: Algoritmo para Calcular el interés simple de un capital, conociendo, la tasa de interés y el tiempo. **Fórmula:** $\text{Interés} = \text{Capital} \times \text{Tasa} \times \text{Tiempo}$.

1. Leer el capital
2. Leer la tasa de interés
3. Leer el tiempo
4. Multiplicar Capital por Tasa por Tiempo
5. Mostrar: el interés simple calculado
6. Proceso finalizado

Ejemplo: Asignamos cualquier valor a kilómetros

capital = 5000 tasa de interés = 0,08 tiempo = 3 años

interés = $5000 * 0,08 * 3 = 800$

Resultado: 1200

Ejercicios propuestos:

Problema 1: Elabore un algoritmo que calcule la suma de dos números.

Problema 2: Construya un algoritmo que calcule el promedio de cuatro calificaciones.

Problema 3: Construya un algoritmo que calcule el área y el perímetro de un rectángulo.

Problema 4: Construya un algoritmo que calcule el área total de un cilindro.

Problema 5: Elabore un algoritmo que convierta horas a minutos.

Problema 6: Algoritmo que, dado un valor, permita calcular su cuadrado y su cubo.

Problema 7: Algoritmo que dado un valor en pies lo convierta a centímetros.

Problema 8: Diseñar un algoritmo que muestre el siguiente número de un valor previamente ingresado.

Problema 9: Diseñar un algoritmo que muestre el CUADRADO del número siguiente a un valor ingresado.

Problema 10: Algoritmo que dado el peso de una persona en kilogramos, permita calcular y mostrar dicho peso en libras.

Problema 11: Algoritmo que dado un valor correspondiente a una distancia en millas marinas, la transforme en kilómetros.

Problema 12: Diseñe un algoritmo que dado una temperatura en grados Celsius (centígrados) transforme a grados Fahrenheit.

Problema 13: Algoritmo que dado un tiempo en: Horas, Minutos y segundos, se transforme dicho tiempo a SEGUNDOS.

Problema 14: Diseñe un algoritmo que calcule el salario mensual de un trabajador si se conoce el pago por hora y las horas trabajadas.

Problema 15: Elabore un algoritmo que calcule el monto total a pagar si se agrega el 15% de IVA a una compra.

Problema 16: Un comerciante compra un artículo a un costo dado. Determine el precio al cual debe venderlo si desea ganar el 25 %. Haga el algoritmo respectivo.

Problema 17: Diseñar un algoritmo para calcular el porcentaje de hombres y de mujeres que hayan un grupo, dados los totales de hombres y de mujeres.

Problema 18: Diseñe un algoritmo que dado un tiempo en semanas se transforme a: Días, Horas, Minutos y segundos.

Problema 19: Algoritmo que dada una velocidad en Kilómetros por horas la transforme en: Kilómetros por minutos y Kilómetros por segundos.

Problema 20: Diseñar un algoritmo que dado un valor (N), permita calcular su sumatoria:

$$S = 1+2+3+...+ N$$

1.15.6 Formas de Representar un Algoritmo

Para representar un algoritmo existen diversas formas, y usar la apropiada dependerá del nivel de formalidad y propósito, pues, cada forma responde a necesidades del contexto en donde se utilice al algoritmo. Para el propósito de formación, en este texto se aplicarán las siguientes:

1.15.6.1 Representación en Lenguaje Natural

Utiliza la descripción escrita mediante lenguaje natural (palabras) para describir paso a paso las actividades y/o procesos de un algoritmo. Aquí, se enumeran la lista de instrucciones o acciones a seguir. La descripción de las instrucciones debe ser lo más precisa posible, evitando ambigüedad. Tiene las siguientes características:

- Es secuencial.
- No formal.
- Es ambiguo si no se redacta cuidadosamente.
- Útil para introducción conceptual y aprendizaje inicial de los algoritmos.
- Tiene la desventaja de no ser ejecutable ni formalmente verificable.

Todos los ejercicios desarrollados y propuesto en el texto hasta ahora, han sido creados mediante el lenguaje natural. Ejemplos:

Cambiar una bombilla fundida

- 1) Ubicar una escalera debajo de la bombilla fundida
- 2) Tomar una bombilla nueva
- 3) Subir por la escalera
- 4) Girar la bombilla fundida hacia la izquierda hasta soltar
- 5) Enroscar la bombilla nueva en el plafón hasta apretarla
- 6) Bajar de la escalera

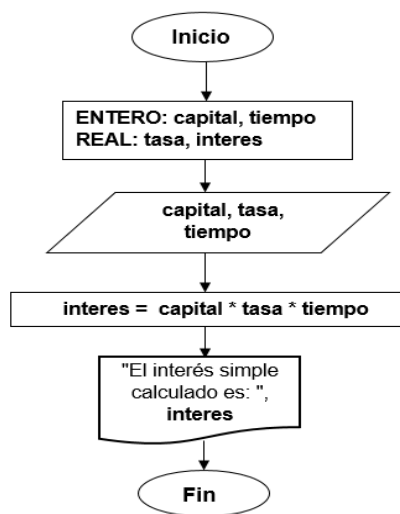
Calcular el interés simple

- 1) Leer el capital
- 2) Leer la tasa de interés
- 3) Leer el tiempo
- 4) Multiplicar Capital por Tasa por Tiempo
- 5) Mostrar: el interés simple calculado
- 6) Proceso finalizado

1.15.6.2 Diagrama de Flujo

Un diagrama de flujo es una forma de representar “gráficamente” a un algoritmo, mediante símbolos estandarizados que representan las diferentes acciones en la resolución de un problema. Este tipo de representación permite visualizar de forma clara la estructura lógica de un algoritmo, incluyendo secuencias, decisiones y ciclos. Su principal ventaja es la claridad visual, especialmente en etapas iniciales de aprendizaje de los algoritmos.

Ejemplo: Calcular el interés simple



Los diagramas de flujo serán estudiados con más profundidad en el capítulo 2.

1.15.6.3 Pseudocódigo

Es una forma estructurada de describir los pasos de un algoritmo, utilizando un lenguaje intermedio entre el lenguaje natural y un lenguaje de programación. Aunque no está estandarizado, utiliza una serie de “palabras” que representan las acciones del algoritmo, entre las que se encuentran las notaciones para representar condiciones, ciclos, funciones y otros aspectos que no dependen de un lenguaje de

programación en concreto. Esto permite, representar la lógica del algoritmo de forma clara y facilitar la traducción posterior a código real en un lenguaje de programación. Tiene las siguientes características:

- Tiene una estructura de sintaxis semiformal, pero más clara y precisa que el lenguaje natural.
-
- Es independiente del lenguaje computacional, aunque muy cercano a un lenguaje de programación.

Ejemplo: Calcular el interés simple

INICIO

LEER capital

LEER tasa

LEER tiempo

interes $\leftarrow$ capital * tasa * tiempo

ESCRIBIR "El interés simple calculado es: ", interes

FIN

En los siguientes capítulos, se analizará con aplicación práctica el uso de Pseudocódigos a problemas resueltos.

1.15.6.4 Representación en Código (programa en un lenguaje de programación)

Corresponde a la implementación del algoritmo a código real. Este caso, las acciones del algoritmo se transforman a un conjunto de instrucciones detalladas pertenecientes a un Lenguaje de Programación, que al ejecutarlas en una computadora realizan una tarea específica, objetivo funcional del algoritmo.

Ejemplo: Calcular el interés simple

Inicio

capital = float(input("Ingrese el capital: "))

tasa = float(input("Ingrese la tasa de interés: "))

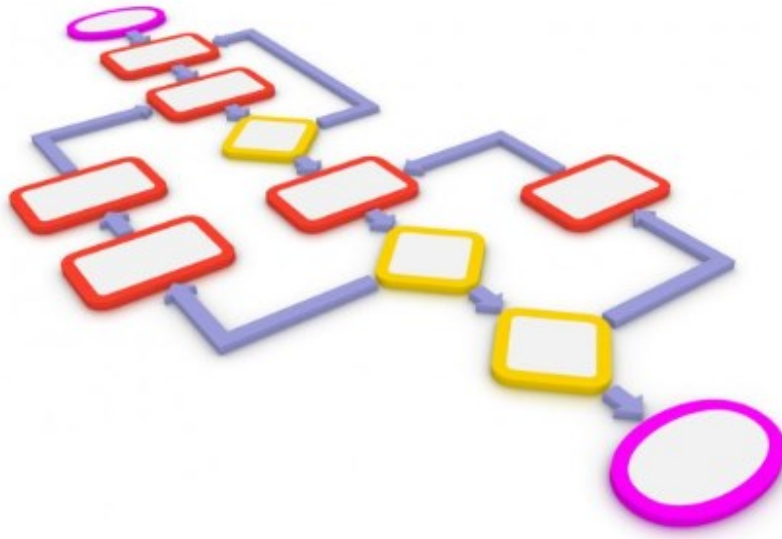
tiempo = float(input("Ingrese el tiempo: "))

interes = capital * tasa * tiempo

print("El interés simple calculado es:", interes)

Fin

Capítulo II. Representación formal de algoritmos



Objetivo General del Capítulo

Desarrollar la capacidad de analizar y representar soluciones mediante algoritmos, utilizando diagramas de flujo y pseudocódigos como herramientas formales, a través del uso de las estructuras de control fundamentales (secuenciales, condicionales y repetitivas), con el propósito de potenciar el pensamiento lógico y algorítmico en la resolución estructurada de problemas.

2.1 Diagrama de Flujo

Un Diagrama de Flujo (flowchart, del inglés), es una representación “gráfica” de los procesos de un Algoritmo. En lugar de explicar las acciones de los procesos solo con palabras, se utilizan “figuras (símbolos)” que representan dichas acciones que se ejecutan del algoritmo (entrada-salida, procesos, decisiones, entre otros), así como flechas que indican el orden o secuencia del flujo a seguir dentro del diagrama.

Una de las ventajas de utilizar un diagrama de flujo es el fortalecimiento del pensamiento lógico y analítico dentro del proceso de creación de un algoritmo. Por otra parte, dada su naturaleza visual, permite una mejor comprensión de los procesos y secuencias de un algoritmo, lo que los hace útiles para enseñar y aprender programación. De esta manera, permite a un principiante visualizar el proceso de solución del algoritmo antes de comenzar a escribir el código o instrucciones en un lenguaje de programación, lo que puede prevenir errores y ahorrar tiempo en la creación de programas eficientes.

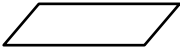
2.2 Símbolos básicos del diagrama de flujo

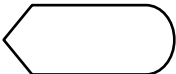
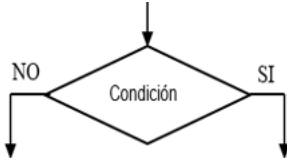
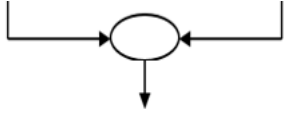
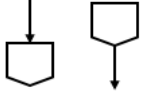
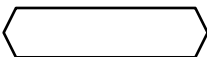
Son figuras estandarizadas (tabla 4) que representan diferentes tipos de acciones o elementos dentro de un proceso. Su uso permite que cualquier persona pueda comprender de forma clara y lógica el diagrama.

La información específica que se incluye dentro de cada símbolo varía según las acciones del algoritmo a desarrollar y de quien va a diseñar el diagrama de flujo.

Tabla 4

Símbolos Básicos del Diagrama de Flujo

Símbolo	Figura (Representación)	Uso
Óvalo		Indica el inicio y/o fin del diagrama. Debe ser el primer (inicio) y último (fin) símbolo del diagrama.
Paralelogramo		Representa una entrada de datos.
Rectángulo		Indica una operación de tipo aritmética (cálculo) o un proceso de asignación.

Símbolo	Figura (Representación)	Uso
Salida		Representa una salida de información
Rombo		Indica una comparación entre datos (pregunta o condición). Tendrá siempre dos salidas que requiere una elección (si/no) y cuyas acciones dependen del resultado de la comparación.
Conector de reagrupamiento		Permite la reagrupación de flujos en una salida. Generalmente se lo utiliza para cerrar las condiciones.
Conectores de página		Indica la continuidad de un diagrama en otro sitio (página). Utilice números (o letras), para indicar la secuencia.
Ciclo "Para"		Permite la realización de ciclos o bucles "para" dentro del diagrama.
Módulo o subrutina		Indica un proceso externo de llamado a un módulo, subrutina o procedimiento.
Flechas de Flujo		Indican la progresión o dirección del flujo de un paso al otro dentro del diagrama.

Nota. Fuente: Elaboración propia



Existen **herramientas para digitalizar y construir los diagramas** de flujo, pero se recomienda al principiante empezar con el diseño manual pues, el uso del software optimiza solo la representación visual y técnica del diagrama.

2.3 Reglas para elaborar diagramas de flujo

Las siguientes **reglas para elaborar diagramas de flujo** deben aplicarse desde la etapa inicial de diseño, independientemente del medio utilizado para su

elaboración. En un proceso formativo, es recomendable construir primero los diagramas de manera **manual**, ya que esto facilita la comprensión de la lógica del proceso y el significado de cada símbolo antes de utilizar herramientas digitales.

- Al dibujar un diagrama, sus símbolos siempre deberán estar unidos mediante las flechas de flujo y las líneas de flujo no pueden cruzarse.
- Todo símbolo (excepto las líneas de flujo) llevará en su interior información que indique su función dentro del diagrama.
- Se recomienda que los diagramas, se grafiquen de arriba hacia abajo o de izquierda a derecha. Esto para un buen diseño del algoritmo y ayudar a la codificación posterior.
- El inicio y final del diagrama solo aparecen una vez.
- En la parte superior se encuentran los soportes de entrada de **datos**, en la parte central estarán los **procesos** y en la parte final se encontrará la **salida** o resultados.
- Cada símbolo se utilizará de acuerdo a su propósito. Por ejemplo, en el símbolo condicional, no deben incluirse asignaciones; ni en el símbolo de proceso hacer condiciones.
- Un símbolo no puede tener más de una salida, si no es un elemento de decisión.
- Si se requiere más de una hoja para su construcción, use conectores de página y numerar las mismas de forma conveniente.



Las reglas sirven como guía para que el proceso sea claro, ordenado y fácil de entender.

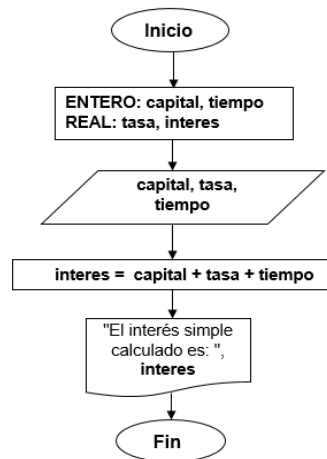
2.4 Primer acercamiento a los diagramas de flujo

Como se describió, los diagramas de flujo representan gráficamente a los algoritmos, pero es necesario empezar a comprender, que los algoritmos y en consecuencia los diagramas, tienen procesos con diferentes niveles de complejidad lógica. En una etapa inicial de aprendizaje, se emplearán **diagramas con procesos secuenciales solo a manera de ejemplos**, los cuales permiten comprender la estructura básica de un algoritmo sin incorporar aún estructuras condiciones ni de repeticiones o bucles.

Un diagrama secuencial describe a un algoritmo, en el que las acciones se ejecutan de forma **lineal y ordenada**, donde cada actividad se realiza y conduce directamente a la siguiente, lo que favorece la comprensión del flujo lógico y en la identificación de entradas, procesos y salidas. Ejemplo:

Calcular el interés simple

- 1) Leer el capital
- 2) Leer la tasa de interés
- 3) Leer el tiempo
- 4) Multiplicar Capital por Tasa por Tiempo
- 5) Mostrar: el interés simple calculado
- 6) Proceso finalizado



Como se puede observar, los símbolos de un diagrama de flujo se utilizan para representar diferentes tipos de acciones del algoritmo y cada uno tiene su propio significado. Por lo tanto, nunca estarán “vacíos”, ya que en ellos convergen: secuencia de operaciones, manejo de datos que deben ser ingresados, procesados y mostrados como resultados, así como estructuras lógicas de control que serán estudiadas más adelante en texto.

2.5 Elementos básicos de los algoritmos en diagramas de flujo

Un diagrama de flujo no solo muestra la secuencia de pasos, sino también una serie de elementos lógicos que intervienen en su ejecución, según las “acciones” del algoritmo a desarrollar y de quien va a diseñar el diagrama de flujo. Entre ellos se encuentran los datos, las variables, constantes, expresiones, operadores, las estructuras de control, entre otros elementos, que permiten representar de forma clara cómo se define y procesa la información dentro de un diagrama.

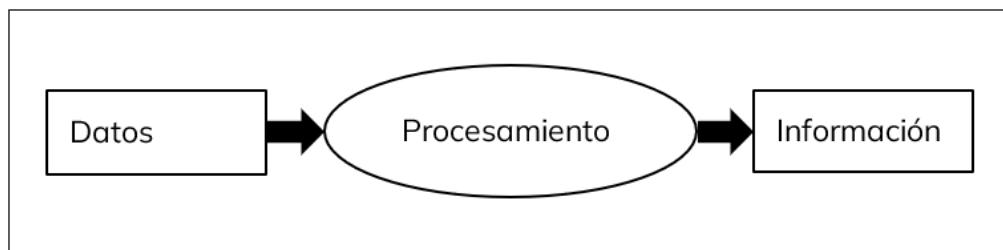
2.5.1 Dato y Tipos de Datos

2.5.1.1 Dato

En forma general, un **dato** representa “algo” que puede ser una “cantidad, hecho o acontecimiento sin analizar”, por lo que es considerado la “unidad mínima de información”. En el contexto informático, un **dato** es un valor o referente que recibe la computadora por diferentes medios (entrada) y que agrupados y procesados con otros datos representan la **información** (figura 10), que el desarrollador manipula en la construcción de una solución mediante un algoritmo.

Figura 10

Datos e información



Nota. Fuente: Elaboración propia

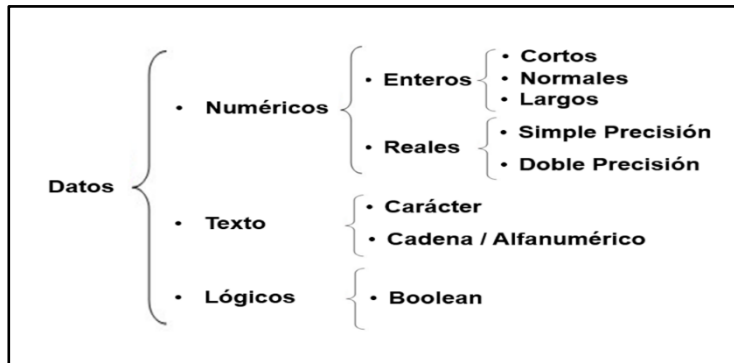
2.5.1.2 Tipos de Datos Básicos (datos primitivos)

Un diagrama de flujo, sin importar su tamaño o complejidad, siempre requiere manejar información, ya que sus acciones operan sobre valores que poseen características específicas. Estos valores pueden representar cantidades numéricas, textos, caracteres o condiciones lógicas, y su correcta identificación es fundamental para diseñar algoritmos consistentes. Por lo expuesto, al crear un diagrama de flujo (o en programación) es indispensable determinar primero a qué tipo o categoría corresponden los datos que se utilizarán.

En este contexto, los **datos primitivos** toman importancia al ser considerados los **tipos de datos básicos y fundamentales** que utiliza un lenguaje de programación (o diagrama) para representar información, ya que la clasifican según su naturaleza y establecen las bases para su manipulación (figura 11). Por lo tanto, comprender estos tipos de datos facilita la definición precisa de las operaciones que se realizan en cada etapa del diagrama (o programa) y define el tipo de valor que puede almacenarse y las operaciones que pueden realizarse sobre él.

Figura 11

Clasificación de Tipos de Datos



Los tipos de datos permiten comprender **qué información se ingresará y procesará, por lo tanto el Programador/Desarrollador es quien deberá determinar el tipo de datos con los que trabajará su algoritmo.**

Nota. Fuente: Elaboración propia

Se consideran “primitivos” porque no están formados por otros tipos de datos y sirven como base para construir estructuras más complejas.

2.5.1.3 Características y propiedades de los Tipos de Datos Primitivos

Cada tipo de dato primitivo define el tipo de valor que puede almacenarse y las operaciones que pueden realizarse sobre él (tabla 5). Por esta razón, es necesario conocer las características y propiedades de los datos que circulan dentro del flujo del diagrama y/o programa, antes de aprender a almacenar esa información en variables.

Tabla 5

Características y Propiedades de los Tipos de Datos

Tipo de Dato	Descripción	Número de Bytes en memoria	Rangos de valores	de Operaciones permitidas
Enteros	Corto (Byte)	1 Byte	0 a 255	Sumar, restar, dividir, multiplicar, residuo
Permiten almacenar valores numéricos sin parte decimal	Normal (Integer)	2 Bytes	- 32768 a + 32767	
	Largo (Long)	4 Bytes	- 2147483648 a	

Tipo de Dato	Descripción	Número de Bytes en memoria	Rangos de valores	de Operaciones permitidas
			+ 2147483647	
Reales Permiten almacenar valores numéricos con decimales	Simple Precisión (Float) Doble Precisión (Double)	4 Bytes (7 dígitos decimales) 8 Bytes (14 dígitos decimales)	-3.4E+38 a +3.4E+38 -1.7E+308 a +1.7E+308	Sumar, restar, dividir, multiplicar
Texto El tipo de dato texto corresponde a cualquier carácter(es) no numérico.	Tipo de dato Carácter es una unidad o símbolo que puede ser una letra, un número o cualquier carácter especial. (Char)	1 Byte	Cualquier carácter que forme parte del conjunto ASCII reconocido por la computadora.	Sumar caracteres (concatenación), longitud y extraer. Estas operaciones se analizarán más adelante en este texto
	Tipo de dato Cadena o Alfanumérico es usado para identificar aquella información compuesta por la combinación de números, letras y otros caracteres dispuestos de forma consecutiva. (String)	Dependerá del tamaño de la cadena	Hasta 255 caracteres, aunque dependerá del lenguaje de programación	
Lógico	Tipo de dato que puede representar dos valores: verdadero o falso (Boolean)	2 Bytes	Verdadero o Falso	And, Or, Not

Nota. Fuente: Elaboración propia basada en diversas fuentes disponibles en la Web.

2.5.2 Almacenamiento de información: Variables

2.5.2.1 Definición

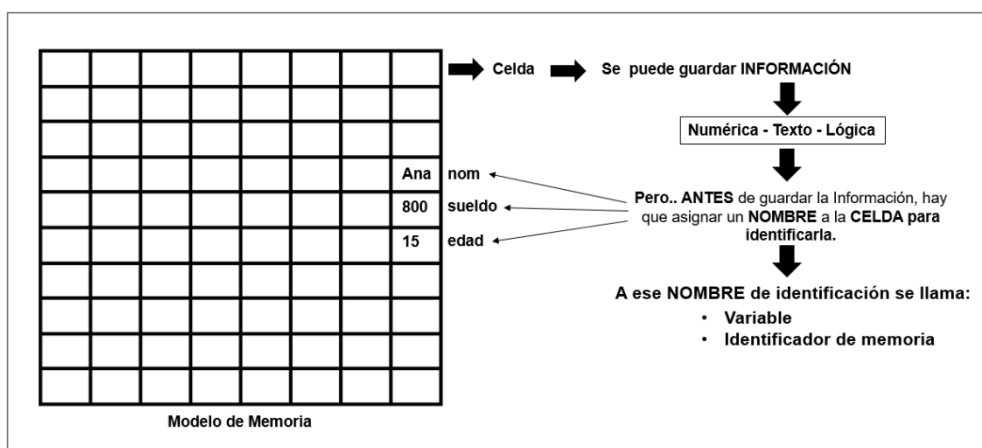
Una variable es un nombre utilizado para almacenar datos dentro de un espacio de la memoria mientras se procesa. Cada variable tiene un nombre que permite identificarla y un tipo de dato que determina la clase de información que puede contener. Se llama variable ya que esta área de memoria puede almacenar, de manera temporal, diferentes valores durante la ejecución de un programa.

Como “ilustración didáctica”, se utiliza un “modelo de memoria” (la memoria físicamente no es así), para comprender mejor la definición de **variable**. En este modelo (figura 12), cada celda representa en forma ilustrativa una “posición o espacio de memoria”, en donde a manera de ejemplo se requiere almacenar los datos de una persona. Para ello, se debe:

- En primer lugar, asignar nombres a esas celdas o espacios para lo cual se han creado a tres variables, cuyas identificaciones son: nom, sueldo y edad.
- En segundo lugar, una vez creadas las tres variables en memoria, se ingresan los datos a cada una de ellas (Ana, 800, 15). A nivel de diagramación, este ingreso se hará con el símbolo de “paralelogramo” y en programación se usa la instrucción respectiva de “ingreso”, según el lenguaje de programación a utilizar.

Figura 12

Ilustración de Memoria como Modelo Didáctico



Nota. Fuente: Elaboración propia.

2.5.2.2 Características principales de una variable

La variable debe ser creada por el programador/desarrollador y debe cumplir las siguientes características:

- Tiene un **nombre** que la identifica.
- Almacena un **valor** que puede cambiar con el tiempo.
- Está asociada a un **tipo de dato** (entero, real, texto, lógico, etc.).
- Permite realizar **operaciones y cálculos** dependiendo de su tipo.



Recuerde **siempre.... Primero se crea(n)** la(s) variable(s) y **segundo, se ingresará** los valores de acuerdo al tipo de dato con la cual se creó la(s) variable(s).

2.5.2.3 Clasificación de las Variables según el Tipo de Información

Según el tipo de dato que almacena o guarda una variable, estas se clasifican en forma general en: Numérica, Alfanumérica y Lógicas (booleanas). Esta clasificación permite identificar la naturaleza de los datos que serán utilizados dentro del algoritmo y facilita su correcta representación en un diagrama de flujo y/o programa.

2.5.2.3.1 Variables Numéricas

Son aquellas que sólo pueden contener o almacenar valores numéricos. También pueden contener un signo más (+) o el signo menos (-) y un punto (.) para separar valores enteros con valores decimales. Incluyen variables en base a la clasificación establecida en la tabla 2.x. Ejemplos:

suelo = **500**

auxiliar = **+15**

contador = **-1**

pi = **3.1416**

2.5.2.3.2 Variables Alfanuméricas

Son aquellas que pueden contener en su interior información textual (letras, palabras, símbolos o combinaciones de caracteres). Incluyen variables de carácter y cadenas de texto. Ejemplos:

nombre = "María"

dirección = "Calle 5 Av. 40"

letra = "W"

clave = "ABC12345"

2.5.2.3.3 Variables Lógicas

Son aquellas que pueden contener en su interior solo dos estados (verdadero o falso).

aprobado = verdadero

acceso = falso

En este texto, en los diagramas de flujo se utilizará principalmente las variables con los tipos de datos: Numéricas (Entero y Real) y Alfanuméricas (Cadena y Carácter).



Una variable, al ser de un determinado tipo, **sólo puede tomar valores de ese mismo tipo.**

2.5.2.4 Asignación de nombres a las variables

El nombre que se asigne a una variable debe ser “descriptivo”, es decir, que describa la información que almacena en forma clara, lo que facilita la lectura, comprensión y mantenimiento del algoritmo por cualquier programador/desarrollador.

En los diagramas de flujo, las siguientes reglas básicas para nombrar variables tienen un carácter formativo y buscan mejorar la claridad y comprensión del algoritmo. Estas reglas, no están sujetas todavía a la sintaxis estricta de un lenguaje de programación; sin embargo, es recomendable aplicarla desde esta etapa como

“buenas prácticas de nombramiento”, para facilitar la transición posterior hacia el uso de software o lenguajes de programación:

- El nombre a la variable debe iniciar siempre con una letra (A – Z), sin incluir la letra ñ y no deben contener espacios en blanco. Ejemplos:

nombre

sueldo

edad

cubo

contador

- Pueden combinarse letras y números. Pero el primer carácter debe ser letra. ejemplos:

contador1

auxiliar2

calculo5

- El nombre de una variable debe ser descriptivo o nemotécnico; es decir que, con solo ver el nombre de la variable, se comprenda la información que va a almacenar. Ejemplos:

nombreproducto

correoelectronico

sueldogerente

aporteiess

impuestoiva

- El nombre de una variable, no puede ser ninguna palabra reservada del lenguaje de programación en que se está codificando el diagrama. Ejemplos:

If main for else

- Se puede usar un guion bajo (_) para que el nombre sea más descriptivo. Ejemplos.

aporte_iess

sueldo_netto

impuesto_iva

En programación, cada lenguaje tiene sus propias reglas (revisar manuales), adicionalmente a las ya descritas anteriormente, lo que las convierte en obligatorias, al formar parte de la sintaxis del lenguaje.



- En un diagrama, pseudocódigo o programa siempre se referencia el nombre de la variable, ya sea para: ingresar, procesar o mostrar un valor.
- La correcta definición del tipo y del nombramiento de la variable, facilita su identificación y uso adecuado en las operaciones donde se las utilice.

2.5.2.5 Cómo Declarar Variables en un Diagrama de Flujo

Según el tipo de dato que va a contener la variable en un diagrama, se debe proceder:

- a) Si es un **Dato Numérico**, definir la variable como: **Entero** o **Real**, según la información a guardar y/o procesar.
- b) Si es un dato **Alfanumérico**, definir la variable como: **Cadena** o **Caracter**, según la cantidad de información a guardar y/o manipular.

Para declarar o definir cualquiera sea el tipo de variable, considere lo siguiente:

1. Toda variable que se utilice en un diagrama/pseudocódigo/programa, debe ser declarada con su respectivo tipo de dato. Usted debe analizar el tipo de dato, según la información a procesar (ver figura 2.x).
2. En un diagrama de flujo, la declaración de variables se hace dentro de un símbolo de proceso, inmediatamente después del símbolo de inicio y de la siguiente forma:

Tipo de dato: nombre variable1, nombre de la variable2,...

Ejemplo 1: Crear las variables numéricas y enteras: horas y sueldo.

Entero: horas, sueldo

Dado que las dos variables tienen el mismo tipo se pueden crear, separándolas con la coma (",").

o también en forma individual

Entero: horas

Entero: sueldo

Ejemplo 2: Crear las variables numéricas: horas, valor por hora, sueldo y aporte al IESE. En este caso se asume que en las variables para almacenar las horas, valor por

hora y sueldo, no tendrán decimales y serán de tipo ENTERO. Mientras que el aporte al IESS, también es numérica, pero de tipo REAL, ya que se almacenará con valor decimal al realizar el cálculo respectivo en el diagrama.

Entero: horas, valorhora, sueldo

Real: aporteieess

Ejemplo 3: Utilizando el ejemplo anterior, se solicita aumentar la variable: nombre, para almacenar el nombre del trabajador.

Entero: horas, valorhora, sueldo

Real: aporteieess

Cadena: nombretabajador

Actividades con ejercicios propuestos

Práctica: Escriba la declaración más apropiada para cada una de la siguiente información (tabla 6). La definición estará dada por el tipo de dato (vea la información) y el nombre de la variable que debe ser descriptiva (se recomienda no usar nombres de variables muy largos en cantidad de caracteres).

Tabla 6

Ejercicios para identificar Tipo de Datos y asignar Nombres a Variables

Información de referencia	Definición de Tipo de dato y nombre de Variable
nombre	Cadena: nombre
edad	Entero: edad
altura	
género (valores: H: hombre, M: mujer)	
númeroDeHijos	
sueldo	
Iva	
tallaCamisa (valores: S,L, M, XL)	
Peso (en kilos)	
precio	

Información de referencia	Definición de Tipo de dato y nombre de Variable
alumnoRepetidor (valores: SI/NO)	
Mensaje	
numeroDías	
minutos	
placaVehículo	
ContadorTurnos	
totalSueldosUniversidad	
baseCuadrado (en centímetros)	
mayorDeEdad (valores: verdadero/falso)	

Nota. Fuente: Elaboración propia

2.5.2.6 Paradigma de Programación a utilizar en los Diagramas de Flujo

Como se mencionó en el capítulo anterior, en la programación los paradigmas representan un “modelo” o “estilo” de programación para la construcción de un programa informático, determinando así su estructura”. En este texto, se utilizará el **Paradigma Imperativo** que describe el “¿Cómo se hace?” y se basa en una secuencia de acciones (procesos y/o instrucciones), en las que se detalla “**paso a paso qué debe hacer**” para resolver un problema mediante diagrama y/o programa. En este paradigma se debe determinar:

1. **¿Qué datos se utilizarán?:** Se trata de definir o declarar las variables con su respectivo tipo de dato que serán requeridos y utilizados en toda la estructura del algoritmo. A este proceso se denominará como: “**La Estructura de Datos**”.
2. **¿Qué Algoritmo se aplicará?:** Se trata de definir el **Conjunto finito** de pasos, acciones u instrucciones **lógicas**, que siguiendo un orden específico conducen a la **solución** de un **problema**. Aquí en donde se procesan las variables según un control de flujo establecido por el **algoritmo** desarrollado.

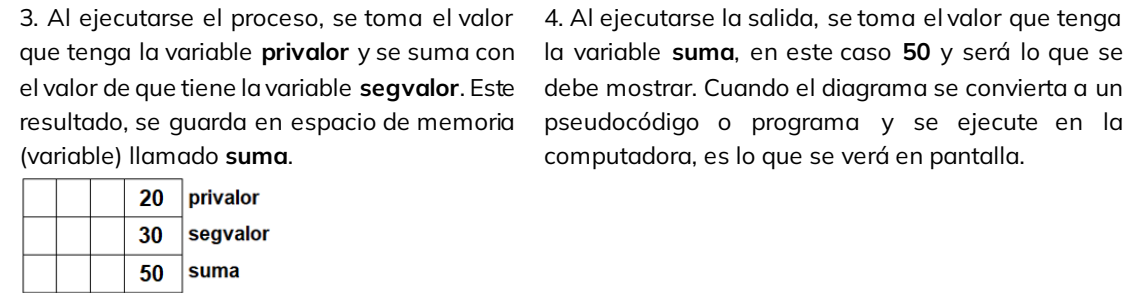
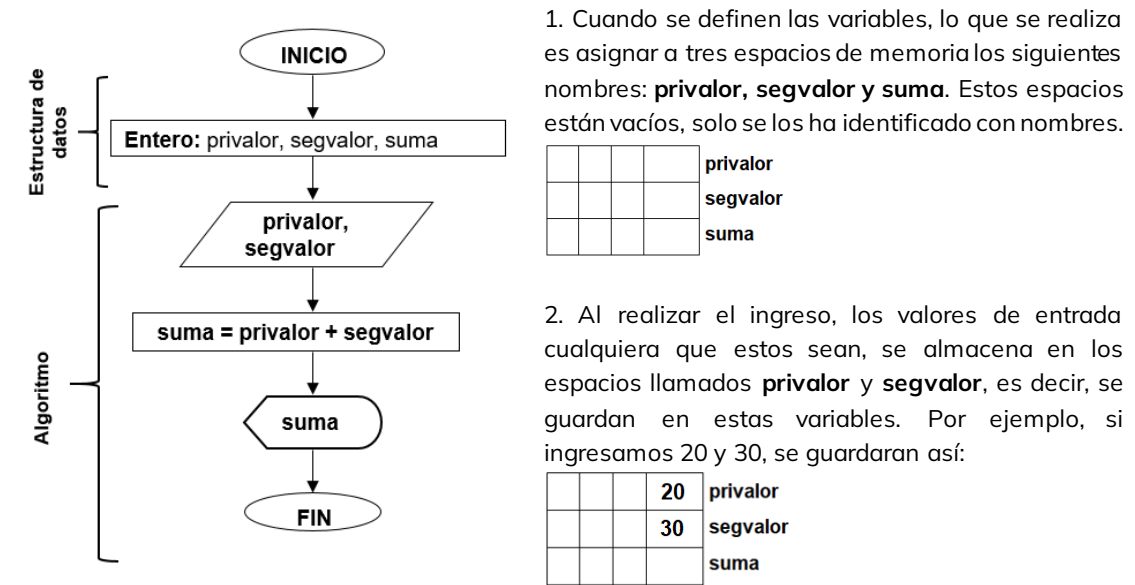
En forma sintetizada, este paradigma se lo representa de la siguiente forma, que será la que aplicará en los diagramas de flujo en este texto:

Programa = Estructura de Datos + Algoritmo

Actividades con problemas resueltos

Ejemplo 1: Crear un diagrama que permita ingresar dos valores, se pide sumar estos valores y mostrar el resultado de dicha suma. Posterior a esto terminar el proceso.

Para entender este diagrama, se debe seguir el flujo inmediatamente después del símbolo de inicio:



Como se puede observar, el diagrama está conformado por dos partes: La primera parte contiene la **estructura de datos**, que es donde se define o declaran las variables. La segunda parte, contiene el **algoritmo**, es decir, la lógica de la solución al problema planteado. De esta forma se cumple con el **Paradigma Imperativo**:

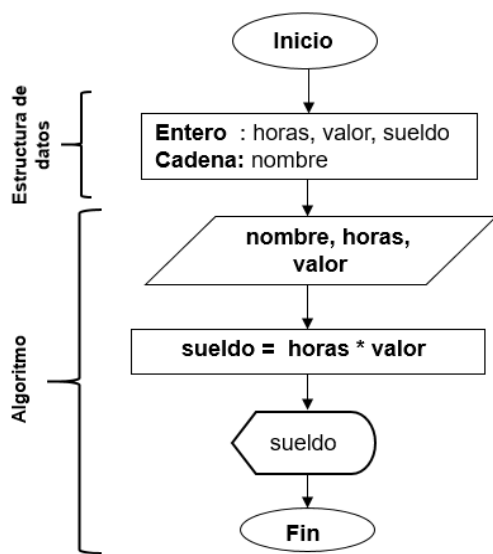
Programa = Estructura de Datos + Algoritmo



En los diagramas, pseudocódigos y programas **SIEMPRE se hace referencia al nombre de las variables**, pues, en ellas están **ALMACENADOS** los valores a ingresar y/o procesar. Tal como se puede observar en el diagrama anterior.

Ejemplo 2: Desarrollar un diagrama que permita calcular y mostrar el sueldo semanal de un trabajador conociendo su nombre, horas trabajadas y valor por hora.

Para entender este diagrama, se debe seguir el flujo inmediatamente después del símbolo de inicio:



1. Cuando se definen las variables, lo que se realiza es asignar a cuatro espacios de memoria los siguientes nombres: **horas**, **valor**, **sueldo** y **nombre**. Estos espacios están vacíos, solo se los ha identificado con nombres.

				nombre
				horas
				valor
				sueldo

2. Al realizar el ingreso, los valores de entrada cualquiera que estos sean, se almacena en los espacios llamados **nombre**, **horas**, **valor**, es decir, se guardan en estas variables. Por ejemplo, si ingresamos María, 40 y 8, se guardarán así:

			María	nombre
			40	horas
			8	valor
				sueldo

3. Al ejecutarse el proceso, se toma el valor que tenga la variable **horas** y se multiplica por el valor que tiene la variable **valor**. Este resultado, se guarda en espacio de memoria (variable) llamado **sueldo**.

			María	nombre
			40	horas
			8	valor
			320	sueldo

4. Al ejecutarse la salida, se toma el valor que tenga la variable **sueldo**, en este caso **320** y será lo que se debe mostrar. Cuando el diagrama se convierta a un pseudocódigo o programa y se ejecute en la computadora, es lo que se verá en pantalla.

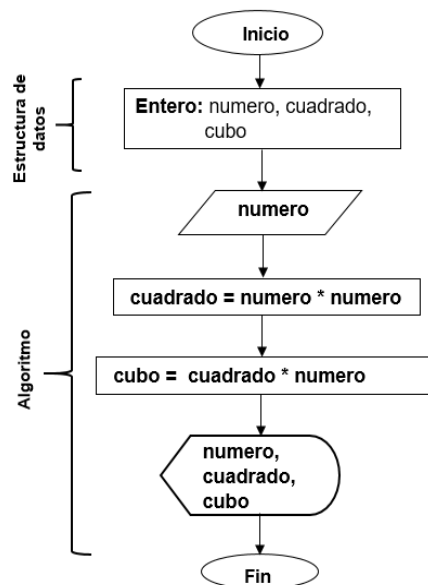
Ejemplo 3: Diseñe un diagrama para calcular y mostrar el cuadrado y el cubo de un número previamente ingresado.

Para este problema, vamos a iniciar con el “algoritmo cuantitativo” donde se tiene la solución mediante lenguaje natural.

1. Leer el número.
2. Cuadrado: Multiplicar el número por sí mismo.
3. Cubo: Multiplicar el cuadrado por el número leído
4. Mostrar el numero leído, cuadrado y cubo calculado
5. Proceso finalizado

Convertido el algoritmo cuantitativo a diagrama, quedará así:

Para entender este diagrama, se debe seguir el flujo inmediatamente después del símbolo de inicio:



1. Cuando se definen las variables, lo que se realiza es asignar a tres espacios de memoria los siguientes nombres: **numero**, **cuadrado** y **cubo**. Estos espacios están vacíos, solo se los ha identificado con nombres.

				numero
				cuadrado
				cubo

2. Al realizar el ingreso, el valor de entrada cualquiera que este sea, se almacena en el espacio de memoria llamado **numero**, es decir, se guarda en esta variable. Por ejemplo, si ingresamos 5, se guardará así:

			5	numero
				cuadrado
				cubo

3. Al ejecutarse el primer proceso, se toma el valor que tenga el **número** y se multiplica por sí mismo, guardando el resultado en el espacio de memoria (variable) llamado **cuadrado**.

Al ejecutarse el segundo proceso, se toma el valor que tenga la variable **cuadrado** y se multiplica por el contenido de la variable **numero**, guardando el resultado en el espacio de memoria (variable) llamado **cubo**.

			5	numero
			25	cuadrado
			125	cubo

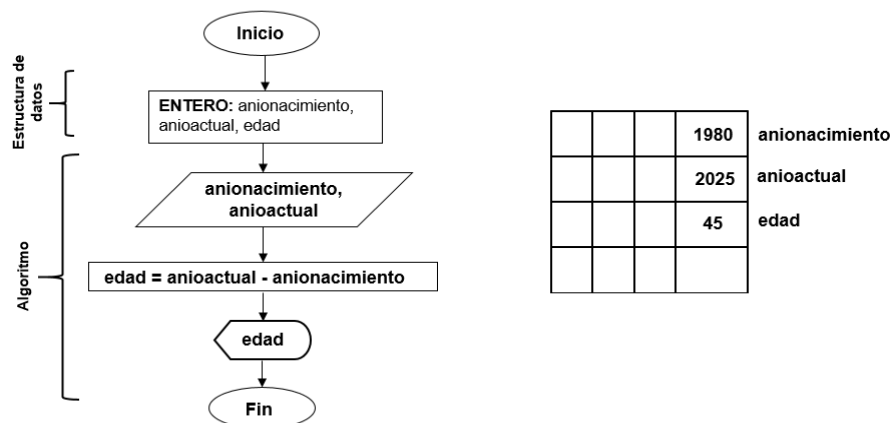
4. Al ejecutarse la salida, se toman los valores que tengan las variables **numero**, **cuadrado** y **cubo**, en este caso **5**, **25** y **125** y será lo que se debe mostrar. Cuando el diagrama se convierta a un pseudocódigo o programa y se ejecute en la computadora, es lo que se verá en pantalla.

Ejemplo 4: Diagrama que permita calcular una edad, conociendo el año actual y el año de nacimiento.

Para este problema, vamos a iniciar con el “algoritmo cuantitativo” donde se tiene la solución mediante lenguaje natural.

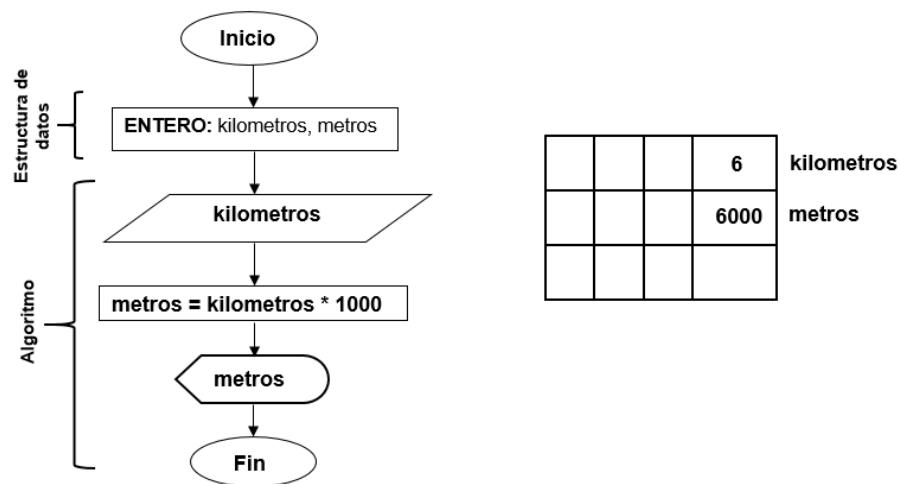
1. Leer el año actual y el año de nacimiento
2. Calcular Edad: Restar del año actual el año de nacimiento
3. Mostrar la Edad
4. Finalizar el proceso

En este ejemplo no se explicará la ilustración de memoria, solo se muestra como quedan las variables como resultado del proceso. En los problemas anteriores se realizó la explicación para comprender el concepto de variable. Convertido el algoritmo cuantitativo a diagrama, quedará así:



Ejemplo 5: Diagrama que permita calcular cuántos metros hay en una cantidad de kilómetros ingresados. **Observación:** 1 kilómetro es igual a 1000 metros

1. Leer el número de kilómetros
2. Multiplicar kilómetros por 1000 metros
3. Mostrar: el número de metros
4. Proceso finalizado



Ejercicios propuestos:

Para resolver los siguientes problemas, tome como referencia los ejercicios resueltos y planteados en los **Algoritmos Cuantitativos** del capítulo I, así como las **Metodologías Cognitivas para la resolución de problemas** en este mismo capítulo.

Problema 1: Diagrama para calcular y mostrar el cuadrado un número, previamente ingresado.

Problema 2: Diagrama para calcular y mostrar el número de horas en una cantidad de días ingresados.

Problema 3: Diagrama que permita calcular y mostrar el área de un triángulo, conociendo su base y altura. **Fórmula:** $\text{área} = (\text{base} * \text{altura}) / 2$.

Problema 4: Diagrama para Calcular el interés simple de un capital, conociendo, la tasa de interés y el tiempo. **Fórmula:** $\text{Interés} = \text{Capital} \times \text{Tasa} \times \text{Tiempo}$.

Problema 5: Diagrama que dado el peso de una persona en kilogramos, permita calcular y mostrar dicho peso en libras.

Problema 6: Diseñe un diagrama que dado un tiempo en número de semanas se conviertan y muestren a: Días y Horas.

Problema 7: Diagrama que dado un tiempo en: horas y minutos se transforme dicho tiempo a segundos.

Problema 8: Diagrama que dado un valor correspondiente a una distancia en millas marinas, la convierta y muestre a kilómetros.

Problema 9: Diagrama que dado un valor en pulgadas, se convierta y muestre a centímetros.

Problema 10: Diseñe un diagrama que dado una temperatura en grados Fahrenheit, se convierta y muestre a grados Celsius (centígrados).

Problema 11: Diseñe un diagrama que calcule y muestre el monto total a pagar si se agrega el 15 % de IVA a una compra.

Problema 12: Diseñar un diagrama para calcular el porcentaje de hombres y de mujeres que asisten a una clase de programación, dados los totales de hombres y de mujeres.

Problema 13: Diseñe un diagrama que permita calcular y mostrar el Área y el Perímetro de un terreno.

Problema 14: Un comerciante compra un artículo a un costo dado. Determine el precio al cual debe venderlo si desea ganar el 25 %. Haga el diagrama respectivo.

Problema 15: Diseñe un diagrama que dados el pago por hora y las horas trabajadas mensualmente, calcule y muestre: salario mensual, aporte al IESS (9.35 del sueldo mensual) y el sueldo neto a recibir.

2.5.3 Las Constantes

2.5.3.1 Definición

Una constante, es un valor que permanece inalterable en la ejecución de un diagrama y/o programa. Es decir, que ese valor no va a cambiar durante toda la ejecución del algoritmo. A diferencia de una variable, cuyo contenido puede modificarse, la constante mantiene el mismo valor desde que se define hasta que finaliza el proceso.

En forma general, según la naturaleza del valor constante, se clasifican en Numéricas y Alfanuméricas (aunque también se puede utilizar constantes Lógicas).

2.5.3.2 Constantes Numéricas

Son valores numéricos que no se alteran y que pueden ser positivos, negativos, enteros o reales. Ejemplos:

Enteras: 1 5 0 - 2 100

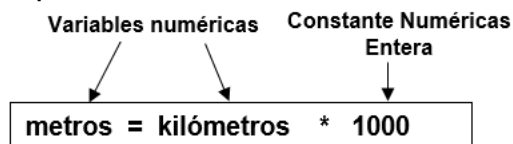
Real: 9.35 0.5 -1.5

Generalmente se las utiliza para:

- “Ayudar” a realizar procesos matemáticos, donde se necesiten emplear valores ya conocidos.
- Para la inicialización de variables, según el contexto del algoritmo.

Según su forma de definición, las constantes pueden ser usadas directamente en las operaciones del algoritmo o diagrama (o programa), y toman el nombre de “**constantes literales**”. Esta forma de aplicación es la que se ha utilizado en algunos diagramas de flujo.

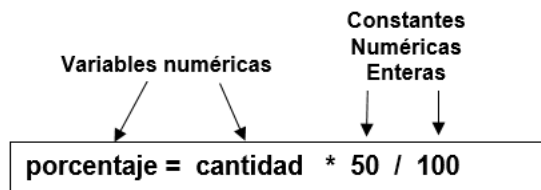
Ejemplo 1:



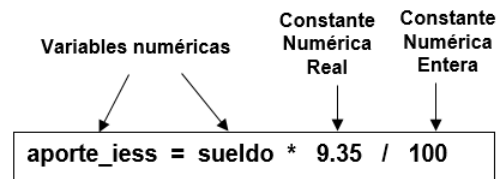
Ejemplo 2:



Ejemplo 3:

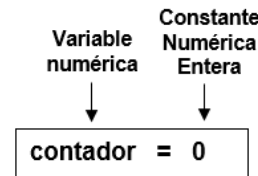


Ejemplo 4:



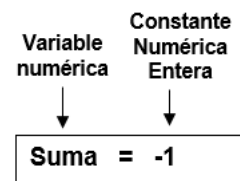
Ejemplo 5:

Inicialización de una variable

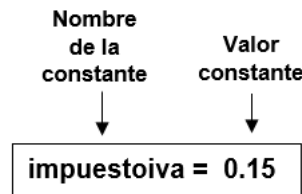
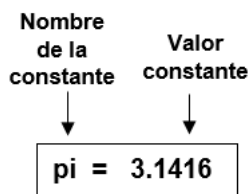


Ejemplo 6:

Inicialización de una variable



Las constantes numéricas también pueden ser usadas identificándolas con un nombre, en este caso son llamadas **“constantes simbólicas”**. Ejemplos:



2.5.3.3 Constantes Alfanuméricas

Son aquellas constantes que están formadas por una secuencia o hilera de caracteres que se encuentran siempre dentro de comillas dobles o comillas simples (apóstrofe). Generalmente se las utiliza para enviar títulos o mensajes de salida por pantalla o impresora.

Ejemplo 1:

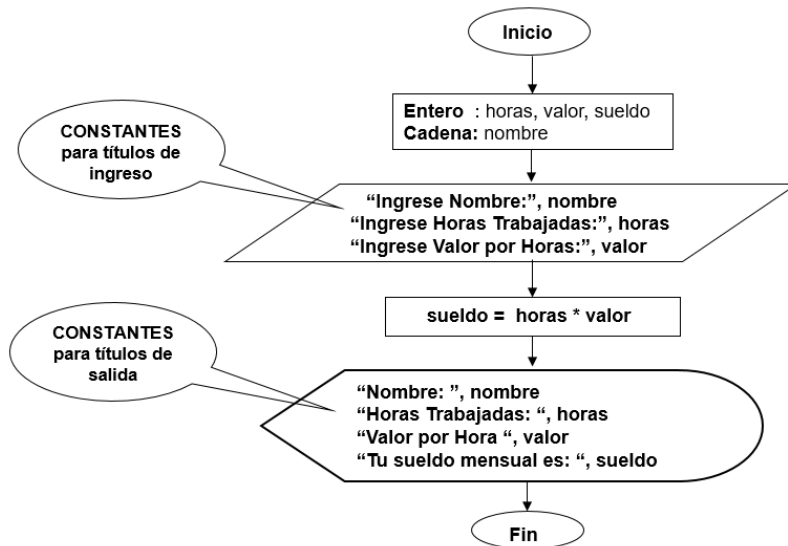
"Universidad Laica Eloy Alfaro de Manabí"
"\$ 25.000,00"
"Ingrese su Clave"

Ejemplos 2:

clave = "abcABC123"
titulo = "Facultad de Ciencias Informáticas"

Constantes con
nombres o
identificadores

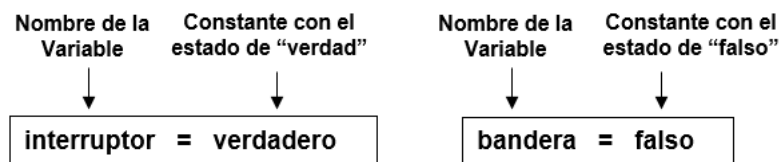
Las constantes alfanuméricas también pueden ser utilizadas como mensajes informativos o descriptivos en los símbolos (o instrucciones en el caso de programación) de entrada y salida en los diagramas de flujo. Ello permite orientar al usuario sobre la información que debe ingresar o interpretar los resultados obtenidos. Ejemplo:



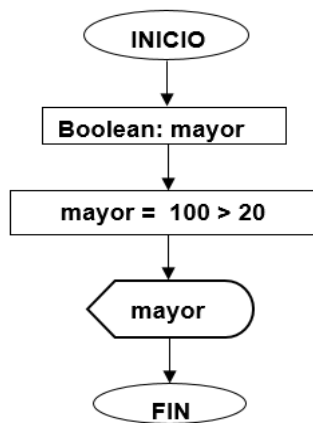
Al utilizar a las constantes alfanuméricas como **mensajes informativos o descriptivos en los ingresos o salidas**, estas etiquetas o títulos deben tener relación con la información que ingresa o se muestra como resultado.

2.5.3.4 Constante Lógica

Es un tipo especial de constante, que contiene un valor fijo que representa un estado de verdad o falsedad, y que no cambia durante la ejecución del algoritmo o programa. Por lo tanto, solo puede contener uno de dos valores posibles: verdadero o falso. **Ejemplos:**



Las constantes lógicas, se utilizan principalmente en estructuras condicionales y también como una forma de expresión lógica. A manera de ilustración, se presenta el siguiente ejemplo:



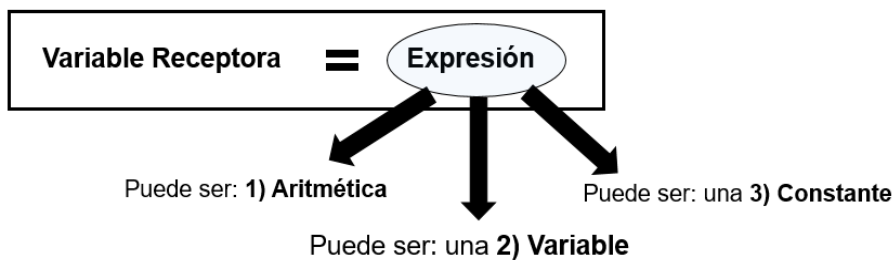
El resultado que se almacena en la variable de tipo booleana llamada “mayor” sería **verdadero**, puesto que el valor 100 es mayor que 20. Por lo tanto, ese sería el resultado que se mostraría.



La variable que contenga cualquiera de estos dos estados, debe ser declarada con el tipo de dato lógico o booleano.

2.5.4 Las Expresiones

En programación, es un proceso que se utiliza para indicar una operación de cálculo y/o asignación (en diagramación utilice el símbolo de rectángulo). El resultado de la evaluación de la expresión genera un valor que se almacenará en una variable receptora. La forma para utilizar una expresión en un diagrama de flujo es el siguiente:



El signo igual (=) significa que primero se realiza lo que está a la derecha de la expresión y el resultado que se genera se almacena en la variable receptora que se encuentre a la izquierda de la expresión.

Las expresiones en un diagrama y/o programa, pueden ser de tres tipos: formada por procesos aritméticos, formada solo por variable y formada solo por constante.

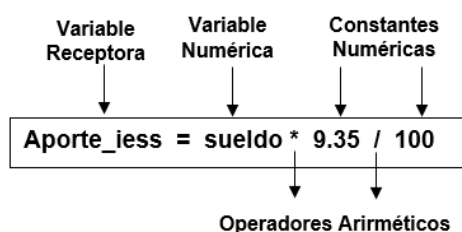
2.5.4.1 Expresiones formadas por Procesos Aritméticos

Este tipo de expresión se utiliza al hacer un cálculo matemático o una llamada a una función (esto último será explicado más adelante en este texto). Considere lo siguiente para su uso:

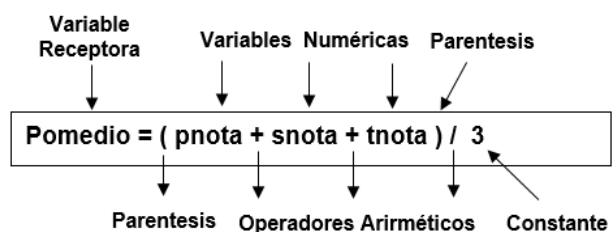
- Estará formada por operandos y operadores. **Los Operandos** pueden ser: variables y/o constantes y **los Operadores** deben ser Aritméticos (operaciones matemáticas comunes). También puede haber paréntesis y nombre de funciones especiales.
- La expresión genera un valor resultante que se almacena en la variable receptora, de acuerdo a las operaciones aritméticas establecidas.
- El valor resultante de la expresión debe ser del mismo tipo que la variable receptora.

En los algoritmos y diagramas ya revisados hasta ahora, en su mayoría se ha utilizado expresiones aritméticas.

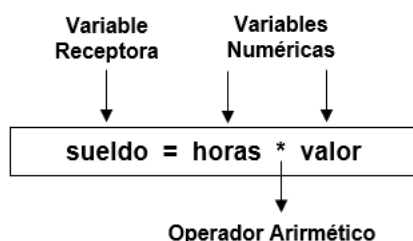
Ejemplo 1:



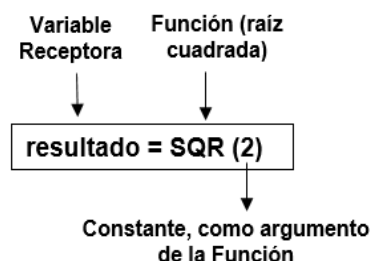
Ejemplo 2:



Ejemplo 3:



Ejemplo 4:



2.5.4.2 Expresión formada solo por Variable

A esta forma de expresión se la conoce también como **Asignación**. Una operación de asignación, consiste en asignar el valor del operando derecho que corresponde a una variable a la ubicación de almacenamiento representado por la variable receptora.

Reglas para la operación de Asignación:

- Tanto la variable del lado derecho, como la variable receptora deben ser del mismo tipo.
- La variable receptora (izquierda) es la única que cambia de valor cuando ya ha tenido un valor asignado. Por esta razón se considera un proceso **“destructivo”**.

Ejemplo: $a = 2$ y $b = 5$

2
5

 $\begin{matrix} a \\ b \end{matrix}$ Al hacer la expresión $a = b$ quedará:

5
5

 $\begin{matrix} a \\ b \end{matrix}$

El valor que tenía la variable “a” que era 2, se pierde en memoria, quedando con 5.

- La variable del lado derecho conserva su valor después de ejecutarse la asignación.

Ejemplo: $a = 2$ y $b = 5$

2
5

 $\begin{matrix} a \\ b \end{matrix}$ Al hacer la expresión $a = b$ quedará:

5
5

 $\begin{matrix} a \\ b \end{matrix}$

El valor que tiene variable “b” que es 5, no se pierde en memoria, quedando con 5.

- Si la variable receptora, está también del lado derecho, cambia de valor por el hecho de aparecer a la izquierda.

Ejemplo 1: $a = 2$ y $b = 5$

2
5

 $\begin{matrix} a \\ b \end{matrix}$ Al hacer la expresión $a = a + b$ quedará:

7
5

 $\begin{matrix} a \\ b \end{matrix}$

Ejemplo 2: $a = 10$

10

 a Al hacer la expresión $a = a + 1$ quedará:

11

 a

En los temas de **contadores** y **acumuladores** que serán explicados más adelante en este texto, se utilizará el **literal d)** como parte de estas temáticas para su entendimiento y aplicación.

2.5.4.3 Expresión formada solo por Constante

También se considera una operación de asignación que puede:

Asignar un nuevo valor en forma de constante a una variable receptora. Ejemplos:

titulo = "Universidad Laica Eloy Alfaro de Manabí"

pi = 3.1416

iva = 0.15

- c) Asignar un valor de inicialización a una variable, especialmente en los procesos en que se deba usar acumuladores y contadores. Ejemplos:

Contador = 1 totalsueldo = 0

Ejercicios propuestos con uso de expresiones:

Ejercicio 1

¿Cuál será el valor que tomarán las siguientes variables, tras la ejecución estas instrucciones?

A = 25
B = A + 8
C = B - 5
D = C + (A + B)

B =
C =
D =

Ejercicio 2

Evaluar las siguientes expresiones aritméticas, considerando los valores:

a = 5, b = 10, c = 2, d = 12, e = 5, R1 = 9, R3 = -1

R1 = R1 + (a + b)	R1 =
R2 = R1 + c	R2 =
R3 = R3 + d	R3 =
R4 = R3 + e	R4 =
R1 = R1 + (d / 2)	R1 =

Ejercicio 3

¿Qué valores quedan almacenados en las variables a, b y c ?

a = 10, b = 5, c = 3

a = a + 3

b = b + 4 - a

c = a + b + c

a = a + c

b = 4

c = c + 3 - b + 2

a = ____

b = ____

c = ____

2.5.5 Los Operadores

Son elementos de un diagrama y/o programa que realizan operaciones matemáticas y lógicas, principalmente sobre valores numéricos. Son fundamentales para realizar cálculos, comparaciones y tomar decisiones para producir resultados.

Los operadores se clasifican en:

- Aritméticos
- Relacionales
- Lógicos
- Asignación

El operador de asignación que corresponde al “=” ya fue descrito anteriormente en las expresiones y sus formas. En el apartado de las estructuras de control se analizan los operadores de relación y lógicos.



En un diagrama de flujo los operadores aritméticos y asignación van dentro de símbolo de proceso (rectángulo). Los operadores relacionales y lógicos van el símbolo de condición (rombo)

2.5.5.1 Los Operadores Aritméticos

Los operadores aritméticos son símbolos que permiten realizar las operaciones matemáticas comunes (tabla 7) sobre valores numéricos en diagrama y/o programa. Los operadores aritméticos se utilizan para calcular un valor con base en una serie de “operandos” en una expresión. Por lo tanto, se utilizan para producir resultados a partir de variables y/o constantes.

Tabla 7

Operadores Aritméticos

Operador	Función
$\wedge$ o $**$	Potenciación
$*$	Multiplicación
$/$	División real
$\backslash$	División entera
MOD o $\%$	Residuo de una división
$+$	Suma
$-$	Resta

Nota. Fuente: Elaboración propia

Es importante mencionar que para realizar la potenciación, es necesario consultar el manual del lenguaje de programación a utilizar ya que el símbolo de la tabla anterior puede no ser compatible. A nivel de diagramación se podrán utilizar sin inconvenientes.



En los diagramas de flujo, los operadores aritméticos aparecen dentro del **símbolo de proceso (rectángulo)**.

2.5.5.1.1 Jerarquía de los Operadores Aritméticos

Cuando una expresión contiene varios operadores aritméticos, el resultado depende del **orden en que se ejecutan las operaciones**. A este orden se le conoce como **jerarquía de operadores o precedencia**. Por lo tanto, las operaciones se ejecutan por prioridad (tabla 8), de mayor a menor.

Tabla 8

Orden de Jerarquía de los Operadores Aritméticos

Precedencia prioridad	o Operador	Función
1	()	Paréntesis
2	$\wedge$ o $**$	Potenciación
3	* / \	Multiplicación División Real División Entera
	MOD o %	Residuo de división
4	+ -	Suma Resta



Si es una expresión, **no se respeta esta jerarquía**, el resultado puede ser incorrecto.

2.5.5.1.2 Consideraciones en una Expresión Aritmética

Cuando se combinan diferentes operadores en una expresión, el valor determinado por una expresión se calcula siguiendo un orden de evaluación en función de las siguientes reglas de precedencia:

- a) Para alterar el orden de precedencia entre los operadores aritméticos use los paréntesis **()**.
- b) Ante la ausencia de paréntesis en la expresión, las operaciones se realizan tomando en consideración las prioridades descritas en tabla 2.x.

Ejemplo: Calcular el promedio de tres notas

$$\text{promedio} = \frac{\text{pnota} + \text{snota} + \text{tnpta}}{3}$$

1) Divide
2) Suma

Se obtiene un resultado, pero no sería correcto para calcular el promedio, ya que primero divide y después suma.

- a) Si la expresión contiene paréntesis, se ejecutará primero la operación(es) dentro de ellos. Si existen paréntesis anidados estos se resolverán de adentro hacia afuera. Ejemplo: Calcular el promedio de tres notas

$$\text{promedio} = \frac{(\text{pnota} + \text{snota} + \text{tnpta})}{3}$$

2) Divide
1) Suma

Se obtiene el resultado correcto para calcular el promedio, ya que primero suma los valores de las variables entre paréntesis y al final divide.

- b) Si la expresión contiene operadores de igual jerarquía, estos se resuelven de izquierda a derecha.

Ejemplo: Calcular el aporte al IESS

$$\text{aporte\_iess} = \text{suelo} * 9.35 / 100$$

→

Ejercicios resueltos de conversiones de expresiones algebraicas a aritméticas

Considerando el orden de precedencia de los operadores aritméticos, se muestran los siguientes ejemplos, para convertir **expresiones algebraicas** a **expresiones aritméticas** que puedan ser procesadas en un diagrama y/o programa para computadora.

Expresiones Algebraicas**Expresiones Aritméticas**

Ejercicio 1:	$Y = \frac{A+B}{2}$	→	$Y = (A + B) / 2$
Ejercicio 2:	$Y = (A+B)^2$	→	$Y = (A + B) ^ 2$
Ejercicio 3:	$C = 4x - (2y + 7)$	→	$C = (4 * x) - ((2 * y) + 7)$
Ejercicio 4:	$Y = \frac{a+b}{c-d}$	→	$Y = (a + b) / (c - d)$
Ejercicio 5:	$C = \frac{x}{y} (z + w) \pi$	→	$C = x / y * (z + w) * 3.1416$

Ejemplos propuestos de conversiones de expresiones algebraicas a aritméticas

Transformar las siguientes expresiones algebraicas a expresiones aritméticas que puedan ser procesadas en un diagrama y/o programa para computadora.

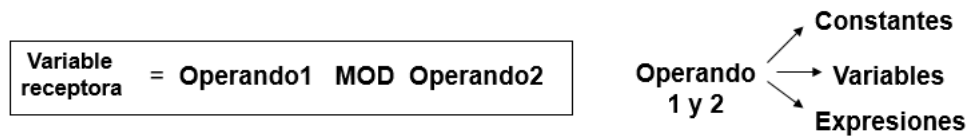
Expresiones Algebraicas**Expresiones Aritméticas**

Ejercicio 1:	$Y = A^2 + B^2$	→	$Y =$
Ejercicio 2:	$X = \frac{Y+1}{2}$	→	$X =$
Ejercicio 3:	$Z = \frac{a+b}{c-d}$	→	$Z =$
Ejercicio 4:	$Y = \frac{m}{n} + p$	→	$Y =$
Ejercicio 5:	$Y = \frac{a^2}{b-c}$	→	$Y =$
Ejercicio 6:	$Z = 4x - 2y + 7$	→	$Z =$
Ejercicio 7:	$Y = m + \frac{n}{p-q}$	→	$Y =$
Ejercicio 8:	$Y = \frac{a^2 + b^2}{\frac{1}{x^2}}$	→	$Y =$
Ejercicio 9:	$w = \frac{x+y}{x} + \frac{2x}{x-2}$	→	$W =$
Ejercicio 10:	$Y = \frac{x+y}{u + \frac{w}{b}}$	→	$Y =$

2.5.5.1.3 Operador Aritmético MOD (Módulo o Residuo)

El operador aritmético **MOD** (o también **%**) se utiliza para obtener el residuo de una división entera entre dos números (operandos). Para utilizar este operador se

debe usar el siguiente formato, considerando que los operandos deben ser valores **enteros**:



Para comprender como funciona este operador, considere los siguientes aspectos:

- a) Si el valor de Operando1 es **Mayor o Igual** al valor de Operando2, el valor resultante será el **resto o residuo** de la división entre Operando1 y Operando2.

Ejemplo 1: $a = 7 \text{ MOD } 4$ El resultado es 3 porque $7 / 4 = 1$, con residuo de 3.

Ejemplos 2:

$R = 10 \text{ MOD } 4$	$R = 15 \text{ MOD } 3$	$R = 7 \text{ MOD } 3$	$R = 20 \text{ MOD } 7$
2	0	1	6

- b) Si el valor de Operando1 es **Menor** al valor de Operando2, el valor resultante será lo que contiene Operando1, ya que no alcanza a dividirlo ni una sola vez.

Ejemplo 1: $A = 2 \text{ MOD } 4$ El resultado es 2

Ejemplos 2:

$R = 10 \text{ MOD } 100$	$R = 3 \text{ MOD } 15$
10	3

- c) Caso especiales con los valores de Operando1 y Operando2.

- Si Operando2 tiene como valor cero (0), el lenguaje genera un mensaje de error.

Ejemplo: $A = 7 \text{ MOD } 0$ Error en el lenguaje de programación

- Si Operando1 tiene como valor cero (0), el resultado siempre será cero.

Ejemplo: $A = 0 \text{ MOD } 7$ El resultado es 0

En programación, el operador MOD es una herramienta muy importante ya que permite trabajar con problemas que involucran principalmente: residuos, divisibilidad y ciclos dentro de un algoritmo. Ejemplos de algunos problemas clásicos de aplicación:

- Determinar si un número es par o impar.
- Determinar si un número es primo.
- Mostrar divisores de un número.
- Determinar si un número es múltiplo de otro
- Detectar posiciones específicas dentro de una secuencia.
- Validaciones numéricas (verificar si un año es o bisiesto)
- Alternar comportamientos (por ejemplo, acciones en posiciones pares e impares).



Este operador, dependiendo del lenguaje o la herramienta para crear pseudocódigo, puede representarse como: MOD o %.

2.5.6 La Prueba de Escritorio

Antes de ejecutar un algoritmo mediante un programa en la computadora, es necesario y recomendable comprobar que su lógica de resolución funciona correctamente. La **prueba de escritorio** es el procedimiento “manual” de prueba algorítmica, que consiste en la validación y verificación del algoritmo a través de la ejecución de las acciones que lo componen (procesos), para determinar sus resultados (salida) a partir de un conjunto inicial de datos (entrada).

2.5.6.1 La importancia de la Prueba de Escritorio

Las pruebas de escritorio, al permitir hacer un seguimiento paso a paso de cada actividad del diagrama, ayuda a determinar:

- El correcto funcionamiento y efectividad del algoritmo representado en el diagrama.
- Identificar posibles errores de lógica que puedan provocar resultados incorrectos.
- Verificar todos los procesos que tengan cálculos.

- Validar las condiciones, en el caso que el diagrama las tuviere.
- Comprobar la efectividad de los ciclos, en el caso que el diagrama los utilice.
- Asegurarse que el algoritmo produce el resultado esperado.
- Entender la lógica interna del algoritmo.

Al realizar la prueba de escritorio, quien la realice se debe cuestionar (figura 13) acerca de la efectividad del proceso y funcionamiento del algoritmo.

Figura 13

Validación y Verificación del Algoritmo



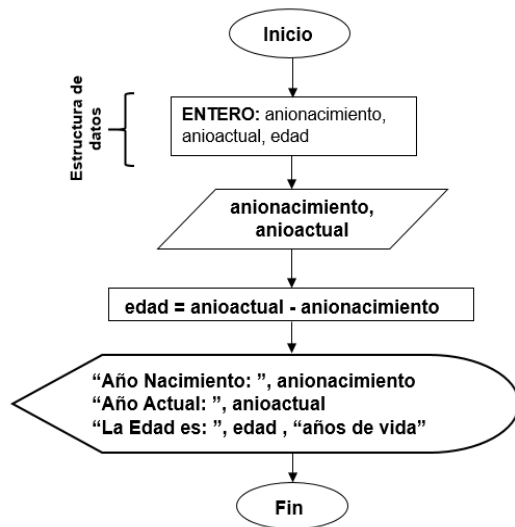
Nota. Fuente: Elaboración propia con gráfico disponible en la Web

2.5.6.2 Pasos para realizar una prueba de escritorio

Para simular manualmente la ejecución del algoritmo, hay que seguir los siguientes pasos:

- Hacer una lista de todas las variables que se usan en el diagrama (aquí es importante ver la estructura de los datos en la primera parte del diagrama). Al final de la lista, se utiliza por lo general la palabra “salida”, para indicar los resultados que se mostrarán.
- Dar valor(es) de entrada a la(s) variable (s) que sólo se ingresan (símbolo de entrada).
- Evaluar el/los proceso(s) en base a la secuencia del diagrama.
- Evaluar condiciones, bucles u otras acciones dentro del diagrama, si los tuviere.
- Verificar si los resultados cumplen con las especificaciones solicitadas en el enunciado del problema.

Ejemplo 1: Diagrama que permita calcular una edad, conociendo el año actual y el año de nacimiento.



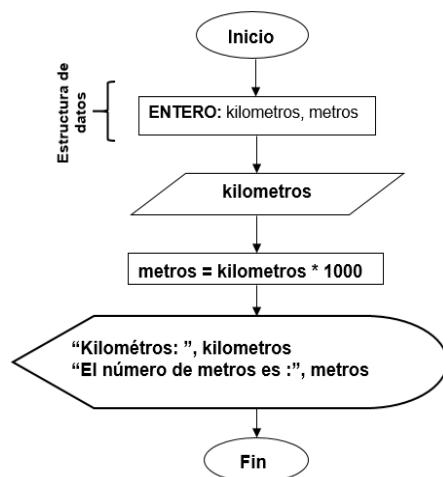
Prueba de Escritorio

anionacimiento	anioactual	SALIDA
1980	2026	Año Nacimiento: 1980 Año Actual: 2026 La Edad es: 46 años de vida

Al iniciar la prueba considere:

1. Se crea la estructura de datos, con todas las variables a utilizar con sus respectivos tipos de datos. En este caso: anionacimiento, anioactual y edad.
2. Se ingresa valor solo a las variables de entrada. En este caso se ingresa 1980 a la variable anionacimiento y 2026 a la variable anioactual.
3. En el proceso, se calcula la edad en base a los datos ingresados (a la variable anionacimiento que tiene 1980 se resta lo que tiene la variable anioactual que 2026 5). El resultado obtenido es 46, se guarda en la variable edad.
4. Finalmente, se muestran los contenidos de las variables anionacimiento, anioactual y edad, utilizando etiquetas descriptivas (mensajes) para interpretar mejor los resultados obtenidos.

Ejemplo 2: Diagrama que permita calcular cuántos metros hay en una cantidad de kilómetros ingresados. **Observación:** 1 kilómetro es igual a 1000 metros



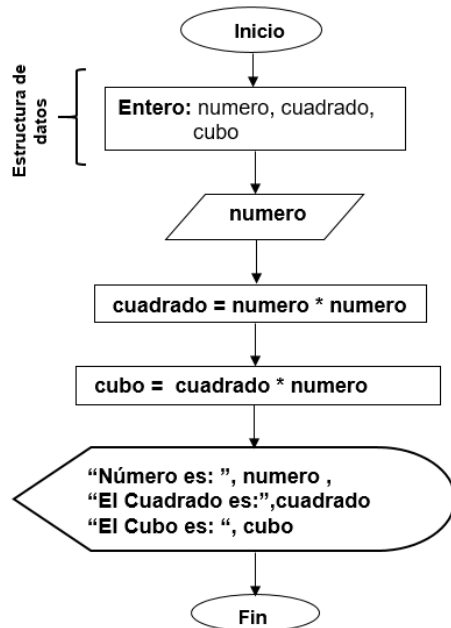
Prueba de Escritorio

kilometros	metros	SALIDA
5	5000	Kilómetros : 5 El número de metros es : 5000

Al iniciar la prueba considere:

1. Se crea la estructura de datos, con todas las variables a utilizar con sus respectivos tipos de datos. En este caso: kilometros y metros.
2. Se ingresa valor solo a la variable de entrada. En este caso se ingresa 5 a la variable kilometros.
3. En el proceso, se calcula el número de metros en base al valor ingresado (la variable kilometros tiene 5, que se multiplica por la constante numerica 1000). El resultado obtenido es 5000, que se guarda en la variable metros.
4. Finalmente, se muestran los contenidos de las variables kilometros y metros, utilizando etiquetas descriptivas (mensajes) para interpretar mejor los resultados obtenidos.

Ejemplo 3: Diseñe un diagrama para calcular y mostrar el cuadrado y el cubo de un número previamente ingresado.



Prueba de Escritorio

numero	cuadrado	cubo	SALIDA
3	9	27	Número es: 3 El Cuadrado es: 9 El Cubo es: 27

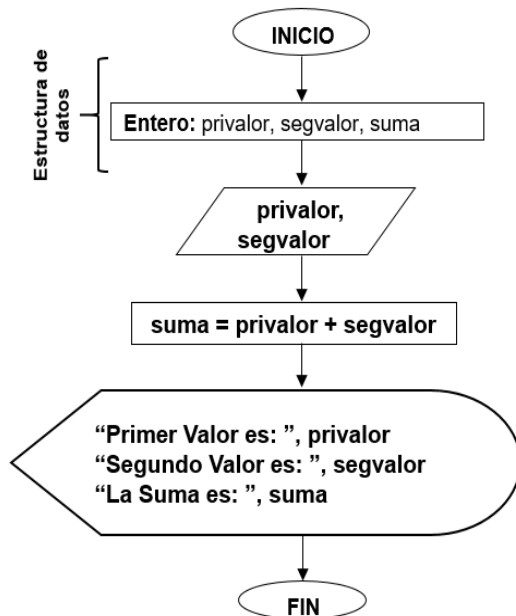
Al iniciar la prueba considere:

1. Se crea la estructura de datos, con todas las variables a utilizar con sus respectivos tipos de datos. En este caso: numero, cuadrado y cubo.
2. Se ingresa valor solo a la variable de entrada. En este caso se ingresa 3 a la variable numero.
3. En el primer proceso, se calcula el cuadrado del valor ingresado (la variable numero tiene 3, se multiplica por sí mismo). El resultado, obtenido es 9, se guarda en la variable cuadrado.
4. En el segundo proceso, se calcula el cubo del valor ingresado (la variable cuadrado tiene 9, que se multiplica por la variable numero que tiene 3). El resultado, obtenido es 27, se guarda en la variable cubo.
5. Finalmente, se muestran los contenidos de las variables numero, cuadrado y cubo, utilizando etiquetas descriptivas (mensajes) para interpretar mejor los resultados obtenidos.

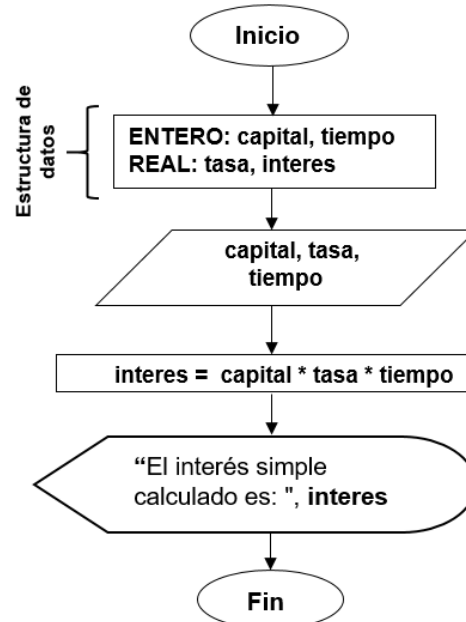
Actividad propuesta.

Realizar la prueba de escritorio para los siguientes diagramas de flujo:

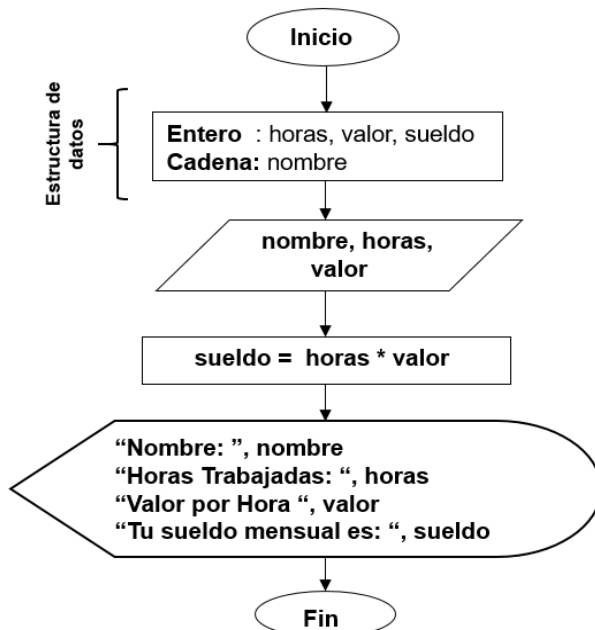
Problema 1: Diagrama que permite ingresar dos valores, se pide sumar estos valores y mostrar el resultado de dicha suma. Posterior a esto terminar el proceso.



Problema 2: Diagrama que permite calcular y mostrar el interés simple, conociendo el capital, la tasa y tiempo.



Problema 3: Diagrama que permite calcular y mostrar el sueldo semanal de un trabajador conociendo su nombre, horas trabajadas y valor por hora.



2.5.7 El Pseudocódigo

El Pseudocódigo es una de las representaciones que se utilizan en el aprendizaje de la programación. Se trata de un pseudo (falso) lenguaje que utiliza instrucciones (palabras) de lenguaje natural, comprensibles para cualquier persona, pero los más parecido y simulando la estructura típica de un programa realizado en un lenguaje de programación. Es decir, se trata de una forma sencilla y estructura de representar la solución de un algoritmo para que pueda ser leído por una persona, pero sin las restricciones que impone un lenguaje de programación.

2.5.7.1 Características de un Pseudocódigo

- Utiliza una estructura lógica bien definida y parecida a un lenguaje de programación, mediante secuencias ordenadas de acciones para resolver un problema.
- El pseudocódigo es un paso intermedio entre los diagramas de flujo y los lenguajes de programación.
- Al ser una descripción en lenguaje natural de un algoritmo, resulta independiente de los lenguajes de programación.
- Emplean palabras claras y sencillas como parte de sus instrucciones, tales como: Inicio, Fin, Si, Entonces, Mientras, Leer, Escribir, entre otras.
- El pseudocódigo facilita la comprensión del algoritmo, lo que permite identificar errores en la lógica en su construcción.

2.5.7.2 Estructura de un Pseudocódigo

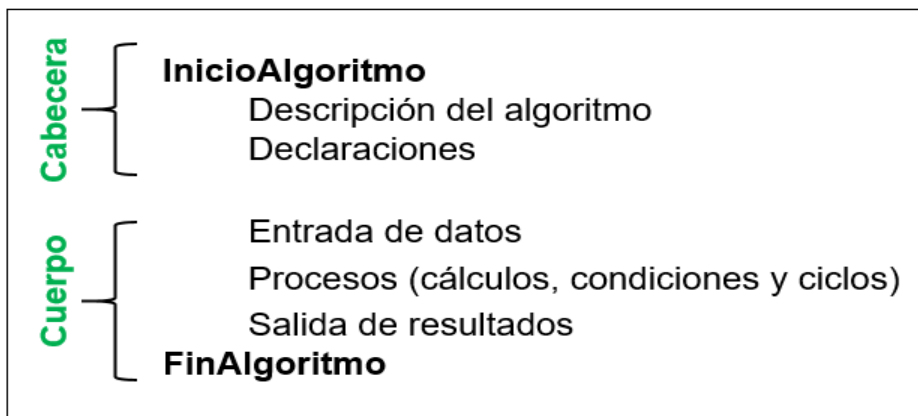
Todo pseudocódigo, tiene una estructura formal que sigue un orden lógico para representar un algoritmo. Generalmente se compone de las siguientes partes, que pueden variar según el contexto de desarrollo utilizado: la cabecera y el cuerpo del algoritmo (figura 2.5).

- a) **La Cabecera:** Esta sección sirve para identificar y contextualizar el pseudocódigo antes de escribir sus instrucciones. Generalmente, está formada por:
- El inicio y la descripción de la funcionalidad del algoritmo.
 - La declaración de variables y/o constantes con sus tipos de datos

- Declaración de módulos y/o funciones (serán analizadas más adelante)
- b) **El cuerpo:** Esta sección es representa la parte principal del pseudocódigo, ya que define la lógica del algoritmo y contiene todas las instrucciones para resolver el problema. Está formado por:
- Entrada de datos.
 - Procesos (las operaciones de cálculos).
 - Estructuras condicionales y repetitivas.
 - Salida de resultados.
 - Finalización.

Figura 14

Estructura General de un Pseudocódigo



Nota. Fuente: Elaboración propia

2.5.7.3 Instrucciones formales y básicas de un Pseudocódigo

El pseudocódigo está formado por instrucciones (palabras) que se emplean para describir las acciones del paso a paso en la solución del problema. Estas instrucciones son similares a las utilizadas en los lenguajes de programación, pero escritas en un lenguaje de fácil lectura para una persona. Entre las principales se tienen:

2.5.7.3.1 Para comentarios

A lo largo del pseudocódigo, se pueden agregar comentarios para explicaciones o aclaraciones que no afectan en la ejecución. Solo sirven para mejorar la comprensión en la lógica detrás de un proceso o uso de las instrucciones.

En pseudocódigo no existe una regla universal para comentarios, pero comúnmente se utilizan los siguientes caracteres según el contexto: #, // o /*.

Ejemplo:

```
# Algoritmo para calcular el área de un triángulo
o
// Algoritmo para calcular el área de un triángulo
o
/* Algoritmo para calcular el área de un triángulo */
```

2.5.7.3.2 Para Definir variables y sus Tipos de Datos

En pseudocódigo, para crear una variable y definir su tipo de datos que va almacenar, se utiliza la instrucción **definir** de la siguiente forma:

Definir nombre_variable **Como** Tipo

El Tipo hace referencia a los tipos de datos ya mencionados anteriormente: **Entero** (números sin decimales), **Real** (números con decimales), **Cadena** (texto) y **Lógico** (verdadero o falso).

Ejemplos:

Definir sueldo **Como** Entero

Definir area **Como** Real

Definir privalor, segvalor, tervalor **Como** Entero

Definir nombre **Como** Cadena

Definir bandera **Como** Lógico

Puede definir más de una variable del mismo tipo, separando por comas (,).

2.5.7.3.3 Para Ingresar Datos

En pseudocódigo, la instrucción que generalmente se utiliza para que el usuario pueda introducir o ingresar datos es la palabra: Leer

Se utiliza de la siguiente forma: **Leer** nombre_variable

Ejemplos:

Leer nombre

Leer sueldo

Leer base

2.5.7.3.4 Para Mostrar Datos o Información

En pseudocódigo, para mostrar datos o información se utilizan generalmente las instrucciones Mostrar o Escribir, según el contexto de desarrollo quedando a elección de la persona. Se pueden utilizar de la siguiente forma, según lo que se quiera mostrar.

Mostrar / Escribir nombre_variable

Mostrar / Escribir "título o descripción"

Mostrar / Escribir "título o descripción", nombre_variable

Ejemplos:

Mostrar sueldo o también **Escribir** sueldo

Escribir "Ingrese la base del triángulo"

Escribir "El área del triángulo es: ", area

Escribir "Tienes: ", tiempo, "de trabajo en la empresa"

Escribir "Tú saldo es: ", valor, "Hoy: ", fecha

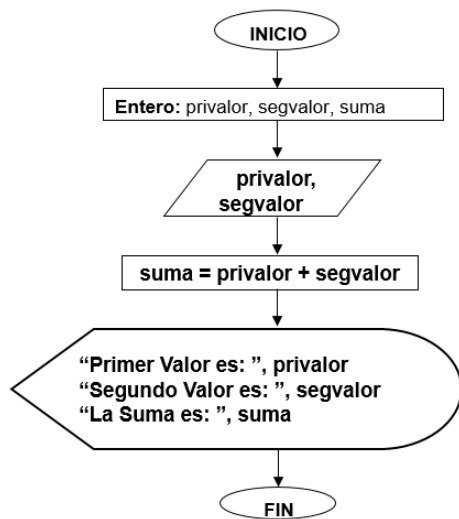
Escribir "Ingrese su nombre"

Leer nombre

2.5.7.3.5 Para hacer Cálculos Aritméticos

En pseudocódigo, por lo general los cálculos aritméticos se los utiliza mediante una asignación con los operadores aritméticos, tal como se han realizado en los problemas resueltos mediante los diagramas. Por lo tanto, no hay una instrucción exclusiva para los cálculos.

Ejemplos de Diagramas con Pseudocódigos



// Sumar dos valores ingresados

Algoritmo Problema1

Definir privalor, segvalor, suma **Como Entero**

Escribir "Ingrese primer valor"

Leer privalor

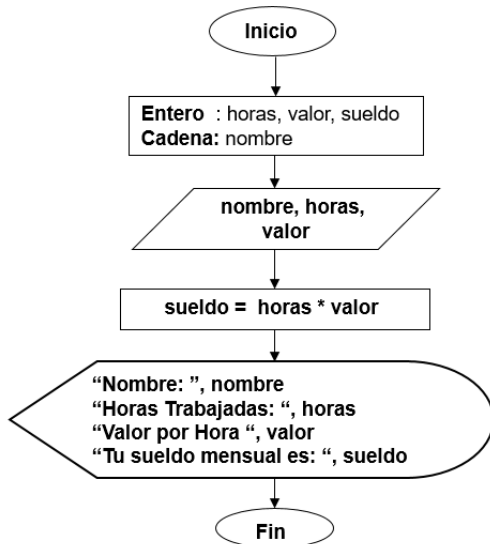
Escribir "Ingrese segundo valor"

Leer segvalor

suma = privalor + segvalor

Escribir "La suma es: ", suma

FinAlgoritmo



Calcular el sueldo de un trabajador

Algoritmo problema2

Definir horas, valor, sueldo **Como Entero**

Definir nombre **Como Cadena**

Escribir "Ingrese Nombre"

Leer nombre

Escribir "Ingrese Horas Trabajadas"

Leer horas

Escribir "Ingrese Valor por Horas de Trabajo"

Leer valor

Sueldo = valor * horas

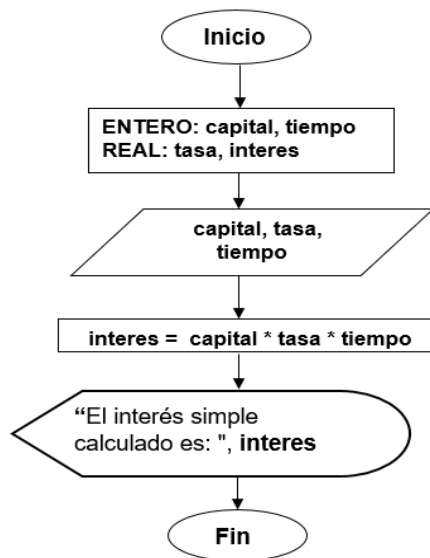
Escribir "Nombre es: ", nombre

Escribir "Horas Trabajadas: ", horas

Escribir "Valor por Horas: ", valor

Escribir "El Sueldo es: ", sueldo

FinAlgoritmo



// Calcular el sueldo de un trabajador

Algoritmo problema3

Definir capital, tiempo **Como Entero**

Definir tasa, interes **Como Real**

Escribir "Ingrese Capital"

Leer capital

Escribir "Ingrese Tasa"

Leer tasa

Escribir "Ingrese Tiempo"

Leer tiempo

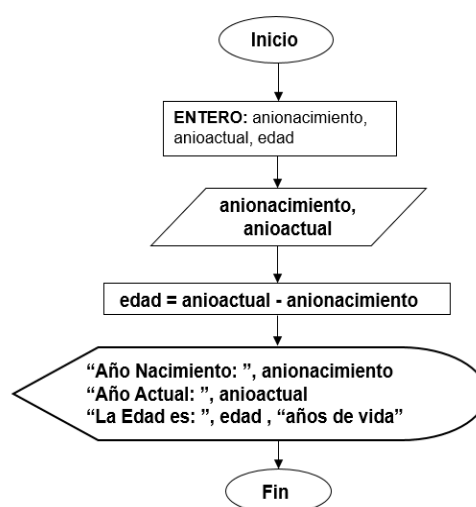
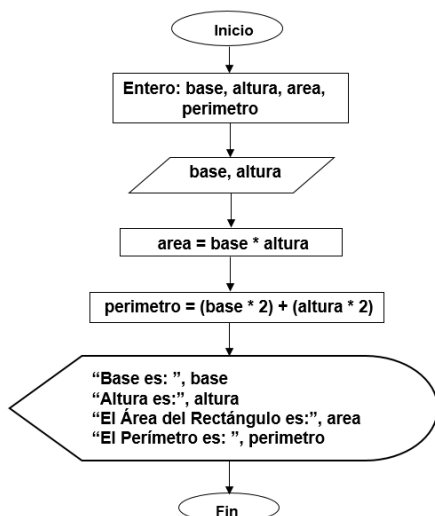
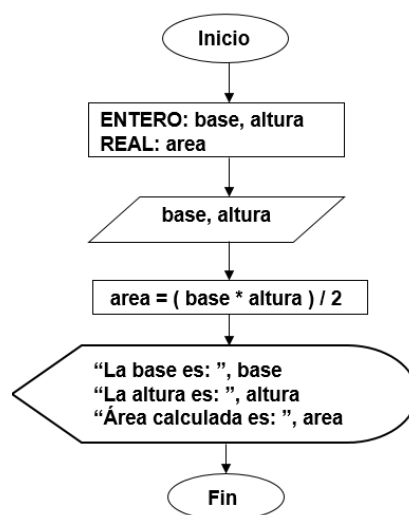
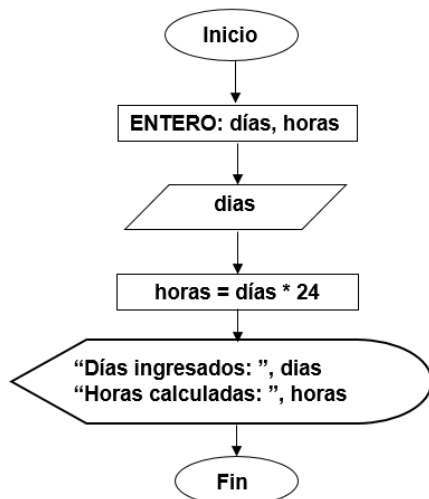
interes = capital * tasa * tiempo

Escribir "El interés simple calculado es: ", interes

FinAlgoritmo

Actividad Propuesta

Realizar los Pseudocódigos a los siguientes diagramas de flujo y problemas:



Problema 1: Dado un valor correspondiente a una distancia en kilómetros, la transforme y muestre a millas marinas. Haga el Pseudocódigo correspondiente.

Problema 2: Diseñar un Pseudocódigo que, dados cuatro valores, calcule y muestre el promedio de dichos valores.

Problema 3: Diseñar un Pseudocódigo muestre el número anterior a un valor ingresado.

Problema 4: Pseudocódigo que permita ingresar dos números y luego calcular y mostrar su cociente y su resto o residuo (haga uso del operador aritmético MOD).

Problema 5: Dado el peso de una persona en kilogramos, calcular y mostrar su peso en libras y onzas. Hacer el Pseudocódigo respectivo.

2.5.7.3.6 Para realizar Condiciones

Aunque el tratamiento y uso de condiciones serán analizadas más adelante, para su empleo se utiliza la instrucción **Si – Entonces – SiNo (Caso Contrario) – FinSi** de la siguiente forma:

Si condición **Entonces**

instrucciones a realizar si se cumple la condición

SiNo

instrucciones a realizar si NO se cumple la condición

FinSi

2.5.7.3.7 Para realizar Bucles o Repeticiones

Para bucles o procesos repetitivos que se utilizarán las siguientes instrucciones que serán motivos de explicación más adelante en este texto:

a) Ciclos PARA:

PARA variable = vinicial, vfinal, incremento

instrucciones a realizar en el ciclo

FIN PARA

b) Ciclos MIENTRAS:

MIENTRAS condición

instrucciones a realizar mientras la condición sea verdadera

FIN MIENTRAS

c) Ciclos REPETIR:

REPETIR

instrucciones a realizar hasta que la condición se cumpla

HASTA QUE condición

2.5.7.3.8 La Indentación

La indentación consiste en insertar márgenes o tabulaciones (espacios) para estructurar las diferentes partes o bloques (condiciones, bucles entre otros elementos) de un pseudocódigo y/o programa. Esta técnica es fundamental en la programación ya que permite organizar la jerarquización del código y mejorar visualmente la legibilidad y lectura de un programa.

Ejemplo:

```
Algoritmo EvaluarEdad
    Definir edad Como Entero
    Leer edad
    Si edad >= 18 Entonces
        Escribir "Es mayor de edad"
        Si edad >= 65 Entonces
            Escribir "Es adulto mayor"
        FinSi
    FinSi
FinAlgoritmo
```

2.5.8 Estructuras de Control

Las estructuras de control son mecanismos que tienen como objetivo controlar el orden de ejecución de las distintas acciones en un diagrama y/o programa. Es decir, “controlan el flujo de ejecución de las instrucciones” del algoritmo mediante secuencias, decisiones y repeticiones.

En los diagramas, al igual que todos los lenguajes de programación existen estructuras de control básicas. Estas estructuras son de tres tipos:

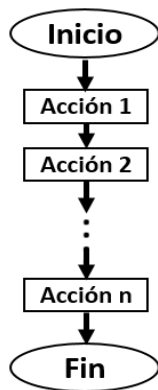
- Estructura Secuencial
- Estructuras condicionales
- Estructuras de Repetición



Hay un **teorema estructurado** que menciona: Cualquier Diagrama, Pseudocódigo y Programa, se puede **construir utilizando únicamente las tres estructuras** antes mencionadas.

2.5.8.1 Estructura Secuencial

Es la estructura de control más básica y común en programación. Se compone de acciones (procesos o instrucciones) que se deben **ejecutar en forma consecutiva**, una después de otra, siguiendo una y solo una línea de flujo. También es conocida como “estructura lineal”. La representación gráfica de la estructura secuencial en diagramación es:

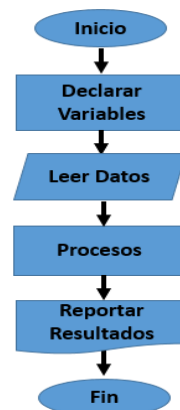


Las acciones suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. De esta manera se entiende la **secuencialidad** en la ejecución de las acciones.

2.5.8.1.1 Consideraciones al utilizar una estructura secuencial

- La estructura secuencial se emplea para resolver problemas sencillos.
- La estructura secuencial, puede contener cualquiera de los siguientes procesos y/o instrucciones, como parte de sus “acciones”:

- Declaración de Variables
- Asignación de valores
- Entrada de datos
- Procesamiento de datos
- Reporte de resultados

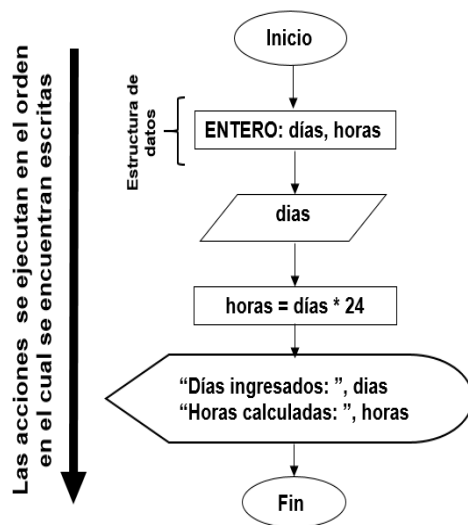


- Para construir una secuencia de acciones basta con escribir cada acción en un símbolo diferente, dependiendo del algoritmo a representar.
- Cada símbolo (excepto el óvalo) tiene siempre una sola entrada y una sola salida.

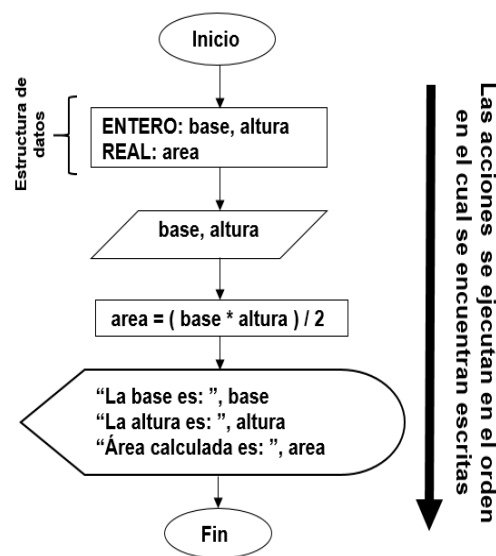
- e) Las “acciones” del diagrama, pseudocódigo o programa son ejecutadas en el orden en el cual se encuentran escritas.

Hasta ahora, todos los ejercicios planteados como actividades de ejemplos y propuestos han sido contruidos utilizando estructura secuencial. A continúan se presentan dos ejemplos de problemas sobre la secuencialidad en la ejecución de las instrucciones en algoritmos.

Problema 1: Diagrama que convierte un número de días a horas. Mostrar los dos valores, el ingresado y calculado.



Problema 2: Diagrama calcular y mostrar el área de un triángulo, conociendo su base y altura.



Problemas Propuestos

Resolver los siguientes problemas mediante diagramas de flujo, aplicando la estructura secuencial. Para esta actividad considere lo siguiente:

- Todo diagrama de flujo, como parte de su estructura, debe tener obligatoriamente:
 - La **estructura de sus datos** (declaración de los tipos de variables).
 - El **algoritmo** correspondiente a su solución.
- Todo diagrama de flujo, debe la **prueba de escritorio**. De esa forma, se verificará y comprobará su funcionalidad.

- c) Los resultados de cada diagrama de flujo, deben mostrarse con **títulos con mensajes informativos o descriptivos**, utilizando **constantes alfanuméricas**, como parte de las salidas en los resultados.

Problema 1: Diagrama que permita ingresar su nombre, edad, carrera que estudia, nivel que cursa y ciudad de origen. Luego muestre con mensajes los datos ingresados de la siguiente forma: "Hola," {nombre}. "Tienes" {edad} "años", "Tú carrera de estudio es," {carrera}, "Estás en el nivel de tu carrera:" {nivel} y "Tu ciudad procedencia es:" {ciudad}. Haga uso de constantes alfanuméricas, para las salidas solicitadas.

Problema 2: Diagrama que permita ingresar un valor. Se pide calcular y mostrar el 50% a dicho valor.

Problema 3: Diagrama que ingrese un número, disminúyalo en 30 %. Mostrar el valor de la disminución y el nuevo valor que toma el número ingresado.

Problema 4: Mediante un diagrama, ingrese el radio de un círculo, calcular y mostrar su superficie y la longitud de la circunferencia.

Problema 5: Diagrama que permita leer un número entero, calcular y mostrar el doble y el triple de ese número.

Problema 6: Escriba un algoritmo que dado un valor en centímetros lo convierta y muestre a: pulgadas y pies.

Problema 7: Crear un diagrama que permita ingresar dos valores que representan la base y altura de un rectángulo. Se solicita calcular y mostrar el área y perímetro del rectángulo.

Problema 8: Diagrama que permita solicitar un número que representa kilómetros. Se solicita transformar y mostrar dicho valor en: metros, decímetros, centímetros y milímetros.

Problema 9: Crear un diagrama que permita ingresar dos números y luego calcule y muestre su cociente y su resto o residuo (haga uso del operador aritmético MOD).

Problema 10: Crear un diagrama que convierta euros a dólares. Se debe ingresar la cantidad en euros y el diagrama debe mostrar su equivalente en dólares (consulte el cambio actual).

Problema 11: Un comerciante compra un artículo a un costo dado, el cual se debe ingresar. Determine el precio al cual debe venderlo si desea ganar el 15 %. Haga el diagrama respectivo.

Problema 12: Modifique el problema anterior, de tal manera que no se conoce el porcentaje de ganancia.

Problema 13: Un trabajador recibe mensualmente un bono por cargas familiares. Dicho bono se calcula de la siguiente forma: El 10 % del sueldo por la primera carga y un 5 % del sueldo por cada carga adicional a partir de la segunda. Realice un diagrama que previo al ingreso del sueldo y el número de cargas familiares, calcule y muestre: el valor a pagar por bono, el valor a recibir del trabajador al mes.

Problema 14: Un vendedor recibe un sueldo base más un 10 % extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.

Problema 15: La empresa EMAPAM requiere determinar el pago que debe realizar una persona por el total de metros cúbicos que consume de agua al llenar una piscina. Realice un diagrama que permita determinar ese pago, considerando que como datos de entrada se tienen: altura, largo y ancho de la piscina y el costo del metro cúbico.

Problema 16: Una persona recibe un préstamo de "X valor" de un banco y desea saber cuánto pagará de interés, conociendo que el banco le cobra una tasa del "Y por

ciento anual”. Muestre los datos ingresados y el interés calculado. Haga el diagrama respectivo.

Problema 17: El Hotel “5 Estrellas” tiene una promoción para sus clientes: cobra por una habitación \$20 el primer día, y por el resto de los días cobra \$15 diario. Realice un diagrama que determine el monto a pagar por la habitación si la estadía fue de varios días.

Problema 18: El profesor de la asignatura de Algoritmos, prepara tres cuestionarios para el examen final de los paralelos: A, B y C. Se sabe que se tarda 5 minutos en revisar cada cuestionario A, 8 minutos en revisar el cuestionario B y 6 minutos en el C. La cantidad de cuestionarios para cada paralelo se deben ingresar. Hacer el diagrama respectivo que muestre ¿Cuántas horas y cuántos minutos se tardará en revisar todas las evaluaciones?

Problema 19: En un curso de programación hay N cantidad de alumnos, de los cuales en la evaluación escrita de “Algoritmos”, se obtuvieron las siguientes notas:

- N número de notas Sobresalientes
- N número de notas Muy buenas
- N número de notas Buenas
- N número de notas Regulares

Diseñar un diagrama para calcular y mostrar dichos notas en porcentajes

Problema 20: Haga un diagrama que permita calcular y mostrar el aumento TOTAL por alquiler de tres locales comerciales. Ingresará el pago mensual de cada uno de los tres locales y aplicará un aumento del 10, 15 y 20 % respectivamente.

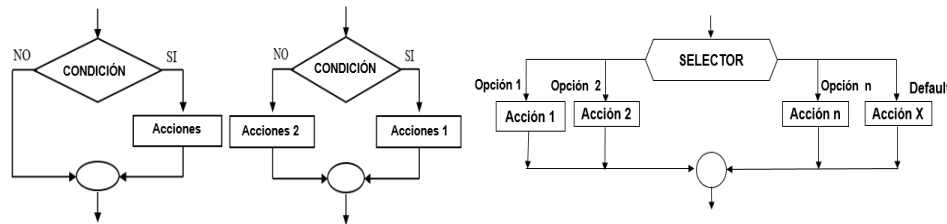
2.5.8.2 Estructuras Condicionales

2.5.8.2.1 Introducción y Definición

En Diagramación y/o Programación no todo es indicar que se hagan acciones en forma secuencial. La mayoría de las veces es necesario, según el contexto del algoritmo, dar varias opciones o alternativas, en donde se decide hacer entre “Acción

1”, “Acción 2”, ... “Acción n”, dependiendo de cierta condición. Es aquí, donde las estructuras condicionales cobran importancia, pues, todo sistema de software las utiliza, por lo que son parte fundamental de cualquier lenguaje de programación.

Las estructuras condicionales **son mecanismos de control** que permiten **decidir** qué alternativa seguirá el flujo del diagrama y/o programa dependiendo del resultado de la evaluación de una condición, en caso de cumplirse una o varias opciones posibles.



El resultado de la condición que viene dado por un valor booleano (verdadero o falso), que determina si un conjunto de acciones se ejecuta o no. Al utilizar las estructuras condicionales cobran importancia los datos booleanos y el uso de los siguientes operadores:

- Operadores de relación
- Operadores lógicos



Las estructuras condicionales, establecen el control de flujo de diagrama/programa basado en la evaluación booleana de una expresión dentro de una condición.

2.5.8.2.2 Operadores de Relación

Los operadores de relación, también conocidos como **relacionales**, son símbolos que permiten comparar valores (deben ser del mismo tipo de datos), que al determinar la relación, devuelven siempre un resultado de tipo booleano que puede ser: **verdadero (true)** o **falso (false)**.

Los operadores de relación (tabla 9), son signos utilizados para comparar valores o expresiones y determinar la relación existente en dicha comparación (mayor, menor, igual, etc.). El resultado de la comparación (verdadero o falso) se utilizará para tomar decisiones con respecto al flujo y ejecución de acciones del algoritmo.

Tabla 9

Operadores de Relación

Operador	Función (relación)
= o ==	Igualdad
<> o !=	Desigualdad o diferencia
>	Mayor que
<	Menor que
> =	Mayor o igual que
< =	Menor o igual que

Nota. Fuente: Elaboración propia



Sin los **Operadores de Relación**, los programas computacionales no podrían “tomar decisiones”.

2.5.8.2.3 Tipos de Estructuras Condicionales

Los **tipos de estructuras condicionales** son las distintas formas en que un diagrama y/o programa puede tomar decisiones según se cumpla o no una condición lógica, permitiendo controlar el flujo de ejecución de un algoritmo.

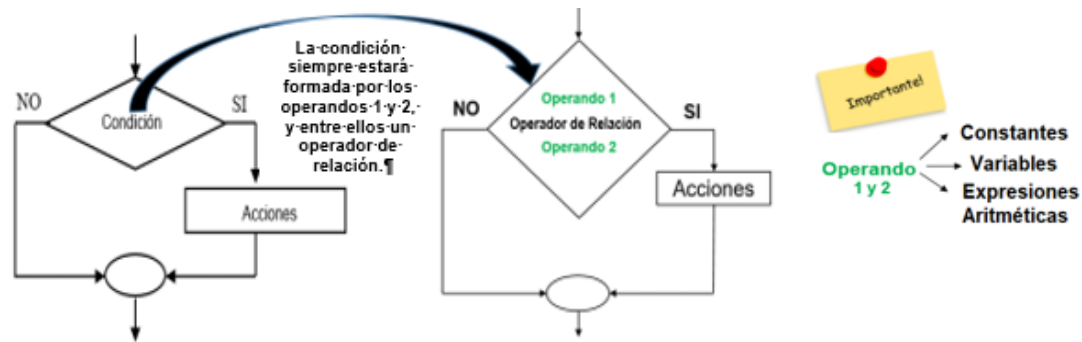
Los tipos de estructura condicionales, son las distintas formas en que un diagrama y/o programa puede comparar y tomar decisiones para determinar el flujo de control del algoritmo. Existen los siguientes tipos de estructuras condicionales:

- Condiciones simples
- Condiciones dobles
- Condiciones Anidadas
- Condiciones múltiples

2.5.8.2.3.1 Estructuras Condicionales Simples

Las estructuras condicionales simples, son aquellas que tienen dos salidas (“sí” - “no” o también “verdadero” o “falso”) y que ejecutan un bloque de acciones **SOLO**

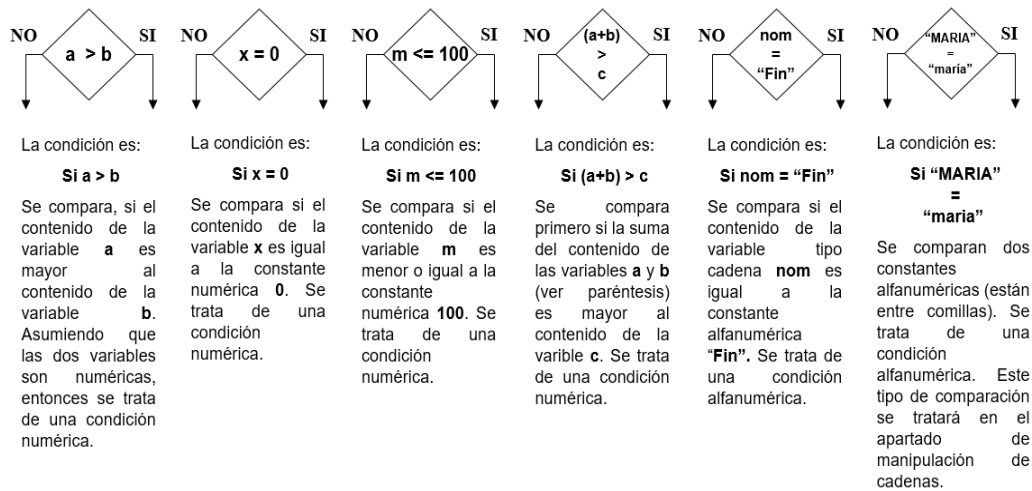
cuando el resultado de la condición es **verdadera**; si el resultado es falso, no hace nada. Este tipo de comparación, tiene la siguiente forma:



Al utilizar las Condiciones Simples, es fundamental tener en consideración los siguientes aspectos:

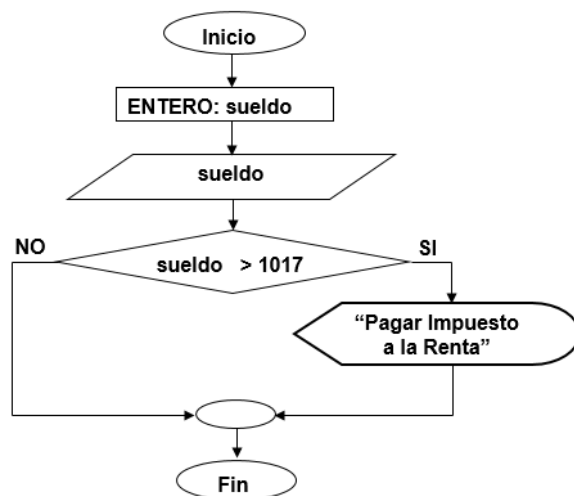
- Todos los operadores relacionales son binarios, es decir, siempre tienen dos operandos.
- Los valores de los operandos que forman la condición, deben ser homogéneos, es decir, deben ser del mismo tipo de datos.
- Los Operandos pueden ser: Constantes, Variables o Expresiones Aritméticas.
- Si los operandos son expresiones, utilizar paréntesis ().
- Si en la condición, combina operadores aritméticos y operadores de relación, la mayor precedencia (jerarquía) la tienen los operadores aritméticos.
- La evaluación de la condición, siempre se realiza de izquierda a derecha.
- Toda condición que se abre, se debe cerrar con el símbolo de reagrupamiento.

Ejemplos:



Problemas Resueltos

Problema 1: Diagrama que ingresa el sueldo mensual de una persona, si supera los \$ 1017 mostrar un mensaje en pantalla indicando que debe "Pagar Impuesto a la Renta".



PRUEBA DE ESCRITORIO

Sueldo	Salida
~~500~~	
~~1500~~	Pagar Impuesto a la Renta
~~450~~	
1050	Pagar Impuesto a la Renta

// Determinar el Pago a la Renta

Algoritmo problema1

Definir capital, tiempo **Como Entero**

Definir tasa, interes **Como Real**

Escribir "Ingrese Sueldo Mensual"

Leer sueldo

Si sueldo > 1017 **Entonces**

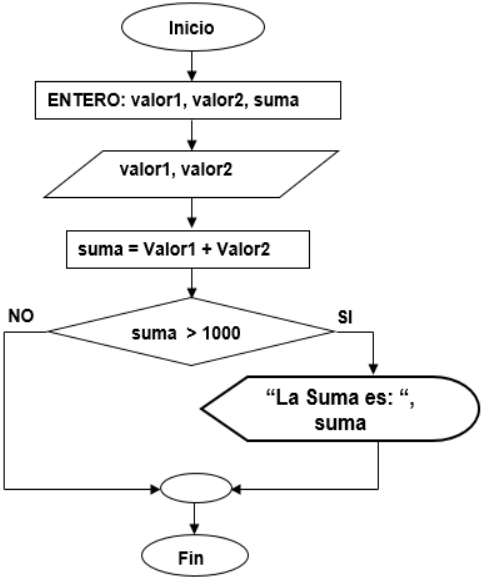
Escribir "Pagar Impuesto a la Renta"

FinSi

FinAlgoritmo

Las líneas rojas en la prueba de escritorio, indican que han procesado varios ingresos, para comprobar la efectividad del diagrama de flujo

Problema 2: Diagrama que permita el ingreso de dos valores, sumarlos y mostrar dicha suma siempre y cuando esta sea mayor a 1000.



// Determina si la suma es mayor a 1000

Algoritmo Sumavalores

Definir valor1, valor2, suma **Como Entero**

Escribir "Ingrese Primer Valor"

Leer valor1

Escribir "Ingrese Segundo Valor"

Leer valor2

suma = valor1 + valor2

Si suma > 1000 **Entonces**

Escribir "La Suma es", suma

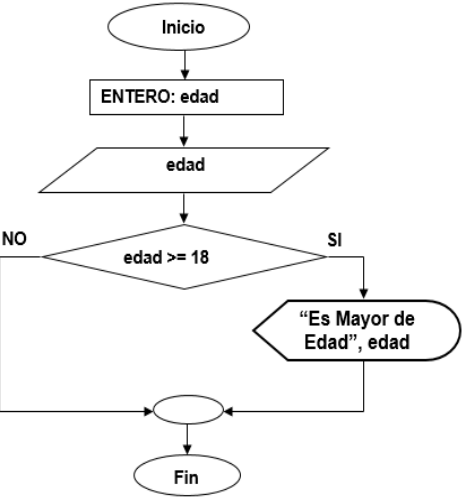
FinSi

FinAlgoritmo

PRUEBA DE ESCRITORIO

valor1	valor2	suma	Salida
~~600~~	~~500~~	1100	La Suma es 1100
~~700~~	~~800~~	1500	La Suma es 1500
700	50	750	

Problema 3: Diagrama que conociendo la edad de una persona, determina si es "Mayor de Edad". Haga el diagrama respectivo.



PRUEBA DE ESCRITORIO

Edad	Salida
~~30~~	Es Mayor de Edad 30
~~25~~	Es Mayor de Edad 25
~~18~~	Es Mayor de Edad 18
16	

// Determina si es mayor de Edad

Algoritmo MayorEdad

Definir edad **Como Entero**

Escribir "Ingrese la Edad"

Leer edad

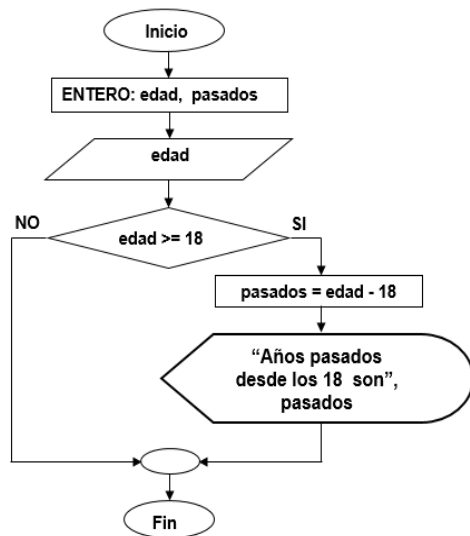
Si edad >= 18 **Entonces**

Escribir "Es Mayor de Edad", edad

FinSi

FinAlgoritmo

Problema 4: Si la edad de una persona es mayor o igual a 18 años, calcular y mostrar cuántos años han pasado desde que cumplió la mayoría de edad. Haga el diagrama respectivo.



PRUEBA DE ESCRITORIO

edad	pasados	Salida
~~30~~	~~12~~	Años pasados desde los 18 son 12
~~45~~	~~27~~	Años pasados desde los 18 son 27
~~18~~	~~0~~	Años pasados desde los 18 son 0
16		

// Determina años pasada la mayoría edad

Algoritmo AñosdeMayoríaEdad

Definir edad, pasados Como Entero

Escribir "Ingrese la Edad"

Leer edad

Si edad >= 18 Entonces

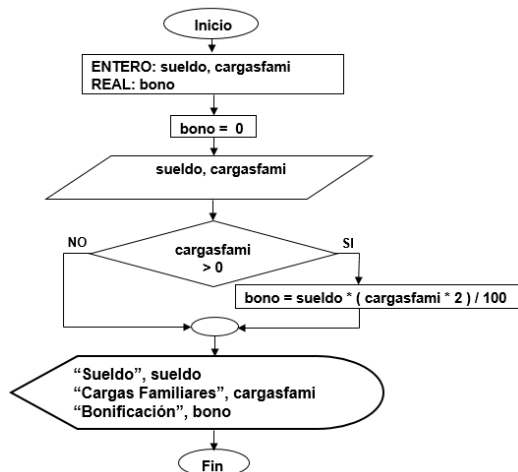
Pasados = edad - 18

Escribir "Años pasados desde los 18 son",
pasados

FinSi

FinAlgoritmo

Problema 5: Diagrama que determina si un empleado debe o no ganar un bono del 2 % de su sueldo por cada carga familiar bajo su responsabilidad. Por pantalla se muestra la información ingresada y el valor de la bonificación. En caso de no tener cargas familiares, la bonificación será de cero (0).



PRUEBA DE ESCRITORIO

sueldo	cargafami	Bono	Salida
~~600~~	~~0~~	~~0~~	Sueldo 600 Cargas Familiares 0 Bonificación 0
800	5	~~0~~ 80	Sueldo 800 Cargas Familiares 5 Bonificación 80

// Cálculo de Bonificación por Cargas Familiares

Algoritmo Bonificaciónporcargas

Definir sueldo, cargafami Como Entero

Definir bono Como Real

Bono = 0

Escribir "Ingrese Sueldo"

Leer sueldo

Escribir "Ingrese Número de Cargas Familiares"

Leer cargafami

Si cargafami > 0 Entonces

bono = sueldo * (cargafami * 2) / 100

FinSi

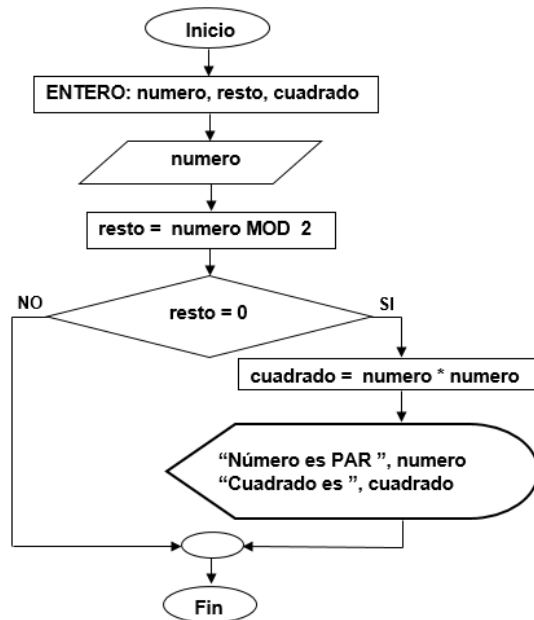
Escribir "Sueldo", sueldo

Escribir "Cargas Familiares", cargafami

Escribir "Bonificación", bono

FinAlgoritmo

Problema 6: Diagrama que permita el ingreso de un número. Verificar si dicho número es PAR. Si lo es, calcular el cuadrado del número y mostrarlo. En caso de ser IMPAR, no haga nada y finalizar.



PRUEBA DE ESCRITORIO			
numero	resto	cuadrado	Salida
6	0	36	Número es PAR 6 Cuadrado es 36
9	1		
10	0	100	Número es PAR 10 Cuadrado es 100

// Cálculo de cuadrado si número es PAR

Algoritmo NumeroPar

Definir numero, resto, cuadrado **Como** Entero

Escribir "Ingrese un Número"

Leer numero

resto = numero **MOD** 2

Si resto = 0 Entonces

Cuadrado = numero * numero

Escribir "Número es PAR, numero

Escribir "Cuadrado es ", cuadrado

FinSi

FinAlgoritmo

Problemas Propuestos

Crear los diagramas respectivos para cada problema que se plantea utilizando Estructuras Condicionales Simples. Adicionalmente realice la Prueba de Escritorio y el Pseudocódigo correspondiente.

Problema 1: Diagrama que permita calcular el descuento del 20 % que hace un almacén a sus clientes, cuya compra supere los \$1000. Muestre el valor del descuento y cuanto pagará el cliente por su compra.

Problema 2: Modificar el Diagrama anterior, de tal forma, que el porcentaje de descuento sea ingresado también.

Problema 3: Desarrolle un diagrama que permita el ingreso de dos números. Si el primero es mayor al segundo debe calcular y mostrar la media de los dos números.

Problema 4: Diagrama que ingrese dos números y muestre su división. Si el divisor es cero se debe mostrar un mensaje de ERROR.

Ejercicio 5: Diagrama que ingrese un número positivo (representa centímetros), en caso de no serlo envíe un el mensaje “error” y finalice. El diagrama debe convertir de centímetros a **kens** y **shakus**, unidades japonesas de longitud. Se recuerda que un ken son seis shakus y que un shaku equivale a 30,3 cm.

Problema 6: Diagrama que permita el ingreso de un número. Verificar si dicho número es mayor a 10. Si lo es, calcular y mostrar su cuadrado.

Ejercicio 7: Realizar un diagrama para que dados dos números enteros nos diga si el primero es divisible exactamente por el segundo o viceversa.

Problema 8: Diagrama que permita el ingreso de un número. Verificar si dicho número es IMPAR. Si lo es, calcular y mostrar su Cuadrado y su Cubo. En caso de ser PAR, no haga nada y finalizar.

Ejercicio 9: Diagrama que permita el ingreso de un número. Verificar si dicho número es PAR. Si lo es, sumarle 1 y mostrarlo. En caso de ser IMPAR, no haga nada y finalizar.

Problema 10: Desarrolle un diagrama que permita el ingreso de un número. Se solicita determinar mediante un mensaje, si dicho número es divisible exactamente para 10.

Problema 11: Si un terreno tiene más de 100 metros cuadrados, calcular y mostrar su área y perímetro. Determine los datos a ingresar y haga el diagrama respectivo.

Problema 12: Si un automóvil tiene más de 50.000 kilómetros recorridos, calcular su depreciación según una tasa del 5 % anual. Haga el diagrama respectivo.

Ejercicio 13: En un almacén se tiene una promoción que consiste en un descuento del 20 % por el consumo de tres (3) productos siempre y cuando sus costos totales sea mayor a \$ 100. Haga el diagrama respectivo que permita generar y mostrar el total por venta de los tres (3) productos, el descuento y el valor total a pagar.

Problema 14: Elaborar un diagrama para calcular el sueldo neto de un trabajador. Como datos de entrada, se tiene: Sueldo diario, Días trabajados y Días extras trabajados.

Nota: Considere que para pagar los días extras trabajados, este valor debe ser mayor a cero. Considere que el tiempo extra es pagado al doble del sueldo diario.

Problema 15: Diagrama que permita el ingreso de dos valores, que representan el numerador y denominador de una fracción. Solo si el denominador es menor que 5, calcular y mostrar la inversa de la fracción. En caso contrario finalizar.

Problema 16: Diagrama que permita calcular el número de litros de pintura, siempre y cuando la superficie de una pared sea mayor a 5 metros cuadrados. Como datos de entrada se tendrá: el ancho y alto de una pared, conociendo que la pintura a utilizar rinde 5 metros cuadrados por litro.

Problema 17: Si un paquete pesa más de 10 kilogramos, calcular el costo del envío, considerando una tarifa de \$ 2 por cada kilogramo adicional al 10. A ello agregue \$ 5. Muestre cuanto deberá cancelar.

Ejercicio 18: Diagrama, que dada la duración en minutos de una llamada calcular y mostrar su costo, considerando:

- a) Hasta tres minutos el costo es 0.50 centavos
- b) Por encima de tres minutos es 0.50 más 0.1 centavos por cada minuto adicional a los tres primeros.

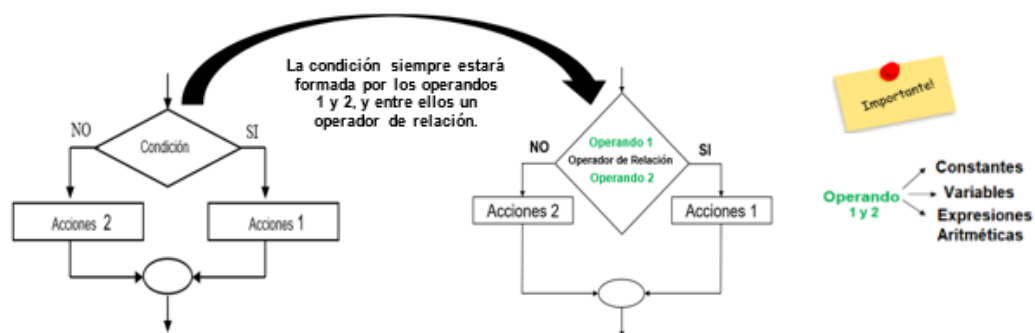
Ejercicio 19: Elaborar un diagrama que intercambie los valores entre dos variables numéricas, solo si el valor de la primera es mayor a la segunda. Muestre los

contenidos antes y después del intercambio. Asuma que se ingresarán valores diferentes.

Ejercicio 20: La empresa eléctrica desea establecer un subsidio por consumo a sus usuarios. Teniendo como datos de entrada la lectura inicial y la lectura final en kilovatios de un medidor, determine si el consumo en kilovatios es mayor a 100 para ser beneficiario del subsidio. En caso de serlo, el subsidio será del 20 % del monto a pagar por dicho consumo, conociendo que cada kilovatio tiene un valor de \$0,10. Considere que todos los usuarios pagan una tasa del 5 % por aseo urbano y un 25 % por alumbrado público. Haga el Diagrama que muestre y calcule lo establecido en el enunciado, a más del valor total que debe pagar el usuario.

2.5.8.2.3.2 Estructuras Condicionales Dobles

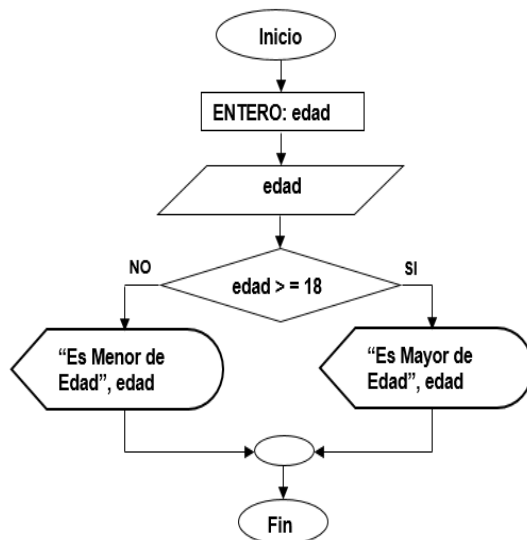
Las estructuras condicionales dobles, son aquellas condiciones que tienen dos salidas (“si” – “no” o también “verdadero” o “falso”) y que ejecutan un bloque acciones (1) cuando el resultado de la condición es verdadera y un bloque diferente de acciones (2) cuando el resultado es falso. Este tipo de comparación tiene la siguiente forma:



Cuando se utiliza este tipo de condiciones, muchas veces por cualquiera de las dos acciones, se deben realizar otras condiciones. En este caso es muy importante mencionar, que las condiciones se cierran de adentro hacia fuera, es decir, la última condición en abrirse, será la primera en cerrarse (ver problema resuelto de ejemplo número 3).

Problemas Resueltos

Problema 1: Se desea determinar conociendo la edad de una persona, si es “Mayor” o “Menor” de edad. Haga el diagrama respectivo.



PRUEBA DE ESCRITORIO

edad	Salida
~~30~~	Es Mayor de Edad 30
~~15~~	Es Menor de Edad 15
~~18~~	Es Mayor de Edad 18
16	Es Menor de Edad 16

// Determina mayoría de edad

Algoritmo Edad

Definir edad Como Entero

Escribir "Ingrese una Edad"

Leer edad

Si edad >= 18 Entonces

Escribir "Es Mayor de Edad", edad

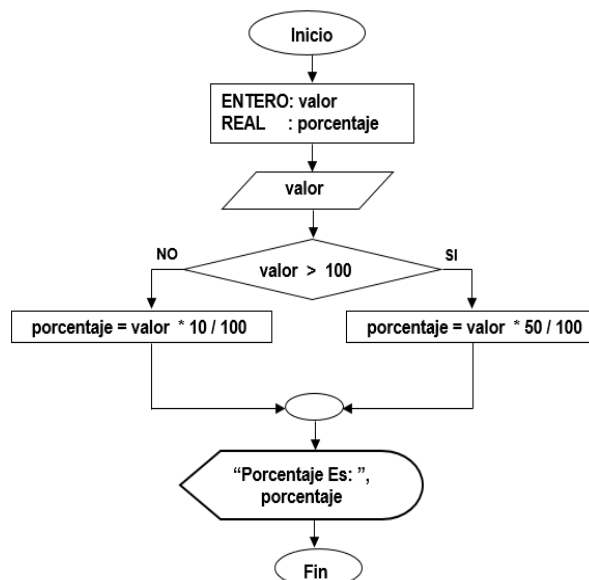
SiNo

Escribir "Es Menor de Edad", edad

FinSi

FinAlgoritmo

Problema 2: Diagrama que permita el ingreso de un valor, si el valor es mayor a 100, calcular el 50 % de este valor, caso contrario calcular 10 % al valor. Por ambos casos muestre el porcentaje calculado.



PRUEBA DE ESCRITORIO

valor	porcentaje	Salida
~~600~~	~~300~~	Porcentaje Es: 300
~~70~~	~~7~~	Porcentaje Es: 7
500	250	Porcentaje Es: 250

// Determina porcentaje de acuerdo a valor

Algoritmo Porcentajes

Definir valor Como Entero

Definir porcentaje Como Real

Escribir "Ingrese un Valor"

Leer valor

Si valor > 100 Entonces

porcentaje = valor * 50 / 100

SiNo

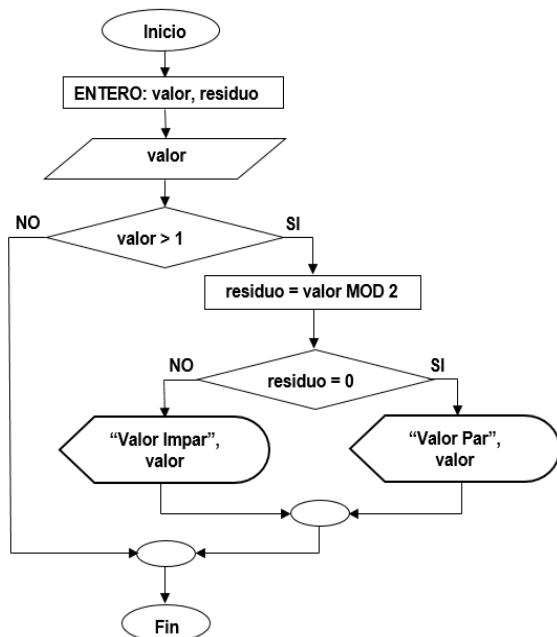
porcentaje = valor * 10 / 100

FinSi

Escribir "Porcentaje Es: ", porcentaje

FinAlgoritmo

Problema 3: Diagrama que permita leer un valor, si dicho valor es mayor 1 determine con un mensaje si el valor ingresado es “Par” o “Impar”. En caso contrario finalice el proceso.



PRUEBA DE ESCRITORIO

valor	residuo	Salida
~~6~~	~~0~~	Valor Par 6
~~15~~	~~1~~	Valor Impar 15
~~8~~	~~0~~	Valor Par 8
0		

// Determina si valor es Par o Impar

Algoritmo ParImpar

Definir valor, residuo Como Entero

Escribir "Ingrese un Valor"

Leer valor

Si valor > 1 Entonces

residuo = valor MOD 2

Si residuo = 0 Entonces

Escribir "Valor Par", valor

SiNo

Escribir "Valor Impar", valor

FinSi

FinSi

FinAlgoritmo

Problema 4: Diagrama que permita el ingreso de dos valores, si los números son iguales envíe el mensaje “Son iguales”, en caso contrario determine y muestre cuál es el “mayor” de los dos números.

PRUEBA DE ESCRITORIO

va1	va2	Salida
~~6~~	~~6~~	Valores Iguales
~~15~~	~~10~~	Mayor 15
8	20	Mayor 20

// Encuentra y muestra el mayor valor ingresado

Algoritmo MayoryMenor

Definir va1, va2 Como Entero

Escribir "Ingrese Primer Valor"

Leer va1

Escribir "Ingrese Segundo Valor"

Leer va2

Si va1 = va2 Entonces

Escribir "Valores Iguales"

SiNo

Si va1 > va2 Entonces

Escribir "Mayor ", va1

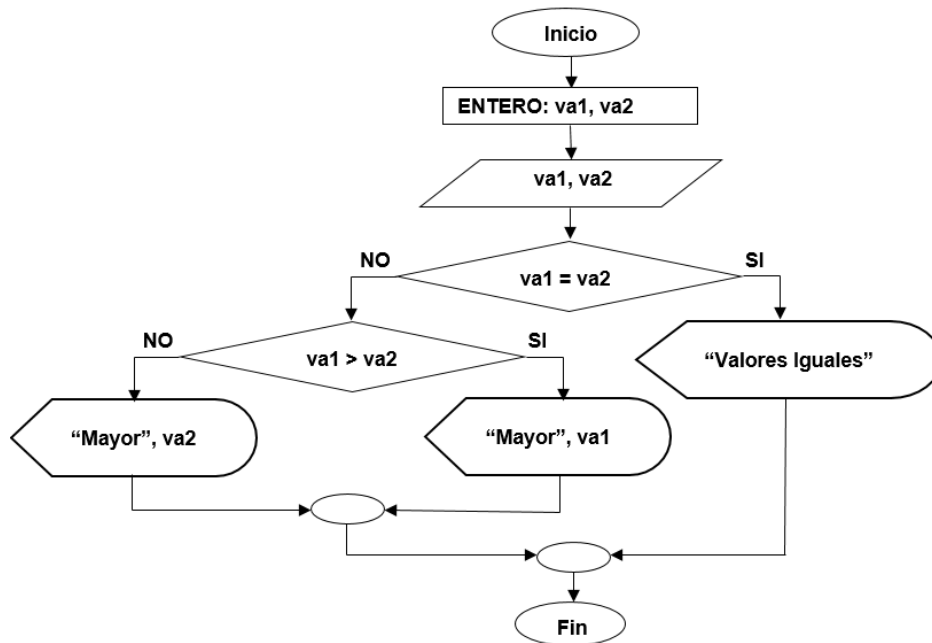
SiNo

Escribir "Mayor ", va2

FinSi

FinSi

FinAlgoritmo



Problemas Propuestos

Problema 1: Modificar el problema resuelto número 1, de tal forma que se utilice solo un símbolo de salida, para determinar la edad de la persona.

Problema 2: Diagrama que lea una clave de usuario y determine con un mensaje si es o no correcta. Considere, que la clave es correcta cuando se ingrese "123ABC".

Problema 3: Diagrama que lea una calificación y determine con un mensaje si la calificación es o no correcta. Considere, que la calificación es correcta en el rango de 1 a 10, ambos inclusive.

Problema 4: Diagrama que permita el ingreso del nombre y género de una persona. Determine con un mensaje si es hombre o mujer.

Problema 5: Diagrama que permita el ingreso de dos números y determine mediante un mensaje si son iguales o diferentes.

Problema 6: Crear un diagrama que ingrese su edad y muestre un mensaje que diga "Eres un adulto mayor" si tiene más de 65 años, y "No eres un adulto mayor" si tiene 65 años o menos.

Problema 7: Diagrama que permita el ingreso de tres (3) números, si el primero de ellos es negativo, calcular el producto de los otros dos, en caso contrario calcular y mostrar el promedio aritmético de los tres (3).

Problema 8: Diagrama que permita leer un valor, determine con un mensaje si el valor es “Positivo” o “Negativo” o igual a “Cero”.

Problema 9: Diagrama que permita leer dos valores, determine con un mensaje si los tienen o no el mismo signo (positivo o negativo).

Problema 10: Un cliente se hospeda varias noches en un hotel, el precio por noche es de 100 dólares. Si el usuario se hospeda más de 3 noches se le aplica un porcentaje de descuento del 5 % y si no se aplica un porcentaje de descuento del 0 %. Muestre el monto total que debe pagar el cliente, sabiendo que él indica cuántas noches se va a hospedar.

Ejercicio 11: En la ciudad de Manta, hay una empresa para envío de paquetes a las diferentes regiones del país. Para el costo por envío se considera:

- a) Paquetes $\leq$ a 5 kilos, el costo será de 5 dólares
- b) Por encima de este peso, se cobra 50 centavos por cada kilo adicional al quinto kilo.

El diagrama debe mostrar: La información ingresada y el costo por el envío del paquete.

Problema 12: Construir un diagrama que permita determinar si tres valores ingresados son o no un “trío pitagórico”. Un trío pitagórico se define como un conjunto de tres números, a, b y c que cumplen con la relación $a^2 + b^2 = c^2$.

Problema 13: Diagrama que calcule el total a pagar por la compra de camisas. Si se compran tres (3) camisas o más se aplica un descuento del 20 % sobre el total de la compra y si son menos de tres (3) un descuento del 10 %. Determine los datos de entrada.

Problema 14: Realice un diagrama para determinar y mostrar cuánto se debe pagar por equis cantidad de marcadores considerando que si son 100 o más el costo es de 0,50 centavos; de lo contrario, el precio es de 60 centavos.

Problema 15: Diagrama que permita ingresar dos números enteros y que calcule su división. Contemplar: Que no existe división para cero y en ese caso mostrar el mensaje de "ERROR", y si los dos números son cero desplegar un mensaje "INDETERMINACIÓN", caso contrario realizar la operación.

Problema 16: Diagrama que permita el ingreso de dos valores, si los números son iguales envíe el mensaje "Son iguales", en caso contrario, determine y muestre los valores en orden de mayor a menor.

Problema 17: Diagrama que permita el ingreso de un número, si el número es igual a cero, envíe el mensaje "Cero", en caso contrario determine y muestre si dicho número es positivo o negativo.

Problema 18: Elaborar un diagrama que intercambie los valores entre dos variables numéricas, sólo si el valor de la primera es mayor a la segunda. Muestre los contenidos antes y después del intercambio. Asuma que se ingresarán valores diferentes.

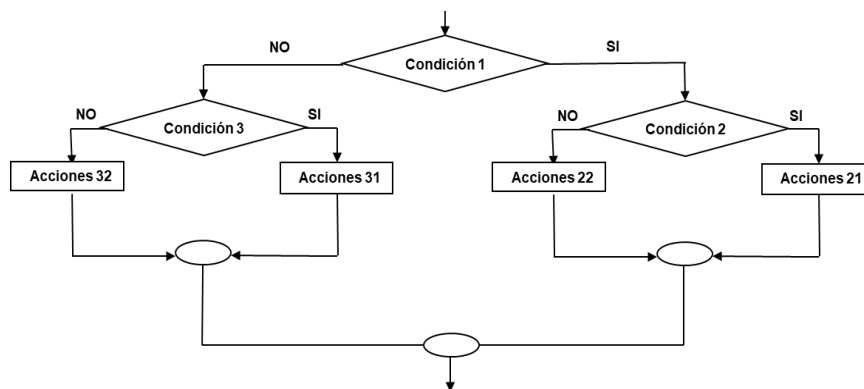
Problema 19: Diagrama que ingrese un número positivo. Si dicho número es menor a 10 o mayor a 99 enviar el mensaje "Proceso no Permitido" y finalice. En caso contrario, muestre la suma de los dígitos que conforman el número ingresado.

Problema 20: Crear un diagrama donde se ingresa dos calificaciones, correspondientes a los dos parciales de la asignatura de programación de un estudiante de informática. Determine con un mensaje si el estudiante "aprueba", "reaprueba" o se queda a "recuperación". Para aprobar la materia debe completar 14 o más puntos. Si tiene menos de 4 puntos "la reaprueba", en caso contrario va a recuperación.

2.5.8.2.3.3 Estructuras Condicionales Anidadas

Aunque técnicamente no constituyen una clasificación independiente de las estructuras condicionales, sino una forma de establecer condiciones jerarquizadas que dependen de la naturaleza del algoritmo a resolver. Considerando lo expuesto, se dice que:

Una estructura condicional es anidada cuando por las opciones del verdadero o el falso de una estructura condicional hay otra(s) estructura(s) condicional(es). Es decir, que la anidación ocurre cuando dentro de un bloque de ejecución (por sí o por no), se incorpora(n) otra(s) condición(es), que depende del resultado de primera, estructurando así un flujo lógico jerarquizado. La cantidad de condiciones dependerá de la naturaleza del problema a resolver. Gráficamente, se puede representar de la siguiente forma:



Un aspecto muy importante a considerar en este tipo de estructuración condicional, es que las condiciones se cierran de **adentro hacia fuera**, es decir, la **ÚLTIMA** condición en abrirse es la **PRIMERA** en cerrarse. Debemos considerar que la ejecución de una o más “acciones” (instrucciones o procesos) depende de la evaluación de una o más condiciones.

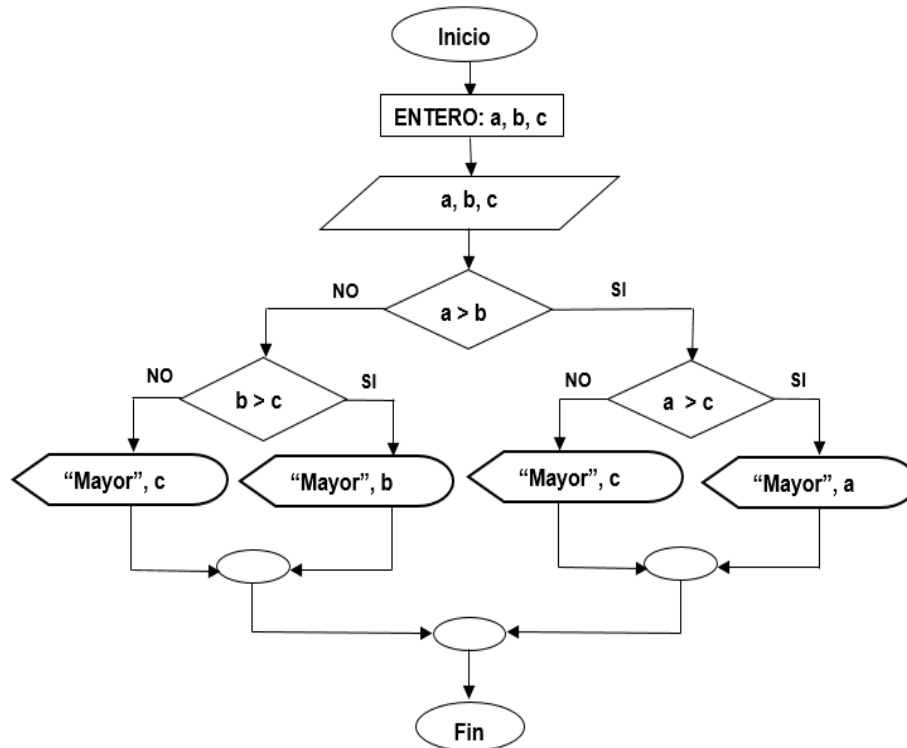
En programación, será común el uso de estructuras condicionales anidadas, por tal motivo, en algunos problemas resueltos (problemas 3 y 4) y propuestos en las estructuras condicionales dobles, ya se aplicó en forma básica el anidamiento de condiciones.



Las condiciones anidadas son especialmente útiles cuando la resolución de un problema requiere validar múltiples criterios relacionados entre sí.

Problemas Resueltos

Problema 1: Diagrama que permite leer tres valores (se usan las variables a, b y c), se busca y muestra el mayor valor ingresado. Se asume que ingresan valores diferentes.



// Encuentra y muestra el mayor valor ingresado

Algoritmo Mayorvalor

Definir a, b, c Como Entero

Escribir "Ingrese Primer Valor"

Leer a

Escribir "Ingrese Segundo Valor"

Leer b

Escribir "Ingrese Tercer Valor"

Leer c

Si $a > b$ Entonces

Si $a > c$ Entonces

Escribir "Mayor", a

SiNo

Escribir "Mayor", c

FinSi

SiNo

Si $b > c$ Entonces

Escribir "Mayor ", b

SiNo

Escribir "Mayor ", c

FinSi

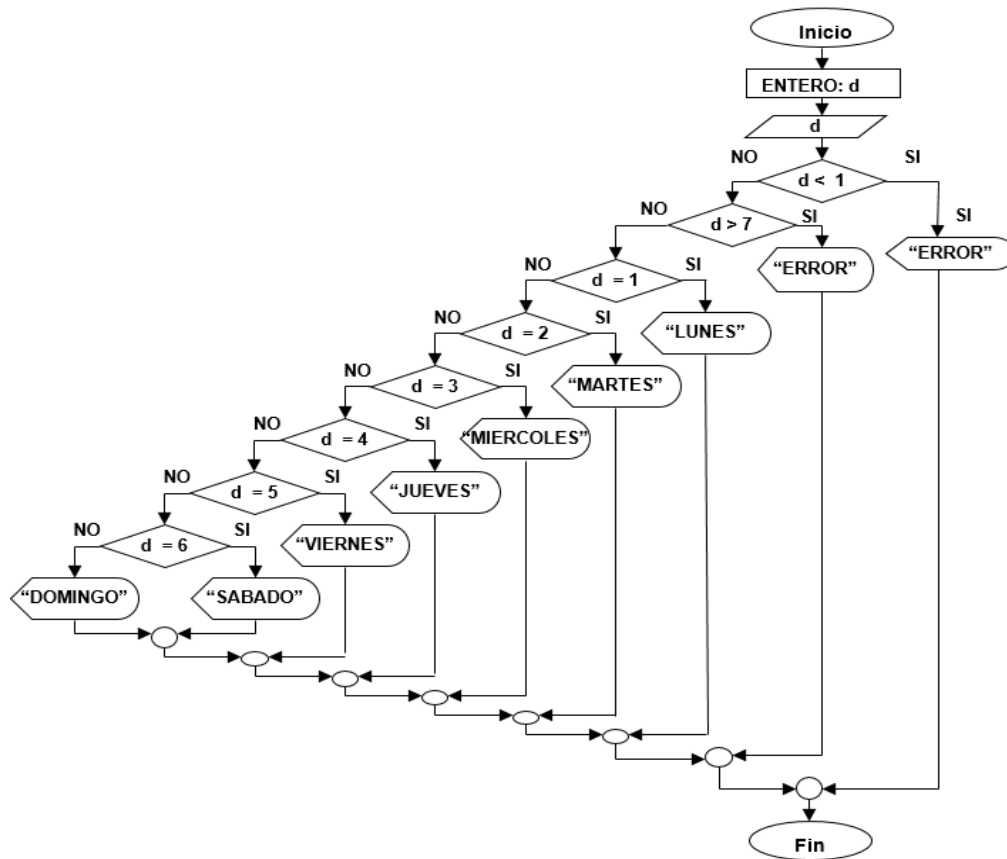
FinSi

FinAlgoritmo

PRUEBA DE ESCRITORIO

a	b	c	SALIDA
15	10	5	Mayor 15
15	12	50	Mayor 50
20	100	50	Mayor 100
20	100	500	Mayor 500

Problema 2: Diagrama que permite el ingreso de un valor que representa un día cualquiera. Si el valor es menor 1 o mayor a 7 se envía un mensaje de error, caso contrario se muestra el día de la semana a que corresponde el número ingresado.



// Muestra el día de la semana

Algoritmo Díasemana

Definir d Como Entero

Escribir "Ingresa Valor (1-7)"

Leer d

Si d < 1 Entonces

Escribir "ERROR"

SiNo

Si d > 7 Entonces

Escribir "ERROR"

SiNo

Si d = 1 Entonces

Escribir "LUNES"

SiNo

Si d = 2 Entonces

Escribir "MARTES"

SiNo

Si d = 3 Entonces

Escribir "MIÉRCOLES"

SiNo

PRUEBA DE ESCRITORIO

D	SALIDA
~~9~~	ERROR
~~0~~	ERROR
~~2~~	MARTES
~~5~~	VIERNES
~~1~~	LUNES
7	DOMINGO

```

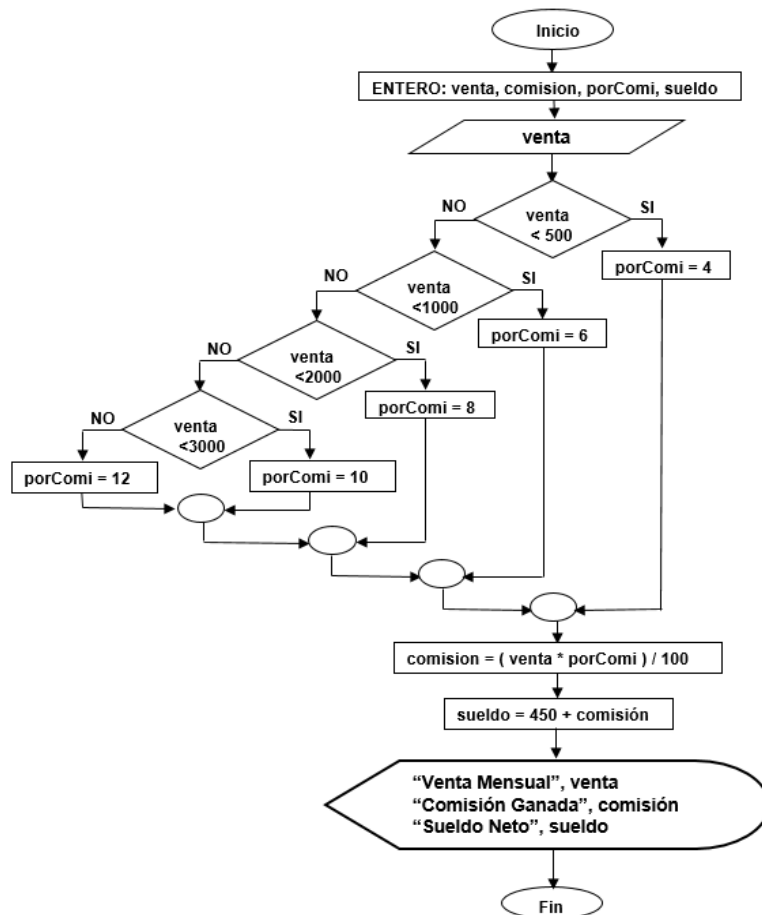
Si d = 4 Entonces
  Escribir "JUEVES"
  SiNo
    Si d = 5 Entonces
      Escribir "VIERNES"
      SiNo
        Si d = 6 Entonces
          Escribir "SÁBADO"
          SiNo
            Escribir "DOMINGO"
            FinSi
          FinSi
        FinSi
      FinSi
    FinSi
  FinSi
FinSi
FinAlgoritmo

```

Problema 3: Una empresa dedicada a la venta de productos cosméticos, paga a sus vendedores un salario básico de \$450, a lo que se suma el porcentaje de la comisión por ventas mensuales que realiza el vendedor. Para calcular dicha comisión se considera:

- Si las ventas mensuales son menores a \$500 ganará el 4 % del valor de las ventas mensuales.
- Si las ventas mensuales son menores a \$1000 ganará el 6 % del valor de las ventas mensuales.
- Si las ventas mensuales son menores a \$2000 ganará el 8 % del valor de las ventas mensuales.
- Si las ventas mensuales son menores a \$3000 ganará el 10 % del valor de las ventas mensuales.
- Si las ventas mensuales son mayores o iguales a 3000 ganará el 12 % del valor de las ventas mensuales.

Muestre las ventas mensuales del vendedor, su comisión y el sueldo neto que recibirá mensualmente.



PRUEBA DE ESCRITORIO

Venta	porComi	comisión	sueldo	SALIDA
~~400~~	~~4~~	~~16~~	~~466~~	Venta Mensual 400 Comisión Ganada 16 Sueldo Neto 466
~~900~~	~~6~~	~~54~~	~~504~~	Venta Mensual 900 Comisión Ganada 54 Sueldo Neto 504
~~3100~~	~~12~~	~~372~~	~~822~~	Venta Mensual 3100 Comisión Ganada 372 Sueldo Neto 822

// Cálculo de comisiones por ventas

Algoritmo Comisiones

Definir venta, porComi Como Entero

Definir comision, sueldo Como Real

Escribir "Ingrese Venta Mensual"

Leer venta

Si venta < 500 Entonces

porComi = 4

SiNo

Si venta < 1000 Entonces

porComi = 6

SiNo

Si venta < 2000 Entonces

porComi = 8

SiNo

```

        Si venta < 3000 Entonces
            porComi = 10
        SiNo
            porComi = 12
        FinSi
    FinSi
FinSi
comision = (venta * porComi) / 100
Sueldo = 450 + comisi3n
Escribir "Venta Mensual", venta
Escribir "Comisi3n Ganada", comision
Escribir "Sueldo Neto", sueldo
FinAlgoritmo

```

Problemas Propuestos

Ejercicio 1: Diagrama que solicite: nombre de un estudiante y 3 calificaciones. Si la suma de estas calificaciones es mayor o igual a 21 desplegar el mensaje de “APROBADO”. Si la suma de estas calificaciones es mayor o igual a 14 desplegar el mensaje de “RECUPERACI3N”. Si la suma de estas calificaciones es menor a 14 desplegar el mensaje de “REPROB3”.

Ejercicio 2: Diagrama que permita calcular lo que hay que pagar a un trabajador teniendo en cuenta su sueldo y las horas extras trabajadas. Para el pago de horas extras se tiene en cuenta adem3s la categor3a del trabajador.

Categor3a Precio hora extra

1	\$ 10
2	\$ 9
3	\$ 8
4	\$ 7

Nota: Cada trabajador puede tener como m3ximo 30 horas extras, si tienen m3s s3lo se les pagar3n 30. A los trabajadores con categor3a mayor a 4 no se paga horas extras.

Ejercicio 3: Diagrama que permita el ingreso de dos n3meros y un operador aritm3tico (tipo car3cter, que puede ser +, -, * y /) y muestre el resultado de la

operación, según el operador ingresado. En caso de ingresar un operador diferente a los indicados envíe el mensaje “Operación no Permitida” y finalice.

Ejercicio 4: Crear un diagrama que permita mostrar cuánto cobra de cuota un club deportivo a sus socios. Se deberá ingresar la edad de un socio, y se debe mostrar cuanto debe pagar de acuerdo lo siguiente:

- a) \$ 10 a los socios junior (menores de 15 años)
- b) \$ 15 a los socios juveniles (menores de 21 años)
- c) \$ 20 a los socios activos (menores a 50 años)
- d) Gratis para los mayores de 50 años.

Ejercicio 5: En la ciudad de Manta, hay una empresa para envío de paquetes a las diferentes regiones del país. Para el costo por envío se considera:

- a) Paquetes $\leq$ a 5 kilos, el costo será de 5 dólares
- b) Por encima de este peso, se cobra 50 centavos por cada kilo adicional al quinto kilo.
- c) Dependiendo de la región destino se tendrá un costo adicional: Costa (2 dólares), Sierra (3 dólares), Oriente (5 dólares) e Insular (15 dólares).

El diagrama debe mostrar: La información ingresada (Peso del paquete y Región) y el costo por el envío del paquete.

Ejercicio 6: Diagrama que ingrese un número. Si dicho número es menor a 10 o mayor a 99 enviar el mensaje “Proceso no Permitido” y finalice. En caso contrario, muestre los números en letras por separado.

Ejemplo: 25 Debe mostrar: Dos Cinco

Ejemplo: 78 Debe mostrar: Siete Ocho

Ejercicio 7: En un almacén se efectúa una promoción, la cual consiste en hacer un descuento sobre el valor total de la compra, según el color de una “bolita” que el cliente saca de un ánfora. Si la bolita es de color **blanco**, no se hará descuento alguno. Si es **verde** se hará un descuento del 2 %. Si es **amarilla** se hará un descuento del 3 %. Si es **azul** se hará un descuento del 4 %. Si es **roja** se hará un descuento del 5 %.

Calcular y mostrar, la cantidad final que el cliente deberá pagar por su compra. Haga el diagrama respectivo.

Ejercicio 8: Diagrama que ingrese un número. Si dicho número es menor a 10 o mayor a 99 enviar el mensaje “Proceso no Permitido” y finalice. En caso contrario, muestre la suma de los dígitos que conforman el número ingresado.

Ejemplo: 25 Debe mostrar: 7

Ejemplo: 78 Debe mostrar: 15

Ejercicio 9: Diagrama que ingrese un valor. Si es igual a cero, envíe el mensaje “cero” y finalice. En caso contrario determine si es “par” positivo o negativo y si es “impar” positivo o negativo.

Ejercicio 10: Diagrama que ingrese dos números positivos. Si el primero número es mayor al segundo, seleccione su último dígito y súmelo al segundo número, en caso contrario tome el primer dígito del segundo número y súmelo al primero número. Por ambos casos, se deben mostrar los nuevos números.

Nota: En caso de que uno o los dos números sean menores a 10 enviar el mensaje “Proceso no Permitido” y finalice.

Ejercicio 11: Diagrama que ingrese tres valores que representan los lados de un triángulo. Determine mediante un mensaje si forman un triángulo equilátero, isósceles o escaleno.

Ejercicio 12: Crear un diagrama que ingrese tres (3) números y los muestre ordenados de forma ascendente.

Ejercicio 13: Diseñe un diagrama que muestre los días de la semana luego de ingresar en una variable “las dos primeras letras del día”. Asuma que la primera letra es ingresada en “mayúscula” y la segunda “minúscula”. En caso de ingresar letras diferentes envíe el mensaje “Día no existe”.

Ejercicio 14: Diagrama que permita leer tres valores (use las variables A, B y C), de tal manera que se encuentre muestre el valor medio ingresado. Ejemplos:

A	B	C	
20	100	5	deberá mostrar 20
4	10	50	deberá mostrar 10
500	800	700	deberá mostrar 700

Nota: Asuma que ingresará valores diferentes en las variables.

Ejercicio 15: Diagrama que permita leer tres valores (use las variables A, B y C), de tal manera que se encuentre muestre el menor valor ingresado.

Ejercicio 16: Diagrama que permita leer un valor de temperatura en grados Celsius (°C). Se pide enviar un mensaje indicando si la temperatura es fría (menor a 15 °C), templado (entre 15 °C y 25 °C) y calor (mayor a 25 °C).

Ejercicio 17: Diagrama que permita calcular y mostrar la depreciación anual de un artículo previamente ingresado. Dicha depreciación se determina dividiendo el valor de compra del artículo y el número de años de vida útil. La vida de utilidad se determina en base a la clase de artículo. Considere los siguientes artículos:

- Computadora → 3 años
- Vehículo → 5 años
- Maquinaria → 8 años
- Mobiliario → 10 años
- Casa → 25 años

Ejercicio 18: Diagrama que permita el ingreso un valor (asuma que es entero). Si el valor es menor 1 enviar un mensaje de “error” y finalice, Si es mayor 10000 también enviar un mensaje de “error” y finalice. Caso contrario muestre cuantos dígitos tiene el valor ingresado.

Ejemplo: 4 → muestra “Un dígito”

Ejemplo: 70 → muestra “Dos dígitos”

Ejemplo: 350 → muestra “Tres dígitos”

Ejercicio 19: Diagrama que permita el ingreso un valor (asuma que es entero) y que representa el día de un año “no bisiesto”. Si el valor es menor 1 enviar un mensaje de “error” y finalice, Si es mayor 365 también enviar un mensaje de “error” y finalice. Caso contrario determine y muestre a que “Mes” y a que “Día” de ese mes, corresponde el valor ingresado.

Ejemplo: 15 → muestra: Mes de Enero, día 15

Ejemplo: 40 → muestra: Mes de Febrero, día 9

Ejemplo: 65 → muestra: Mes de Marzo, día 6

Ejemplo: 359 → muestra: Mes de Diciembre, día 25

Ejercicio 20: Diagrama que permita el ingreso de dos valores que representan Mes y Día respectivamente de un año “no bisiesto”. Se solicita determinar a qué día del año corresponde la fecha ingresada.

Ejemplo: Mes 1, día 15 Deberá mostrar: Día del año: 15

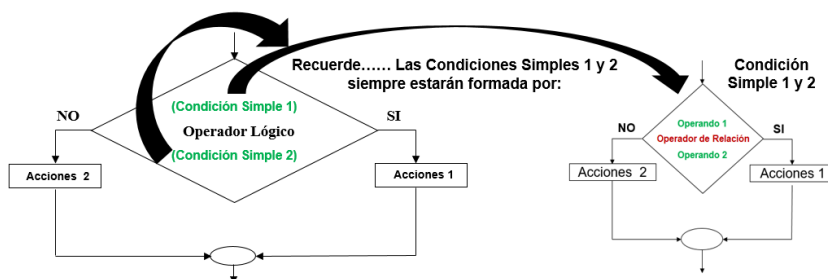
Ejemplo: Mes 2, día 9 Deberá mostrar: Día del año: 40

Ejemplo: Mes 3, día 6 Deberá mostrar: Día del año: 65

Ejemplo: Mes 12, día 25 Deberá mostrar: Día del año: 359

2.5.8.2.3.4 Estructuras Condicionales Dobles - Compuestas

Son aquellas en las cuales pueden tomarse dos caminos (“si” – “no” o también “verdadero” o “falso”), según se cumplan **dos o más condiciones simples**, es decir, están formadas por la unión lógica de dos o más condiciones simples. Dicha unión se realiza mediante los operadores lógicos. El esquema de este tipo de condición es el siguiente:



- La ejecución de las “acciones” (instrucciones o procesos) **depende de la evaluación de las condiciones simples**.
- Se recomienda que las **Condiciones Simples vayan encerradas entre PARENTESIS**, para una mejor lectura de la pregunta.

2.5.8.2.3.5 Operadores Lógicos

En programación, se utilizan para **evaluar más de una condición al mismo tiempo** y tomar decisiones en función del resultado de ellas. Es decir, que se usan habitualmente para conectar dos o más condiciones simples y retornan como resultado un valor booleano (“verdadero” o “falso”), que será utilizado en la decisión para ejecutar determinadas acciones. Se clasifican en:

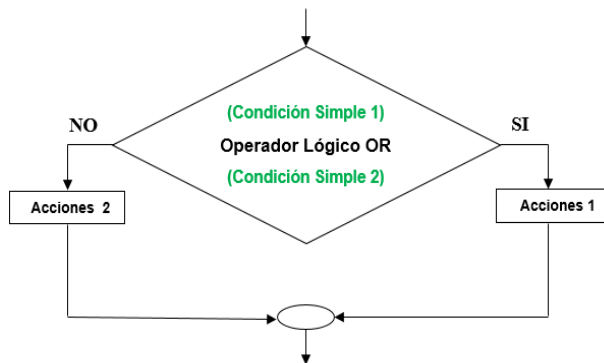
- OR
- AND
- NOT

2.5.8.2.3.5.1 Operador lógico OR

Conecta lógicamente **dos o más Condiciones Simples** y retorna como resultado un valor booleano (verdadero o falso) que será utilizado en una decisión o pregunta, para ejecutar Acciones 1 o Acciones 2.

Funciona de la siguiente manera:

"Si se cumple **UNA O TODAS LAS CONDICIONES SIMPLES** dentro de una pregunta, el resultado de la misma será **VERDADERO**, si no se cumple **NINGUNA CONDICIÓN SIMPLE** la pregunta es **FALSA**".



Para utilizar el operador lógico **OR** se aplica una tabla de la verdad (tabla 10) que muestra cómo se comporta el operador al combinar todos los posibles resultados (verdadero y falso) con dos condiciones simples, facilitando así la comprensión de su funcionamiento dentro de expresiones condicionales.

Tabla 10

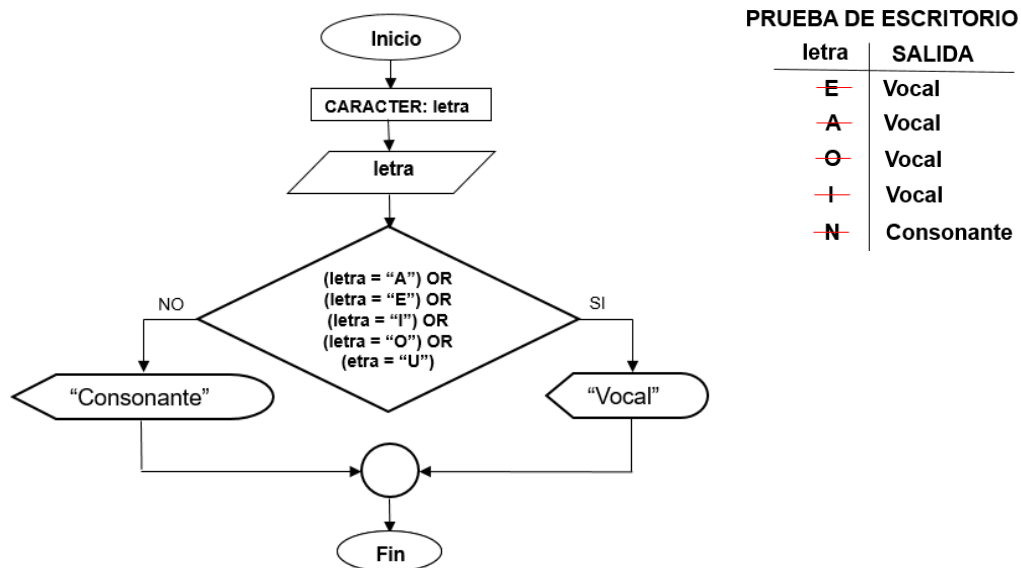
Tabla de la verdad del operador lógico OR

Condición 1	Condición 2	Resultado con OR
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Falso	Verdadero	Verdadero
Falso	Falso	Falso

Nota. Fuente: Elaboración propia

Problemas Resueltos

Problema 1: Diagrama que permita ingresar un carácter en mayúscula en una variable tipo cadena. Determine con un mensaje si el carácter ingresado es “vocal” o “consonante”.



// Determina si es vocal o consonante

Algoritmo OperadorOR

Definir letra Como Caracter

Escribir "Ingresa un Carácter en Mayúscula"

Leer letra

Si (letra = "A") OR (letra = "E") OR (letra = "I") OR (letra = "O") OR (letra = "U") Entonces
Escribir "Vocal"

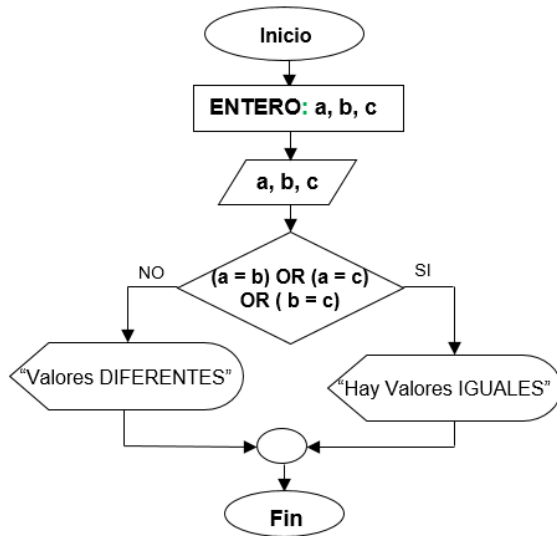
SiNo

Escribir "Consonante"

FinSi

FinAlgoritmo

Problema 2: Diagrama que permite ingresar tres valores, se solicita enviar un mensaje indicando, si entre ellos hay valores “iguales” o son valores “diferentes”.



PRUEBA DE ESCRITORIO

a	b	c	SALIDA
~~10~~	~~20~~	~~10~~	~~Hay Valores IGUALES~~
~~20~~	~~10~~	~~10~~	~~Hay Valores IGUALES~~
~~10~~	~~10~~	~~20~~	~~Hay Valores IGUALES~~
~~10~~	~~10~~	~~10~~	~~Hay Valores IGUALES~~
10	20	30	Valores DIFERENTES

// Determina si hay valores iguales o diferentes

Algoritmo OperadorOR

Definir a, b, c Como Entero

Escribir "Ingrese Primer Valor"

Leer a

Escribir "Ingrese Segundo Valor"

Leer b

Escribir "Ingrese Tercer Valor"

Leer c

Si (a = b) OR (a = c) OR (b = c) Entonces

Escribir "Hay Valores IGUALES"

SiNo

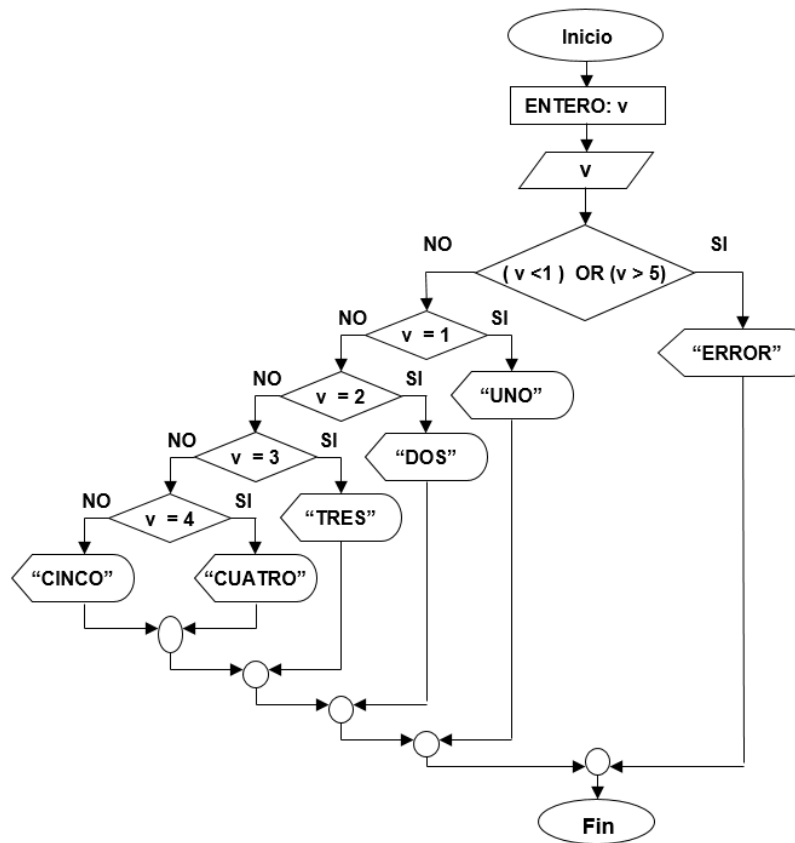
Escribir "Valores DIFERENTES"

FinSi

FinAlgoritmo

Problema 3: Diagrama que permite ingresar un valor, si es menor a uno o mayor a cinco enviar un mensaje de error, caso contrario muestre el valor ingresado en letras.

Nota: En este problema se combinan condiciones compuestas y condiciones simples.



PRUEBA DE ESCRITORIO

v	SALIDA
~~6~~	ERROR
~~0~~	ERROR
2	DOS
~~5~~	CINCO
3	TRES

// Números de 1 a 5 en letras

Algoritmo OperadorOR

Definir v Como Entero

Escribir "Ingrese Valor (1-5)"

Leer v

Si (v < 1) OR (v > 5) **Entonces**

Escribir "ERROR"

SiNo

Si v = 1 **Entonces**

Escribir "UNO"

SiNo

Si v = 2 **Entonces**

Escribir "DOS"

SiNo

Si v = 3 **Entonces**

Escribir "TRES"

SiNo

Si v = 4 **Entonces**

Escribir "CUATRO"

SiNo

Escribir "CINCO"

FinSi

FinSi

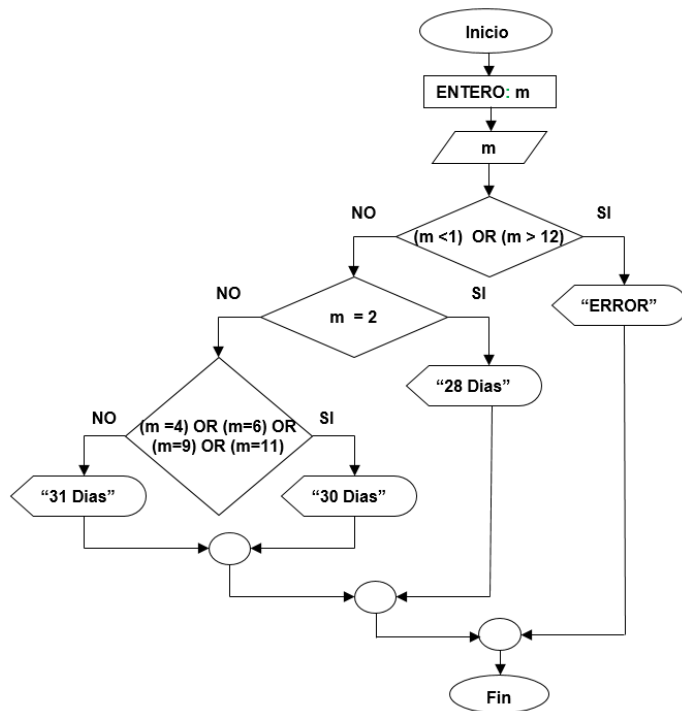
FinSi

FinSi

FinSi

FinAlgoritmo

Problema 4: Diagrama que permite el ingreso un valor que representa un MES cualquiera de un año No Bisiesto. Si el valor es menor a 1 o mayor a 12 enviar un mensaje de ERROR y finalizar, caso contrario muestre cuántos días tiene el mes ingresado.



PRUEBA DE ESCRITORIO

m	SALIDA
~~-6~~	30 dias
~~0~~	ERROR
~~-2~~	28 dias
~~-5~~	31 dias
11	30 dias

// Determina cuantos días tiene un mes

Algoritmo OperadorOR

Definir m **Como** Entero

Escribir "Ingrese Número de Mes (1-12)"

Leer m

Si (m < 1) **OR** (m > 12) **Entonces**

Escribir "ERROR"

SiNo

Si m = 2 **Entonces**

Escribir "28 Días"

SiNo

Si (m = 4) **OR** (m = 6) **OR** (m = 9) **OR** (m = 11) **Entonces**

Escribir "30 Días"

SiNo

Escribir "31 Días"

FinSi

FinSi

FinSi

FinAlgoritmo

Problemas Propuestos

Problema 1: En una cooperativa de transporte, una persona puede acceder al descuento del 50% del costo del pasaje, si cumple al menos una condición:

- Tiene 65 años o más...
- posee un carnet de discapacidad.

El diagrama debe determinar los datos a ingresar y si el usuario recibe el descuento y a cuanto equivale el mismo.

Problema 2: En una tienda para venta de artículos de mascotas, se ofrece un descuento especial del 10 % de la compra a los clientes que cumplan al menos uno de estos requisitos:

- Realizar una compra mayor a \$50...
- tener tarjeta de cliente frecuente de la tienda.

El diagrama debe determinar los datos a ingresar y si el cliente recibe el descuento y a cuanto equivale el mismo.

Problema 3: Diagrama que permita determinar con un mensaje, si un cliente puede o no acceder a un crédito. Para ello debe considerar que tenga un sueldo mayor o igual a \$482 **o si tiene un garante**. Adicionalmente debe considerar:

- Si accede por el sueldo, no debe tener deudas mayores al 40 % de este salario.
- Si accede por garante, el historial crediticio no debe ser "Malo".

Determine que datos deben ser ingresados y enviar el mensaje solicitado en base a lo establecido en el enunciado.

Problema 4: En un centro recreacional, hay una piscina donde solo puede ingresar una persona que cumplan al menos una de las siguientes condiciones:

- Tener 15 años o más...
- estar acompañadas por un adulto responsable...
- que sepa nadar.

El diagrama debe determinar con un mensaje si una persona puede o ingresar a la piscina y adicionalmente los datos que deben ser ingresados.

Problema 5: Crear un diagrama que determine si un número es considerado “especial”. Un número será especial si:

- Es múltiplo de 3...
- es múltiplo de 5.

Problema 6: Diagrama que permita ingresar tres números enteros y mostrar "Al menos uno de los números es par" si alguno de los números es par.

Problema 7: Diagrama que permita la lectura de un valor (representa un mes). Si dicho valor es menor < 1 o > 12 envíe un mensaje de “Error”, caso contrario determine a que trimestre del año, corresponde el mes ingresado.

Ejemplo: Si el valor ingresado es 2: Corresponde al “Primer Trimestre del Año”

Ejemplo: Si el valor ingresado es 11: Corresponde al “Cuarto Trimestre del Año”

Problema 8: Diagrama que lea dos calificaciones correspondientes a las notas en los dos parciales de un estudiante en la asignatura de ALGORITMOS de una universidad. Las calificaciones no deben ser < 1 ni > 10 , de ser así envíe un mensaje de “ERROR” y finalice. En caso contrario determine con un mensaje si dicho estudiante “Aprueba la asignatura” (suma de notas mayores o iguales a 14) o “Pierde la asignatura” (suma de notas menores a 4) o “Va a recuperación en la asignatura”.

Problema 9: Diagrama que le solicite al usuario el ingreso de un valor. Si dicho valor es menor a 10 o mayor a 99, envíe el mensaje “error” y finalice. En caso contrario, determine:

- Si los dos dígitos del valor son iguales, muestre el mensaje: “valores iguales”.
- Si los dos dígitos del valor son diferentes, muestre el mayor dígito.

Nota: Haga uso del operador aritmético MOD para obtener los dígitos del valor ingresado.

Problema 10: Diagrama que le solicite al usuario el ingreso de una cantidad. Si dicha cantidad es menor a 1 o mayor a 99999, envíe el mensaje “error” y finalice. En caso

contrario, determinar cuántos dígitos que posee la cantidad y a partir de esto mostrar también si pertenece a alguna de estas denominaciones:

1: unidad

10: decena

100: centena

1000: unidades de miles

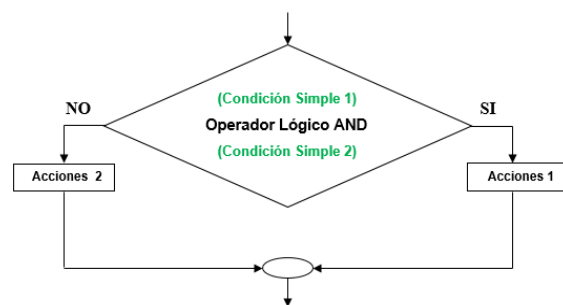
10000: decenas de miles

2.5.8.2.3.5.2 Operador lógico AND

Conecta lógicamente **dos o más Condiciones Simples** y retorna como resultado un valor booleano (verdadero o falso) que será utilizado en una decisión o pregunta, para ejecutar Acciones 1 o Acciones 2.

Funciona de la siguiente manera:

"Si se cumplen **TODAS LAS CONDICIONES SIMPLES** dentro de la pregunta, el resultado de la misma será **VERDADERA**, en caso contrario, al no cumplirse **una o todas** las condiciones, el resultado de la pregunta será **FALSA**".



Para utilizar el operador lógico **AND** se aplica una tabla de la verdad (tabla 11) que muestra cómo se comporta el operador al combinar todos los posibles resultados (verdadero y falso) con dos condiciones simples, facilitando así la comprensión de su funcionamiento dentro de expresiones condicionales.

Tabla 11

Tabla de la verdad del operador lógico AND

Condición 1	Condición 2	Resultado con AND
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso
Falso	Verdadero	Falso
Falso	Falso	Falso

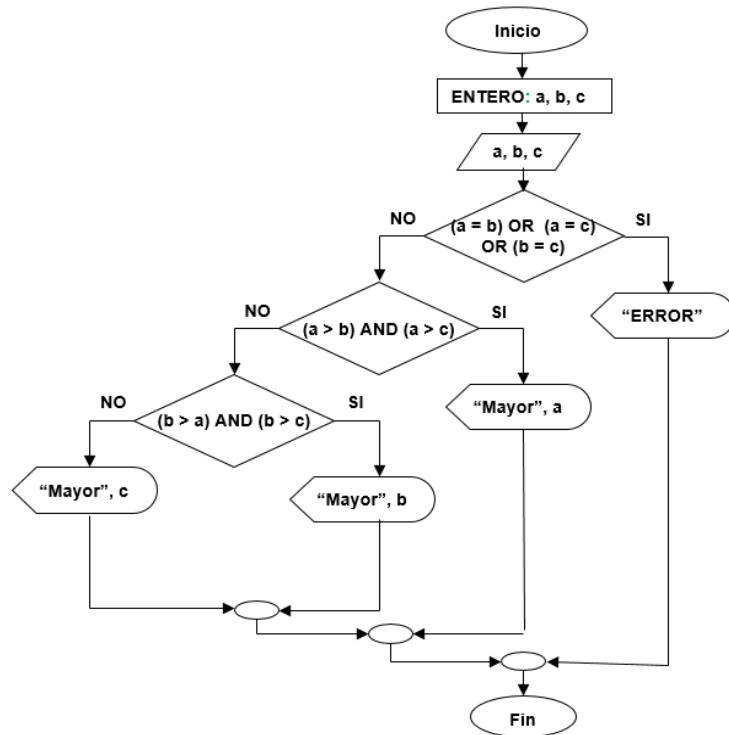
Nota. Fuente: Elaboración propia

Problemas Resueltos

Problema 1: Diagrama que permite el ingreso de tres valores diferentes, encuentra y muestra cual es el mayor valor ingresado. Envía un mensaje de error en el caso de ingresar valores iguales.

PRUEBA DE ESCRITORIO

a	b	c	SALIDA
~~5~~	~~5~~	~~3~~	ERROR
~~8~~	~~4~~	~~8~~	ERROR
~~7~~	~~2~~	~~2~~	ERROR
~~9~~	~~6~~	~~2~~	Mayor 9
~~3~~	~~8~~	~~4~~	Mayor 8
7	1	9	Mayor 9



// Determina el mayor valor ingresado

Algoritmo OperadorAND

Definir a, b, c **Como** Entero

Escribir "Ingrese Primer Valor"

Leer a

Escribir "Ingrese Segundo Valor"

Leer b

Escribir "Ingrese Tercer Valor"

Leer c

Si (a = b) **OR** (a = c) **OR** (b = c) **Entonces**

Escribir "ERROR"

SiNo

Si (a > b) **AND** (a > c) **Entonces**

Escribir "Mayor", a

SiNo

Si (b > a) **AND** (b > c) **Entonces**

Escribir "Mayor", b

SiNo

Escribir "Mayor", c

FinSi

FinSi

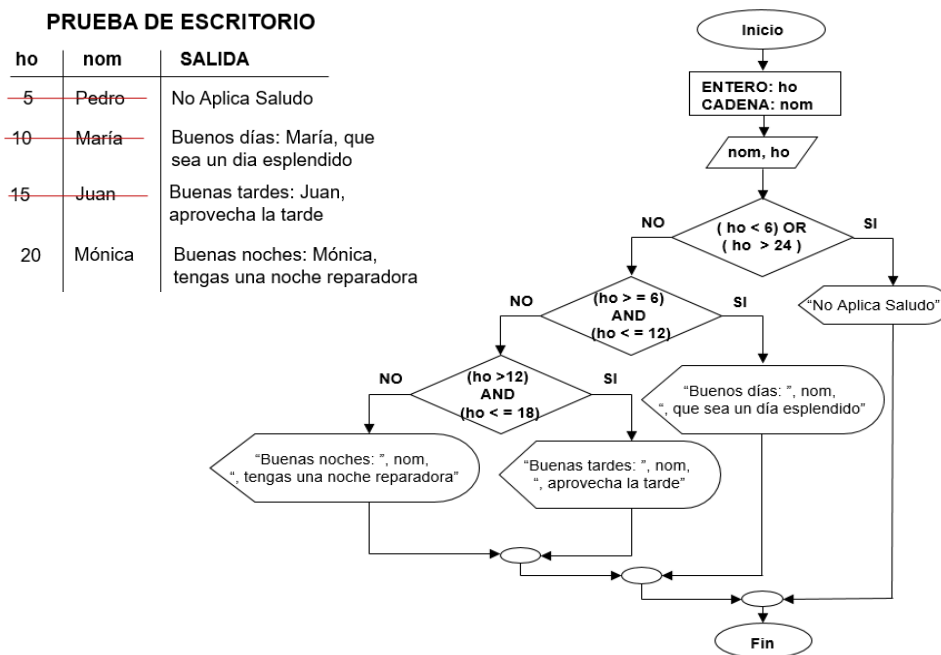
FinSi

FinAlgoritmo

Problema 2: Diagrama que permita ingresar un nombre y un valor entero que representa una hora (formato de 24 horas). Si la hora es menor a 6 o mayor a 24, envíe el mensaje “No Aplica saludo”, caso contrario desplegará un saludo, dependiendo de la hora ingresada. Para ello, el mensaje debe considerar:

- a) 06:00 a 12:00 Buenos días: *Nombre*, que sea un día espléndido.
- b) 12:01 a 18:00 Buenas tardes: *Nombre*, aprovecha la tarde.
- c) 18:01 a 23:59 Buenas noches: *Nombre*, tengas una noche reparadora.

Ejemplo: Nombre: María Hora: 10 Buenos días María, que sea un día espléndido



// Determina el saludo de acuerdo a la hora

Algoritmo OperadorAND

Definir ho **Como** Entero

Definir nom **Como** Cadena

Escribir "Ingrese Nombre"

Leer nom

Escribir "Ingrese Hora (formato 24 horas)"

Leer ho

Si (ho < 6) **OR** (ho > 24) **Entonces**

Escribir "No Aplica Saludo"

SiNo

Si (ho >= 6) **AND** (ho <= 12) **Entonces**

Escribir "Buenos días: ", nom , "que sea un día espléndido"

SiNo

Si (ho > 12) **AND** (ho <= 18) **Entonces**

Escribir "Buenas tardes: ", nom , "aprovecha la tarde"

SiNo

Escribir "Buenas noches: ", nom , "tengas una noche reparadora"

FinSi

FinSi

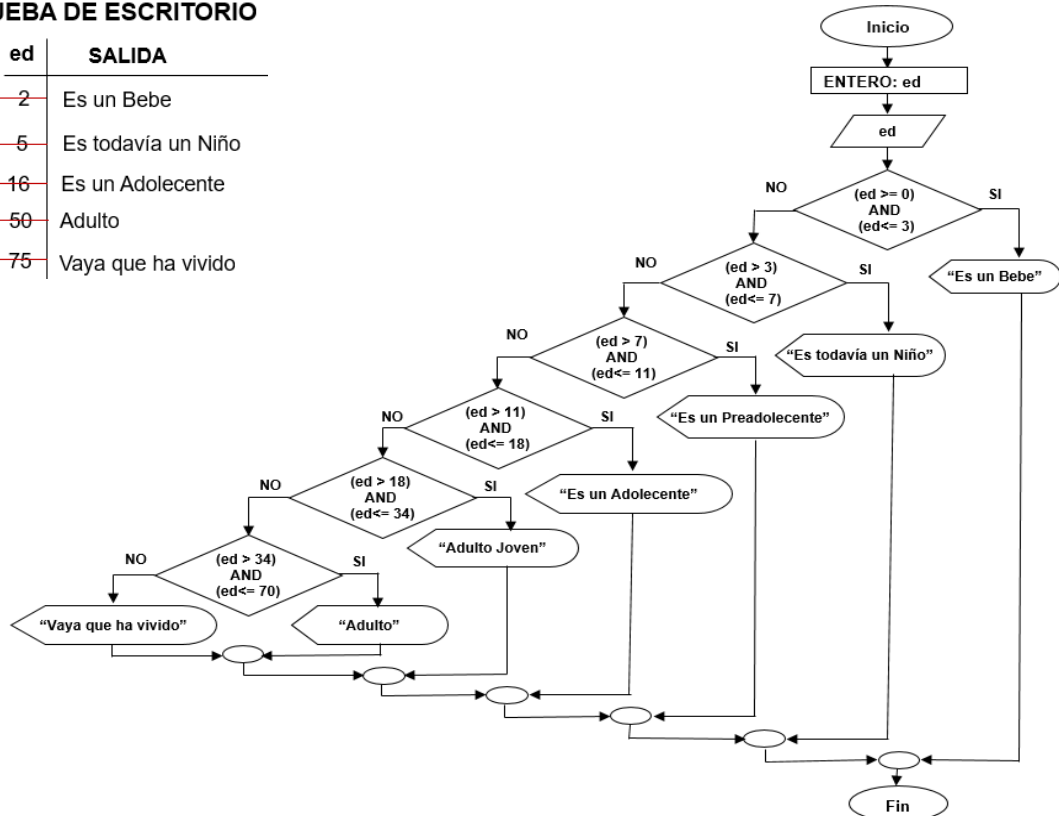
FinSi

FinAlgoritmo

Problema 3: Haga un diagrama que muestre lo siguiente: dada la edad de la persona si está entre 0 y 3 años, escriba “Es un bebe”, si está entre los 3 y 7 años, escriba “Es todavía un niño”, si está entre los 7 y 11 años, escriba “Es un preadolescente”, si está entre los 11 y 18 años, escriba “Es un adolescente”, si está entre los 18 y 34 años, escriba “Adulto joven”, si está entre los 34 y 70 años, escriba “Adulto”, si es mayor a 70 años escriba “Vaya que ha vivido”.

PRUEBA DE ESCRITORIO

ed	SALIDA
-2	Es un Bebe
-5	Es todavía un Niño
-16	Es un Adolescente
-50	Adulto
-75	Vaya que ha vivido



Algoritmo OperadorAND

Definir ed Como Entero

Escribir "Ingrese Edad"

Leer ed

Si (ed >= 0) AND (ed <= 3) Entonces

 Escribir "Es un Bebe"

SiNo

 Si (ed > 3) AND (ed <= 7) Entonces

 Escribir "Es todavía un Niño"

 SiNo

 Si (ed > 7) AND (ed <= 11) Entonces

 Escribir "Es un Preadolecente"

 SiNo

 Si (ed > 11) AND (ed <= 18) Entonces

 Escribir "Es una Adolescente"

 SiNo

 Si (ed > 18) AND (ed <= 34) Entonces

 Escribir "Adulto Joven"

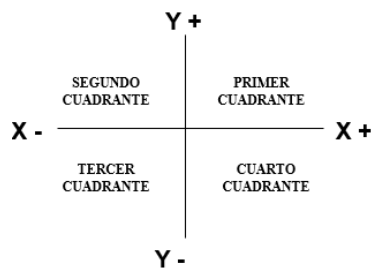
 SiNo

 Si (ed > 34) AND (ed <= 70) Entonces

 Escribir "Adulto"

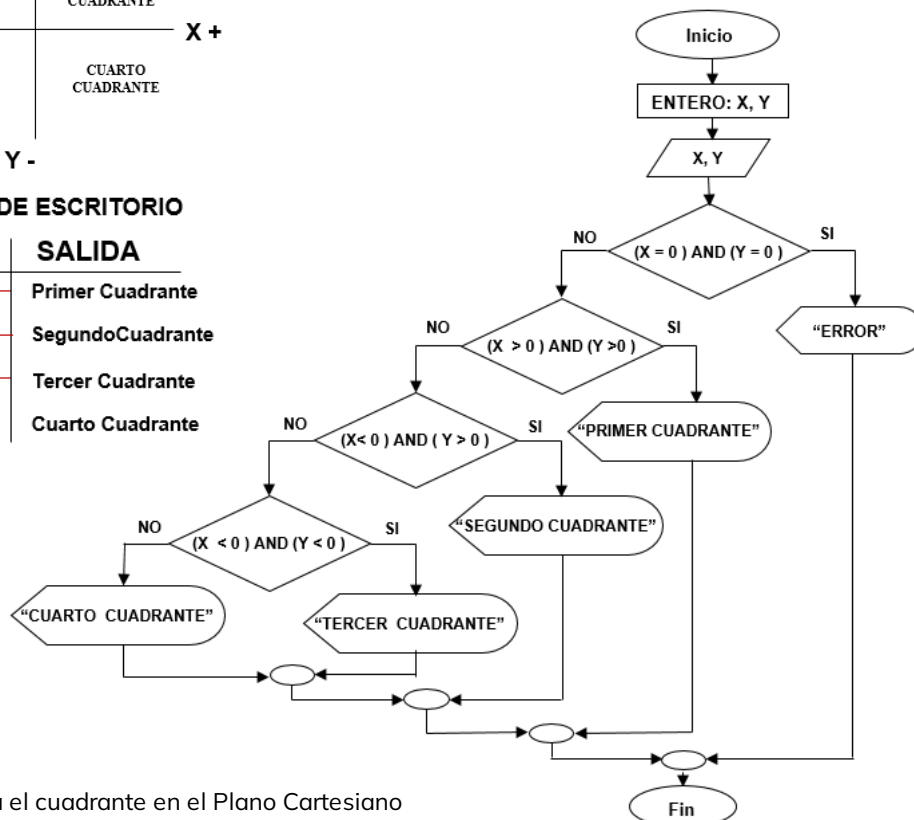
SiNo
Escribir "Vaya que ha vivido"
FinSi
FinSi
FinSi
FinSi
FinSi
FinSi
FinAlgoritmo

Problema 4: Diagrama que permite ingresar dos valores enteros que representan el eje de las X y el eje de las Y en un Plano Cartesiano. De acuerdo a los valores ingresados, indicar a qué cuadrante del plano cartesiano, corresponden dichos valores. Considere finalizar si ambos valores son iguales a cero muestre "ERROR" y finalice.



PRUEBA DE ESCRITORIO

X	Y	SALIDA
6	2	Primer Cuadrante
-8	4	Segundo Cuadrante
-5	-6	Tercer Cuadrante
9	-5	Cuarto Cuadrante



// Determina el cuadrante en el Plano Cartesiano

Algoritmo OperadorAND

Definir x, y Como Entero

Escribir "Ingrese Valor Eje X"

Leer x

Escribir "Ingrese Valor Eje Y"

Leer y

Si (x = 0) AND (y = 0) Entonces

Escribir "ERROR"

SiNo

Si (x > 0) AND (y > 0) Entonces

```

        Escribir "PRIMER CUADRANTES"
    SiNo
        Si  $(x < 0)$  AND  $(y > 0)$  Entonces
            Escribir "SEGUNDO CUADRANTE"
        SiNo
            Si  $(x < 0)$  AND  $(y < 0)$  Entonces
                Escribir "TERCER CUADRANTE"
            SiNo
                Escribir "CUARTO CUADRANTE"
        FinSi
    FinSi
FinSi
FinAlgoritmo

```

Problemas Propuestos

Ejercicio 1: Diagrama que solicite ingresar un nombre de usuario y una contraseña. Si el nombre es “María” y la contraseña es “uleam”, mostrar el mensaje “Usuario y contraseña correctos”. Si el nombre o la contraseña no coinciden, mostrar “Acceso denegado”.

Ejercicio 2: Diagrama que permita el ingreso de un valor. Si el número tiene un dígito elevarlo al cuadrado. Si tiene dos dígitos elevarlo al cubo. Si tiene tres dígitos, restarle 100. Por cualquiera de las posibilidades anteriores, utilice una sola salida en la que se muestre el valor ingresado, el número de dígitos del valor y el resultado del proceso solicitado. Finalmente considere que si el número es menor a 1 o mayor a tres dígitos, muestre el mensaje “Número no valido” y finalice.

Ejercicio 3: Diagrama que determine la calificación final de un estudiante en base a su puntaje en tres exámenes, donde si el promedio de sus puntajes es mayor o igual a 90 la calificación es **A**, si está entre 80 y 89 la calificación es **B**, si está entre 70 y 79 la calificación es **C**, si está entre 60 y 69 la calificación es **D**, y si está por debajo de 60 la calificación es **F**.

Ejercicio 4: Diagrama que lea tres números enteros HO, MI, SE correspondientes a hora, minutos y segundos respectivamente. Determine y compruebe si la hora que se ingresa es o no válida. Considere que se lee la hora en modo 24 Horas, es decir, el valor válido para las horas será mayor o igual que cero y menor que 24. El valor

válido para los minutos y segundos estará comprendido entre 0 y 59 ambos incluidos.

Problema 5: Diagrama que determine con un mensaje, si un paciente que llega a un centro médico, debe ser atendido de inmediato si presenta **fiebre mayor a 39° o presión arterial mayor a 150**. Sin embargo considere:

- a) Si cumple ambas condiciones, se clasifica como “Urgencia crítica”.
- b) Si cumple solo una, se clasifica como “Urgencia moderada”.

Problema 6: “Delicias” es un negocio dedicado a ofrecer banquetes; sus tarifas son las siguientes: el costo de platillo por persona es de \$20, pero si el número de personas está entre 100 y 200, el costo es de \$18. Si el número de personas está entre 201 y 300, el costo es de \$16. Si el número de personas está entre 301 y 400, el costo es de \$14. Para más de 400 personas el costo por platillo es de \$10. Se requiere un diagrama que ingrese el número de personas (en caso de ser menor a 1, enviar un mensaje de “error” y finalice) y determine el presupuesto correspondiente.

Problema 7: Diagrama que solicite la edad, el sexo de una persona y el número de aportaciones de su seguro. El diagrama debe determinar si ya puede o no jubilarse. Tome en cuenta que un hombre se puede jubilar cuando tenga 60 años o más, en cambio, una mujer se jubilará si tiene más de 55 años. Otro factor a considerar es que sus aportaciones para jubilarse deben ser mayores a 400 sin importar el sexo de la persona.

Problema 8: Diagrama que solicite tres números y diga cuál es el mayor y cuál es el menor. Considerar que si entre los tres números hay valores “iguales” enviar el mensaje “Igualdad” y termine el proceso.

Problema 9: En el estatuto de la universidad se establece la siguiente “escala” para determinar la “equivalencia” de evaluación de notas de los estudiantes en un parcial. Para ello se debe ingresar la nota del parcial del estudiante (debe estar entre 1 y 10,

ambos inclusive. De no ser así envíe el mensaje “error” y finalice) y se pide determinar la “equivalencia” correspondiente.

Puntaje parcial	Equivalencia
De 9.50 a 10	Excelente
De 8.50 a 9.49	Muy Bueno
De 7.50 a 8.49	Bueno
De 7.00 a 7.99	Regular
Menos de 7	Deficiente

Ejercicio 10: Diagrama que permita ingresar dos valores que representan un mes y día (una fecha) respectivamente. Determine mediante un mensaje a que estación del año corresponde la fecha ingresada. Considere: Primavera: inicia el 20 de marzo y finaliza el 21 de junio. Verano: inicia el 21 de junio y finaliza el 23 de septiembre. Otoño: inicia el 23 de septiembre y finaliza el 21 de diciembre. Invierno: inicia el 21 de diciembre y finaliza el 20 de marzo.

2.5.8.2.3.5.3 Operador lógico NOT

El operador lógico NOT (también conocido como negación) es un **operador unario** (usa un solo operando) que se utiliza para invertir el resultado de la condición. En otras palabras, si la expresión es verdadera, NOT la convierte en falsa, y si la expresión es falsa, NOT la convierte en verdadera. Funciona de la siguiente forma:

"Si la condición es verdadera la pregunta será **FALSA**. Si la condición es falsa, la pregunta será **VERDADERA**".

Para utilizar el operador lógico **NOT** se aplica una tabla de la verdad (tabla 12) que muestra cómo se comporta el operador con los dos los posibles resultados (verdadero y falso) al negar una condición

Tabla 12

Tabla de la verdad del operador lógico NOT

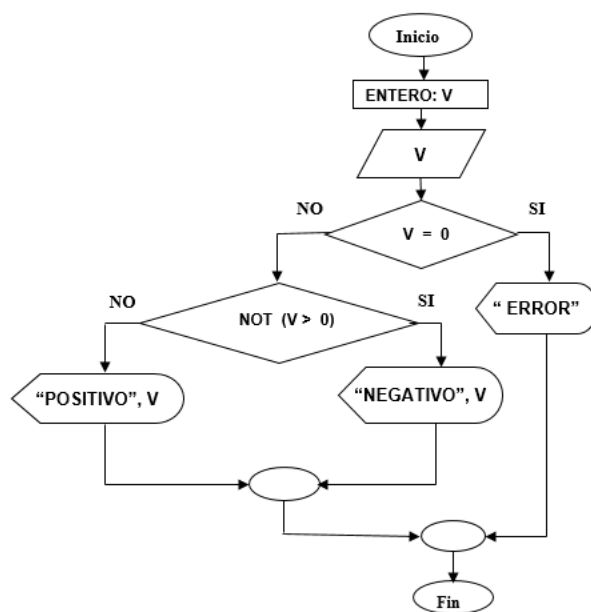
Condición	Resultado con NOT
Verdadero	Falso
Falso	Verdadero

Nota. Fuente: Elaboración propia

El operador NOT se usa para crear una “negación lógica”, por lo que se puede combinar con otros operadores lógicos como AND y OR para formar expresiones booleanas más complejas.

Problemas Resueltos

Problema 1: Diagrama que permita el ingreso de Valor, si este valor es igual a cero enviar un mensaje de error, caso contrario muestre por pantalla un mensaje indicando si el valor es positivo o negativo.



PRUEBA DE ESCRITORIO

V	SALIDA
~~-300~~	POSITIVO 300
~~-5~~	NEGATIVO -5
0	ERROR

// Determina la negación de un valor

Algoritmo OperadorNOT

Definir v Como Entero

Escribir "Ingrese Valor"

Leer v

Si v = 0 **Entonces**

Escribir "ERROR"

SiNo

Si NOT (v > 0) **Entonces**

Escribir "NEGATIVO", v

SiNo

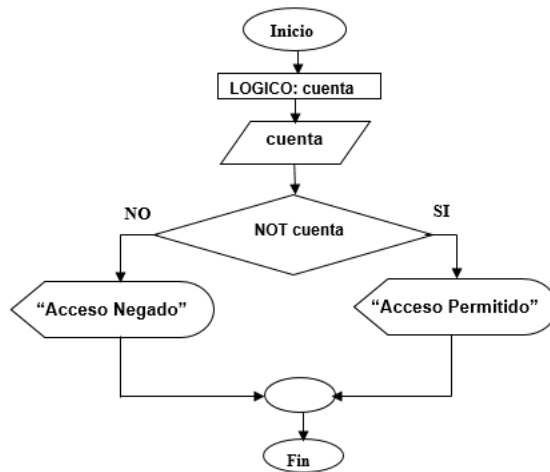
Escribir "POSITIVO", v

FinSi

FinSi

FinAlgoritmo

Problema 2: Diagrama que permita determinar si un usuario puede acceder a su cuenta de correo institucional, siempre que su cuenta no esté bloqueada.
El diagrama debe preguntar si la cuenta está bloqueada y mostrar si el acceso es permitido o denegado.



PRUEBA DE ESCRITORIO

cuenta	SALIDA
~~VERDADERO~~	Acceso Negado
~~FALSO~~	Acceso Permitido

// Determina el acceso a correo

Algoritmo OperadorNOT

Definir cuenta **Como** Logico

Escribir "¿La cuenta está bloqueada?"

(Verdadero/Falso)"

Leer cuenta

Si NOT cuenta **Entonces**

Escribir "Acceso permitido"

Sino

Escribir "Acceso Negado"

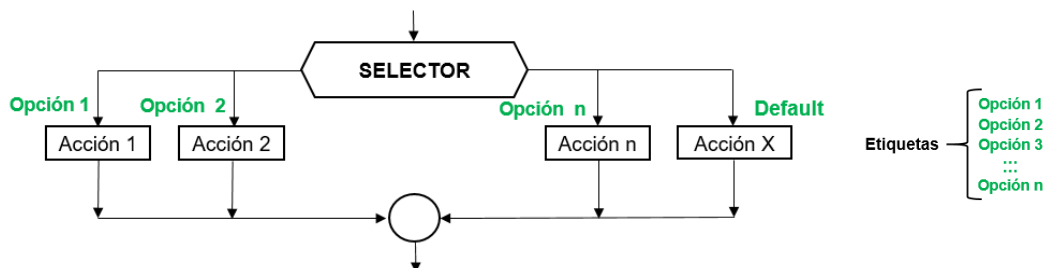
FinSi

FinAlgoritmo

2.5.8.2.3.5 Estructuras Condicionales Múltiples o Selectivas

Esta estructura permite seleccionar y ejecutar una “acción determinada” de entre múltiples alternativas posibles. Por lo tanto, el flujo del diagrama se bifurca por varias ramas u “opciones”, dependiendo del “contenido” de una variable y/o expresión de control llamada “**SELECTOR**”. El camino u “opción” elegido se identifica como “**ETIQUETA**”, y su ejecución dependerá del valor que tome el selector.

Su representación gráfica para utilizarse en diagramación es la siguiente:



Funciona de la siguiente forma:

Si el **valor del selector** (que puede ser una variable o una expresión) “coincide con la opción 1 (etiqueta)”, se bifurca el control a ejecutar la Acción 1; si el valor del selector “coincide con la opción 2 (etiqueta)”, se bifurca el control a

ejecutar la Acción 2, y si el valor del selector “coincide con la opción n (etiqueta)”, se bifurca el control a ejecutar la Acción N. Si el valor del selector “No coincide con ninguna opción”, entonces se ejecutan las “acciones por **DEFAULT**”. La evaluación de valores del selector con los valores de las etiquetas se hace de izquierda a derecha.

Esta estructura es especialmente útil cuando:

- Hay varias opciones posibles.
- Se evalúa una sola variable u expresión que será el **SELECTOR**.
- Se comparan valores exactos, entre el selector y las etiquetas.
- Se quiere un código o programa más claro y organizado.
- El programa requiere trabajar con menús u opciones numeradas.



A diferencia de las estructuras condicionales, que solo permiten **dos opciones de salida (Si – No)**, las estructuras múltiples permiten **más de dos opciones u alternativas posibles**.

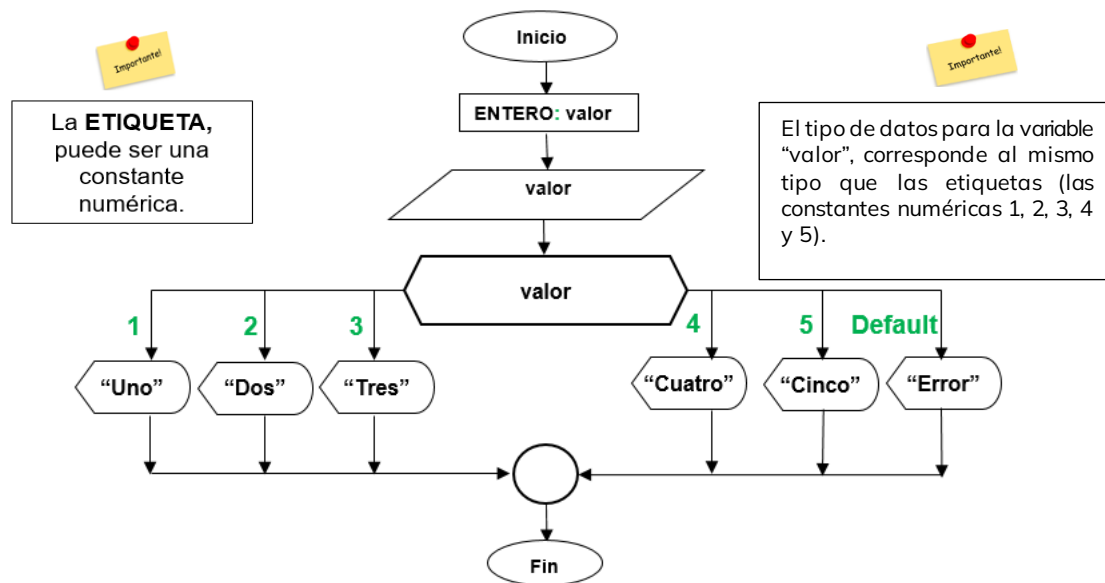
2.5.8.2.3.6 Consideraciones al utilizar condiciones Múltiples o Selectivas

- **EL SELECTOR:** Indica o representa el criterio de evaluación condicional de la estructura. Puede ser el valor de una Variable (Numérica entera o de Carácter) o el resultado de una expresión aritmética.
- **La ETIQUETA:** Identifica la opción que es representada por un valor (constante numérica, alfanumérica o una expresión), hacia donde se bifurca el flujo, por lo que debe ser del mismo tipo de datos del valor del selector. El tipo de etiqueta dependerá del problema a resolver.
- En caso que el valor del selector no coincida con ningún valor de las etiquetas, se ejecuta la acción por DEFAULT (defecto) u “De Otro Modo”.
- Solo se ejecuta un caso, es decir, cuando se encuentra un caso que coincide se ejecutan sus instrucciones, después de eso la estructura termina. No se ejecutan los demás casos.
- No tiene sentido usar una condición selectiva, cuando hay muy pocas opciones. Por ejemplo sexo (1; hombre, 2: Mujer), en este caso es más óptimo y eficiente usar una condición doble.
- Al iniciar la ejecución de la estructura se puede pronunciar: “Dependiendo del valor de....”, “Según el valor de ...”, “En el caso del valor de.....”.

- En programación, dependiendo del lenguaje a utilizar se la denomina: Switch (Java, C#, C++, PHP, etc... , Select Case (Python)). En Pseudocódigo se utiliza la estructura “Según”.

Problemas Resueltos

Problema 1: Diagrama que permita ingresar un valor entre 1 y 5 (ambos inclusive) muestre el dígito ingresado en letras. En caso de ingresar un valor fuera de rango envíe un mensaje de “Error” y finalice.



// Muestra números (1 a 5) en letras

Algoritmo EstructuraSelectiva

Definir valor **Como** Entero

Escribir "Ingrese valor entre 1 y 5"

Leer valor

Según valor **Hacer**

Caso 1:

Escribir "Uno"

Caso 2:

Escribir "Dos"

Caso 3:

Escribir "Tres"

Caso 4:

Escribir "Cuatro"

Caso 5:

Escribir "Cinco"

De Otro Modo:

Escribir "Error"

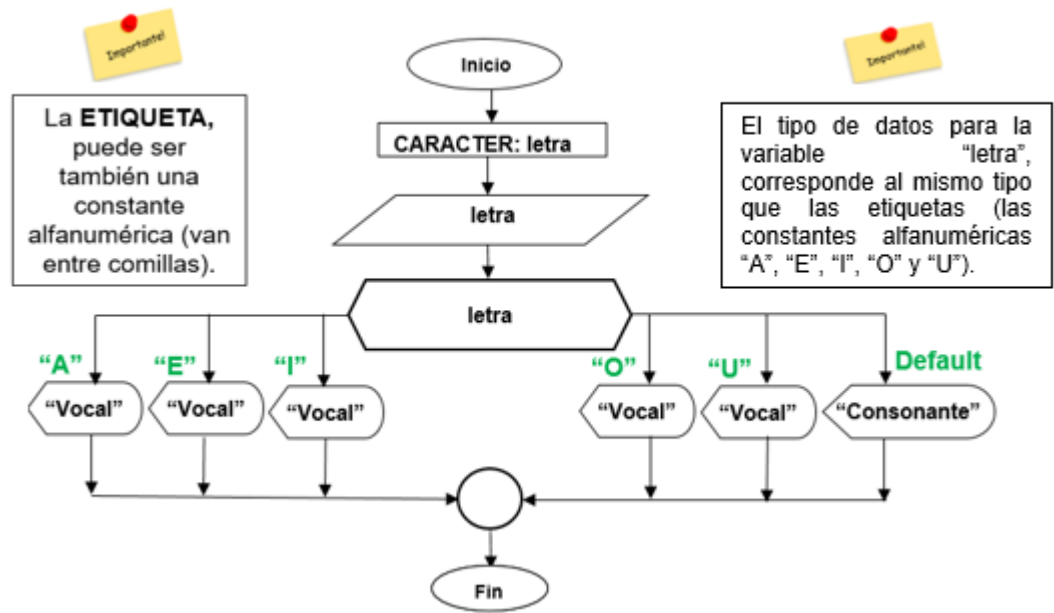
FinSegún

FinAlgoritmo

PRUEBA DE ESCRITORIO

valor	SALIDA
~~-2~~	Dos
~~-5~~	Cinco
~~-1~~	Uno
~~-4~~	Cuatro
~~-3~~	Tres
~~-10~~	Error
-5	Error

Problema 2: Diagrama que permita ingresar un carácter en mayúscula. Determina con un mensaje si el carácter ingresado es “vocal” o “consonante”.

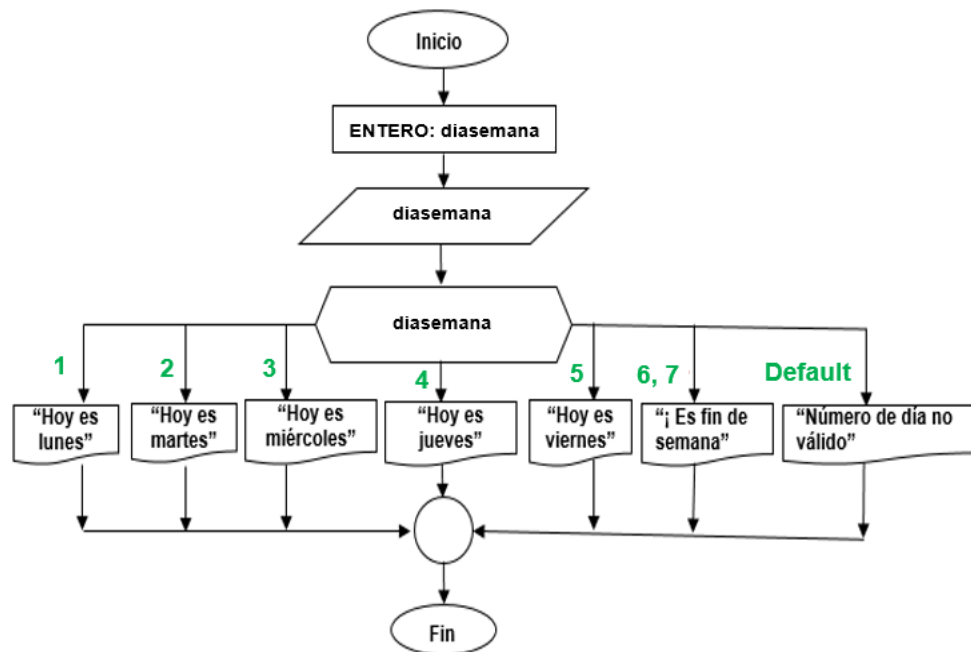


```
// Muestra Vocal o Consonante
Algoritmo EstructuraSelectiva
Definir letra Como Caracter
Escribir "Ingrese letra Mayúscula"
Leer letra
Según letra Hacer
    Caso "A":
        Escribir "Vocal"
    Caso "E":
        Escribir "Vocal"
    Caso "I":
        Escribir "Vocal"
    Caso "O":
        Escribir "Vocal"
    Caso "U":
        Escribir "Vocal"
    De Otro Modo:
        Escribir "Consonante"
FinSegún
FinAlgoritmo
```

PRUEBA DE ESCRITORIO

letra	SALIDA
~~E~~	Vocal
~~A~~	Vocal
~~U~~	Vocal
~~X~~	Consonante
O	Vocal

Problema 3: Diagrama que permita ingresar un valor entre 1 y 7 (ambos inclusive). Se pide mostrar (se puede usar también un nuevo símbolo) a que día de la semana corresponde dicho valor, considerando mostrar el mensaje “¡ Es fin de semana”, cuando se ingrese sábado o domingo. En caso de ingresar un valor fuera de rango envíe el mensaje “Número de día no válido” y finalice.



// Muestra el día de la semana

Algoritmo EstructuraSelectiva

Definir diasemana **Como Entero**

Escribir "Ingrese el día se semana en número"

Leer diasemana

Según letra **Hacer**

Caso 1:

Escribir "Hoy es lunes"

Caso 2:

Escribir "Hoy es martes"

Caso 3:

Escribir "Hoy es miércoles"

Caso 4:

Escribir "Hoy es jueves"

Caso 5:

Escribir "Hoy es viernes"

Caso 6, 7:

Escribir "¡ Es fin de semana"

De Otro Modo:

Escribir "Número de día no válido"

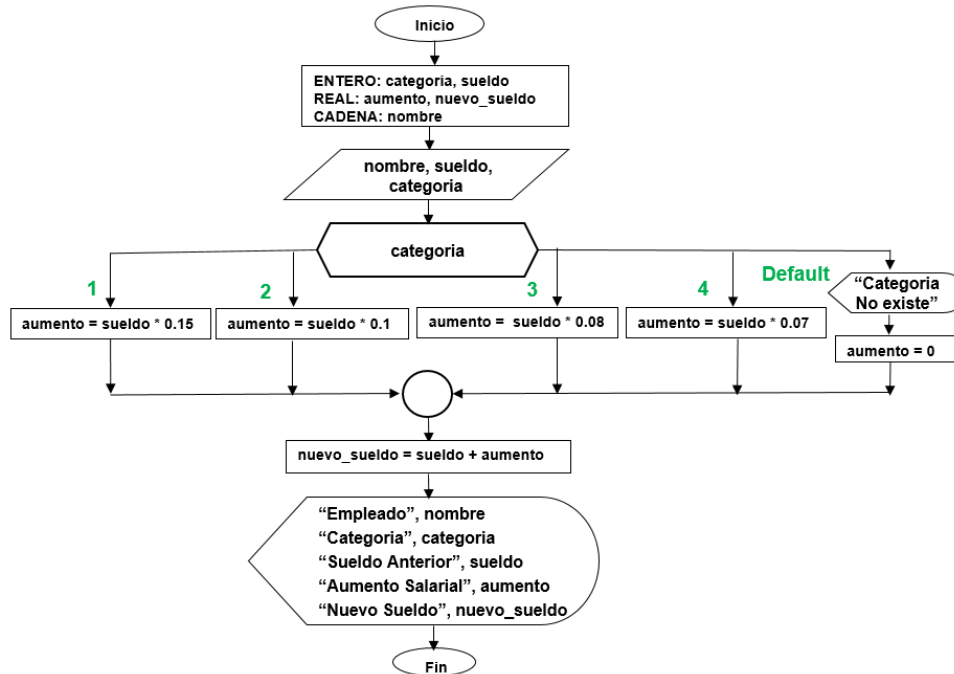
FinSegún

FinAlgoritmo

PRUEBA DE ESCRITORIO

diasemana	SALIDA
~~2~~	Hoy es martes
~~5~~	Hoy es viernes
~~1~~	Hoy es lunes
~~4~~	Hoy es jueves
~~3~~	Hoy es miércoles
~~6~~	¡ Es fin de semana
10	Número de día no válido

Problema 4: Dados como datos la categoría y el sueldo de un trabajador, calcule el aumento salarial correspondiente, teniendo en cuenta la siguiente tabla. Muestre la categoría del trabajador y el nuevo sueldo.



INCREMENTOS	
CATEGORIA	AUMENTO
1	15%
2	10%
3	8%
4	7%

// Calcula aumento salarial en base a categoría

Algoritmo EstructuraSelectiva

Definir categoría, sueldo **Como Entero**

Definir aumento, nuevo_sueldo **Como Real**

Definir nombre **Como Cadena**

Escribir "Ingrese Nombre Empleado"

Leer nombre

Escribir "Ingrese Sueldo"

Leer sueldo

Escribir "Ingrese Categoría (1 – 4)"

Leer categoria

Según categoria **Hacer**

Caso 1:

aumento = sueldo * 0.15

Caso 2:

aumento = sueldo * 0.10

Caso 3:

aumento = sueldo * 0.08

Caso 4:

aumento = sueldo * 0.07

De Otro Modo:

Escribir "Categoría no existe"

aumento = 0

FinSegún

nuevo_sueldo = sueldo + aumento

Escribir "Empleado", nombre

Escribir "Categoría", categoria

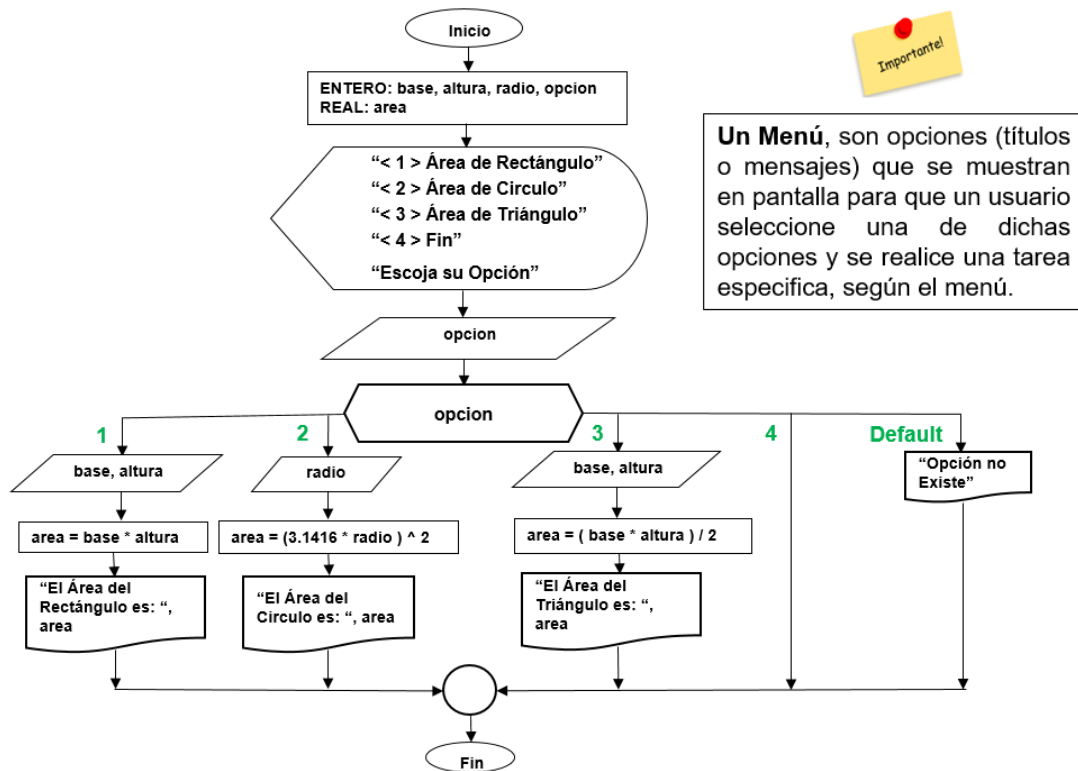
Escribir "Sueldo Anterior", sueldo

Escribir "Aumento salarial", aumento

Escribir "Nuevo Sueldo", nuevo_sueldo

FinAlgoritmo

Problema 5: Diagrama que permita efectuar el cálculo del área de un: Rectángulo, Círculo y Triángulo según la opción seleccionada por el usuario a través de un menú. Además se deben ingresar los datos adicionales que se requieran para el cálculo del área según opción seleccionada.



// Calcula aumento salarial en base a categoría

Algoritmo EstructuraSelectiva

Definir base, altura, radio, opcion **Como**

Entero

Definir area **Como** Real

Escribir "< 1 > Área de Rectángulo"

Escribir "< 2 > Área de Círculo"

Escribir "< 3 > Área de Triángulo"

Escribir "< 4 > Fin"

Escribir "Escoja Opción"

Leer opcion

Según opcion **Hacer**

Caso 1:

Escribir "Ingrese Base Rectángulo"

Leer base

Escribir "Ingrese Altura Rectángulo"

Leer altura

area = base * Altura

Escribir "El Área del Rectángulo es: ",

area

Caso 2:

Escribir "Ingrese Radio del Círculo"

Leer radio

area = (3.1416 * radio) ^ 2

Escribir "El Área del Círculo es: ", area

Caso 3:

Escribir "Ingrese Base Triángulo"

Leer base

Escribir "Ingrese Altura Triángulo"

Leer altura

area = (base * Altura) / 2

Escribir "El Área del Triángulo es: ", area

Caso 4:

De Otro Modo:

Escribir "Opción no existe"

FinSegún

FinAlgoritmo

Problemas Propuestos

Problema 1: Diseñar un diagrama que utilizando “Condiciones Selectiva”, permita ingresar un número entero (1 a 10), muestre su equivalente en números romanos.

Problema 2: Diagrama que utilizando “Condiciones Selectiva”, solicite el ingreso un número del 1 al 12 y que muestre a qué trimestre del año corresponde.

Problema 3: Una empresa otorga bonos según el tipo de empleado:

1. Administrativo (10 % del salario)
2. Técnico (15 % del salario)
3. Profesional (20 % del salario)
4. Directivo (25 % del salario)

Elaborar el diagrama utilizando “Condiciones Selectiva”, permita ingresar el tipo de empleado y su salario y mostrar el bono correspondiente.

Problema 4: Diseñar un diagrama que utilizando “Condiciones Selectiva”, permita calcular la tarifa eléctrica según el consumo en base a las siguientes categorías:

1. Residencial (\$0.25 por kWh)
2. Comercial (\$0.50 por kWh)
3. Industrial (\$ 1 por kWh)

Se ingresará el consumo en kWh y calcular y mostrar el valor total a pagar.

Problema 5: Diagrama que permita ingresar un carácter en una variable. Determine con un mensaje si el carácter ingresado es “vocal” (sin importar que sea mayúsculas o minúsculas) o “Cualquier otro caracter”. Utilice una “Condición Selectiva” para lo solicitado.

Problema 6: En una carretera, el peaje depende del tipo de vehículo:

- 1: Motocicleta (\$1)
- 2: Automóvil (\$2)
- 3: Camioneta (\$3)
- 4: Bus (\$5)
- 5: Camión (\$7)

El diagrama debe mostrar en menú el tipo de vehículo y calcular el valor a pagar de acuerdo a la tarifa establecida. Considere que si ingresa una opción incorrecta, mostrar “Categoría no válida”.

Problema 7: Diagrama que utilizando una “Condición Selectiva”, permita enviar un menú con las opciones de: Suma, Resta, Multiplicación, División y Salir. Por cualquiera de las operaciones básicas matemáticas, ingresar dos números mediante las variables numéricas “valor1” y “valor2”. Se pide realizar y mostrar el resultado de la operación seleccionada (usar 1 solo símbolo de salida para mostrar) según la OPERACIÓN escogida del menú. **Nota:** En caso de dividir para CERO envíe un mensaje de “Error”.

Problema 8: La pizzería Bella Napoli ofrece pizzas vegetarianas y no vegetarianas a sus clientes. Los ingredientes para cada tipo de pizza aparecen a continuación.

- Ingredientes vegetarianos: Pimiento y tofu.
- Ingredientes no vegetarianos: Peperoni, Jamón y Salmón.

Hacer el diagrama que pregunte al usuario si quiere una pizza vegetariana o no, y en función de su respuesta le muestre un menú con los ingredientes disponibles para que elija. Solo se puede elegir un ingrediente además de la mozzarella y el tomate que están en todas las pizzas. Al final se debe mostrar si la pizza elegida es vegetariana o no y todos los ingredientes que lleva.

Problema 9: Diagrama que ingrese el nombre del empleado, la zona de trabajo, las ventas del empleado, luego calcular su comisión en base a un porcentaje basado en la zona de trabajo, luego determinar el aporte al IESS (9.45 % de los ingresos totales) y el total a recibir del empleado. Muestre los datos ingresados y calculados. Use una “Condición Selectiva”. Se considera la siguiente tabla para el cálculo de la comisión.

Zona	Porcentaje de Comisión
A	6 %
B	8 %
C	9 %

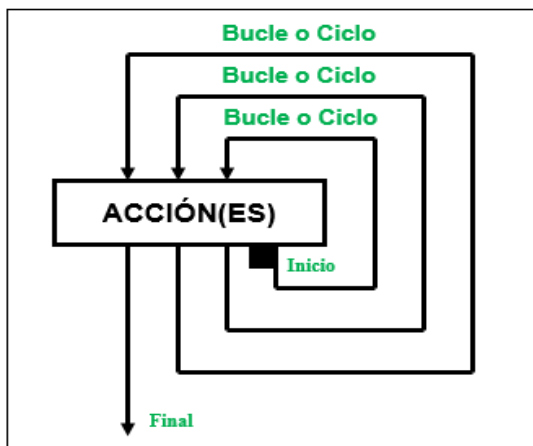
Problema 10: Diseñar un diagrama que utilizando una “Condición Selectiva”, elabore un conversor de medidas de longitud. El usuario podrá elegir mediante un menú entre las siguientes unidades de medida: 1) Metros 2) Pies 3) Centímetros 4) Pulgadas. Así mismo el usuario tendrá que indicar la equivalencia a la que desea convertir (serán las mismas mostradas en el menú, menos la opción que ya ha sido seleccionada). Analice y determine qué datos se necesitan.

2.5.8.3 Estructuras Repetitivas o de Iteración

Las estructuras repetitivas son llamadas también estructuras cíclicas, iterativas o de bucles. Permiten ejecutar una “acción” o “conjunto de acciones” de manera repetitiva (o cíclica) un cierto número finito de veces (figura 15). Es decir, permiten realizar **bucles** en programación.

Figura 15

Representación gráfica de un proceso repetitivo



Nota. Fuente: Elaboración propia

Un bucle (del inglés *loop*) o ciclo (en programación, **ciclo y bucle significan exactamente lo mismo**), es una estructura de control que permite repetir la ejecución de un conjunto instrucciones o acciones tantas veces, hasta que se cumpla una determinada condición de salida del bucle. El proceso de repetir la ejecución de una secuencia de acciones se denomina también **iteración**.

Un aspecto importante a considerar es que todo bucle tiene un **principio** y un **final** que representa la “condición de salida del bucle”. En programación la salida del bucle, se puede originar por los siguientes factores:

- Al cumplirse un **Número Determinado de Veces** la ejecución del conjunto de acciones.
- Ejecutar el conjunto de acciones repetidamente **Mientras** se cumpla una condición.
- Ejecutar el conjunto de acciones repetidamente **Hasta que** ocurra algo específico en una condición.

2.5.8.3.1 Tipos de Estructuras repetitivas en programación

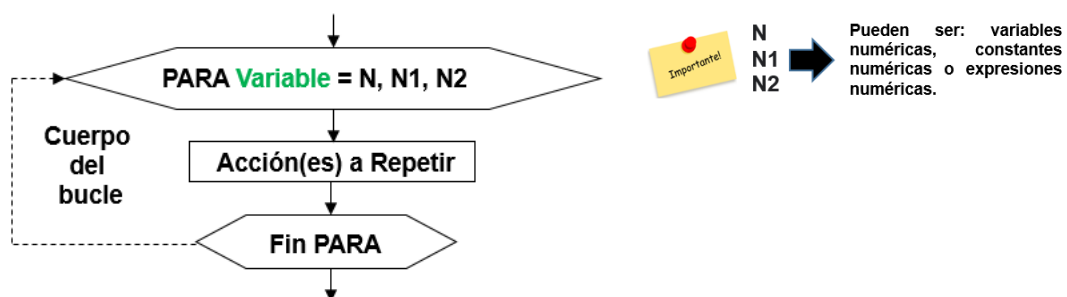
Los tipos de estructuras repetitivas más comunes son:

- Desde / Para (FOR)
- Mientras (WHILE)
- Repetir Hasta Que (Do While / Repeat Until)

Todos los lenguajes de programación utilizan las estructuras repetitivas comunes y adicionalmente aplican variantes de las mismas, por lo que se debe consultar la documentación respectiva del lenguaje de programación que se utilizará.

2.5.8.3.1.1 Estructura Repetitiva Para / Desde (For)

Permite realizar un bucle, conociendo de antemano el número de ciclos a realizar. Por lo tanto, ejecuta en forma repetitiva la(s) acción(es) un número especificado de veces, controlando de modo “automático” las iteraciones a través del cuerpo del bucle. Usa el siguiente esquema:



En este gráfico:

Variable: Debe ser numérica y entera, controla el bucle y se la conoce como "corredor" o "Contador Automático" o "variable controladora del bucle".

N: Indica el **valor inicial** desde donde comienza la "variable controladora del bucle".

N1: Indica el **valor final**, hasta donde debe llegar “la variable controladora del bucle”.

N2: Es un valor que indica el **Incremento o Decremento** que tendrá la “variable controladora del bucle”. Es decir, cómo aumenta o disminuye ese valor en la variable controladora. Por lo expuesto, este valor puede ser positivo o negativo y depende de:

- a) Si N (valor inicial) $\leq N1$ (valor final), entonces **N2** será un valor “**POSITIVO**” y se **sumará** al contenido de la “variable controladora del bucle”.
- b) Si N (valor inicial) $> N1$ (valor final), entonces **N2** será un valor “**NEGATIVO**” y se **restará** al contenido de la “variable controladora del bucle”.

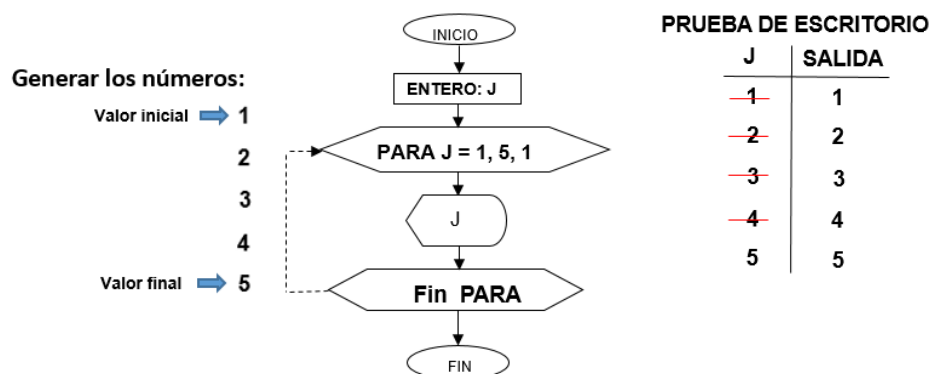
Importante: **N2** será opcional cuando el incremento sea de uno positivo (**+1**), de lo contrario su uso es obligatorio.

¿Cómo funciona?

Al llegar el flujo la primera vez a la estructura, la “variable controladora del bucle” toma el valor inicial (N) y entra al “cuerpo del ciclo” a ejecutar la “acción o conjunto de acciones a repetir” comprendidos entre **PARA** y **Fin PARA**. Al llegar a este último, verifica si el valor de la variable controladora llegó a su valor final ($N1$). Si es así, sale del bucle, caso contrario regresa nuevamente a **PARA** a sumar o restar el valor de **N2** a la variable controladora y entra nuevamente al bucle a repetir el proceso.

Problemas Desarrollados

Problema 1: Diagrama que use una estructura repetitiva **PARA**. Se solicita automáticamente generar los números del uno al cinco.



En este diagrama, la “variable controladora del bucle” es **J**. Esta variable a lo largo del flujo tomará los valores de: 1, 2, 3, 4 y 5, por lo tanto, su contenido será mostrado en forma REPETITIVA en el cuerpo del bucle.

El ciclo **PARA** será controlado por la variable **J**. Esta variable va de **1** que será su valor inicial, hasta **5** que será su valor final. El incremento que tendrá la variable controladora será de **1** (positivo). De acuerdo al enunciado los números a mostrar van de 1,2,3,4,5.

¿Cómo Funciona este Diagrama? – Prueba de Escritorio

1. Al llegar a la estructura la **primera vez**, la variable controladora **toma el valor inicial que es 1** y entra al bucle a mostrar lo que tiene es decir **1**.
2. Cuando llega el flujo a **Fin PARA**, internamente se verifica si la variable controladora llegó a **5**. Como no es así, automáticamente el flujo retorna a **PARA**, en donde a la variable controladora se le **suma el valor del incremento que es 1**, por lo tanto ahora la variable controladora **tiene 2** y vuelve a entrar al bucle a **mostrar 2** y ejecuta nuevamente el **Fin PARA** para evaluar si la variable controladora que tiene **2** llegó a **5**.
3. El proceso se vuelve **REPETITIVO**, hasta que la variable controladora llegue a **5**. Al **evaluar el Fin PARA** y llegar al valor final, automáticamente sale del bucle y ejecuta la siguiente instrucción que sigue a **Fin PARA**, en este caso a finalizar el proceso.

// Genera automáticamente los números 1 a 5

Algoritmo EstructuraPARA/DESDE

Definir J Como Entero

Para J ← 1 Hasta 5 Con Paso 1

Escribir J

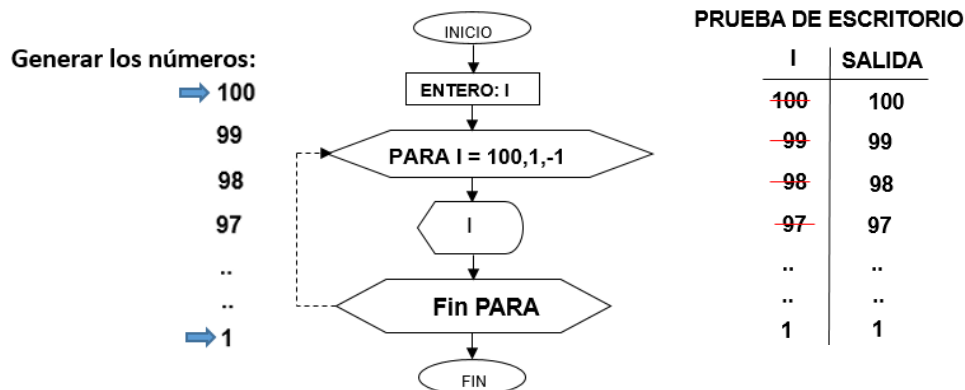
Fin Para

FinAlgoritmo



En este ejercicio, el valor inicial (1) **es menor** al valor final (5), por lo tanto el incremento es **POSITIVO** (1), valor que se **SUMA a la “variable controladora del bucle”**. Considere que el incremento, se puede excluir, dado que es +1.

Problema 2: Diagrama que use una estructura repetitiva **PARA**. Se solicita automáticamente generar los números del cien al uno.



En este diagrama, la “variable controladora del bucle” es **I**. Esta variable a lo largo del flujo tomará los valores de: 100, 99, 98... 1, por lo tanto, su contenido será mostrado en forma REPETITIVA en el cuerpo del bucle.

El ciclo **PARA** será controlado por la variable **I**. Esta variable va de **100** que será su valor inicial, hasta **1** que será su valor final. El incremento que tendrá la variable controladora será de **-1** (negativo). De acuerdo al enunciado los números a mostrar van de 100, 99, 98... 1.

¿Cómo Funciona este Diagrama? – Prueba de Escritorio

1. Al llegar a la estructura la **primera vez**, la variable controladora **toma el valor inicial que es 100** y entra al bucle a mostrar lo que tiene es decir **100**.
2. Cuando llega el flujo a **Fin PARA**, internamente se verifica si la variable controladora llegó a **1**. Como no es así, automáticamente el flujo retorna a **PARA**, en donde a la variable controladora se le **resta el valor del incremento que es - 1**, por lo tanto ahora la variable controladora **tiene 99** y vuelve a entrar al bucle a **mostrar 99** y ejecuta nuevamente el **Fin PARA** para evaluar si la variable controladora que tiene **99** llegó a **1**.
3. El proceso se vuelve **REPETITIVO**, hasta que la variable controladora llegue a **1**. Al **evaluar el Fin PARA** y llegar al valor final, automáticamente sale del bucle y ejecuta la siguiente instrucción que sigue a **Fin PARA**, en este caso a finalizar el proceso.

// Genera automáticamente los números 100 a 1

Algoritmo EstructuraPARA/DESDE

Definir I Como Entero

Para I $\leftarrow$ 100 Hasta 1 Con Paso -1

Escribir I

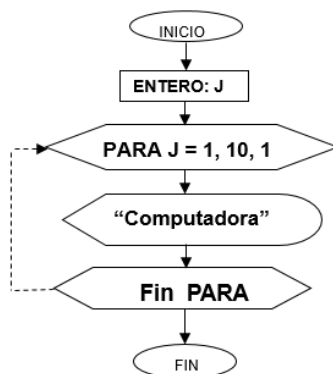
Fin Para

FinAlgoritmo



En este ejercicio, el valor inicial (100) es **mayor** al valor final (1), por lo tanto el incremento es **NEGATIVO** (-1), valor que se **RESTA** a la “variable controladora del bucle”.

Problema 3: Diagrama que use una estructura repetitiva **PARA**. Se solicita mostrar por pantalla automáticamente la palabra “Computadora” diez veces.



PRUEBA DE ESCRITORIO

J	SALIDA
~~1~~	Computadora
~~2~~	Computadora
~~3~~	Computadora
..	..
10	Computadora

// Muestra la palabra Computadora 10 veces

Algoritmo EstructuraPARA/DESDE

Definir J Como Entero

Para J $\leftarrow$ 1 Hasta 10 Con Paso 1

Escribir “Computadora”

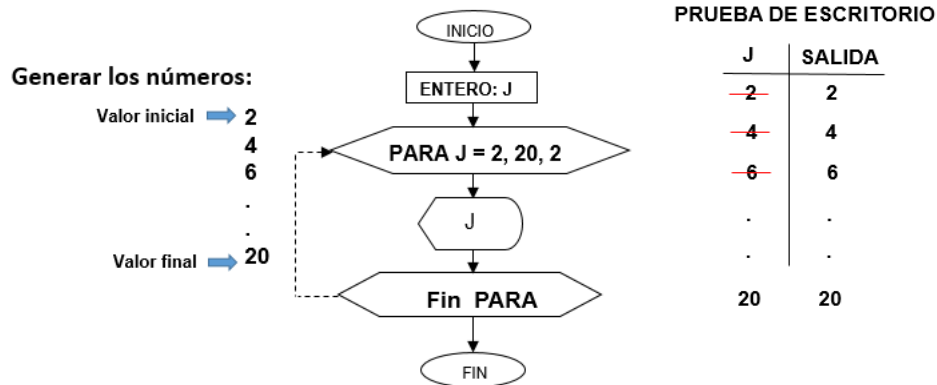
Fin Para

FinAlgoritmo



Este diagrama se realiza **10** ciclos y en cada ciclo, se muestra la constante alfanumérica “Computadora”.

Problema 4: Diagrama que use una estructura repetitiva **PARA**. Se solicita mostrar por pantalla automáticamente los 10 primeros números **pares**.



// Muestra los primeros 10 números pares

Algoritmo EstructuraPARA/DESDE

Definir J Como Entero

Para J ← 2 Hasta 20 Con Paso 2

Escribir J

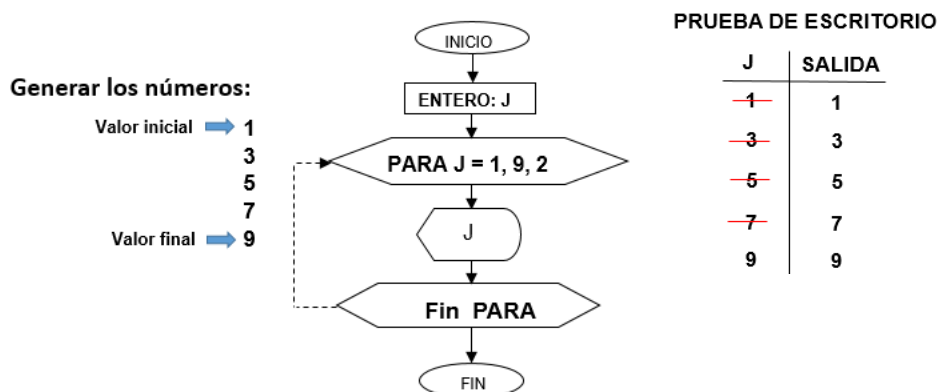
Fin Para

FinAlgoritmo



En este diagrama, el valor inicial de la variable controladora de bucle es **2**, su valor final es **20** y el incremento es de **2**. Por lo tanto, serán **10** ciclos a realizar y en cada ciclo, la variable controladora se incrementa en **2**.

Problema 5: Diagrama que use una estructura repetitiva **PARA**. Se solicita mostrar por pantalla automáticamente los 5 primeros números **impares**.



// Muestra los 5 primeros números impares

Algoritmo EstructuraPARA/DESDE

Definir J Como Entero

Para J ← 1 Hasta 9 Con Paso 2

Escribir J

Fin Para

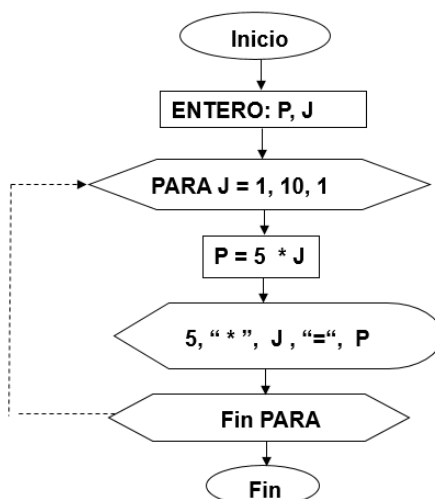
FinAlgoritmo



En este diagrama, el valor inicial de la variable controladora de bucle es **1**, su valor final es **9** y el incremento es de **2**. Por lo tanto, serán **5** ciclos a realizar y en cada ciclo, la variable controladora se incrementa en **2**.

Problema 6: Diagrama que genere automáticamente la tabla de multiplicar del cinco (5). Dicha tabla se debe ir de 1 a 10. Por pantalla se debe mostrar: El multiplicando, el multiplicador y el producto.

5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50



PRUEBA DE ESCRITORIO

J	P	Salida
~~1~~	~~5~~	5 * 1 = 5
~~2~~	~~10~~	5 * 2 = 10
~~3~~	~~15~~	5 * 3 = 15
4	20	5 * 4 = 20
..	..	
..	..	
10	50	5 * 10 = 50

// Muestra la tabla de multiplicar del 5

Algoritmo EstructuraPARA/DESDE

Definir J, P Como Entero

Para J ← 1 Hasta 10 Con Paso 1

P = J * 5

Escribir 5, " ", J, "=", P

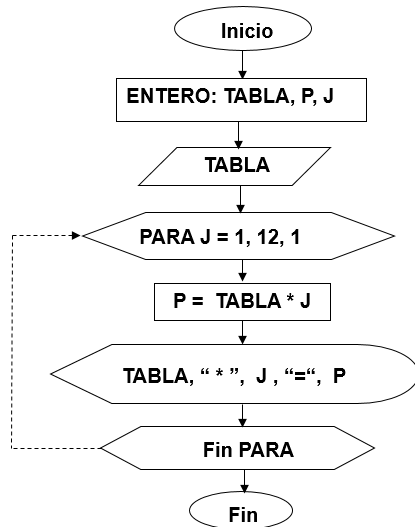
Fin Para

FinAlgoritmo



En este diagrama se realizan **10** ciclos, dado lo solicitado en el enunciado (la tabla va de 1 a 10). Como proceso repetitivo en cada ciclo se multiplica la constante numérica **5**, por el valor de la variable controladora que va a variar de **1 a 10**, dando un **producto diferente**, lo que genera la tabla solicitada.

Problema 7: Diagrama que genere automáticamente cualquier tabla de multiplicar (ingresar la tabla deseada). Dicha tabla se debe ir de 1 a 12. Por pantalla se debe mostrar: El multiplicando, el multiplicador y el producto.



PRUEBA DE ESCRITORIO

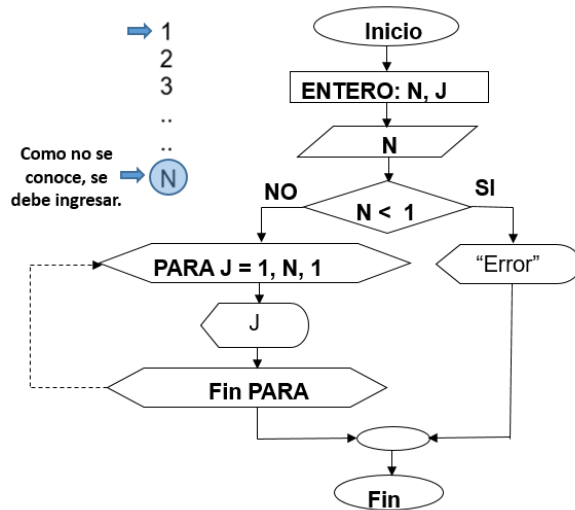
TABLA	J	P	Salida
3	~~1~~	~~3~~	3 * 1 = 3
	~~2~~	~~6~~	3 * 2 = 6
	~~3~~	~~9~~	3 * 3 = 9
	~~4~~	~~12~~	3 * 4 = 12
	..	..	
	..	..	
	12	36	3 * 12 = 36

// Genera una tabla de multiplicar ingresada
Algoritmo EstructuraPARA/DESDE
 Definir TABLA, P, J Como Entero
 Escribir "Ingrese Tabla de Multiplicar"
 Leer TABLA
 Para J ← 1 Hasta 12 Con Paso 1
 P = TABLA * J
 Escribir TABLA, " * ", J, " = ", P
 Fin Para
FinAlgoritmo

Importante!

En este diagrama se realizan **12** ciclos, dado lo solicitado en el enunciado (la tabla va de 1 a 12). Antes de realizar los ciclos, se ingresa la TABLA que se quiere generar (en el caso de prueba fue **3**). Como proceso repetitivo en cada ciclo se multiplica la TABLA ingresada por el valor de la variable controladora que va a variar de **1 a 12**, dando un **producto diferente**, lo que genera lo solicitado.

Problema 8: Diagrama que genere automáticamente los N primeros números. Si el valor de ingresado a N es menor a 1, envíe un mensaje de ERROR y finalice el proceso.



PRUEBA DE ESCRITORIO

N	J	SALIDA
4	~~1~~	1
	~~2~~	2
	~~3~~	3
	4	4

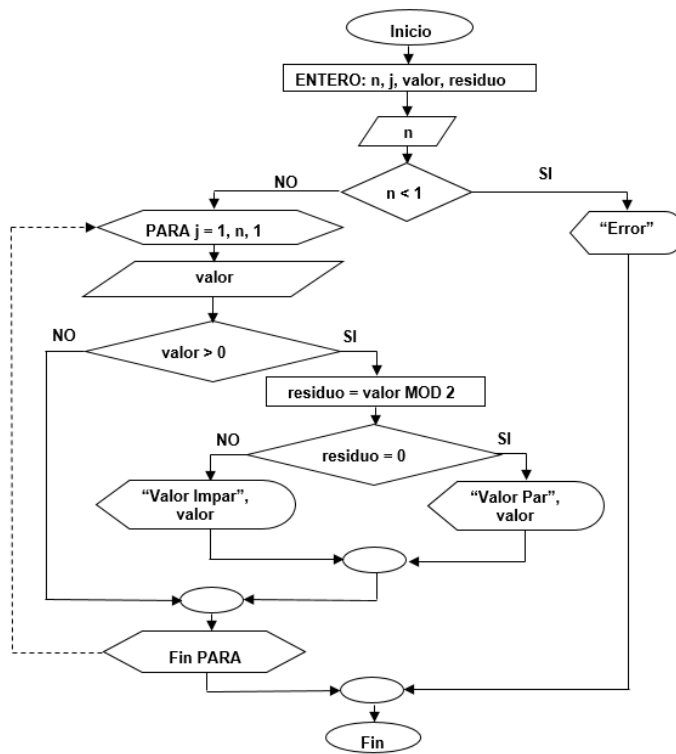
```

// Genera los N primeros números
Algoritmo EstructuraPARA/DESDE
  Definir J, N Como Entero
  Escribir "Ingrese Valor hasta donde generar"
  Leer N
  Si N < 1 Entonces
    Escribir "ERROR"
  SiNo
    Para J ← 1 Hasta N Con Paso 1
      Escribir J
    Fin Para
  FinSi
FinAlgoritmo
  
```



En este ejercicio, el valor final del bucle, corresponde a una variable, cuyo valor es previamente ingresado.

Problema 9: Diagrama que permita ingresar un número N de VALORES. Si el valor de N es menor a 1, envíe un mensaje de ERROR, en caso contrario para cada valor ingresado, determine siempre y cuando sea POSITIVO si es PAR O IMPAR.



PRUEBA DE ESCRITORIO				
n	valor	J	residuo	Salida
- 3				Error
4	7	~~-1~~	~~-1~~	Valor Impar 7
	10	~~-2~~	~~-0~~	Valor Par 10
	- 5	~~-3~~		
	8	4	0	Valor Par 8

// Ingresa N Valores y determina si son Par o Impar a los valores positivos

Algoritmo Estructura PARA/DESDE

Definir n, j, valor, residuo **Como Entero**

Escribir "Ingrese cuantos Valores a Procesar"

Leer N

Si N < 1 **Entonces**

Escribir "Error"

SiNo

Para j ← 1 **Hasta** N **Con Paso** 1

Escribir "Ingrese Valor"

Leer valor

Si valor > 0 **Entonces**

residuo = valor MOD 2

Si residuo = 0 **Entonces**

Escribir "Valor Par", valor

SiNo

Escribir "Valor Impar", valor

FinSi

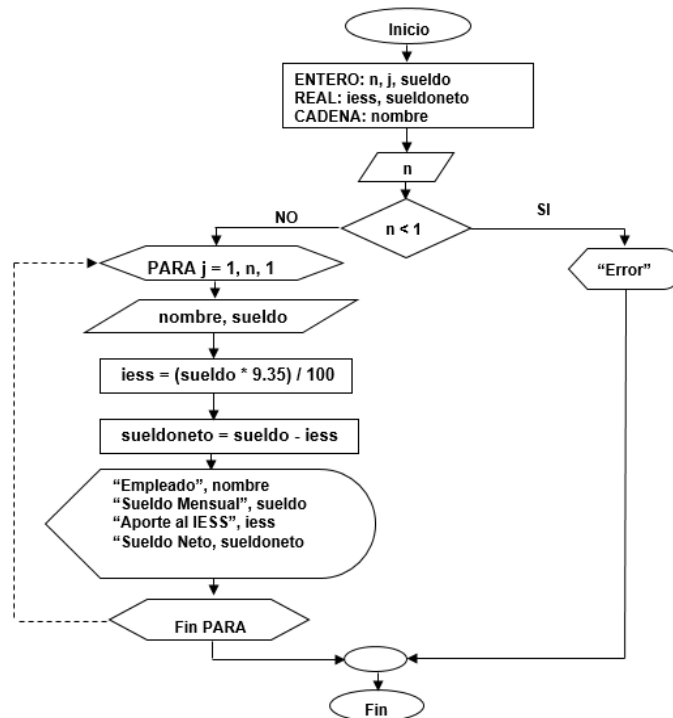
FinSi

Fin Para

FinSi

FinAlgoritmo

Problema 10: Diagrama que permita ingresar un número N de Empleados de una empresa. Si el valor de N (representa el número de empleados) es menor a 1, envíe un mensaje de ERROR y finalice. En caso contrario por cada empleado ingrese: Nombre y Sueldo. Se pide calcular el Aporte al IESS (9.35 % del sueldo) y el valor neto a recibir de cada empleado. Por pantalla muestre los datos ingresados y los cálculos solicitados.



// Procesa N cantidad de empleados

Algoritmo EstructuraPARA/DESDE

Definir n, j, sueldo **Como** Entero

Definir iess, sueldoneto **Como** Real

Definir nombre **Como** Cadena

Escribir "Ingrese cuantos Empleados va a Procesar"

Leer N

Si N < 1 **Entonces**

Escribir "Error"

SiNo

Para j ← 1 **Hasta** N **Con Paso** 1

Escribir "Ingrese Nombre del Empleado"

Leer nombre

Escribir "Ingrese Sueldo Mensual"

Leer sueldo

$iess = (sueldo * 9.35) / 100$

$sueldoneto = sueldo - iess$

Escribir "Empleado", nombre

Escribir "Sueldo Mensual", sueldo

Escribir "Aporte al IESS", iess

Escribir "Sueldo Neto", sueldoneto

Fin PARA

FinSi

FinAlgoritmo

Problemas Propuestos

Problema 1: Diagrama que genere automáticamente los números **PARES** menores a 100.

Problema 2: Diagrama que genere automáticamente los números **IMPARES** positivos menores a 100.

Problema 3: Diagrama que muestre los números del 10 al 500 de 10 en 10.

Problema 4: Utilizando la lógica del problema resuelto No. 3, haga un diagrama que muestre automáticamente la palabra "Manta", N número de veces.

Problema 5: Dado un entero positivo N en el rango de 1 a 100, se alternan los mensajes: "Hola mundo" y "Hello World" una vez en español, la siguiente en inglés, la siguiente en español, y así sucesivamente hasta mostrar N mensajes. Si el valor de N está fuera del rango envíe un mensaje de "Error" y finalice.

Problema 6: Diagrama que dado un número entero N, muestre los 100 números siguientes a N. En caso que el valor ingresado sea menor a 0 envíe un mensaje de "Error" y finalice.

Problema 7: Diagrama que muestre números del 1 al 100 reemplazando los múltiplos de 3 por la palabra "Algoritmo".

Problema 8: Diagrama que permita ingresar dos valores, si el primer valor es mayor al segundo envíe un mensaje de ERROR y finalice, en caso contrario genere los números de forma automática desde el primer valor hasta el segundo valor.

Problema 9: Diagrama que permita mostrar las potencias del número 2 desde 2^1 hasta 2^n . **Considere que** si el valor de ingresado a N es menor a 1, envíe un mensaje de ERROR y finalice el proceso.

Problema 10: Diagrama que genere automáticamente los N primeros números. Si el valor de ingresado a N es menor a 1, envíe un mensaje de ERROR y finalice el proceso. Caso contrario mostrar solo aquellos números generados que sean impares.

Problema 11: Diagrama que permita generar una tabla de multiplicar N desde un valor positivo INICIAL hasta un valor positivo FINAL. En caso que el valor INICIAL, sea mayor al valor FINAL envíe un mensaje de "ERROR" y finalice.

Problema 12: Diagrama que permita generar una tabla de multiplicar previamente ingresada. Si la misma es igual a CERO (0) envíe a pantalla un mensaje de "ERROR", en caso contrario, si la tabla ingresada es **PAR** debe ir de **1 a 12**, si es **IMPAR** de **10 a 1**. Solo debe usar una estructura **PARA** para cualquiera de ambos casos.

Problema 13: En base a la lógica del problema resuelto No. 9, haga un diagrama que permita ingresar un número N de VALORES. Si el valor de N es menor a 1, envíe un mensaje de ERROR, en caso contrario enviar un mensaje indicando si el valor ingresado es: Positivo, Negativo o Cero.

Problema 14: Diagrama que permita mostrar los cuadrados de los números del 1 al 10.

1	1
2	4
3	9
..	..
..	..

10 100

Problema 15: Diagrama que permita transformar grados centígrados desde 0 a 100 a grados Fahrenheit. La fórmula correspondiente para pasar de grados centígrados a fahrenheit es:

$$F = 32 + (9 * C / 5)$$

Problema 16: Diagrama que permita generar los N primeros números (si el valor ingresado es menor a 1, envíe un mensaje de ERROR), de tal forma que calcule y muestre su **cuadrado** y su **cubo**. La información se debe generar de la siguiente forma:

1	1	1
2	4	8
3	9	27
..	..	..
N	..	..

Problema 17: Diagrama que permita generar y mostrar la siguiente serie. La serie se debe generar de la siguiente forma:

1	2	3
4	5	6
7	8	9
..	..	..
..	..	..
98	99	

Problema 18: Diagrama que muestre todos los divisores exactos de un valor previamente ingresado. Considere que si el valor ingresado es menor a 1, envíe un mensaje de error y termine el proceso. Ejemplos:

Si valor = 3, mostrar: 1, 3

Si valor es 6, mostrar: 1, 2, 3, 6

Si valor es 12, mostrar: 1, 2, 3, 4, 6, 12

Problema 19: Diagrama que ingrese un valor, si el valor ingresado es menor a 1, envíe un mensaje de ERROR y finalice el proceso. En caso contrario muestre los primeros múltiplos en base a dicho valor. Ejemplos:

Si valor = 3, mostrar: 3, 6, 9

Si valor = 5, mostrar: 5, 10, 15, 20, 25

Si valor = 8, mostrar: 8, 16, 24, 32, 40, 48, 56, 64

Problema 20: Diagrama que permita ingresar 10 valores diferentes (asumirlos), se solicita encontrar y mostrar el MAYOR valor ingresado.

2.5.8.3.1.2 Contadores y Acumuladores

En programación, a una variable se la clasifica de acuerdo a su **contenido que va almacenar** (numérica, alfanumérica y lógica) y por la **actividad y/o trabajo** que se le dará en el diagrama y/o programa.

En este último caso, se tienen a las **variables de trabajo** (las que se han utilizado para ingresar datos, hacer cálculos, asignaciones, condiciones y mostrar información), las **variables de control de bucles** (controlan el número de ciclos o iteraciones) y las variables de tipo **contador** y **acumulador** que serán estudiadas a continuación.

2.5.8.3.1.2.1 Variable de Tipo Contador

En programación, se llama “contador” a una variable **numérica entera** que permite llevar una cuenta de un **valor constante**, mismo que se **incrementa** (suma) o **decrementa** (resta) al contenido de dicha variable **dentro de un bucle**. Considere lo siguiente para su uso:

- a) La variable que actúa como contador **siempre deberá estar antes y después del signo igual (=) en la expresión**, para que se produzca la cuenta. Es decir, cambia de valor por el hecho de aparecer a la izquierda.

$$\begin{array}{c} \text{Misma Variable} \\ \downarrow \quad \downarrow \\ \text{Variable} = \text{Variable} \left\{ \begin{array}{c} + \\ - \end{array} \right\} \text{Valor Constante} \end{array}$$

Nota: Las llaves indican la selección de una de las opciones.

- b) La variable tipo contador siempre es modificada por el **Valor Constante**, mismo que depende del problema a resolver.

Ejemplos: $c = c + 1$ $x = x + 2$ $n = n + 3$ $w = w - 1$

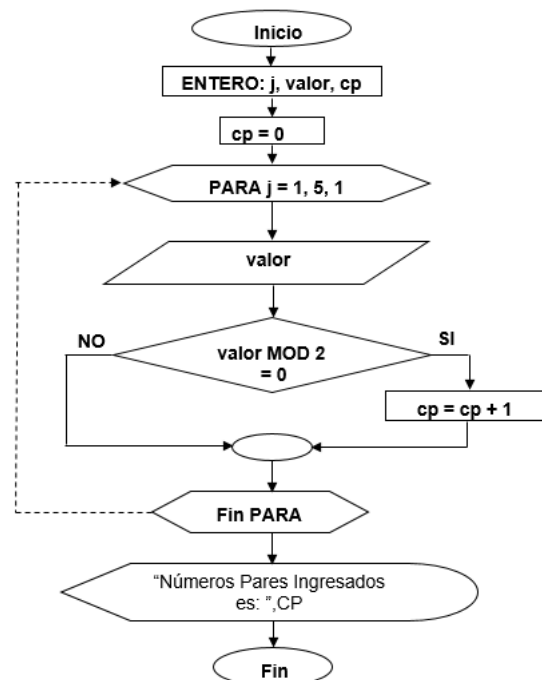
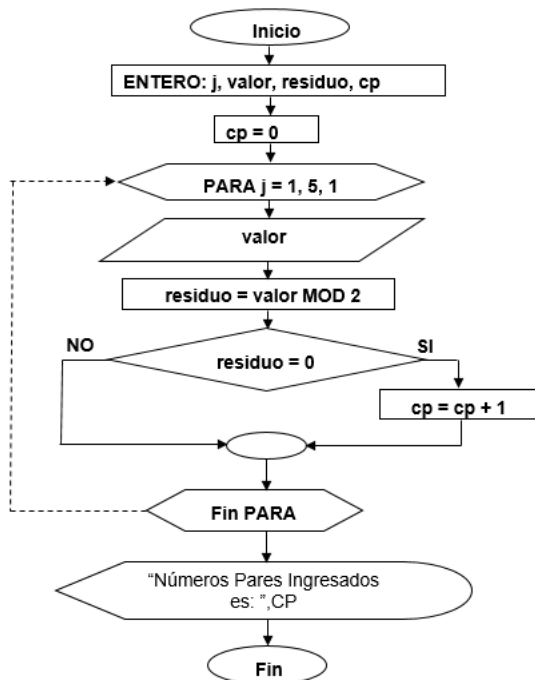
- c) La variable tipo contador **debe ser inicializada** (por lo general al inicio). Este valor inicial dependerá del problema (casi siempre es cero), así también como el lugar a donde debe ser inicializada.

Ejemplos: $c = 0$ $x = 1$ $n = -1$ $w = 2$

- d) La variable tipo contador siempre está vinculada a una condición o a una estructura repetitiva (frecuentemente dentro de un **Para**, **Mientras** o **Repetir Hasta** en diagramación y sus equivalentes en programación).

Problemas con uso de Contadores en la Estructura Repetitiva PARA

Problema 1: Diagrama que permita ingresar cinco valores mediante un bucle. Se solicita contar cuantos valores ingresados son pares.



Como registra la prueba de escritorio, al principio la variable **cp** que es el contador se inicializa con cero antes de ingresar al bucle. Dentro del bucle, a cada valor ingresado, se determina si es divisible exactamente para 2, al guardar el "resto" de la división en la variable "residuo" y preguntar si es igual a cero. Al ser verdadera la condición, el contenido de la variable **cp** se incrementa en 1, produciéndose la cuenta de los valores ingresados que sean pares.

Este diagrama es más eficiente ya que no utiliza la variable "residuo", al hacer directamente la expresión en la condición para determinar el "residuo" mediante el operador MOD.

PRUEBA DE ESCRITORIO

valor	j	residuo	cp	Salida
~~4~~	~~1~~	~~0~~	~~0~~	Números Pares Ingresados es: 3
~~9~~	~~2~~	~~1~~	~~1~~	
~~10~~	~~3~~	~~0~~	~~2~~	
~~3~~	~~4~~	~~1~~	3	
6	5	0		

// Cuenta los valores pares ingresados

Algoritmo EjercicioContador

Definir j, valor, cp **Como** Entero

Para j ← 1 **Hasta** 5 **Con** Paso 1

Escribir "Ingrese Valor"

Leer valor

Si valor MOD 2 = 0 **Entonces**

cp = cp + 1

FinSi

Fin Para

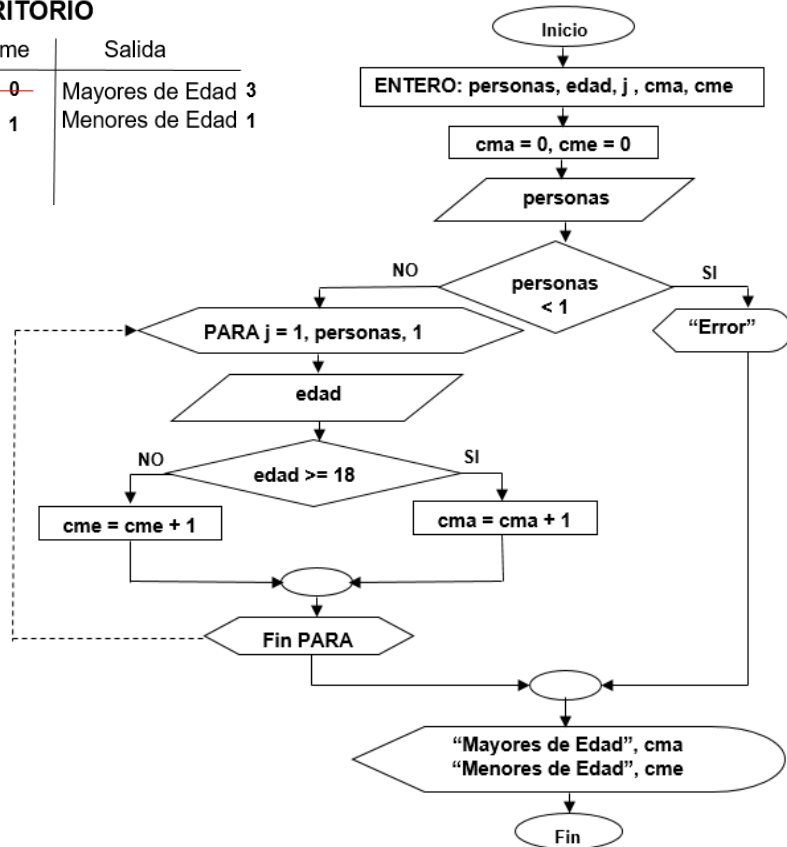
Escribir "Números Pares Ingresados es: ", cp

FinAlgoritmo

Problema 2: Diagrama que permite ingresar las edades de una cantidad de personas. Se solicita contar cuántas son mayores y menores de edad. Considere que si la cantidad de personas es < 1 , enviar un mensaje de Error y finalice el proceso. Al final se muestra los contadores solicitados.

PRUEBA DE ESCRITORIO

personas	edad	j	cma	cme	Salida
4	~~25~~	~~1~~	~~0~~	~~0~~	Mayores de Edad 3
	~~15~~	~~2~~	~~1~~	1	Menores de Edad 1
	~~20~~	~~3~~	~~2~~		
	40	4	3		



En el diagrama se han utilizado dos contadores, que al principio se inicializan con cero:

- **cma**: Para contar los mayores de edad
- **cme**: Para contar los menores de edad

Dependiendo de la edad ingresada, estos contadores de incrementan en 1, mostrando sus valores una vez procesadas todas las personas.

// Cuenta Mayores y Menores de edad de una cantidad de personas

Algoritmo EjercicioContador

Definir personas, edad, j, cma, cme **Como Entero**

cma = 0

cme = 0

Escribir "Ingrese Número de Personas a Procesar"

Leer personas

Si personas < 1 **Entonces**

Escribir "Error"

SiNo

Para j $\leftarrow 1$ **Hasta** personas **Con Paso** 1

Escribir "Ingrese Edad"

Leer edad

Si edad ≥ 18 **Entonces**

cma = cma + 1

SiNo

cme = cme + 1

FinSi

Fin Para

Escribir "Mayores de Edad", cma

Escribir "Menores de Edad", cme

FinSi

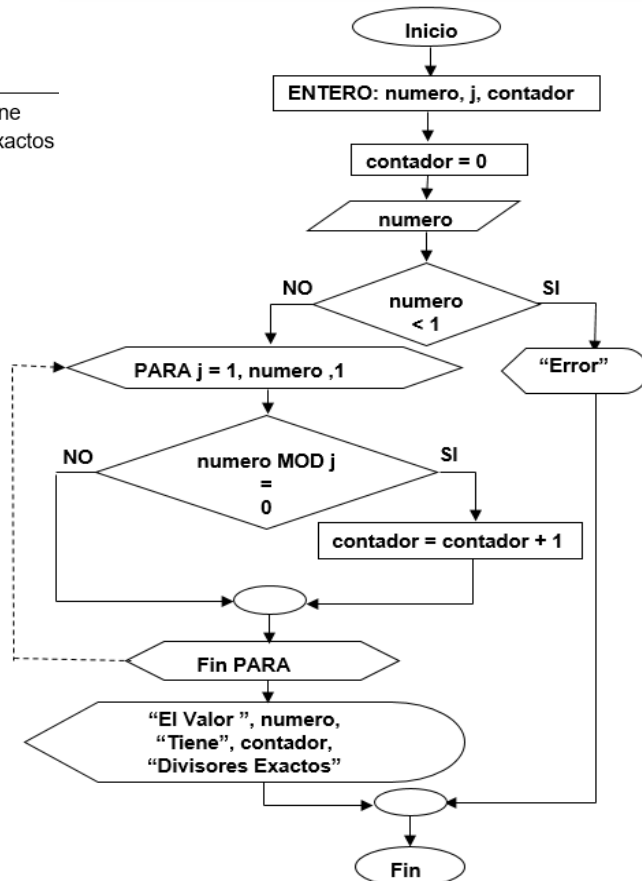
FinAlgoritmo

Problema 3: Diagrama que permite ingresar un número. Si dicho número es menor a 1, envíe un mensaje de Error y finalice el proceso. En caso contrario, calcular y mostrar cuántos divisores exactos tiene el valor ingresado.

Ejemplo: Número 3 ➡ 1,3 Debe mostrar 2
 Número 6 ➡ 1,2,3,6 Debe mostrar 4
 Número 4 ➡ 1,2,4 Debe mostrar 3

PRUEBA DE ESCRITORIO

numero	j	contador	Salida
4	~~1~~	~~0~~	El Valor 4 Tiene 3 Divisores Exactos
	~~2~~	~~1~~	
	~~3~~	~~2~~	
	4	3	



// Cuenta los divisores exactos de un valor ingresado

Algoritmo EjercicioContador

Definir numero, j, contador **Como Entero**

contador = 0

Escribir "Ingrese el Número"

Leer numero

Si numero < 1 **Entonces**

Escribir "Error"

SiNo

Para j ← 1 **Hasta** numero **Con Paso** 1

Si numero MOD j = 0 **Entonces**

contador = contador + 1

FinSi

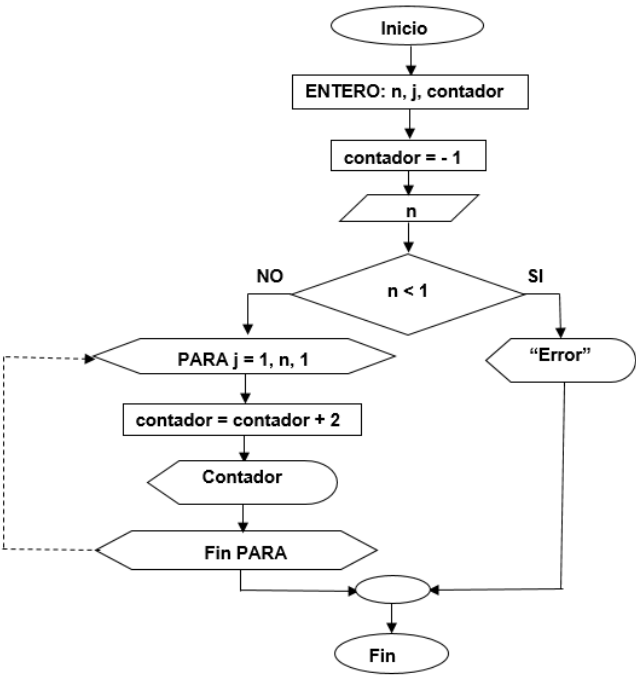
Fin Para

Escribir "El Valor", numero, "Tiene", contador, "Divisores Exactos"

FinSi

FinAlgoritmo

Problema 4: Diagrama que permite generar los N primeros números impares. Si el valor ingresado a N es menor a 1, envía un mensaje de Error y finaliza el proceso.



PRUEBA DE ESCRITORIO

n	j	contador	Salida
5		- 1	
	~~1~~	~~1~~	1
	~~2~~	~~3~~	3
	~~3~~	~~5~~	5
	~~4~~	~~7~~	7
	5	9	9

// Genera los N primeros números impares
Algoritmo EjercicioContador
 Definir n, j, contador **Como Entero**
 contador = - 1
Escribir "Ingrese el Número"
 Leer n
 Si n < 1 Entonces
 Escribir "Error"
 SiNo
 Para j ← 1 **Hasta** n **Con Paso** 1
 contador = contador + 2
 Escribir contador
 Fin Para
FinSi
FinAlgoritmo



En este diagrama el contador se inicializa con un valor de - 1. Al ejecutarse el bucle, el contador se incrementa en 2, con lo que inicia la generación de los valores impares.

Problemas Propuestos con uso de Contadores en la Estructura Repetitiva PARA

Problema 1: Diagrama que permita simular 5 intentos de contraseña ingresados. Contar cuántos intentos fueron correctos, considerando que la contraseña correcta es 123ABC.

Problema 2: Diagrama que permita ingresar N valores. Si el valor ingresado es menor a 1 envíe el mensaje Error y finalice. Caso contrario contar cuantos valores ingresados son: Positivos, Negativos y cuantos son CEROS. Al final muestre lo solicitado.

Problema 3: En un almacén se registran N clientes (si N es menor a 1 envíe un mensaje de error y finalice). Un cliente es frecuente si realizó más de 3 compras en el mes. Haga un diagrama para contar cuántos clientes cumplen esa condición.

Problema 4: En un curso de programación, el profesor desea conocer cuántos estudiantes APRUEBAN y cuantos PIERDEN la asignatura una vez concluido el periodo. El curso está formado por N estudiantes, y por cada estudiante se ingresa la suma total de puntos obtenidos (aprueban con 21 puntos o más). Haga el diagrama respectivo, considerando que si el valor ingresado a N es menor a 1, debe enviar un mensaje de Error. Al final debe mostrar lo solicitado por el docente.

Problema 5: Una empresa tiene N empleados (si N es menor a 1 envíe un mensaje de error y finalice). Haga un diagrama para contar cuántos empleados tienen a) un sueldo menor al salario básico, b) igual al salario básico y c) mayores al salario básico. Al final muestre lo solicitado.

Problema 6: En un almacén se registran N facturas de ventas diarias (si N es menor a 1 envíe un mensaje de error y finalice). Cada venta indica la forma de pago (1=Efectivo, 2=Tarjeta). Haga un diagrama para contar el número de ventas en efectivo y con tarjeta.

Problema 7: Diagrama que usando SOLO una estructura PARA, permita generar y mostrar la siguiente serie, la cual se debe mostrar de la siguiente forma (dos números por línea):

```
1  2
3  4
5  6
.. ..
.. ..
99 100
```

Problema 8: Diagrama que permita contar cuántos números IMPARES existen desde 1 hasta un valor N. Considere, que si el valor ingresado a N es menor a 1 o mayor 100, entonces enviar un mensaje Error y finalizar sin hacer ningún conteo. En caso contrario, haga lo solicitado mostrando el resultado al final.

Problema 9: Diagrama que permita ingresar dos números. Se solicita contar cuántos números hay entre ellos (incluyéndolos).

Problema 10: Crear un diagrama que pida ingresar un valor. Si el valor es menor a 1, entonces finalice. Caso contrario muestre un mensaje indicando si el número es o no primo. Un número es primo, cuando es divisible exactamente para sí mismo y para la unidad.

2.5.8.3.1.2.2 Variables Tipo Acumulador

En programación, se llama “acumulador” a una variable numérica cuya función es “acumular o totalizar” en su interior **valores diferentes**, mismos que se incrementan o decrementan al contenido de la variable dentro de un bucle. Para su uso considere:

- a) La variable que actúa como acumulador **siempre deberá estar antes y después del signo igual (=) en la expresión**, para que se produzca el proceso de acumulación.

$$\begin{array}{c} \text{Misma Variable} \\ \downarrow \quad \downarrow \\ \text{Variable} = \text{Variable} \left\{ \begin{array}{c} + \\ * \\ - \end{array} \right\} \text{Valor Acumular} \end{array}$$

Nota: Las llaves indican la selección de una de las opciones.

- b) La variable tipo acumulador siempre actualiza su contenido usando su propio valor anterior con el **Valor Acumular**. Dicho valor puede ser un dato ingresado o el resultado de una expresión aritmética. Por lo tanto, depende del problema a resolver. Ejemplos:

$\text{total} = \text{total} + \text{suelo}$ $\text{factorial} = \text{factorial} * \text{Valor}$ $x = x - \text{valor}$

- c) La variable tipo acumulador debe **ser inicializada** (por lo general al inicio). Este valor inicial dependerá del problema (casi siempre es cero), así también como el lugar a donde debe ser inicializada. Ejemplos:

$\text{total} = 0$ $\text{factorial} = 1$ $x = 1000$

- d) La variable tipo acumulador siempre está vinculada a una estructura repetitiva (frecuentemente dentro de un **Para**, **Mientras** o **Repetir Hasta** en diagramación y sus equivalentes en programación). Por lo tanto, **recibe valores variables** en cada iteración a medida que se procesan datos.



La diferencia entre un acumulador y un contador radica que **un contador** lleva el incremento o decremento de un **valor constante** y el **acumulador de un valor que varía**.

Problemas con uso de Acumuladores y Estructura Repetitiva PARA

Problema 1: Diagrama que permita calcular y mostrar la suma de los N primeros números positivos. Validar que el valor ingresado a N sea positivo, en caso contrario muestre “Error”. Para realizar la suma el valor a N debe ser positivo.

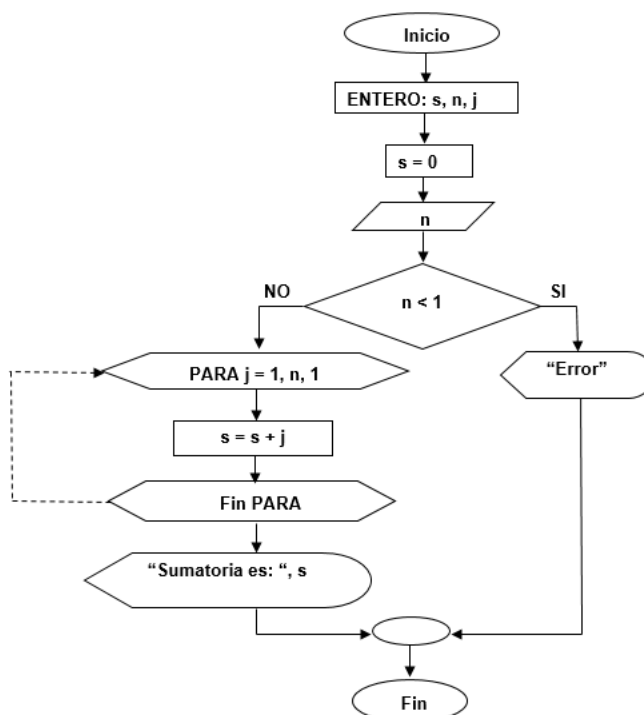
S = 1 + 2 + 3 + ...+ N

PRUEBA DE ESCRITORIO

s	n	j	Salida
~~0~~	5	~~1~~	Sumatoria es: 15
~~1~~		~~2~~	
~~3~~		~~3~~	
~~6~~		~~4~~	
~~10~~		5	
15			

Importante!

Como registra la prueba de escritorio, al principio la variable "s" que es el acumulador se inicializa con cero antes de ingresar al bucle. Dentro del ciclo, cada vez que la variable controladora de bucle se incrementa en 1, dicho contenido se acumula en la variable "s" de forma repetitiva hasta que completa el ciclo.



Importante!

En este diagrama, el valor variable que se acumula corresponde a un valor que se genera mediante la variable controladora de bucle.

// Obtiene la sumatoria de los N primeros números

Algoritmo EjercicioAcumulador

Definir s, n, j **Como** Entero

s = 0

Escribir "Ingrese Valor hasta donde generar"

Leer n

Si n < 1 **Entonces**

Escribir "Error"

SiNo

Para j ← 1 **Hasta** n **Con**

Paso 1

s = s + j

Fin Para

Escribir "Sumatoria es: ", s

FinSi

FinAlgoritmo

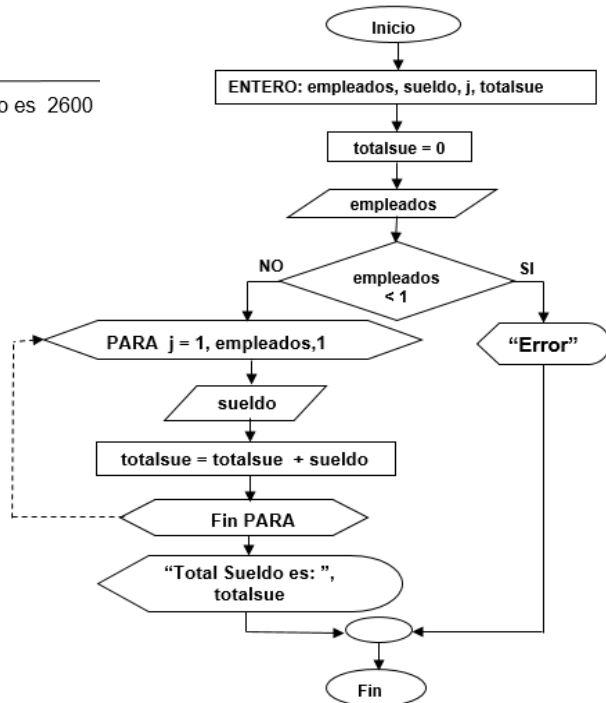
Problema 2: En una empresa que tiene una cantidad de empleados, requiere calcular el total de sueldos que paga mensualmente. El diagrama utiliza una estructura repetitiva para ingresar el sueldo de cada empleado.

PRUEBA DE ESCRITORIO

empleados	sueldo	j	totalsue	Salida
4	~~500~~	~~1~~	~~0~~	Total Sueldo es 2600
	~~400~~	~~2~~	~~500~~	
	~~1000~~	~~3~~	~~900~~	
	700	4	1900	
			2600	



En este diagrama, el valor variable que se acumula corresponde a un dato ingresado, en este caso el sueldo que se ingresa de forma repetitiva.



// Obtiene la sumatoria de sueldos de N empleados

Algoritmo EjercicioAcumulador

Definir empleados, sueldo, j, totalsue **Como Entero**

totalsue = 0

Escribir "Ingrese Número de Empleados"

Leer empleados

Si empleados < 1 **Entonces**

Escribir "Error"

SiNo

Para j ← 1 **Hasta** empleados **Con Paso** 1

Escribir "Ingrese Sueldo"

Leer sueldo

 totalsue = totalsue + sueldo

Fin Para

Escribir "Total Sueldo es: ", totalsue

FinSi

FinAlgoritmo

Problema 3: Diagrama que permita calcular y mostrar el factorial de un valor N previamente ingresado. Validar que el valor ingresado a N sea positivo, en caso contrario muestre "Error". Para calcular el factorial (!) considere el siguiente ejemplo:

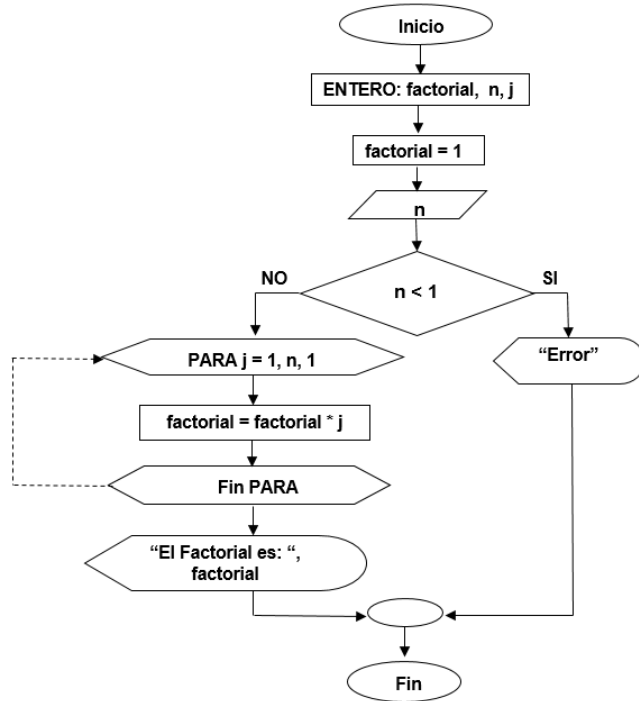
$$n = 5! \quad n = 1 * 2 * 3 * 4 * 5 \quad n = 120$$

PRUEBA DE ESCRITORIO

factorial	n	j	Salida
~~1~~	5	~~1~~	Sumatoria es: 15
~~2~~		~~2~~	
~~6~~		~~3~~	
~~24~~		~~4~~	
120		5	



En este diagrama, el valor variable que se acumula (variable factorial) corresponde a un valor que se genera mediante la variable controladora de bucle. Observe, que para calcular el factorial, dicha variable se inicializa con 1.



// Calcula el factorial de un valor N

Algoritmo EjercicioAcumulador

Definir factorial, n, j **Como** Entero

factorial = 1

Escribir "Ingrese un Valor"

Leer n

Si n < 1 **Entonces**

Escribir "Error"

SiNo

Para j ← 1 **Hasta** n **Con Paso** 1

factorial = factorial * j

Fin Para

Escribir "El Factorial es: ", factorial

FinSi

FinAlgoritmo

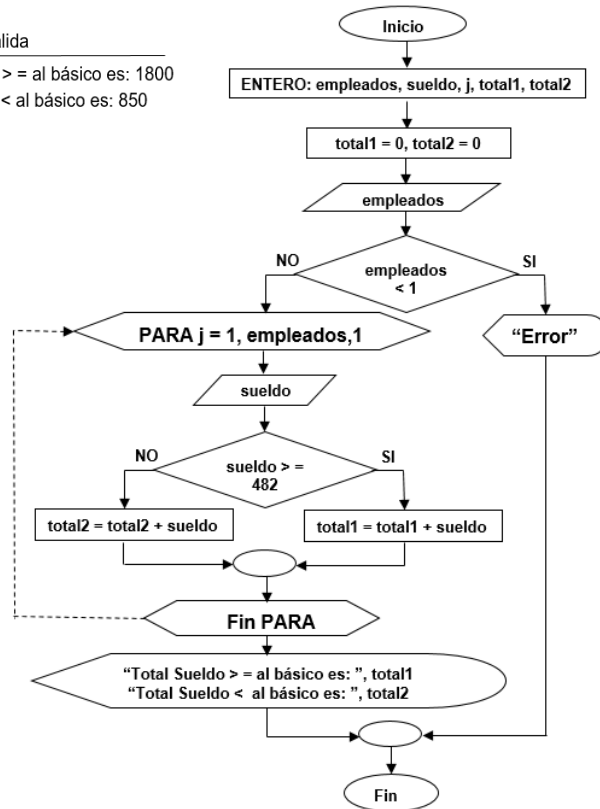
Problema 4: Diagrama que permita ingresar los sueldos mensuales de una cantidad de empleados. Se requiere conocer y mostrar el total de los sueldos mayores y menores al salario mínimo (\$ 482 año, 2026). Considere que si la cantidad de empleados es < 1 , enviar un mensaje de Error y finalizar el proceso.

PRUEBA DE ESCRITORIO

empleados	sueldo	j	total1	total2	Salida
5			-0-	-0-	Total Sueldo $\geq$ al básico es: 1800
	-500-	-1-	-500-		Total Sueldo $<$ al básico es: 850
	-700-	-2-	-1200-		
	-450-	-3-		-450-	
	-600-	-4-	1800		
	400	5		850	



En este diagrama, se utilizan dos acumuladores (total1 y total2), los cuales son inicializados al principio y dentro del ciclo acumulan los sueldos, de acuerdo al valor del salario mínimo. Al final sus contenidos se muestran con los acumulados correspondientes.



// Obtiene los totales de sueldos menores y mayores = al básico

Algoritmo EjercicioAcumulador

Definir empleados, sueldo, j, total1, total2 **Como** Entero

total1 = 0

total2 = 0

Escribir "Ingrese Número de Empleados"

Leer empleados

Si empleados < 1 **Entonces**

Escribir "Error"

SiNo

Para j $\leftarrow 1$ **Hasta** empleados **Con Paso** 1

Escribir "Ingrese Sueldo"

Leer sueldo

Si sueldo ≥ 482 **Entonces**

total1 = total1 + sueldo

SiNo

total2 = total2 + sueldo

FinSi

Fin Para

Escribir "Total Sueldo $\geq$ al básico es: ", total1

Escribir "Total Sueldo $<$ al básico es: ", total2

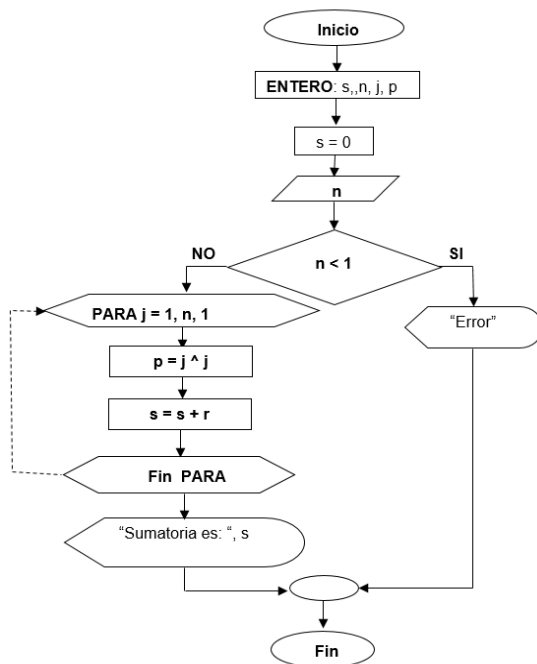
FinSi

FinAlgoritmo

Problema 5: Diagrama que permita calcular la sumatoria de los términos de la siguiente serie:

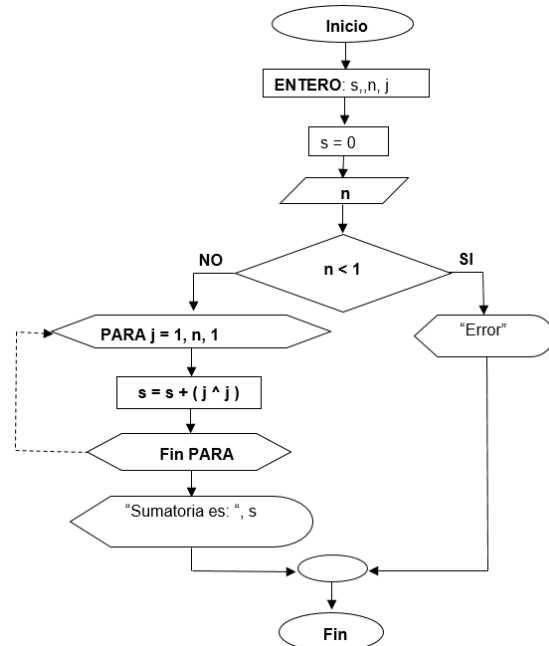
$$S = 1^1 + 2^2 + 3^3 + \dots + N^N$$

Considere que para realizar la serie y obtener la sumatoria el valor de N, que indica la cantidad de términos debe ser positiva, en caso contrario, se envía un mensaje de Error y finalizar el proceso.



En este diagrama, el número de términos de la serie, depende del valor ingresado a la variable "n". En la prueba se ingresó 4, por lo tanto, ese el número de términos de la serie.

Observe que valor que se acumula corresponde a un dato que se calcula, en este caso, el valor de la variable controladora de bucle se eleva a su propio valor (la base siempre será el exponente.. ver cada término de la serie) y dicho resultado se acumula en la variable "s" de forma repetitiva.



PRUEBA DE ESCRITORIO

n	s	r	j	Salida
4	~~0~~			Sumatoria es: 30
	~~1~~	1	~~1~~	
	~~5~~	4	~~2~~	
	~~14~~	9	~~3~~	
	30	16	4	



En este diagrama es más eficiente ya que el valor que se acumula en "s" corresponde a una expresión (base elevada a potencia), con lo que se evita crear una variable y un cálculo adicional (ver diagrama anterior).

```

// Calcula la sumatoria de una serie
Algoritmo EjercicioAcumulador
  Definir s, n, j, p Como Entero
  s = 0
  Escribir "Ingrese Número de Términos de Serie"
  Leer n
  Si n < 1 Entonces
    Escribir "Error"
  SiNo
    Para j ← 1 Hasta n Con Paso 1
      p = j ^ j
      s = s + p
    Fin Para
    Escribir "Sumatoria es: ", s
  FinSi
FinAlgoritmo

```

Problemas Propuestos con uso de Acumuladores y Estructura Repetitiva PARA

Ejercicio 1: El instituto de Meteorología desea conocer el promedio de la temperatura correspondiente a los 31 días del mes de marzo del 2026. Realizar el diagrama correspondiente para ingresar las temperaturas diarias y calcular y mostrar lo solicitado.

Ejercicio 2: Un curso de programación tiene N número de estudiantes (Si el valor ingresado a N es menor a 1, envíe un mensaje de Error y finalice). Se requiere un diagrama que permita ingresar las calificaciones de los N estudiantes y al final calcular y mostrar el promedio aritmético de calificaciones del curso.

Problema 3: Diagrama que permita el ingreso de N número de valores (Si el valor ingresado a N es menor a 1, envíe un mensaje de Error y finalice). Se solicita sumar solo los valores positivos ingresados. Si se ingresa un valor negativo, ignórelo (no lo sume) pero cuente cuántos errores de ingreso hubo. Al final muestre la suma válida y el conteo de errores que se han procesado.

Ejercicio 4: La meta de María es ahorrar \$ 5000 en su cuenta de ahorros en un año (1 depósito por mes). Se pide realizar un diagrama que permita el ingreso de los depósitos y al final determinar:

- Si el total de depósito es igual a meta propuesta.
- Si el total de depósito no llegó a la meta propuesta y cuanto faltó.

- c) Si el total de depósito fue superior a la meta propuesta y en cuanto lo superó.

Ejercicio 5 Juan tiene \$1000 de ahorro y quiere llevar un registro propio de retiros en su cuenta de ahorros. Para ello se pide realizar un diagrama que permita el registro de 10 retiros y si en algún momento del ciclo (de los 10 retiros) el acumulado de registro supera los \$1000, muestre un mensaje “Alerta saldo en negativo” y el valor que “sobrepase al saldo”, dando por finalizado el ciclo antes de llegar a 10 (asigne 10 al valor de la variable controladora de bucle, esta forma termina el ciclo).

Problema 6: Diagrama que permita calcular el interés compuesto mensual de una inversión de \$5000 que genera el 5 % de interés mensual (hágalo para para 12 meses). En cada mes, calcule y muestre el interés, acumúlalo al capital y usa ese nuevo capital para el siguiente mes.

Problema 7: La ciudad de Manta, tiene actualmente una población promedio de 260 000 habitantes y una tasa de crecimiento poblacional estimada del 2 % anual. Se requiere construir un diagrama que permita calcular y mostrar para los siguientes 10 años:

- a) El total de habitantes nuevos por cada año.
- b) El total de habitantes que tendrá la ciudad después de los 10 años.

Problema 8: Haga un diagrama que permita ingresar un número N de valores. Si el valor ingresado a N es menor a 1, envíe un mensaje de Error y finalice, en caso contrario:

- a) Calcular y mostrar la suma de los valores pares ingresados.
- b) Calcular y mostrar la suma de los valores impares ingresados.
- c) Calcular y mostrar la suma total de valores ingresados.

Ejercicio 9: Usando la lógica del problema resuelto No. 4, se pide:

- a) Calcular y mostrar el promedio de los sueldos mayores e iguales al salario básico.
- b) Calcular y mostrar el promedio de los sueldos menores al salario básico.

c) Calcular y mostrar el total de sueldos ingresados.

Ejercicio 10: En un almacén se debe realizar un proceso de facturación de una compra con N cantidad de transacciones (Si el valor ingresado a N es menor a 1, envíe un mensaje de Error y finalice). Por cada transacción se ingresa: El producto, cantidad y precio unitario. Se pide crear un diagrama que permita el ingreso de las N transacciones, calculando y mostrando el pago por cada producto y la información ingresada de él. Adicionalmente al final se pide mostrar y calcular el total de las transacciones, el impuesto al IVA (15 %) y el total que debe pagar el cliente por su factura.

Problema 11: Diagrama que permita generar una tabla de multiplicar N (Si el valor ingresado a N es menor a 1, envíe un mensaje de Error y finalice) que vaya de 1 a 12. Se pide mostrar dicha tabla y al final mostrar la suma de todos los resultados obtenidos de dicha tabla.

Por ejemplo: n con 3, se genera:

3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
.. ..
.. ..
3 * 12 = 36

al final se muestra: 234

Problema 12: Mediante un diagrama resuelva la siguiente serie:

$$S = \frac{1}{2} + \frac{2}{4} + \frac{3}{6} + \dots + \frac{N}{N*2}$$

Nota: Para realizar la serie el valor a N debe ser positivo. En caso contrario, envíe un mensaje de Error y finalice.

Problema 13: Mediante un diagrama resuelva la siguiente serie:

$$S = 1^{1+1} + 2^{2+1} + 3^{3+1} + \dots + N^{N+1}$$

Nota: Para realizar la serie el valor a N debe ser positivo. En caso contrario, envíe un mensaje de Error y finalice.

Problema 14: Mediante un diagrama resuelva la siguiente serie:

$$S = \left(\frac{1}{2}\right)^1 + \left(\frac{2}{4}\right)^2 + \left(\frac{3}{6}\right)^3 + \dots + \left(\frac{N}{N^2}\right)^N$$

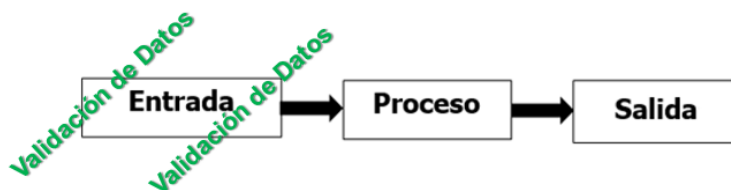
Nota: Para realizar la serie el valor a N debe ser positivo. En caso contrario, envíe un mensaje de Error y finalice.

Problema 15: Diagrama que permita ingresar un número. Si dicho número es menor a 1, envíe un mensaje de Error y finalice el proceso. Caso contrario, determine con un mensaje si un número es o no ABUNDANTE. Un número es ABUNDANTE cuando la suma de sus divisores propios o exactos (excepto él mismo) es mayor que él. Ejemplos:

- Número: **12** 1,2,3,4,6 Suma es **16** Es un número ABUNDANTE
- Número: **9** 1,3 Suma es **4** No es número ABUNDANTE

2.5.8.3.1.3 La Validación de Datos

La validación de datos se define como: “El proceso que consigue evitar y/o minimizar la introducción de datos incorrectos antes de ser procesados en una aplicación”. Es considerada una parte esencial en el procesamiento de datos, por lo que debe ser incluida en la entrada, como elemento en la estructura de un algoritmo.



2.5.8.3.1.4 Importancia de la Validación de Datos

Cuando se usan procesos de validación, el desarrollador y/o programador se asegura que el código escrito para la aplicación incorpore “**acciones de validación**”, lo que garantiza que se cumple con “buenas prácticas” en la construcción del software. Es importante tener presente que la validación:

- Evita y/o minimiza que un programa tenga “fallas” en su ejecución, por los datos proporcionados de entrada.
- Se asegura la entrega de datos limpios y claros a los programas que lo utilizan.

- Garantiza que los datos cumplen con los requisitos y parámetros de calidad en base al problema a resolver.
- No confiar en los datos ingresados por un usuario, por existir posibilidades de equivocación al introducir datos “no válidos”.
- La validación debe ser considerada para todo dato de entrada proporcionado por el usuario.
- Determinadas herramientas de desarrollo incorporan métodos y técnicas de validación.



La **validación** responde a esta pregunta: ¿Este dato es correcto y seguro para que el programa lo utilice?... Aunque **no puede garantizar al 100 %** que los datos sean realmente precisos y consistentes.

2.5.8.3.1.5 Controles de Validación de Datos

En programación, antes de procesar cualquier información, es fundamental **validar que los datos de entrada** sean los correctos, para evitar o minimizar fallas en la ejecución. Por lo tanto, la implementación de controles de validación hace posible que se comprueben la validez de los datos. Esto se logra a través de algunos de los tipos de validación (tabla 13).

Tabla 13

Controles básicos de Validación de Datos

Tipo de Validación	Descripción
Verificación de rango	Comprueba que los datos se encuentran entre un valor aceptable superior e inferior, es decir, dentro de un rango establecido. Ejemplos: Valor sea positivo entre 1 y 999. Sueldo No menor al salario mínimo.
Verificación de tipo de dato	Comprueba que los datos ingresados son de un tipo esperado. Ejemplos: - Un número – Una Fecha – Un tipo cadena
Comprobación de longitud	Comprueba que un dato cumpla con una cantidad mínima o máxima de caracteres. Ejemplos: - Una cédula – Una contraseña de 8 caracteres

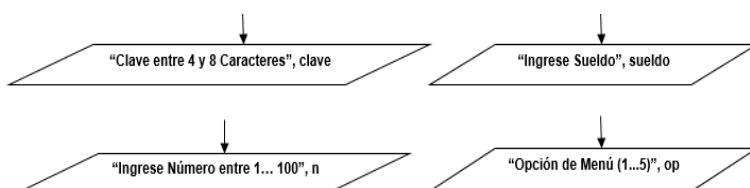
Tipo de Validación	Descripción
Comprobación de presencia	Comprueba que el usuario al menos ha introducido algo como dato, evitando que accidentalmente lo deje vacío.
Cuadro desplegable (En programación gráfica)	Asegura que el usuario solo puede elegir una opción predefinida de una lista, reduciendo las posibilidades de errores ortográficos o respuestas no deseadas. Ejemplo: Combos – Objeto de fechas.

Nota. Fuente: Elaboración propia en base a información consultada en diferentes sitios en la WEB

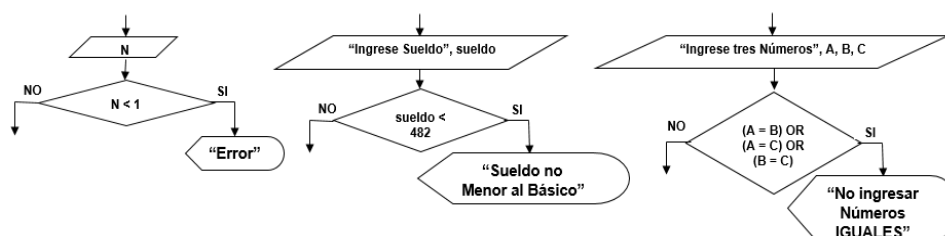
2.5.8.3.1.6 Tipos de Mensajes en la Validación de Datos

Se pueden mostrar dos (2) mensajes:

- a) Uno que aparece **ANTES** que el usuario introduzca los datos. Estas etiquetas o mensajes se ubican como **“títulos”** en los ingresos, dando al usuario información de referencia acerca de lo que debe ingresar. Ejemplos:



- b) Otro que aparece **DESPUÉS** que el usuario intente introducir datos incorrectos o que no cumplan los requisitos de procesamiento. Estas etiquetas o mensajes enviados a pantalla, dan a conocer que se ha producido algún inconveniente con el dato ingresado, por lo tanto, la validación se ha ejecutado y avisa al usuario. Ejemplos:

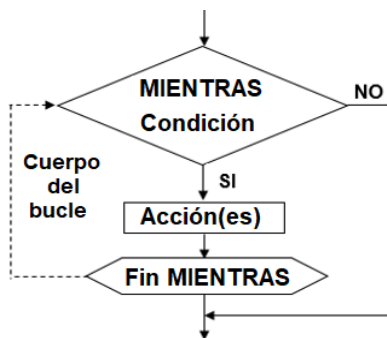


Con el uso de estructuras repetitivas como **“mientras”** y **“repetir hasta”**, generalmente ya no se emplean la forma de mensaje del literal **“b”** cuando los datos no cumplan con los requisitos de procesamiento, en lugar de aquello, se prefiere que

el dato a ingresar este dentro de un bucle y se válida hasta que el usuario lo ingrese correctamente, como se verá más adelante.

2.5.8.3.1.7 Estructura Repetitiva Mientras (While)

La estructura repetitiva MIENTRAS (WHILE) permite realizar un bucle, sin conocer el número de ciclos a ejecutar. Por lo tanto, ejecuta un bloque de instrucciones mientras una condición sea verdadera. En diagramación se usa el siguiente esquema:



Como se observa en el esquema, la estructura **MIENTRAS** evalúa una condición **antes** de ejecutar el bloque de instrucciones.

¿Cómo funciona?

Al ejecutarse **por primera vez** la estructura, la condición que va en **MIENTRAS** es evaluada. Si la condición resulta verdadera, se ejecuta la secuencia de "Acción(es)" que forman el cuerpo del ciclo. Al llegar a **Fin MIENTRAS**, se vuelve a evaluar la condición y si es verdadera, la ejecución del cuerpo de bucle se repite, en caso contrario sale del bucle.

Consideraciones para utilizar la estructura MIENTRAS

- Se puede dar el escenario donde la secuencia de "Acción(es)" del bucle no se ejecute nunca, si al evaluar **por primera vez** la condición resulta ser falsa.
- Se recomienda que la(s) variable(s) que actúa(n) en la condición (controladoras del ciclo) **siempre** debe(n) ser inicializada(s), para que dicha condición sea **verdadera** la primera vez y pueda entrar al ciclo **MIENTRAS**.
- Generalmente es aplicada para procesos de validación de datos de entrada y para procesos repetitivos que requieren ser controlados por una condición.

La estructura MIENTRAS en Pseudocódigo, se presenta de la siguiente forma:

Mientras condición Hacer

Bloque de instrucciones

FinMientras



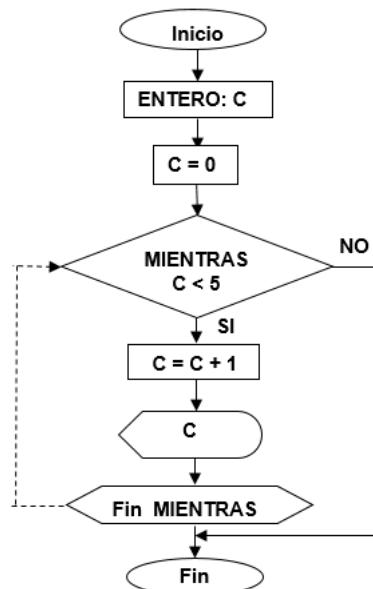
MIENTRAS es una estructura de repetición **PRECONDICIONAL**, es decir, que **depende de que una condición se cumpla** para ejecutar el bloque de instrucciones.

Problemas Resueltos con uso de la estructura MIENTRAS

Problema 1: Diagrama que genere automáticamente los números del uno al cinco.



El diagrama se inicia con el contador "c" con cero, evalúa la condición y al ser verdadera entra al bucle a incrementar y mostrar el contador. El proceso se repite, hasta que la condición sea falsa.



PRUEBA DE ESCRITORIO

C	SALIDA
~~0~~	
~~1~~	1
~~2~~	2
~~3~~	3
~~4~~	4
5	5

// Genera los 5 primeros números con un contador

Algoritmo EjercicioMIENTRAS

Definir c Como Entero

c = 0

Mientras c < 5 Hacer

c = c + 1

Escribir c

FinMientras

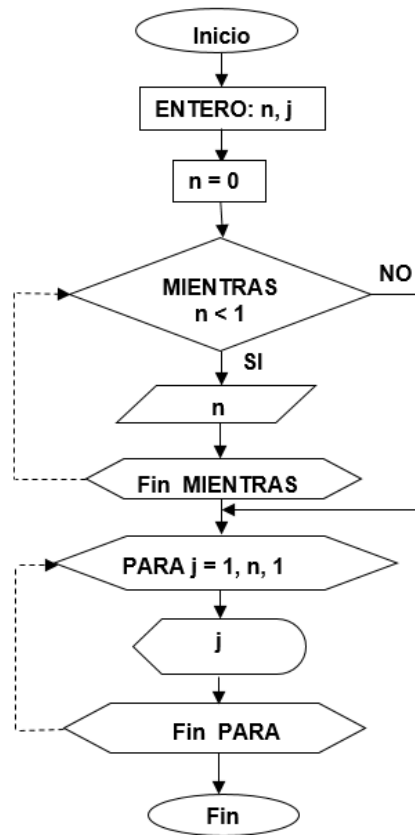
FinAlgoritmo

Problema 2: Diagrama que genera automáticamente los N primeros números. Se valida que el valor ingresado a N no debe ser menor a 1 para realizar lo solicitado.



El diagrama se inicia con la variable “n” con cero. De esta forma se “**obliga**” a que la condición de la estructura sea “**verdadera**” y se entra al bucle. Dentro del bucle se ingresa el valor a la variable “n”, mientras se ingrese un valor “**negativo**”, no saldrá del bucle. Solo cuando a la variable “n” se ingrese un valor “**positivo**” saldrá del bucle a ejecutar la estructura **PARA**, generando los N primeros números.

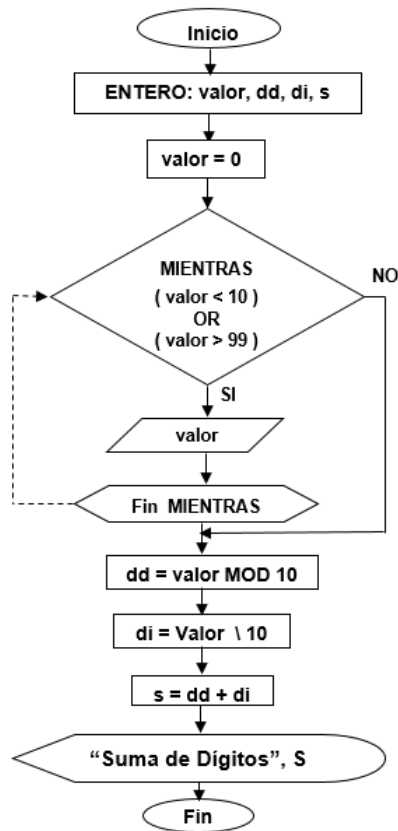
```
// Genera los N primeros números
Algoritmo EjercicioMIENTRAS
  Definir n, j Como Entero
  n = 0
  Mientras n < 1 Hacer
    Escribir “ingrese número”
    Leer n
  FinMientras
  Para j ← 1 Hasta n Con Paso 1
    Escribir j
  Fin Para
FinAlgoritmo
```



PRUEBA DE ESCRITORIO

n	j	SALIDA
~~0~~	~~1~~	1
~~-6~~	~~2~~	2
~~-2~~	~~3~~	3
4	4	4

Problema 3: Diagrama que permita la suma de los dígitos de un valor previamente ingresado. Considere que el valor debe estar entre 10 y 99 ambos incluidos.



PRUEBA DE ESCRITORIO

valor	dd	di	s	Salida
~~0~~				
~~5~~				
~~100~~				
35	5	3	8	Suma de Dígitos 8



El diagrama se inicia con la variable “valor” con cero. De esta forma se “obliga” a que la condición de la estructura sea “verdadera”, validando que mientras se tenga un **valor < 10** o **> 99** se entre al bucle. Dentro del bucle se ingresa un número a la variable “valor” y mientras se siga cumpliendo la condición no saldrá del bucle. Solo cuando a la variable “valor” se ingrese un número entre **10 y 99** (ambos inclusive) saldrá del bucle a ejecutar los procesos que permiten desglosar el número ingresado, sacando el dígito de la derecha (variable **dd**) y el dígito a la izquierda (variable **di**), mostrando la suma respectiva.

// Muestra la suma de los dígitos de un valor entre 10 y 99

Algoritmo EjercicioMIENTRAS

Definir valor, dd, di, s **Como Entero**

valor = 0

Mientras (valor < 1) **OR** (valor > 99) **Hacer**

Escribir “Ingrese número entre 10 y 99”

Leer valor

FinMientras

dd = valor **MOD** 10

di = valor \ 10

s = dd + di

Escribir “Suma de Dígitos”,s

FinAlgoritmo

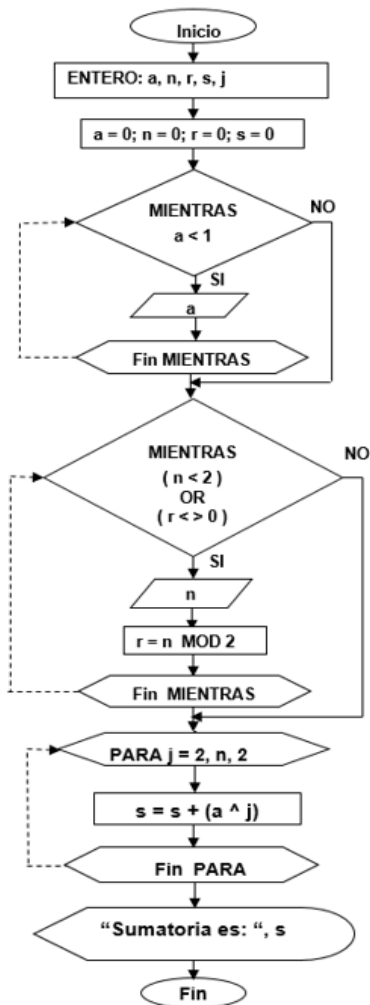
Problema 4: Diagrama que genere la siguiente serie:

$$S = A^2 + A^4 + A^6 + \dots + A^N$$

Considerar que los valores ingresados deben ser **validados**:

A ➡ Representa la Base y debe POSITIVO y Mayor a 1

N ➡ Representa la Máxima potencia y debe ser POSITIVO y PAR.



// Genera una serie de números con su sumatoria

Algoritmo EjercicioMIENTRAS

Definir a, n, r, s, j **Como Entero**

a = 0

n = 0

r = 0

s = 0

Mientras a < 1 **Hacer**

Escribir "Ingrese valor para la Base"

Leer a

FinMientras

Mientras (n < 2) **OR** (r < > 0) **Hacer**

Escribir "Ingrese valor de Máxima Potencia"

Leer n

 r = n MOD 2

FinMientras

Para j ← 2 **Hasta** n **Con Paso** 2

 s = s + (a ^ j)

Fin Para

Escribir "Sumatoria es: ",s

FinAlgoritmo

Problemas Propuestos con uso de la estructura MIENTRAS

Problema 1: Diagrama que utilizando **SOLO contadores y una estructura MIENTRAS**, permita generar los siguientes números (2 números por línea):

1 100
2 99
3 98
.. ..
.. ..

100 1

Problema 2: Diagrama que permita procesar N número de personas (validar que sea > 0). Por cada persona ingresar su edad (validar que estén entre 0 – 100, ambos inclusive). Como proceso calcular y mostrar:

- a) El promedio de edades
- b) Cantidad de personas mayores de edad
- c) Cantidad de personas menores de edad
- d) Cantidad de personas cuya edad sea = a 18 años

Problema 3: Diagrama que permita procesar N número de préstamos en una cooperativa (validar que sea > 0). Ingresar las cantidades prestadas (validar que estén entre 100 y 5000, ambos inclusive). Como proceso calcular y mostrar:

- a) El total prestado
- b) El total de préstamos mayor a \$1000
- c) El total de préstamos menores o iguales a \$1000

Problema 4: Diagrama que permita procesar N número de estudiantes en un curso de programación (validar que N este entre 10 – 50, ambos inclusive). Por cada estudiante ingresar su calificación (validar la calificación 0 – 100, ambos inclusive). Como proceso se pide calcular y mostrar:

- a) Promedio general de calificaciones
- b) Cantidad de aprobados (≥ 70)
- c) Cantidad de reprobados (< 70)
- d) Cantidad de sobresalientes (≥ 90)

- e) Determinar si el curso tiene rendimiento "ALTO" (promedio ≥ 85), "MEDIO" (70–84) o "BAJO" (< 70)

Problema 5: Diagrama que permita procesar N número de vendedores a nivel nacional de un almacén de electrodomésticos (validar que N este entre 10 – 100, ambos inclusive). Por cada vendedor se ingresa su venta mensual (validar la venta entre 100 – 5000, ambos inclusive). Como proceso se pide calcular y mostrar:

- a) Total vendido en el almacén
- b) Cantidad de ventas menores o iguales a 1000
- c) Cantidad de ventas menores o iguales a 2000
- d) Cantidad de ventas menores o iguales a 3000
- e) Cantidad de ventas menores o iguales a 4000
- f) Cantidad de ventas mayores o iguales a 5000
- g) Promedio de ventas del almacén

Problema 6: Diagrama que permita procesar N número de empleados de una empresa procesadora de pescado (validar que N este entre 10 – 100, ambos inclusive). Por cada empleado se ingresa:

- Nombre
- Tipo Empleado (validar entre 1 y 2, ambos inclusive) 1= Administrativo y 2= Planta
- Número de horas trabajadas al mes (validar entre 160 y 180, ambas inclusive)
- Valor por Hora (validar 7 y 8, ambas inclusive para Administrativo) - (validar 9 y 10, ambas inclusive para Planta)

Como proceso se pide calcular y mostrar:

- a) Sueldo
- b) Bonificación (administrativo el 10 % del sueldo, Planta el 15 % del sueldo)
- c) Total ingresos
- d) Iess
- e) Total a Pagar

Problema 7: Diagrama que genere la siguiente serie:

$$S = (1 + 1) + (2 + 2) + (3 + 3) + ... + (N + N)$$

Validar que el valor ingresado a N debe ser positivo

Problema 8: Diagrama que genere la siguiente serie:

$$S = A^0 + A^1 + A^2 + ... + A^N$$

Validar que el valor ingresado a N debe ser mayor o igual a CERO y el valor para A debe estar entre 1 y 9 ambos inclusive.

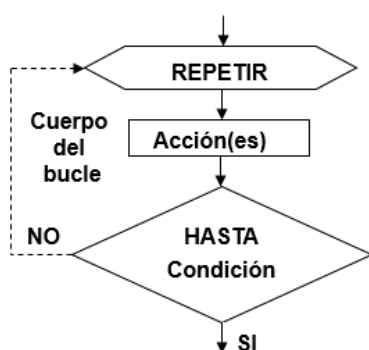
Problema 9: Diagrama que permita generar una tabla de multiplicar cualquiera (validar que el valor sea positivo y se encuentre entre 1 y 20, ambos inclusive). Se pide que la tabla se genere desde un valor "X" (validar que sea positivo) hasta un valor "Y" (validar que sea mayor a "X").

Problema 10: Diagrama que permita ingresar un número (validar que sea positivo). Se pide determinar con un mensaje si un número es o no PERFECTO. Un número es PERFECTO cuando la suma de sus divisores propios o exactos (excepto él mismo) es igual que él. Ejemplos:

- Número: **12** 1,2,3,4,6 Suma es **16** No es un número PERFECTO
- Número: **6** 1,2,3 Suma es **6** Es un número PERFECTO

2.5.8.3.1.8 Estructura Repetitiva Repetir/Hasta (Repeat/Until)

La estructura repetitiva REPETIR/HASTA (REPEAT/UNTIL) permite realizar un bucle sin conocer el número de ciclos a ejecutar. Por lo tanto, ejecuta un bloque de instrucciones hasta que una condición sea verdadera. En diagramación se usa el siguiente esquema:



Como se observa en el esquema, la estructura **REPETIR** ejecuta primero el bloque de instrucciones y **después evalúa** una condición para decidir si continúa o termina el bucle.

¿Cómo funciona?

Al llegar a la estructura, entra al ciclo y ejecuta la secuencia de “Acción(es)” que forman el cuerpo del ciclo. Al llegar a **HASTA**, evalúa la condición. Si la condición es verdadera sale del bucle; si la condición es falsa el flujo de control retorna a **REPETIR** nuevamente el cuerpo de bucle.

Consideraciones para utilizar la estructura REPETIR

- Con esta estructura, el cuerpo del bucle (la acción a repetir) se ejecuta al menos una vez, sin importar el estado de la condición.
- En esta estructura, la **condición** que controla el ciclo va al **final**.
- A diferencia de la estructura **MIENTRAS**, no hay necesidad de inicializar la(s) variable(s) que actúa(n) en la condición de la estructura.
- En algunos lenguajes de programación, se la denomina como: **Repeat/Until** o **Do While**.

La estructura **REPETIR** en Pseudocódigo, se presenta de la siguiente forma:

Repetir

Bloque de instrucciones

Hasta Que Condición



REPETIR es una estructura de repetición **POSTCONDICIONAL**, es decir, que ejecuta primero el bloque de instrucciones y después evalúa la condición.

Problemas Resueltos con uso de la estructura REPETIR

Problema 1: Diagrama que genere los N primeros números. Validar que el valor ingresado a N debe ser POSITIVO. Se utilizarán estructuras REPETIR para validar y generar los números solicitados.

```
graph TD
    Inicio([Inicio]) --> Enter[ENTERO: n, j]
    Enter --> J0[j = 0]
    J0 --> Repetir1{REPETIR}
    Repetir1 --> N[/n/]
    N --> Hasta1{HASTA  
n > 0}
    Hasta1 -- NO --> Repetir1
    Hasta1 -- SI --> Repetir2{REPETIR}
    Repetir2 --> Jplus[j = j + 1]
    Jplus --> J[/j/]
    J --> Hasta2{HASTA  
j = n}
    Hasta2 -- NO --> Repetir2
    Hasta2 -- SI --> Fin([Fin])
```

PRUEBA DE ESCRITORIO

n	j	Salida
~~-6~~	~~0~~	
~~0~~		
4	~~1~~	1
	~~2~~	2
	~~3~~	3
	4	4

 **Importante!**

El diagrama utiliza dos estructuras REPETIR. La primera estructura permite ingresar un valor a la variable “n”. Cuando el control llega a **HASTA** evalúa la condición. Si el contenido de la variable “n”, es mayor a cero (positivo) sale del bucle, en caso de ser falsa la condición, el flujo de control retorna a **REPETIR** nuevamente el ingreso.

La segunda estructura genera los números, para ello la variable “j” que actúa como contador, esta inicializada con cero. Al entrar a la estructura empieza su conteo y se muestra su valor, **HASTA** que su contenido sea igual a la variable “n”, lo que permite salir del bucle, en caso contrario se **REPITE** el proceso de conteo.

// Muestra los N primeros números con 2 estructuras REPETIR

Algoritmo EjercicioREPETIR

Definir n, j Como Entero

Repetir

 Escribir “Ingresa un Número Positivo”

 Leer n

 Hasta Que n > 1

 Repetir

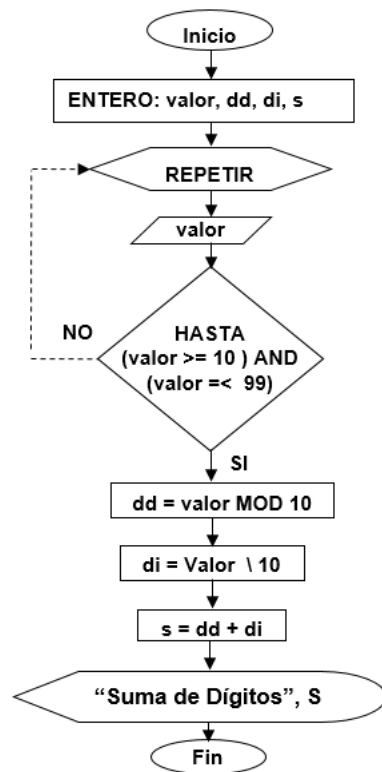
 j = j + 1

 Escribir j

 Hasta Que j = n

FinAlgoritmo

Problema 2: Diagrama que permite la suma de los dígitos de un valor previamente ingresado. Considere que el valor debe estar entre 10 y 99 ambos inclusive.



PRUEBA DE ESCRITORIO

valor	dd	di	s	Salida
~~0~~				
~~5~~				
~~100~~				
35	5	3	8	Suma de Dígitos 8



El diagrama se inicia entrando al bucle e ingresado un número a la variable “**valor**”. Cuando el control llega a **HASTA** evalúa la condición. Si la variable **valor**, tiene un contenido **>=10 y <= 99** sale del bucle, en caso de ser falsa la condición, el flujo de control retorna a **REPETIR** nuevamente el ingreso.

Al salir del bucle, se ejecutan los procesos que permiten desglosar el número ingresado, sacando el dígito de la derecha (variable **dd**) y el dígito a la izquierda (variable **di**), mostrando la suma respectiva.

// Muestra la suma de los dígitos de un valor entre 10 y 99 ambos inclusive

Algoritmo EjercicioREPETIR

Definir **valor, dd, di, s** Como Entero

Repetir

Escribir “Ingresa número entre 10 y 99”

Leer **valor**

Hasta Que (**valor** >=10) **AND** (**valor** <= 99)

dd = **valor** **MOD** 10

di = **valor** \ 10

s = **dd** + **di**

Escribir “Suma de Dígitos”, **s**

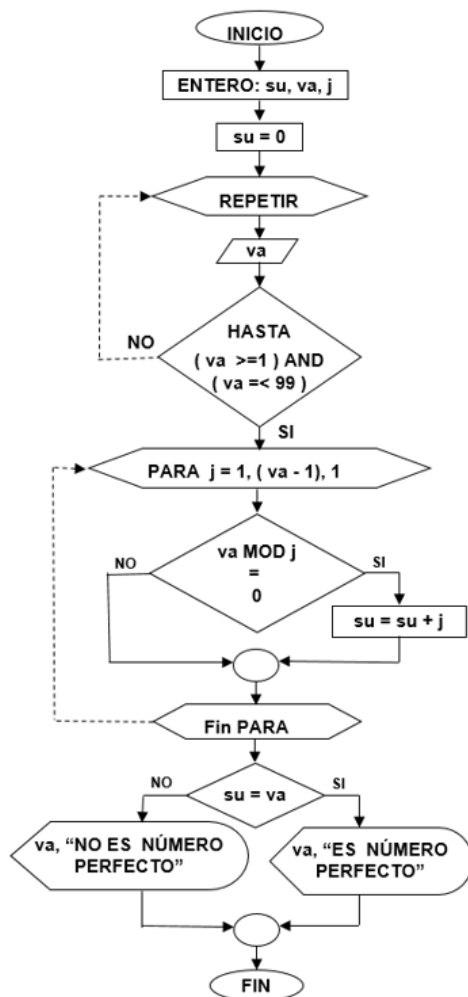
FinAlgoritmo

Problema 3: Diagrama que permita ingresar un número (validar que sea positivo).

Se pide determinar con un mensaje si un número es o no PERFECTO. Un número es PERFECTO cuando la suma de sus divisores propios o exactos (excepto él mismo) es igual que él. Ejemplos:

Número: **12** 1, 2, 3, 4, 6 Suma es **16** No es un número PERFECTO

Número: **6** 1, 2, 3 Suma es **6** Es un número PERFECTO



PRUEBA DE ESCRITORIO

su	va	j	Salida
	~~-5~~		
	~~0~~		
	~~100~~		
~~0~~	6	1	6 ES NÚMERO PERFECTO
~~1~~		2	
~~3~~		3	
6		4	
		5	

// Determina si un número es o no PERFECTO

Algoritmo Ejercicio REPETIR

Definir su, va, j **Como** Entero

su = 0

Repetir

Escribir "Ingrese número entre 1 y 99"

Leer va

Hasta Que (va >= 1) **AND** (va <= 99)

Para j ← 1 **Hasta** (va - 1) **Con Paso** 1

Si va MOD j = 0 **Entonces**

su = su + j

FinSi

Fin Para

Si su = va **Entonces**

Escribir va, "ES NÚMERO PERFECTO"

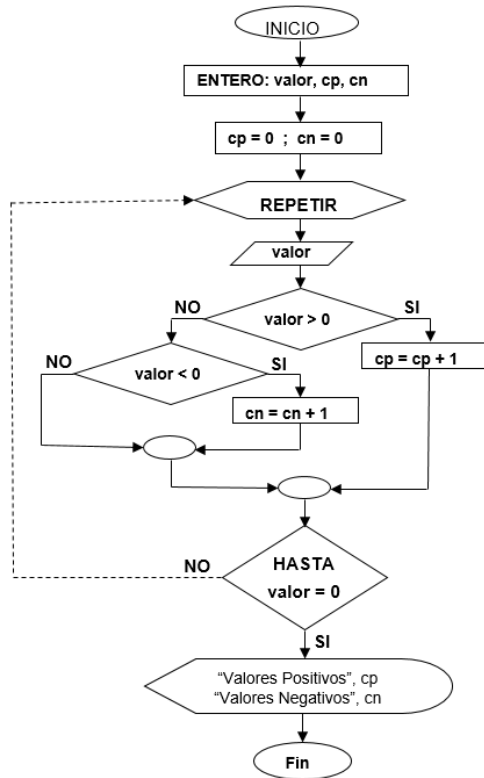
SiNo

Escribir va, "NO ES NÚMERO PERFECTO"

FinSi

FinAlgoritmo

Problema 4: Haga un diagrama que permita ingresar tantos números como se quiera hasta que llegue un cero. Se solicita contar cuantos números ingresados son: Positivos y negativos.



PRUEBA DE ESCRITORIO

valor	cp	cn	Salida
~~25~~	~~0~~	~~0~~	Valores Positivos 3
~~15~~	~~1~~	~~1~~	Valores Negativos 2
~~5~~	~~2~~	2	
~~40~~	3		
~~9~~			
0			

// Cuenta valores Positivos y Negativos hasta ingresar cero.

Algoritmo Ejercicio REPETIR

Definir valor, cp, cn **Como** Entero

cp = 0

cn = 0

Repetir

Escribir "Ingrese un valor"

Leer valor

Si valor > 0 **Entonces**

cp = cp + 1

SiNo

Si valor < 0 **Entonces**

cn = cn + 1

FinSi

FinSi

Hasta Que va = 0

Escribir "Valores Positivos", cp

Escribir "Valores Negativos", cn

FinAlgoritmo

Problemas Propuestos con uso de la estructura REPETIR

Problema 1: Diagrama que solicite el ingreso de un número (validar que este entre 0 y 9999, ambos inclusive). Se pide que muestre un mensaje indicando cuántos dígitos tiene el mismo. Ejemplos

Numero = 78 Muestra: Dos dígitos

Numero = 5000 Muestra: Cuatro dígitos

Numero = 7 Muestra: Un dígito

Problema 2: Haga un diagrama donde se puedan ingresar tantos números como se quiera hasta que llegue a cero. Calcule y muestre el promedio aritmético de los valores ingresados.

Problema 3: Diagrama que permita un control de puntos ganados por turnos de un jugador. Se registran los puntos ganados por turno, hasta que la suma total supere los 1000 puntos. Al final muestre:

- a) El número de rondas que se jugaron.
- b) La suma el total de puntos acumulados.

Problema 4: Diagrama que permita ingresar una serie de números hasta llegar a cero. Como proceso encuentre y muestre el mayor ingresado.

Problema 5: Diagrama que permita el control de peso en una clínica nutricional. Se registran el peso de los pacientes hasta ingresar un peso inválido que puede ser menor 10 o mayor 300 Kg. Se debe calcular y mostrar:

- a) Promedio de peso
- b) Cantidad con obesidad (>100 kg)
- c) Cantidad con bajo peso (<50 kg)

Problema 6: Diagrama que permita ingresar edades hasta llegar a -1, con lo cual termina. Se solicita contar y mostrar al final:

- a) Bebe, si está entre 0 y 3 años
- b) Niño, si está entre los 3 y 7 años
- c) Preadolescente, si está entre los 7 y 11 años
- d) Adolescente, si está entre los 11 y 18 años
- e) Adulto joven, si está entre los 18 y 34 años

- f) Adulto, si está entre los 34 y 70 años
- g) Adulto mayor, si es mayor a 70 años
- h) El promedio de todas las edades.

Problema 7: Diagrama que permita un control de asistencia de estudiantes a un seminario de programación. Se ingresarán tantas asistencias (1 asistió, 0 faltó, 2 justificó) hasta llegar a -1, con lo cual termina. Se solicita contar y mostrar al final:

- a) Total asistencias
- b) Total faltas
- c) Total justificaciones
- d) Porcentaje de asistencia

Problema 8: Diagrama que permita un control de un centro de llamadas. Se registrarán tantas llamadas, hasta que su promedio de duración supere los 50 minutos. Por cada llamada registrar su duración. Al final muestre:

- a) Total llamadas
- b) Cantidad de llamadas largas (mayores a 10 minutos)
- c) Cantidad de llamadas cortas (menores a 5 minutos)
- d) Cantidad de llamadas normales (entre 5 y 10 minutos, ambos inclusive)

Problema 9: Diagrama que permita un control de acceso a un sistema informático. Para ello, el sistema tiene almacenada en una constante con nombre la siguiente contraseña: "321XYZ". El diagrama mediante una estructura REPETIR, permite al usuario intentar ingresar la contraseña en un máximo de 5 intentos. Si acierta la contraseña envíe el mensaje "Contraseña Correcta", en caso contrario muestre "Contraseña Incorrecta". El proceso debe finalizar cuando: La contraseña sea correcta (Envíe este mensaje: "Acceso Concedido"), o se agoten los 5 intentos (Envíe este mensaje: "Cuenta Bloqueada Por Exceso de Intentos").

Problema 10: Diagrama que permita simular en un juego, el control de vida de un personaje. El proceso se inicia con el personaje al cual se le asignan 3 vidas. Se jugarán tantas rondas, hasta que la vida del personaje llegue a cero. Por cada ronda ingresará la "acción contra el personaje" (validar: 1 Vida, 2 Golpe). Por cada "vida" (1) estas se aumentan y por cada "golpe", se disminuye una vida. Al final muestre:

- a) Cuántas vidas recibió.
- b) Cuántos golpes recibió.
- c) Cuántas rondas se jugaron

2.5.8.3.1.9 Bucles Anidados con Estructuras Repetitivas

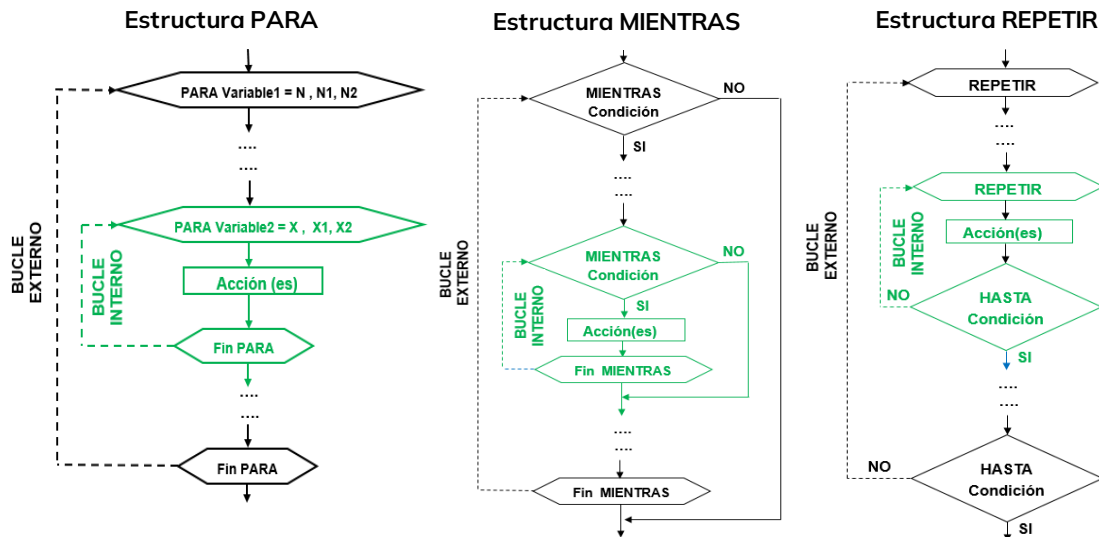
Las estructuras repetitivas constituyen uno de los pilares fundamentales de la programación, ya que permiten ejecutar un conjunto de instrucciones múltiples veces bajo determinadas condiciones.

En este contexto, un bucle anidado ocurre en programación, cuando dentro de un bucle (externo) existe uno o varios bucle(s) interno(s). Esto significa, que un bucle está contenido dentro de otro bucle más grande, por lo tanto, la anidación consiste en colocar una estructura repetitiva dentro de otra, de modo que el ciclo interno se ejecute completamente en cada iteración del ciclo externo. Tenga en consideración los siguientes aspectos para utilizar bucles anidados:

- En los bucles anidados, cuando se ejecuta **una** iteración del **ciclo EXTERNO**, se ejecutarán todas las iteraciones del **ciclo INTERNO**. Por lo tanto, estos se cierran de adentro hacia fuera. Es decir, el **último** bucle en abrirse es el **primero** en cerrarse.
- La cantidad de bucles dependerá de la naturaleza del problema a resolver.
- Las tres estructuras repetitivas (PARA – MIENTRAS – REPETIR) se pueden usar en forma anidada, ya sea de forma independiente o combinando entre ellas.
- El uso de bucles anidados puede aumentar la complejidad y el tiempo de ejecución de un programa. Por lo tanto, es fundamental utilizarlos de manera adecuada para evitar problemas de rendimiento.
- El uso de bucles anidados es común en la programación cuando se trabaja con estructuras de datos multidimensionales, como matrices bidimensionales o simulaciones con múltiples niveles de repetición.

2.5.8.3.1.10 Esquema de las Estructuras Repetitivas Anidadas

En el contexto de la programación, las tres estructuras clásicas: **PARA (FOR)**, **MIENTRAS (WHILE)** y **REPETIR (REPEAT...UNTIL)**, no solo pueden utilizarse de manera independiente, sino que también pueden combinarse entre sí mediante anidación. Gráficamente, se las representa:



Cada una de estas estructuras **conserva su lógica propia** de funcionamiento al anidarse.

Problemas Resueltos con Estructuras Anidadas

Problema 1: Diagrama que muestre los N primeros números (validar que sea POSITIVO), en dónde cada número se muestre tantas veces de acuerdo a su propio valor.

Ejemplo: $n = 5$ **PRUEBA DE ESCRITORIO**

	n	j	k	Salida
1	3	1	1	1
22		2	1	2
333		3	1	3
4444			2	3
55555			3	3

// Genera los N primeros números

Algoritmo EjercicioAnidados

Definir n, j, k **Como Entero**

Repetir

Escribir "Ingrese número"

Leer n

Hasta Que $n > 0$

Para $j \leftarrow 1$ **Hasta** n **Con Paso** 1

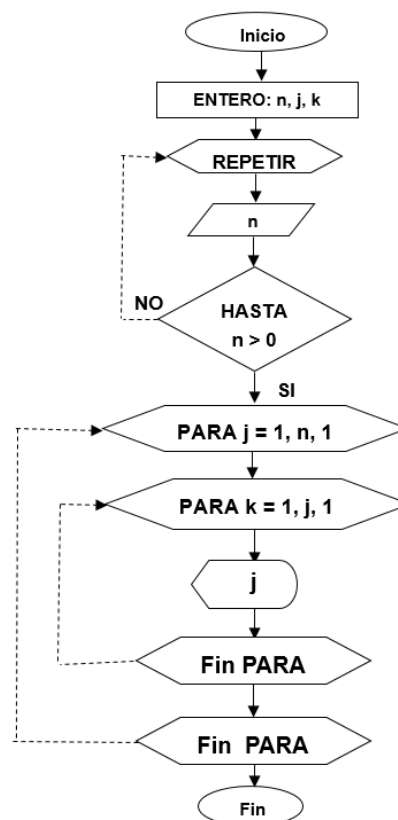
Para $k \leftarrow 1$ **Hasta** j **Con Paso** 1

Escribir j

Fin Para

Fin Para

FinAlgoritmo



En el diagrama una vez ingresado y validado el valor para la variable "n" (positivo) con la estructura **REPETIR**, se inicia el anidamiento utilizando las estructuras **PARA**.

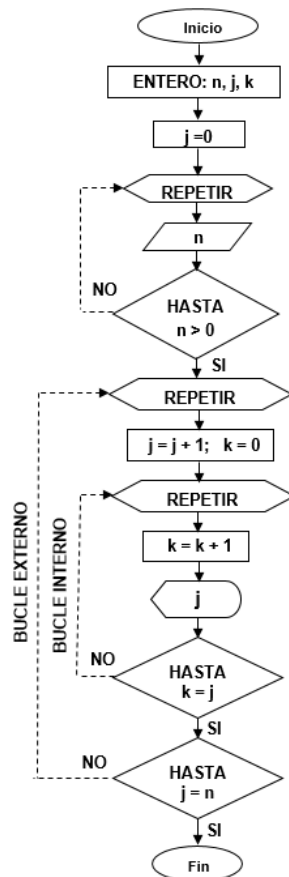
Se puede observar que el bucle externo es controlado por la variable "j" y el bucle interno por la variable "k". El proceso de funcionamiento consiste en que por cada repetición del bucle externo, el bucle interno se ejecuta completamente.

En la prueba se observa que cuando el bucle externo de "j" tenga 1, el bucle interno de "k" dará 1 iteración, cuando el bucle externo de "j" tenga 2, el bucle interno dará 2 iteraciones y así sucesivamente. De esta manera se obtienen estos resultados:

1
2
2
3
3
3

A continuación, se muestra el problema anterior resuelto utilizando las estructuras repetitivas **REPETIR** y **MIENTRAS**. La combinación en el uso de estas estructuras dependerá de la lógica del desarrollador y/o programador y del tipo de problema a resolver.

Lo que sí es importante recomendar, es que cuando se desean crear bucles conociendo el número de iteraciones, es mejor utilizar la estructura repetitiva **PARA**.



// Genera los N primeros números

Algoritmo EjercicioAnidados

Definir n, j, k **Como** Entero

j = 0

Repetir

Escribir "Ingrese número"

Leer n

Hasta Que n > 0

Repetir

 j = j + 1

 k = 0

Repetir

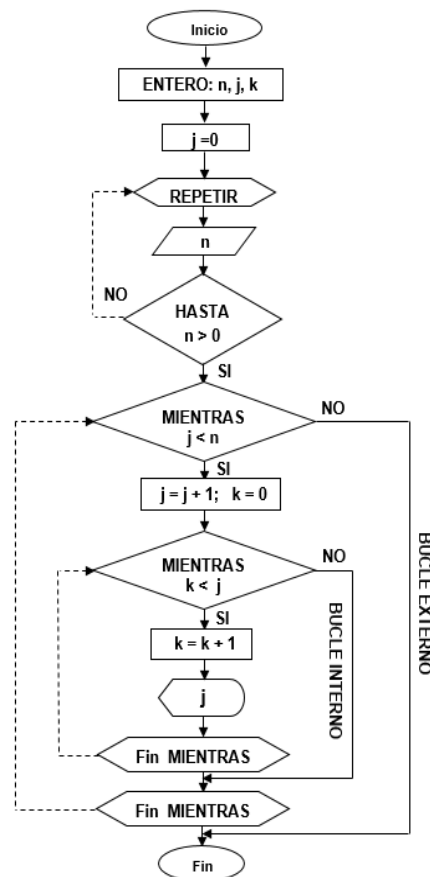
 k = k + 1

Escribir j

Hasta Que k = j

Hasta Que j = n

FinAlgoritmo



// Genera los N primeros números

Algoritmo EjercicioAnidados

Definir n, j, k **Como** Entero

j = 0

Repetir

Escribir "Ingrese número"

Leer n

Hasta Que n > 0

Mientras j < n **Hacer**

 j = j + 1

 k = 0

Mientras k < j **Hacer**

 k = k + 1

Escribir j

FinMientras

FinMientras

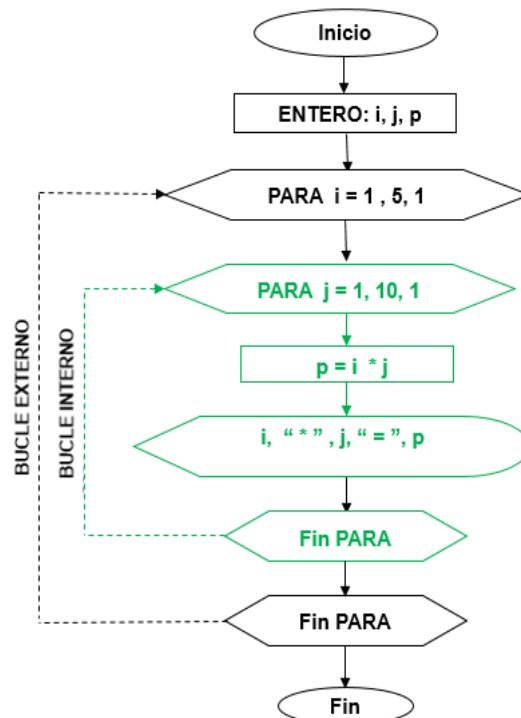
FinAlgoritmo

Problema 2: Diagrama que permita generar automáticamente las cinco (5) primeras tablas de multiplicar. Cada tabla generada debe ir de 1 a 10.

1*	1	=	1
1*	2	=	2
1*	3	=	3
1*	4	=	4
1*	5	=	5
1*	6	=	6
1*	7	=	7
1*	8	=	8
1*	9	=	9
1*	10	=	10

2*	1	=	2
2*	2	=	4
2*	3	=	6
2*	4	=	8
2*	5	=	10
2*	6	=	12
2*	7	=	14
2*	8	=	16
2*	9	=	18
2*	10	=	20

5*	1	=	5
5*	2	=	10
5*	3	=	15
5*	4	=	20
5*	5	=	25
5*	6	=	30
5*	7	=	35
5*	8	=	40
5*	9	=	45
5*	10	=	50



PRUEBA DE ESCRITORIO

i	j	p	Salida
1	1	1	1 * 1 = 1
	2	2	1 * 2 = 2
	3	3	1 * 3 = 3
	4	4	1 * 4 = 4
	5	5	1 * 5 = 5
	6	6	1 * 6 = 6
	7	7	1 * 7 = 7
	8	8	1 * 8 = 8
	9	9	1 * 9 = 9
	10	10	1 * 10 = 10
2	1	2	2 * 1 = 2
	2	4	2 * 2 = 4
	3	6	2 * 3 = 6
	4	8	2 * 4 = 8
	5	10	2 * 5 = 10
	6	12	2 * 6 = 12
	7	14	2 * 7 = 14
	8	16	2 * 8 = 16
	9	18	2 * 9 = 18
	10	20	2 * 10 = 20

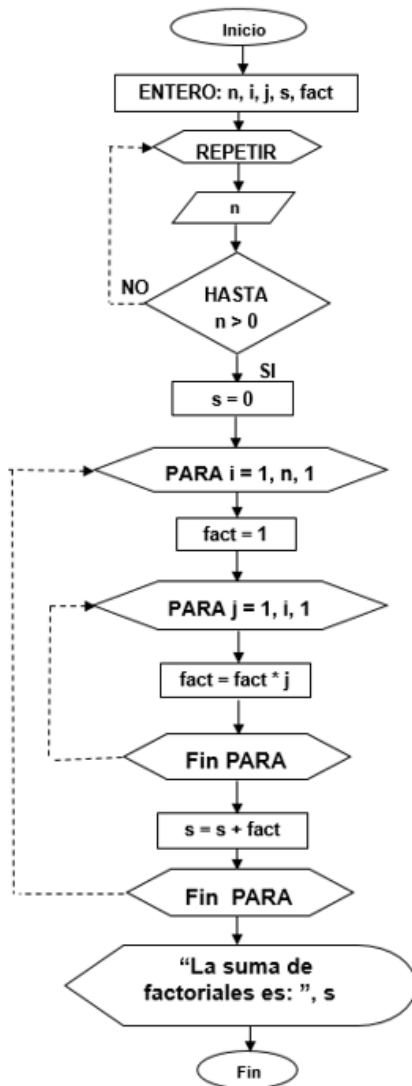
// Genera las 5 primeras tablas de multiplicar
Algoritmo EjercicioAnidados
 Definir i, j, p Como Entero
 Para i ← 1 Hasta 5 Con Paso 1
 Para j ← 1 Hasta 10 Con Paso 1
 p = i * j
 Escribir i, "*", j, "=", p
 Fin Para
Fin Para
FinAlgoritmo



En el diagrama, cada vez que se ejecuta una iteración del bucle externo, se ejecutarán todas las iteraciones del bucle interno.

Problema 3: Diagrama que desarrolle la siguiente fórmula:

$$S = 1! + 2! + 3! + \dots + N!$$



// Genera los factoriales de una fórmula

Algoritmo EjercicioAnidados

Definir n, i, j, s, fact **Como** Entero

Repetir

Escribir "Ingrese número"

Leer n

Hasta Que n > 0

 s = 0

Para i ← 1 **Hasta** n **Con Paso** 1

 fact = 1

Para j ← 1 **Hasta** i **Con Paso** 1

 fact = fact * j

Fin Para

 s = s + fact

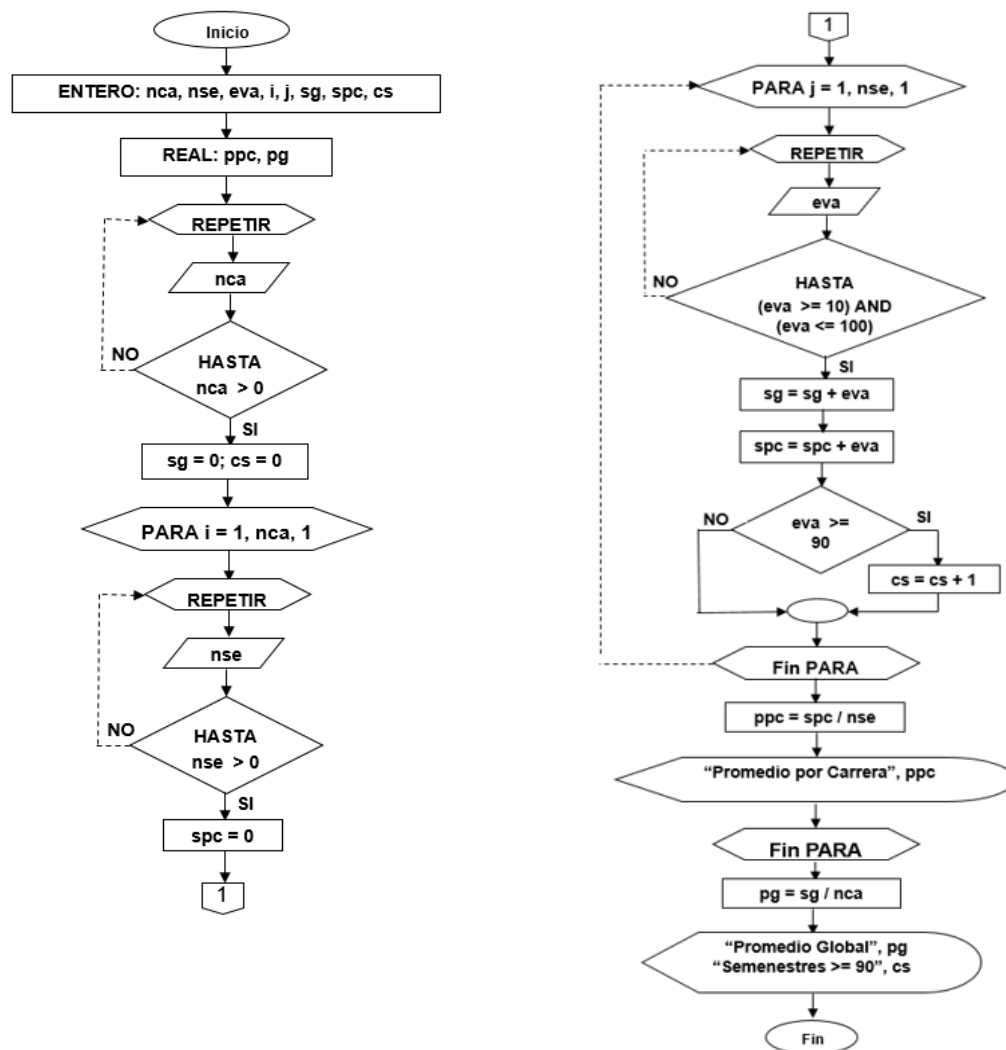
Fin Para

Escribir "La suma de los factoriales es:", s

FinAlgoritmo

Problema 4: Diagrama que permita la evaluación académica por carrera y semestre de una universidad. Se deberá ingresar cuántas carreras serán evaluadas (validar que sea un valor positivo). Por cada carrera (ciclo externo), se deberá ingresar cuántos semestres (validar que sea un valor positivo) se evaluarán (ciclo interno) y por cada semestre la evaluación que tendrá (validar entre 10 y 100, ambos inclusive). Se solicita, calcular y mostrar:

- El promedio por carrera.
- La cantidad de semestres con evaluaciones mayores o iguales a 90.
- Promedio global universitario.



```

// Genera los factoriales de una fórmula
Algoritmo EjercicioAnidados
  Definir nca, nse, eva, i, j, sg, spc, cs Como
Entero
  Definir ppc, pg Como Real
  Repetir
    Escribir "Ingrese Número de Carreras"
    Leer nca
  Hasta Que nca > 0
  sg = 0
  cs = 0
  Para i ← 1 Hasta nca Con Paso 1
    Repetir
      Escribir "Ingrese Números de
Semestre"
      Leer nse
      Hasta Que nse > 0
      spc = 0
      Para j ← 1 Hasta nse Con Paso 1
        Repetir
          Escribir "Ingrese Evaluación"
          Leer eva
          Hasta Que (eva >= 10) AND (eva
<= 100)
          sg = sg + eva
          spc = spc + eva
          SI eva >= 90 Entonces
            cs = cs + 1
          FinSi
        Fin Para
      ppc = spc / nse
      Escribir "Promedio por Carrera", ppc
    Fin Para
  pg = sg / nca
  Escribir "Promedio Global", pg
  Escribir "Semestres >= 90", cs
FinAlgoritmo

```

Problemas Propuestos con Estructuras Anidadas

Problema 1: Diagrama que muestre los N primeros números (validar que sea POSITIVO), en dónde cada número se muestre de la siguiente forma:

Ejemplo: n = 4

```

1      1
2      2 3
3      4 5 6
4      7 8 9 10

```

Problema 2: Usando la lógica del problema resuelto No. 2, desarrollar un Diagrama que permita generar las N (validar con valor positivo) primeras tablas de Multiplicar. Considere que si una tabla generada es **PAR**, debe ir de 1 a 10. Si es **IMPAR** de 15 a 1.

Problema 3: Haga un Diagrama que permita generar las tablas de multiplicar desde un valor **inicial** hasta un valor **final**. Validar que el valor ingresado a **inicial** debe ser menor al valor ingresado a **final**. Considere que cada tabla que se genere debe ir del 1 a 10.

Problema 4: Diagrama que desarrolle la siguiente fórmula:

$$S = 1^{1!} + 2^{2!} + 3^{3!} + \dots + N^{N!}$$

Validar que el valor ingresado a N sea positivo.

Problema 5: Diagrama que permita que una empresa analice las ventas de varios vendedores durante varios días. Se deberá ingresar cuántos vendedores serán evaluados (validar que sea un valor positivo). Por cada vendedor (ciclo externo), se deberá ingresar cuántos días trabajó (validar que sea un valor positivo) y por cada día (ciclo interno), se ingresará el monto vendido (validar entre 10 y 1000, ambos inclusive). Se solicita, calcular y mostrar:

- Total vendido por cada vendedor.
- Promedio por vendedor.
- Cantidad de días con ventas superiores a 1000.
- Total general vendido por la empresa.

Problema 6: Diagrama que permita a una empresa de transporte realizar un análisis de sus rutas. Se deberá ingresar cuántas rutas tienen (validar que sea un valor positivo). Por cada ruta (ciclo externo), se deberá ingresar cuántos viajes se realizaron (validar que sea un valor positivo) y por cada viaje (ciclo interno), se ingresará la cantidad de pasajeros movilizados (validar entre 1 y 40, ambos inclusive). Se solicita, calcular y mostrar:

- Total de pasajeros por ruta.
- Promedio de pasajeros por viaje.
- Total general de pasajeros transportados.

Capítulo III. Manipulación de Cadenas



Objetivo General del Capítulo

En el desarrollo y utilización de programas, la información que se utiliza no siempre se limita a valores numéricos. Este capítulo busca desarrollar en la persona, la capacidad de comprender, analizar y manipular cadenas de caracteres dentro de un algoritmo, aplicando estructuras repetitivas, condicionales, contadores y acumuladores para realizar operaciones como recorrido, comparación, validación, búsqueda y transformación de datos textuales, fortaleciendo así su lógica de programación y la capacidad para resolver problemas prácticos.

3.1 Cadenas

3.1.1 Definición

En programación, una cadena (*string* en inglés) se define como la **secuencia o hilera finita de caracteres** (pueden ser letras, números u otros símbolos) que se almacenan (temporal o permanentemente) para su tratamiento, según las necesidades de una aplicación. Por ejemplo:

- Universidad Laica Eloy Alfaro de Manabí
- Calle 11 y avenida 18, Manta
- Buendía Karen
- +593999123456
- Bienvenido..... Ingrese su Contraseña
- Hoover Max Extract Pressure Pro- modelo 60

3.1.2 Diferencia entre Carácter y Cadena

En la creación de algoritmos y programas, el manejo de texto se basa en dos conceptos fundamentales: el carácter y la cadena. Comprender la diferencia entre ambos es fundamental para poder procesar, comparar y manipular información textual.

Un **carácter** es considerado la **unidad básica** de información textual y se representa por un **único símbolo**, que puede ser: Las letras minúsculas de 'a' - 'z', las letras mayúsculas de 'A' - 'Z'. Los números de '0' a '9', el espacio en blanco ' ', símbolos diversos '!', '@', '%', etcétera. Estos caracteres, como se explicará en lo posterior, forman parte de un conjunto especial de caracteres llamados **ASCII**.

Una **cadena** como ya se describió, consiste en secuencia o hilera de varios caracteres. Por ejemplo:

"123xYz" Cadena de 6 caracteres
"Hola Mundo" Cadena de 10 caracteres
"090866797-4" Cadena de 11 caracteres

En estos ejemplos las comillas (""), son delimitadores y no forman parte de la cadena.



Una cadena puede recorrerse carácter por carácter y permitir realizar operaciones para su manipulación.

3.2 Importancia de las Cadenas en Programación

El manejo y manipulación de cadenas de caracteres es un tema importante en el ámbito informático y muy especialmente en la programación, ya que permiten que los programas trabajen con abundante información textual, que son uno de los tipos de datos más utilizados y procesados en todo sistema informático. Algunas razones de su importancia:

- Para el ingreso, almacenamiento y presentación de datos
- En la validación de datos
- En el procesamiento de datos
- En el tratamiento de cadenas en los Lenguajes de Programación

3.2.1 En el ingreso, almacenamiento y presentación de datos

Cuando se interactúa con un software para ingresar y almacenar información, generalmente se hace a través de cadenas, pues están en formato de texto. Se pueden ingresar y almacenar nombres de usuarios, contraseñas, direcciones de correo electrónico, datos de clientes, información de productos y muchos otros tipos de datos textuales. Del mismo modo, cuando un programa muestra información en la pantalla o la imprime, se utiliza el manejo de cadenas para formatear y presentar los datos de manera legible y estructurada para los usuarios.

3.2.2 En la validación de datos

Antes de procesar o almacenar datos, es esencial realizar validaciones para asegurarse de que la información cumpla con ciertos requerimientos. El manejo de cadenas permite realizar diversas validaciones, como verificar si una cadena tiene una longitud específica, si contiene únicamente caracteres alfabéticos o numéricos, si sigue un patrón de formato determinado (por ejemplo, una dirección de correo electrónico) y mucho más.

3.2.3 En el procesamiento de datos

La información textual no solo se almacena, sino que se analiza y manipula para obtener información útil. Ello implica que una cadena se puede leer para hacer búsquedas (localizar información específica dentro de una cadena), compararla y examinar su contenido para obtener información de utilidad a partir de ella. **La**

forma de cómo realizar estos procesos, serán analizadas en los siguientes apartados de este capítulo.

3.2.4 En el tratamiento de cadenas en los Lenguajes de Programación

El manejo eficiente de cadenas es una competencia esencial para todo desarrollador y/o programador, ya que el texto forma parte central del flujo de información en toda aplicación informática. En este contexto, los lenguajes de programación modernos ofrecen mecanismos que permiten manipular con facilidad la información textual, de acuerdo con los requerimientos de cada aplicación. Ello evidencia la relevancia del tratamiento de cadenas dentro del desarrollo de software.

3.3 Códigos ASCII

Cuando se ingresa, procesa o se almacena una cadena en memoria, estos caracteres no se guardan como elementos visibles (letras, números o signos), sino como **valores numéricos**. Para ello las computadoras y otros dispositivos, utilizan un sistema de codificación, que permite asignar un número específico a cada carácter.

Entre los sistemas de codificación mayormente utilizados, están el **ASCII** que es uno de los primeros sistemas de codificación de caracteres y el **Unicode**, que es el más moderno y completo ya que permite representar los **caracteres de casi todos los idiomas del mundo**, incluidos los Emojis. Para efectos de este texto, la manipulación de cadenas se hará a través del sistema ASCII.

El sistema de codificación ASCII (acrónimo inglés de: American Standard Code for Information Interchange - Código Estándar Estadounidense para el Intercambio de Información). Está formado por un conjunto de 256 de caracteres (numerados de 0 a 255) que es utilizado por los sistemas informáticos para representar textos y para el control y gestión de dispositivos que hacen uso del texto. La figura 16, muestra los caracteres de este sistema, cada uno con su código numérico (también llamado “ordinal”).

Figura 16

E Tabla de caracteres del sistema de codificación ASCII

Ordinal	Caracteres ASCII de control				Caracteres ASCII imprimibles				ASCII extendido											
00	NULL	(carácter nulo)			32	espacio	64	@	96	.			128	Ç	160	à	192	À	224	Ó
01	SOH	(inicio encabezado)			33	!	65	A	97	a			129	ú	161	í	193	Á	225	Ô
02	STX	(inicio texto)			34	"	66	B	98	b			130	ê	162	ô	194	Â	226	Ö
03	ETX	(fin de texto)			35	#	67	C	99	c			131	ë	163	ó	195	Ã	227	Ø
04	EOT	(fin transmisión)			36	\$	68	D	100	d			132	ä	164	ñ	196	Ä	228	Ù
05	ENQ	(consulta)			37	%	69	E	101	e			133	å	165	Ñ	197	Å	229	Ú
06	ACK	(reconocimiento)			38	&	70	F	102	f			134	ä	166	*	198	ä	230	Û
07	BEL	(timbre)			39	'	71	G	103	g			135	ç	167	°	199	Å	231	Ü
08	BS	(retroceso)			40	(	72	H	104	h			136	è	168	¿	200	ä	232	Ý
09	HT	(tab horizontal)			41	)	73	I	105	i			137	é	169	®	201	Ë	233	Û
10	LF	(nueva línea)			42	*	74	J	106	j			138	ê	170	™	202	Ë	234	Ü
11	VT	(tab vertical)			43	+	75	K	107	k			139	ÿ	171	¼	203	Ë	235	Ý
12	FF	(nueva página)			44	,	76	L	108	l			140	ï	172	½	204	Ë	236	ÿ
13	CR	(retorno de carro)			45	-	77	M	109	m			141	ï	173	¾	205	Ë	237	ÿ
14	SO	(desplaza afuera)			46	.	78	N	110	n			142	Ä	174	«	206	Ë	238	ÿ
15	SI	(desplaza adentro)			47	/	79	O	111	o			143	Å	175	»	207	ä	239	ÿ
16	DLE	(esc.vínculo datos)			48	0	80	P	112	p			144	Æ	176		208	ö	240	ä
17	DC1	(control disp. 1)			49	1	81	Q	113	q			145	æ	177		209	ö	241	ä
18	DC2	(control disp. 2)			50	2	82	R	114	r			146	Æ	178		210	ö	242	ä
19	DC3	(control disp. 3)			51	3	83	S	115	s			147	ö	179		211	ö	243	ä
20	DC4	(control disp. 4)			52	4	84	T	116	t			148	ö	180		212	ö	244	ä
21	NAK	(conf. negativa)			53	5	85	U	117	u			149	ö	181		213	ö	245	ä
22	SYN	(inactividad sinc)			54	6	86	V	118	v			150	ü	182		214	ö	246	ä
23	ETB	(fin bloque trans)			55	7	87	W	119	w			151	ü	183		215	ö	247	ä
24	CAN	(cancelar)			56	8	88	X	120	x			152	ÿ	184		216	ö	248	ä
25	EM	(fin del medio)			57	9	89	Y	121	y			153	Ü	185		217	ö	249	ä
26	SUB	(sustitución)			58	:	90	Z	122	z			154	Ü	186		218	ö	250	ä
27	ESC	(escape)			59	;	91	[	123	{			155	ö	187		219	ö	251	ä
28	FS	(sep. archivos)			60	<	92	\	124				156	ö	188		220	ö	252	ä
29	GS	(sep. grupos)			61	=	93	]	125	}			157	ö	189		221	ö	253	ä
30	RS	(sep. registros)			62	>	94	^	126	~			158	x	190		222	ö	254	ä
31	US	(sep. unidades)			63	?	95	_					159	f	191		223	ö	255	nbsp
127	DEL	(suprimir)																		

Nota. Fuente: Tomado de la Web.

Como se observa en la tabla, cada carácter tiene un número asociado que determina su orden dentro del conjunto de caracteres definido por el estándar. Esto es importante, ya que será utilizado en los procesos de: búsqueda, comparación, extracción y análisis en la manipulación de cadenas.



Comprender cómo funciona y se manipula el texto en memoria, fortalece el desarrollo de la **lógica algorítmica**.

3.4 Operaciones Fundamentales con Cadenas

Al considerar las cadenas como un tipo de datos, hay que definir cuáles son las operaciones que es posible hacer con ellas, pues, constituyen la base del procesamiento de texto en programación. Estas operaciones se dan mediante el uso de **funciones especiales**, que acompañadas con estructuras de control como ciclos y condicionales permiten el tratamiento adecuado de cadenas, según la tarea a realizar. Las principales operaciones son:

- Comparación
- Longitud o tamaño
- Concatenación
- Manejo de Sub-cadenas

Los lenguajes de programación incorporan diferentes funciones para manipular cadenas. Si en algún momento se necesitan realizar tareas muy específicas con cadenas, se recomienda acudir a la documentación sobre funciones de cadenas del lenguaje que se esté utilizando.

3.4.1 Operaciones con Cadenas: Comparación

La comparación se realiza entre dos cadenas, para determinar si son iguales o diferentes. Para ello se deben utilizar los operadores relación, los cuales comparan las cadenas carácter por carácter. El resultado de la comparación puede ser **verdadero** o **falso**, y esto dependerá de la posición relativa del **número ordinal** (código ASCII) de los caracteres que se comparan.

Ejemplo 1: **"A" = "a"**

El resultado sería **FALSO** (ordinal (ASCII) de "A" es 65 y el ordinal (ASCII) de "a" es: 97). Por lo tanto, internamente se pregunta: $65 = 97$

Ejemplo 2: **"Z" < "a"**

El resultado sería **VERDADERO** (ordinal (ASCII) de "Z" es 90 y el ordinal (ASCII) de "a" es: 97. Por lo tanto, internamente se pregunta: $90 < 97$

Ejemplo 2: **"Cuenca" > "Cuenca"**

En este caso, la **comparación está** basada en el **valor ordinal (ASCII)** de cada carácter y su posición dentro de la cadena. El resultado sería **VERDADERO**, ya que la cadena más larga se considera la mayor (por el ordinal de "s" de la primera cadena).

Al realizar la comparación de cadenas, es muy importante tener en consideración lo siguiente:

- Generalmente, las comparaciones de cadenas se basan en la **IGUALDAD**.
- La comparación no se hace de forma lexicográfica (como se organizarían las palabras en un diccionario), se hacen considerando los ordinales de los caracteres que se comparan.

.

- En la comparación de ordinales se distingue entre caracteres, pues, las mayúsculas siempre son **menores** que las minúsculas, los números son **menores** que las mayúsculas, y algunos símbolos son **menores** que los números (ver figura 3.x).
- Si las dos cadenas se comparan iguales hasta el final y la otra tiene caracteres restantes, la cadena con los caracteres restantes se considera “mayor”.

3.4.2 Operaciones con Cadenas: Longitud o Tamaño

Toda cadena tiene un atributo llamado **longitud**, que determina la cantidad o número total de caracteres que la conforman.

Muchos esquemas de pseudocódigo incluyen una función para calcular el tamaño de una cadena, por lo tanto, se utilizan funciones como: Longitud, Tamaño. En el caso de este texto, se utilizará la función “Longitud”, de la siguiente forma:

variable = **Longitud** (Cadena)

Dicha función, recibe como parámetro una cadena y, como resultado, devolverá el número de caracteres que tiene.

Ejemplo: 1

l = Longitud (“Manta”)
 5

Ejemplo: 2

x = “Manta, Calle 12 Avenida 24”
 l = Longitud (x)
 26

Ejemplo: 3

x = “”
 l = Longitud (x)
 0

Tenga en cuenta las siguientes consideraciones, al calcular la longitud de una cadena:

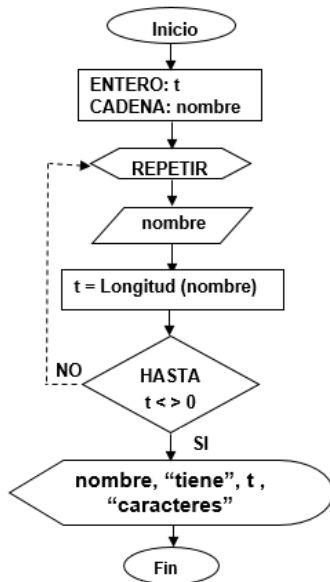
- La cadena puede ser: Una constante alfanumérica o variable alfanumérica.
- La variable que almacena el resultado de la función “**Longitud**” debe ser definida como numérica entera.
- Si el contenido de la variable está vacía (“”), entonces “Longitud”, retorna como resultado cero.



En algunos lenguajes de programación, se utilizan las funciones: Len, strlen o length en lugar de “**Longitud**” para calcular el tamaño de la cadena.

Problemas Resueltos con la Función "Longitud"

Problema 1: Diagrama que permita el ingreso de un nombre. Determinar el número de caracteres que lo conforman. Validar mediante una estructura **REPETIR** que la cadena ingresada no esté vacía.



PRUEBA DE ESCRITORIO

nombre	T	SALIDA
Maria Cedeño	12	Maria Cedeño tiene 12 caracteres
Pedro	5	Pedro tiene 5 caracteres

// Calcula el tamaño de una cadena

Algoritmo EjercicioCadenas

Definir t **Como** Entero

Definir nombre **Como** Cadena

Repetir

Escribir "Ingrese un Nombre"

Leer nombre

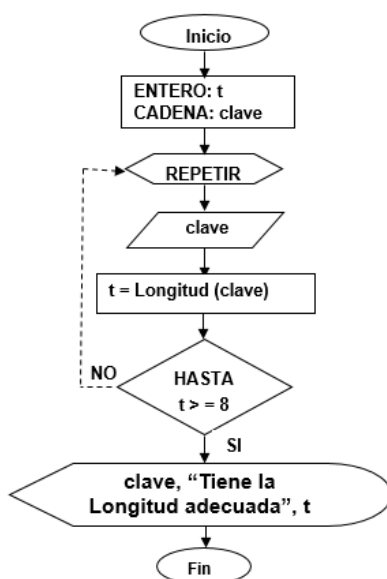
 t = Longitud (nombre)

Hasta Que t <> 0

Escribir nombre, "tiene ", t, "caracteres"

FinAlgoritmo

Problema 2: Diagrama que solicite el ingreso de una contraseña. Se debe verificar si la contraseña tiene **8 o más caracteres**. Si cumple la condición, mostrar un mensaje indicando que la contraseña tiene una longitud adecuada, en caso contrario se volverá a ingresar la contraseña, hasta ingresar el tamaño solicitado.



PRUEBA DE ESCRITORIO

clave	t	Salida
ABC123	6	
lunes	5	
Manta@2026	10	Manta@2026 Tiene la Longitud adecuada 10

// Calcula el tamaño de una contraseña

Algoritmo EjercicioCadenas

Definir t **Como** Entero

Definir clave **Como** Cadena

Repetir

Escribir "Ingrese la Clave (mayor a 7 caracteres)"

Leer clave

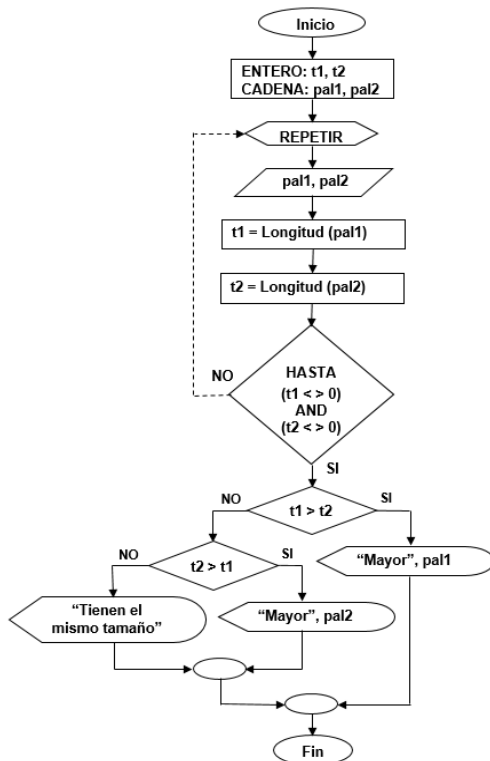
 t = Longitud (clave)

Hasta Que t >= 8

Escribir clave, "Tiene la Longitud adecuada", t

FinAlgoritmo

Problema 3: Diagrama que permite ingresar dos palabras diferentes y determine cuál de las dos tiene mayor cantidad de caracteres. Validar que las variables donde se ingresan las palabras, no estén vacías.



PRUEBA DE ESCRITORIO

pal1	pal2	t1	t2	Salida
Universidad	Casa	11	4	Mayor Universidad
Perro	Gallina	5	7	Mayor Gallina
Silla	Carro	5	5	Tiene el mismo tamaño

// Calcula el tamaño de dos cadenas

Algoritmo EjercicioCadenas

Definir t1, t2 **Como** Entero

Definir pal1, pal2 **Como** Cadena

Repetir

Escribir "Ingrese Primera Palabra"

Leer pal1

Escribir "Ingrese Segunda Palabra"

Leer pal2

 t1 = **Longitud** (pal1)

 t2 = **Longitud** (pal2)

Hasta Que (t1 <> 0) **AND** (t2 <> 0)

Si t1 > t2 **Entonces**

Escribir "Mayor", pal1

SiNo

Si t2 > t1 **Entonces**

Escribir "Mayor", pal2

SiNo

Escribir "Tienen el mismo tamaño"

FinSi

FinSi

FinAlgoritmo

Problemas propuestos con la Función "Longitud"

Problema 1: Elabore un diagrama que permita ingresar una palabra y determine si la cantidad de caracteres es **par o impar**. Validar que la variable de ingreso no esté vacía.

Problema 2: Elabore un diagrama que permita ingresar una palabra cualquiera, se solicita mostrar la palabra ingresada tantas veces de acuerdo al número de caracteres que la componen. Validar que la variable de ingreso no esté vacía.

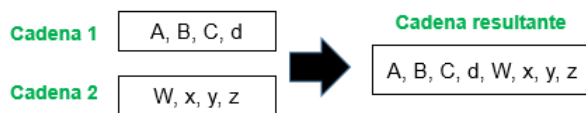
Problema 3: Diseñe un diagrama que permita ingresar dos palabras y calcule la diferencia entre la cantidad de caracteres de ambas. Validar que las variables donde se ingresan las palabras no estén vacías.

Problema 4: Elabore un diagrama que permita ingresar tres palabras y determine cuál es la más corta y cuál es la más larga según su longitud. Validar que las variables donde se ingresan las palabras no estén vacías.

Problema 5: Diseñe un diagrama que permita ingresar dos palabras y determine cuál de ellas tiene una longitud más cercana a **12 caracteres**. Validar que las variables donde se ingresan las palabras no estén vacías.

3.4.3 Operaciones con Cadenas: Concatenación

La concatenación en programación, es la operación que consiste en unir dos o más cadenas para formar una cadena de mayor tamaño, pero conservando el orden de los caracteres en cada una de ellas, es decir, que es un proceso de suma de cadenas.



La concatenación se usa principalmente para formar nuevas cadenas, construir mensajes y crear textos dinámicos en un programa.

Considere los siguientes aspectos para su utilización:

- El símbolo que representa la concatenación varía de un entorno de desarrollo a otro. Los más utilizados son: **+** y **&** (funciones como: **Concat**).
- La longitud de la nueva cadena, será igual a la suma de las longitudes de las cadenas concatenadas.
- Las cadenas para concatenarse pueden estar representadas por constantes alfanuméricas y variables alfanuméricas.

Ejemplo 1:

```

A="Universidad"
B="Laica"
C="Eloy Alfaro de Manabí"
S=A + B + C
└─> UniversidadLaicaEloyAlfaro de Manabí

```

Ejemplo 2:

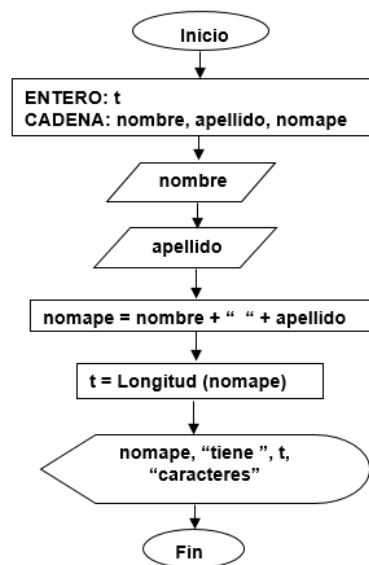
```

A="Universidad"
B="Laica"
C="Eloy Alfaro de Manabí"
S=A + " " + B + " " + C
└─> Universidad Laica Eloy Alfaro de Manabí

```

Problemas Resueltos con uso de Concatenación

Problema 1: Diagrama que permita el ingreso de un nombre y un apellido. Concatenar esta información y mostrarla con su longitud.



```

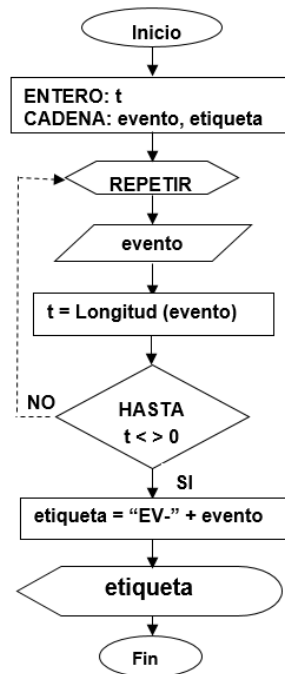
// Concatenas dos cadenas
Algoritmo EjercicioCadenas
Definir t Como Entero
Definir nombre, apellido Como Cadena
Escribir "Ingrese Nombre"
Leer nombre
Escribir "Ingrese Apellido"
nomape = nombre + " " + apellido
t = Longitud (nomape)
Escribir nomape, "tiene ", t, "caracteres"
FinAlgoritmo

```

PRUEBA DE ESCRITORIO

nombre	apellido	nomape	t	SALIDA
Karen	Burgos	Karen Burgos	12	Karen Burgos tiene 12 caracteres
Ian	Sánchez	Ian Sánchez	11	Ian Sánchez tiene 11 caracteres

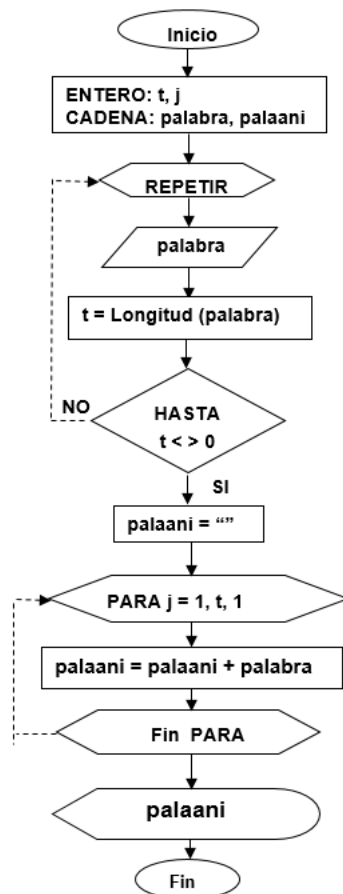
Problema 2: Diagrama que permita generar las etiquetas de los eventos de una empresa promotora. Para ello, cada etiqueta se genera concatenando: a) La cadena “EV” y b) El nombre del evento. Validar que la variable de ingreso del nombre de evento no esté vacía.



PRUEBA DE ESCRITORIO			
evento	t	etiqueta	Salida
Carnaval2026	12	EV-Carnaval2026	EV-Carnaval2026

```
// Crea etiqueta concatenada
Algoritmo EjercicioCadenas
  Definir t Como Entero
  Definir evento, etiqueta Como Cadena
  Repetir
    Escribir "Ingrese nombre del Evento"
    Leer evento
    t = Longitud (evento)
  Hasta Que t <> 0
  etiqueta = "EV-" + evento
  Escribir etiqueta
FinAlgoritmo
```

Problema 3: Diagrama que permite el ingreso de una palabra, como proceso se pide concatenar la palabra ingresada tantas veces de acuerdo a su propia longitud. Validar que la variable de ingreso no esté vacía.



En el diagrama, para hacer el proceso de concatenación, la variable “palaani”, se inicializa con un valor “”, es decir “**vacía con un valor Null**”. Ello permite utilizarla como un “**acumulador**” del valor que tenga la variable “palabra” dentro del bucle.

PRUEBA DE ESCRITORIO

palabra	t	j	palaani	Salida
Sol			Null (vacía)	
	3	1	~~Sol~~	
		2	~~SolSol~~	
		3	SolSolSol	SolSolSol

// Concatena una palabra ingresada

Algoritmo EjercicioCadenas

Definir t, j **Como** Entero

Definir palabra, palaani **Como** Cadena

Repetir

Escribir “Ingrese Palabra”

Leer palabra

 t = **Longitud** (palabra)

Hasta Que t < > 0

 Palaani = “”

Para j ← 1 **Hasta** t **Con** Paso 1

 Palaani = Palaani + palabra

Fin Para

Escribir palaani

FinAlgoritmo

Problemas Propuestos con uso de Concatenación

Problema 1: Diagrama que ingrese tres palabras. Se solicita mostrar una frase formada por concatenación de ellas y luego la longitud de cada palabra, incluyendo la frase concatenada.

Problema 2: Desarrolle un diagrama que permita ingresar tres (3) cadenas. Se pide concatenar las cadenas entre ellas, de tal forma que permita encontrar y mostrar la “pareja de cadenas” que contenga la mayor cantidad de caracteres. Validar mediante estructuras REPETIR que ninguna de las tres (3) cadenas estén vacías.

Problema 3: Diagrama que ingrese una palabra. Se solicita concatenar a la palabra ingresada asteriscos ("*"), de acuerdo al tamaño de la palabra. Los "*" se concatenaran a la izquierda de la palabra. Ejemplos:

Palabra = Ecuador La nueva cadena seria: *****Ecuador

Palabra = Sol La nueva cadena seria: ***Sol

Problema 4: Modificar el diagrama del problema anterior (diagrama No. 3) de tal manera que el usuario ingrese si los asteriscos ("*"), se concatenaran a la izquierda, derecha o en ambos lados de la palabra ingresada. Ejemplos:

Palabra = Ecuador

Lado = Derecho La nueva cadena seria: Ecuador*****

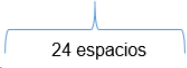
Palabra = Sol

Lado = Ambos La nueva cadena seria: ***Sol***

Problema 5: Se desea "centrar y mostrar", una palabra previamente ingresada en una pantalla que puede mostrar 80 caracteres por líneas. Haga el diagrama correspondiente, validando que la variable de ingreso de la palabra no esté vacía.

Problema 6: Diagrama que ingrese una palabra (validar que no esté vacía) y un número (validar entre 1 y 100, ambos inclusive). Se solicita crear un bucle (1 hasta el número), de tal manera que se concatene a la palabra ingresada el carácter "*" si la variable controladora del bucle es "Par"; en caso de ser "Impar" concatene el carácter "\$". Al final muestre la nueva cadena concatenada.

Problema 7: Diagrama que ingrese una palabra (validar que no esté vacía) y un número (validar entre 1 y 5) que representa la cantidad de "indentación" (tabulación de 8 espacios) que debe tener la palabra desde la izquierda. Muestre la nueva cadena ya indentada. Ejemplo:

Palabra = Casa
Número = 3 La nueva cadena seria:  Casa

Problema 8: Diagrama que genere la siguiente serie de números: 1^1 ; 2^2 ; 3^3 ;
; N^n

De tal manera que el resultado de cada término, corresponda a la cantidad de asteriscos ("*") que se deben generar en una cadena. Validar que el valor ingresado a N debe ser positivo entre 0 y 10, ambos inclusive. Ejemplo: N = 3, se genera la serie:

$1^1; 2^2; 3^3 = 1 = * \quad 4 = **** \quad 27 = *****$

La cadena estaría formada por 32 asteriscos.

3.4.4 Operaciones con Cadenas: Subcadenas

Un substring o subcadena, es una porción o parte de cadena de caracteres que se pueden obtener de una **“cadena más grande o mayor”**. Por ejemplo, la cadena **"abc"** tiene las siguientes subcadenas: "a" , "ab" , "abc" , "b" , "bc" , "c" .

Para obtener subcadenas, es necesario tener en consideración:

Conocer las **“funciones especiales”** que incorporan los lenguajes de programación, para “extraer las subcadenas”. Para ello se debe consultar la “documentación del lenguaje a utilizar”.

d) A estas **“funciones especiales”**, se deben entregar parámetros que establecen de **“que forma y como”** se manipulará la cadena original. Los parámetros pueden ser, según la función a utilizar:

- La **cadena más grande** también llamada “cadena original”. Es decir, la cadena que contiene los caracteres a extraer.
- Las **posiciones iniciales y/o finales** en donde están los **caracteres a obtener**.

3.4.4.1 Función para obtener caracteres: SubCadena

La siguiente función, permite **“localizar y obtener”** dentro de una “cadena mayor” una subcadena más pequeña y/o un carácter. Se utiliza de siguiente forma:
Variable = SubCadena (Cadena, Inicio, Número)

Cadena: Corresponde a la variable o constante que contiene la “cadena mayor” que se va a manipular.

Inicio: Es un valor numérico entero (constante y/o variable) que hace referencia a la “posición” desde donde se va “localizar” los caracteres que formarán la nueva subcadena.

Número: Es un valor entero (constante y/o variable) que hace referencia al “número de caracteres” que se van a “obtener” de la “cadena mayor” a manipular.

Variable: Es una variable de tipo “cadena” y almacena la subcadena obtenida de la función.

Ejemplo 1, utilizando **una constante como cadena mayor**: Obtener la subcadena “Ecuador” de la siguiente cadena.

```
s = SubCadena ("Ecuador País Amazónico", 1, 7)
    ↳ Ecuador
```

Ejemplo 2, utilizando **una variable como cadena mayor**: Obtener la subcadena “Eloy” de la siguiente cadena.

```
x = "Universidad Laica Eloy Alfaro de Manabí"
s = SubCadena (x, 19, 4)
```

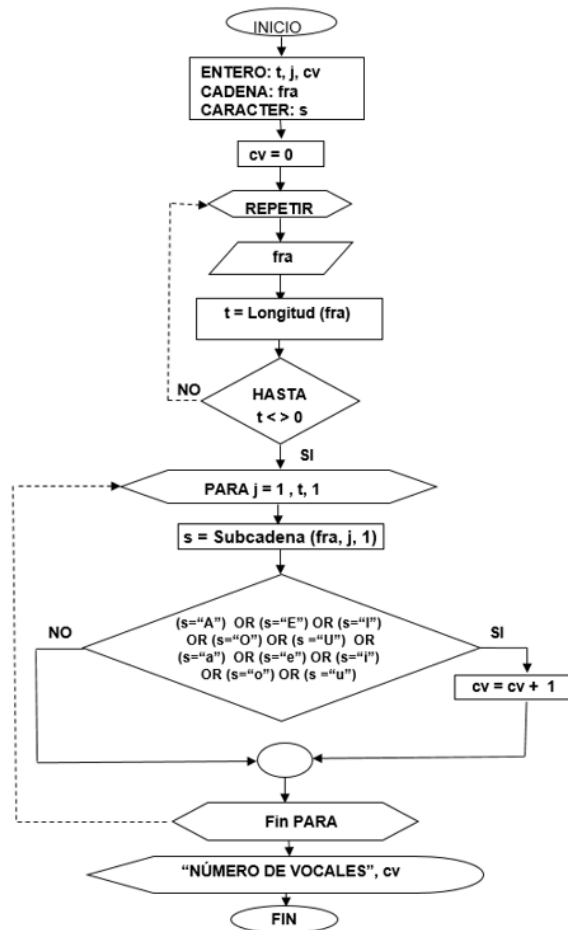
Para utilizar la función **SubCadena**, considere lo siguiente:

```
    ↳ Eloy
```

- Si **Inicio** es mayor que la longitud de la cadena, **Subcadena** devuelve "" (texto vacío).
- Si **Inicio** es menor que la longitud de la cadena, pero **Número** excede la longitud de texto, **Subcadena** devuelve los caracteres hasta el final de la cadena.
- Si **Inicio** es menor que 1, **Subcadena** devuelve el valor de error.
- Si el valor **Número** es negativo, **Subcadena** devuelve el valor de error.

Problemas Resueltos con uso de Subcadenas

Problema 1: Diagrama que permite ingresar una frase (**validar** que la variable no este “vacía”) y como proceso se solicita **contar cuantas vocales** (mayúsculas y minúsculas) existen en ella.



PRUEBA DE ESCRITORIO

t	j	cv	fra	s	Salida
4		0	Gato		NÚMERO DE VOCALES 2
	1			G	
	2	1		a	
	3			t	
	4	2		o	

// Contador de Vocales

Algoritmo EjercicioCadenas

Definir t, j, cv **Como** Entero

Definir fra **Como** Cadena

Definir s **Como** Carácter

cv = 0

Repetir

Escribir “Ingrese Frase”

Leer fra

 t = Longitud (fra)

Hasta Que t <= 0

Para j ← 1 **Hasta** t **Con Paso** 1

 s = Subcadena (fra, j, 1)

Si (s="A") OR (s="E") OR (s="I")
 OR (s="O") OR (s="U") OR
 (s="a") OR (s="e") OR (s="i")
 OR (s="o") OR (s="u") **Entonces**
 cv = cv + 1

FinSi

Fin Para

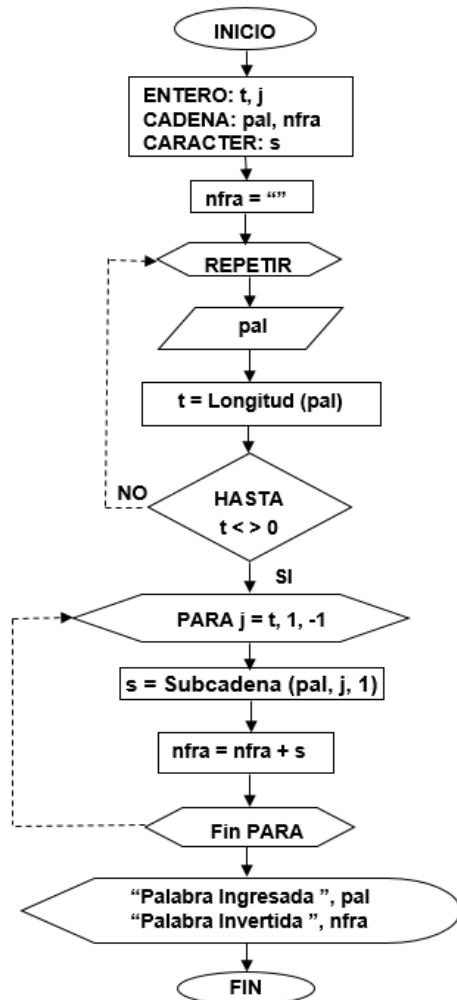
Escribir “Número de Vocales”, cv

FinAlgoritmo



Este diagrama sirve de ejemplo, para entender como “recorrer” en todos los caracteres de una cadena para “rastrear o buscar” determinados caracteres.

Problema 2: Diagrama que ingrese una palabra (validar que la variable de ingreso no esté vacía), como proceso muestre la **palabra invertida**.



PRUEBA DE ESCRITORIO

t	j	fra	nfra	s	Salida
4		Gato	(vacía)		Palabra Ingresada Gato
	~~4~~		~~o~~	~~o~~	Palabra Invertida otaG
	~~3~~		~~ot~~	~~t~~	
	~~2~~		~~ota~~	~~a~~	
	1		otaG	G	

// Invierte Palabra Ingresada

Algoritmo EjercicioCadenas

Definir t, j **Como** Entero

Definir pal, nfra **Como** Cadena

Definir s **Como** Carácter

nfra = ""

Repetir

Escribir "Ingrese Palabra"

Leer fra

 t = Longitud (fra)

Hasta Que t <> 0

Para j ← t **Hasta** 1 **Con** Paso -1

 s = Subcadena (pal, j, 1)

 nfra = nfra + s

Fin Para

Escribir "Palabra Ingresada ", pal

Escribir "Palabra Invertida ", nfra

Fin Algoritmo

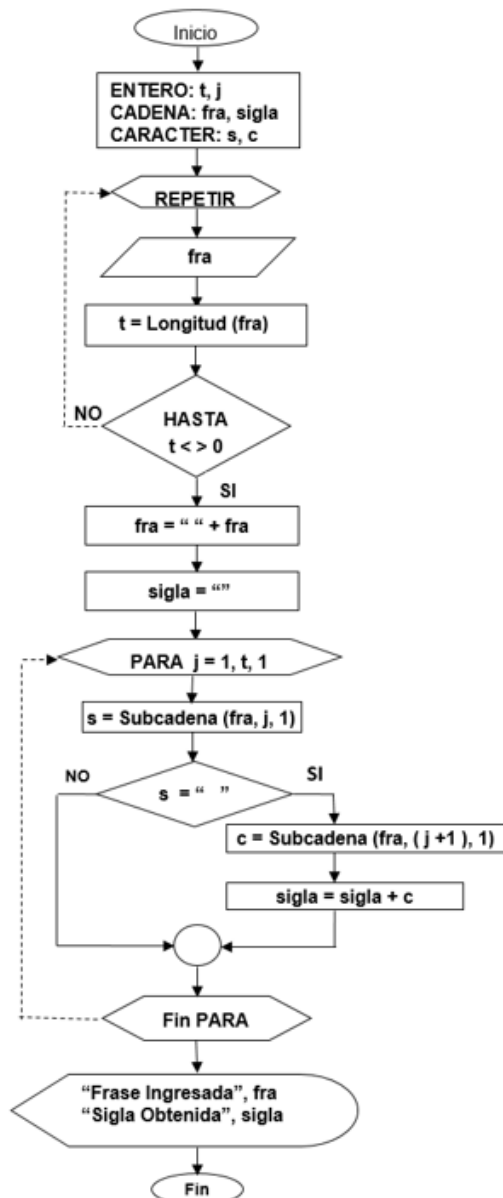


En este diagrama, como proceso principal, se envía el bucle desde el tamaño de la cadena hasta 1, de tal manera que se recorre la cadena carácter por carácter (en forma descendente) y el carácter obtenido se concatena para final mostrar la cadena original de forma invertida.

Problema 3: Diagrama que permite ingreso de una cadena o frase (nombre de una institución y/o organización). Como proceso obtiene de la frase ingresada **su sigla** (abreviatura formada por las letras iniciales de la cadena). Al final muestre lo ingresado y lo solicitado. Ejemplos:

Consejo Nacional Electoral
CNE

Unión Europea
UE



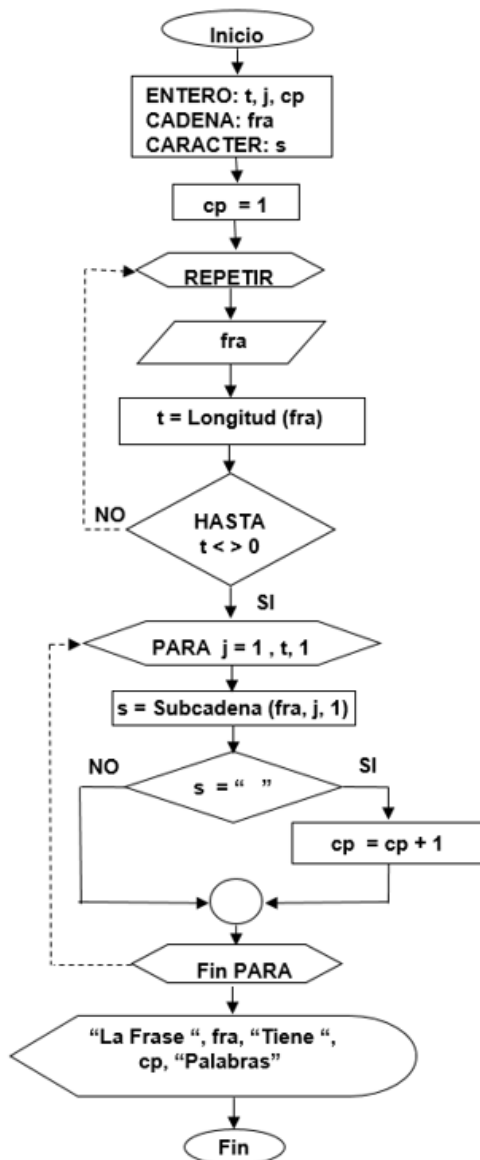
```

// Obtiene la Sigla de una Cadena
Algoritmo EjercicioCadenas
Definir t, j Como Entero
Definir fra, sigla Como Cadena
Definir s, c Como Carácter
fra = " " + fra
Repetir
  Escribir "Ingrese Frase"
  Leer fra
  t = Longitud (fra)
Hasta Que t <> 0
Para j ← 1 Hasta t Con Paso 1
  s = Subcadena (fra, j, 1)
  Si s = " " Entonces
    c = Subcadena (fra, (j + 1), 1)
    sigla = sigla + c
  FinSi
Fin Para
Escribir "Frase Ingresada ", fra
Escribir "Sigla Obtenida ", sigla
FinAlgoritmo
  
```



En este diagrama, a la variable donde se ingresar la cadena, se le concatena al principio un espacio (" "). El propósito es recorrer la cadena carácter por carácter hasta encontrar el espacio (separación de cada palabra en la cadena). Una vez localizado se obtiene el carácter que sigue a él (que es el inicio de cada palabra) y se los concatena para formar la sigla.

Ejercicio 4: Diagrama que permite ingresar una frase (validar que la variable donde se ingresa la frase no este “vacía”). Como proceso se pide **contar** cuantas palabras tiene la frase (detectando espacios). Se asume que la separación entre palabras en la frase es de un espacio.



```

// Contador de Palabras
Algoritmo EjercicioCadenas
  Definir t, j, cp Como Entero
  Definir fra Como Cadena
  Definir s Como Carácter
  cp = 1
  Repetir
    Escribir "Ingrese Frase"
    Leer fra
    t = Longitud(fra)
  Hasta Que t <> 0
  Para j ← 1 Hasta t Con Paso 1
    s = Subcadena(fra, j, 1)
    Si s = " " Entonces
      cp = cp + 1
    FinSi
  Fin Para
  Escribir "La Frase ", fra, "Tiene ", cp,
  "Palabras"
FinAlgoritmo
  
```



En este diagrama, se utiliza un contador de palabras “cp”, que se inicia con 1.

El propósito es recorrer la cadena carácter por carácter hasta encontrar el espacio (separación de cada palabra en la cadena). Una vez localizado se incrementa el contador en 1, hasta recorrer toda la frase. Al final se muestra cuantas palabras existen en la frase.

En caso que la frase tenga una sola palabra, nunca se cumplirá la condición, por lo tanto el número de palabras corresponde al valor inicial del contador que es 1.

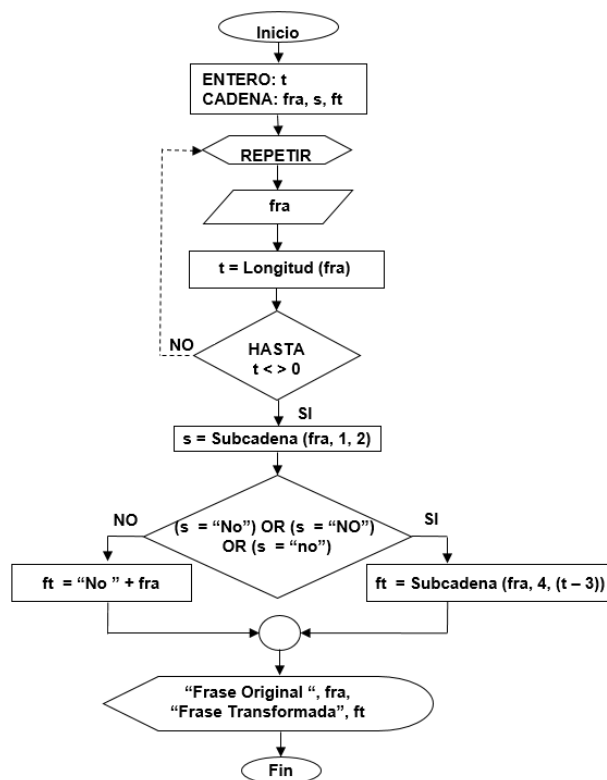
Problema 5: Diagrama que permite ingresar una frase. Se pide “transformar la frase de “positiva” a “negativa” y viceversa. Considere los siguientes aspectos:

- Validar que la variable donde se ingresa la frase no este vacía.
- Asumir que una frase es **“Negativa”**, cuando se inicia con cualquiera de las siguientes cadenas: **“No”, “NO”, “no”**. De no ser así, se asume que la frase será **“Positiva”**.
- Anteponer la cadena **“No”** al inicio de la frase para su transformación a Negativa.
- Quitar la cadena **“No”** al inicio de la frase para su transformación a Positiva.

Ejemplos:

Voy al cine → No Voy al cine

No Hay clases → Hay clases



// Transformador de frase

Algoritmo EjercicioCadenas

Definir t **Como** Entero

Definir fra, s, ft **Como** Cadena

Repetir

Escribir "Ingrese Frase"

Leer fra

 t = Longitud(fra)

Hasta Que t <> 0

 s = Subcadena(fra, 1, 2)

Si (s = "No") **OR** (s = "NO") **OR** (s = "no") **Entonces**

 ft = Subcadena(fra, 4, (t-3))

SiNo

 ft = "No " + fra

FinSi

Escribir "Frase Original", fra

Escribir "Frase Transformada ",ft

FinAlgoritmo



En este diagrama, teniendo en cuenta las consideraciones del enunciado, a la variable de ingreso de la frase ("**fra**"), se le obtienen los 2 primeros caracteres. Si se coincide con cualquiera de las cadenas: **“No”, “NO” o “no”**, entonces a la frase, desde la posición 4 (no se toma en cuenta la negación y el espacio), se obtienen el resto de caracteres de la frase (la longitud de la cadena menos los 3 caracteres que corresponde a la negación y el espacio).

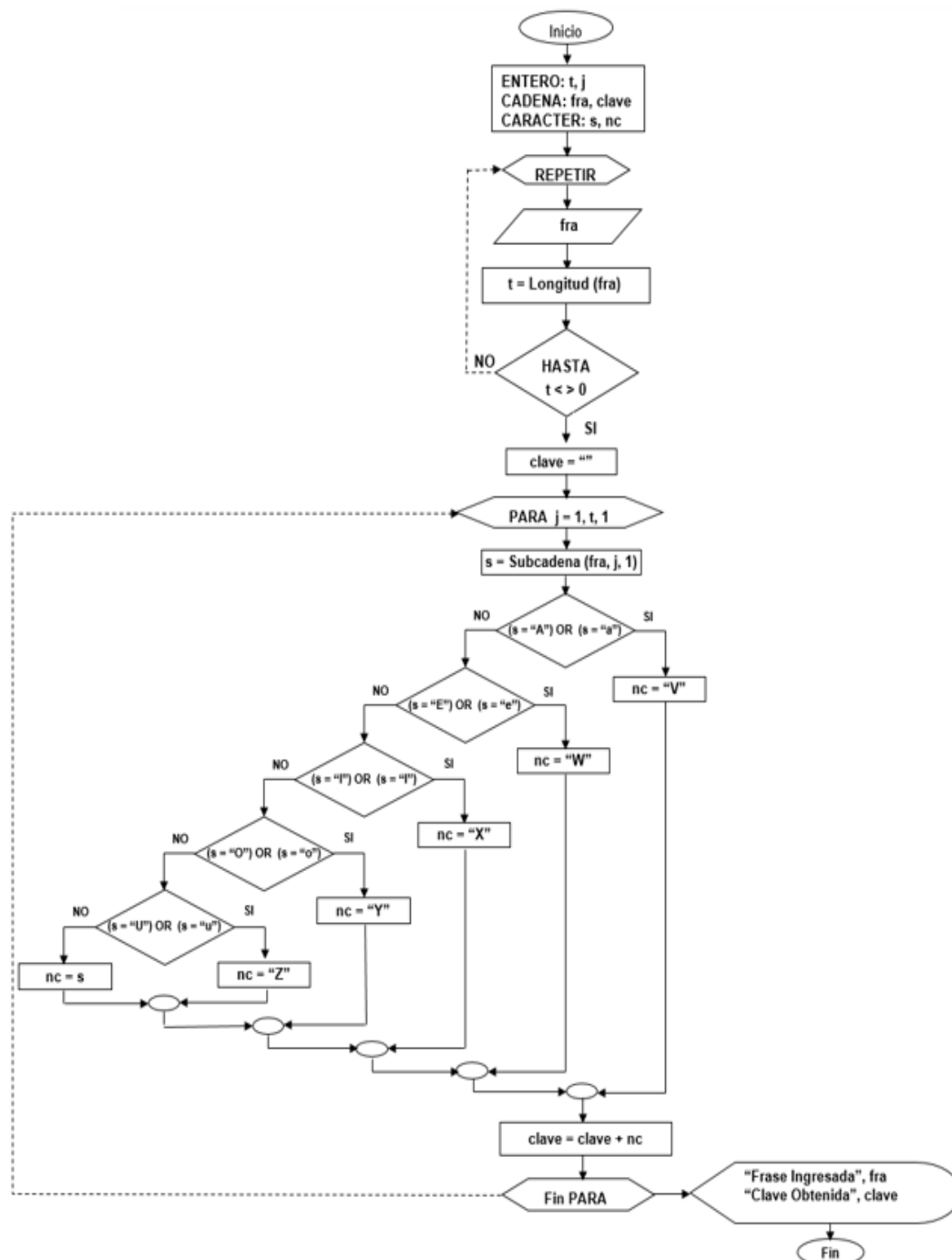
Si la condición es falsa, se asume que la frase es **“Positiva”** y se le concatena al principio la cadena **“No”**. Al final se muestra la frase original y la frase transformada.

Ejercicio 6: Diagrama que permite ingresar una frase. Se pide “transformar la frase” en una clave. Para ello reemplaza cada vocal de la frase (ya sea que este en mayúscula o minúscula) con los siguientes valores:

A - a = **V** E - e = **W** I - i = **X** O - o = **Y** U - u = **Z**

Ejemplo: Hoy si hay clase HYY sX hVy clVsW

Nota: La frase no debe estar vacía y asuma que se excluyen los caracteres tildados.



Problemas Propuestos con uso de Subcadenas

Los siguientes problemas, ayudarán a manipular cadenas utilizando: **subcadenas, longitud, concatenación, condiciones y ciclos.**

Problema 1: Diagrama que permita el ingreso de una palabra (validar que la variable de ingreso no esté vacía), se solicita como proceso mostrar el **primero** y el **último** carácter de la palabra ingresada.

Problema 2: Diagrama que permita el ingreso de una frase (validar que la variable de ingreso no esté vacía), se solicita como proceso **contar** cuantos espacios en blanco hay en la frase ingresada.

Ejercicio 3: Diagrama que permita ingresar un texto. Como proceso se pide contar cuantas oraciones existen en dicho texto (validar que la variable donde se ingresa el texto no este “vacía” y su longitud no sea mayor a 256 caracteres). Considere que la separación de oraciones en el texto se realiza a través de un punto seguido (“.”).

Ejercicio 4: Haga un diagrama que permita ingresar una frase. Se pide **“rellenar”** la frase a la izquierda, con un carácter previamente ingresado, una cantidad N de veces. Considere validar que las variables donde se ingresa la frase y el carácter no deben estar vacías. De igual forma el valor ingresado a N debe ser positivo.

Ejemplo: **frase** → Hola Mundo **caracter** → * **N** → 10 **resultado** → *******Hola Mundo**

Problema 5: Diagrama que permita el ingreso de una palabra (validar que la variable de ingreso no esté vacía). Se solicita como proceso determinar con un mensaje si la palabra es **“palíndromo”** (si una palabra **se lee igual de izquierda a derecha y de derecha a izquierda**).

Ejemplos: radar, oro, ana, ala, oso, reconocer

Problema 6: Diagrama que permita ingresar una frase (validar que la variable donde se ingresa la frase no este vacía). Como proceso se solicita **mostrar en qué posición** de la frase ingresada se encuentra una vocal (mayúscula o minúscula).

Problema 7: Diagrama que permita ingresar una frase cualquiera. Como proceso, se solicita **contar** cuantas veces en la frase se encuentra una letra previamente ingresada (validar que las variables de ingreso no estén vacías). Al final muestre el resultado del conteo solicitado.

Problema 8: Diagrama que permita ingresar una cadena que representa una clave. Como proceso se solicita:

- Validar que la variable donde se ingresa la clave no este vacía y que contenga mínimo 4 y máximo 12 caracteres.
- Mostrar la posición en que se encuentre una letra (debe ser también ingresada), ya sea en Mayúsculas o Minúsculas dentro de la clave ingresada.

Nota: Se excluyen los caracteres tildados

Problema 9: Diagrama que permita ingresar una palabra (validar que la variable de ingreso no esté vacía). Se pide como proceso transformar la palabra en **latín ficticio**. Para ello considere lo siguiente:

- Si una palabra termina en vocal, elimine la vocal y concatene la cadena “us”
- Si una palabra termina en consonante, concatene la cadena “us”

Ejemplo:

Casa	Casus
Viernes	Viernesus

Problema 10: Diagrama que permita ingresar una frase (validar que la variable de ingreso no esté vacía). Se pide como proceso contar cuantas palabras de la frase **INICIAN** con una vocal (sin importar que estén en mayúsculas y minúscula). Al final muestre lo solicitado.

Problema 11: Diagrama que permita ingresar una frase (validar que la variable de ingreso no esté vacía). Se pide como proceso contar cuantas palabras de la frase **TERMINAN** con una vocal (sin importar que estén en mayúsculas y minúscula). Al final muestre lo solicitado.

Problema 12: Diagrama que permita el ingreso de una frase cualquiera (validar que la variable donde se ingresa la frase no este vacía). Se solicita visualizar cada palabra de la frase una debajo de otra, seguida cada palabra del número de caracteres que la componen.

Ejemplo: Hoy es viernes

Hoy	3 Caracteres
es	2 Caracteres
viernes	7 Caracteres

Problema 13: Diagrama que ingrese una dirección email (validar que la variable de ingreso no este vacía), se solicita enviar un mensaje indicando si la dirección es válida o no. Se considerará que una dirección es válida si contiene el símbolo "@" una vez.

Problema 14: Diagrama que permita ingresar una frase (validar que la variable de ingreso no esté vacía). Se pide como proceso contar cuantas palabras de la frase son "agudas con tilde" (aquellas palabras que terminan en "n", "s" o "vocal tildada a la derecha").

Ejemplos: acción, avión, balón, además, anís, bebé, colibrí, café.

Problema 15: Diagrama que permita el ingreso de una frase cualquiera (validar que la variable de ingreso no esté vacía), se solicita lo siguiente:

- Añadir al FINAL de la frase ingresada el PRIMER carácter de la primera palabra de la frase un número N de veces.
- Añadir al PRINCIPIO de la frase ingresada el ÚLTIMO carácter de la última palabra de la frase un número N de veces.

- Validar que el valor ingresado a N debe ser positivo.
- Al final muestre la nueva frase creada.

Problema 16: Diagrama que permita el ingreso de una frase (validar que la variable de ingreso no esté vacía), se solicita como proceso **eliminar** todos los espacios en blanco de la frase ingresada.

Problema 17: Diagrama que ingrese una dirección email (validar que la variable de ingreso no esté vacía). Como proceso se pide obtener el “dominio del correo”, es decir, solo su proveedor:

Ejemplo1: cuenta123@gmail.com Se obtiene: gmail

Ejemplo2: manta@hotmail.com Se obtiene: hotmail

Ejemplo3: algoritmos@yahoo.es Se obtiene: yahoo

Ejercicio 18: Haga un diagrama que permita ingresar una frase. Se pide “eliminar” la **PRIMERA** palabra de la frase. Considere validar que la variable donde se ingresa la frase debe tener mínimo dos (2) palabras.

Ejercicio 19: Diagrama que permita ingresar una frase. Se pide “eliminar” la **ÚLTIMA** palabra de la frase. Considere validar que la variable donde se ingresa la frase debe tener mínimo dos (2) palabras.

Problema 20: Diagrama que permita el ingreso de una palabra y de un carácter cualquiera (validar que las variables de ingreso no estén vacías), se solicita como proceso eliminar todas las apariciones de ese carácter en la palabra ingresada.

3.4.5. Otras Funciones para Manipular Cadenas

3.4.5.1 Función CHR o caracter

La función CHR devuelve el carácter ASCII asociado al número ordinal del carácter. Usa el siguiente esquema:

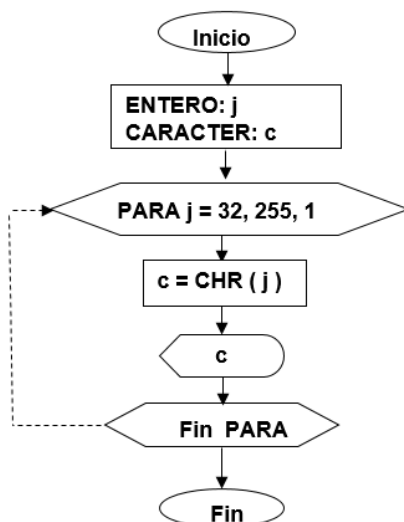
Variable = **CHR** (número ordinal)

Variable: Debe ser tipo cadena, almacena el carácter ASCII correspondiente.

Número: Es un valor numérico entero cuyo rango es de 0 a 255.

Ejemplos: X = CHR (78) X = CHR (165) X = CHR (64) X = CHR (57)
 ↓ ↓ ↓ ↓
 N Ñ @ 9

Ejemplo: Diagrama que permita mostrar los caracteres imprimibles (32 – 255) de la tabla de códigos ASCII.



En este diagrama, el ordinal para los códigos ASCII corresponde a la variable controladora de bucle.

3.4.5.2 Función ASC o ASCII

La función ASC devuelve un valor entero (ordinal) que representa el código de un carácter. Usa el siguiente esquema:

Variable = **ASC** (caracter)

Variable: Debe ser tipo numérico entero.

Carácter: Corresponde a un carácter del conjunto ASCII.

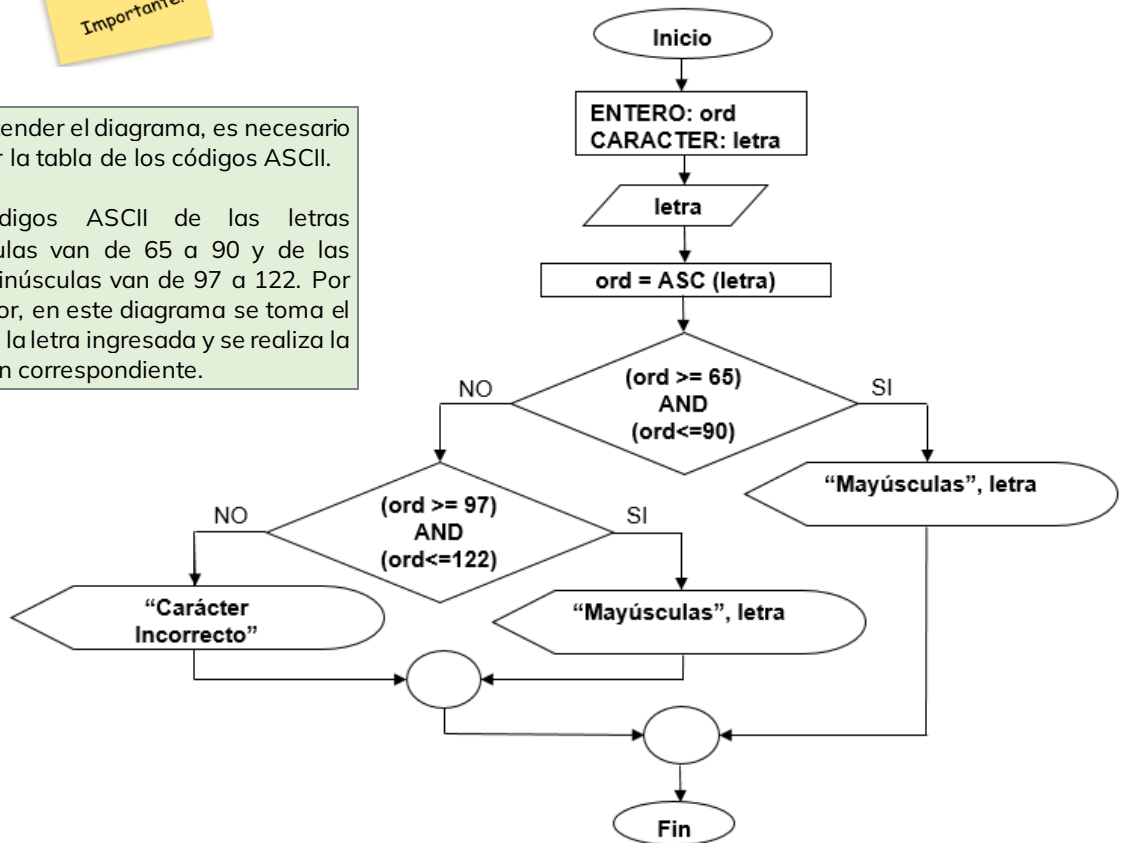
Ejemplos: A = ASC ("N") A = ASC ("Ñ") A = ASC ("@") A = ASC ("9")
 ↓ ↓ ↓ ↓
 78 165 64 57

Problema 1: Diagrama que permita ingresar una letra cualquiera, como proceso determina con un mensaje si la letra ingresada está en “mayúscula o minúscula”. En caso de ingresar cualquier otro carácter (incluyendo los tildados), envíe un mensaje “carácter incorrecto” y finalice.



Para entender el diagrama, es necesario observar la tabla de los códigos ASCII.

Los códigos ASCII de las letras mayúsculas van de 65 a 90 y de las letras minúsculas van de 97 a 122. Por lo anterior, en este diagrama se toma el ASCII de la letra ingresada y se realiza la condición correspondiente.

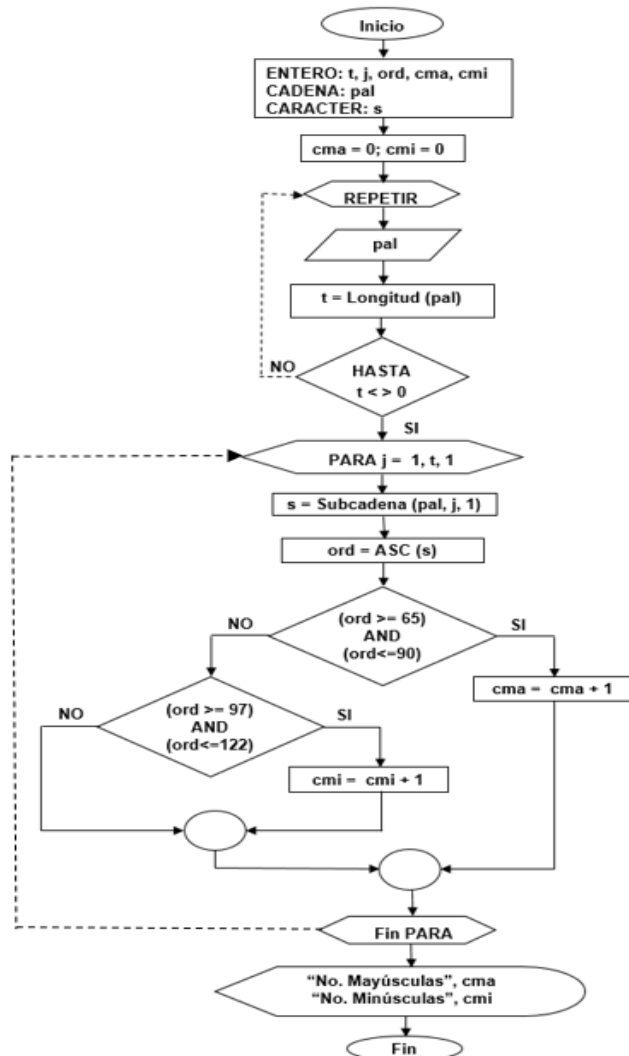


Problema 2: Diagrama que permita el ingreso de una palabra (validar que la variable de ingreso no esté vacía), se solicita como proceso contar cuantas letras hay en **minúsculas y mayúsculas** en la palabra ingresada.



En este diagrama, se recorre la palabra carácter por carácter. Una vez obtenido el carácter, se toma su respectivo "ordinal o código ASCII" y se verifica si está en el rango de las mayúsculas (65 – 90) o de las minúsculas (97-122) realizando el conteo respectivo.

Una vez terminado el recorrido, se muestran el número de mayúsculas y minúscula encontrados en la palabra ingresada.



3.4.5.5 Función VAL o ConvertirANumero

La función **VAL** (o **ConvertirANumero**) convierte una cadena que contiene números a tipo numérico. Usa el siguiente esquema:

Variable = **VAL** (cadena)

Cadena: Corresponde al *string* que contiene números y se convertirá a tipo numérico.

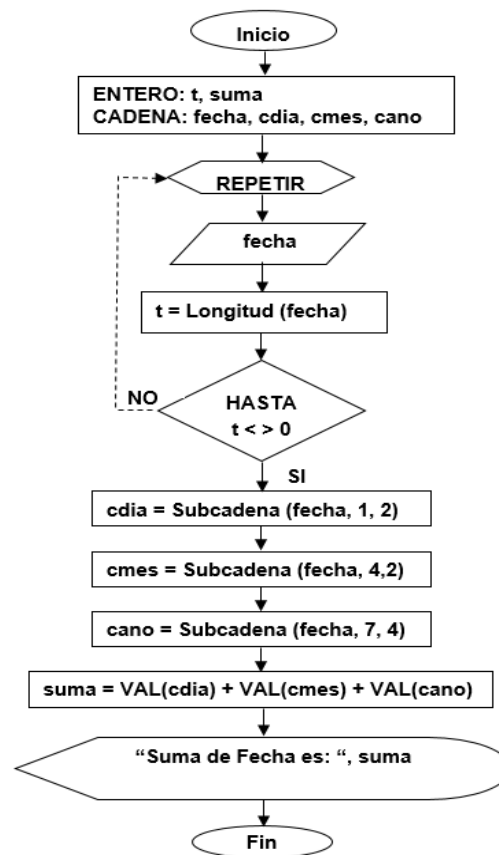
Variable: Debe ser de tipo numérica y almacena la cadena ya convertida a número.

Para realizar este tipo de conversión, considere:

- La función **VAL** deja de leer la cadena en el primer carácter que no reconoce como parte de un número. No se reconocen los símbolos y caracteres que a menudo se consideran partes de valores numéricos, como los signos de dólar y las comas.
- Los caracteres de espacio, tabulación y salto de línea se eliminan del argumento.
- La función **VAL** solo reconoce el punto (.) como un separador decimal válido.

Ejemplo 1: X = VAL("123")	➡	Retorna: 123
Ejemplo 2: X = VAL("25 and 50")	➡	Retorna: 25
Ejemplo 3: X = VAL(" 2 32 5")	➡	Retorna: 2325
Ejemplo 4: X = VAL("2.80")	➡	Retorna: 2.80

Problema: Diagrama que sume los valores de una fecha ingresada (validar que la variable de ingreso no esté vacía). Se asume que el formato de ingreso será en: dd/mm/año. Por ejemplo: 25/12/2025 dará como resultado: $25+12+2025 = 2062$.



3.4.5.6 Función STR o ConvertirATexto

La función **STR (o ConvertirATexto)** convierte números a cadenas. Usa el siguiente esquema:

Variable = **STR** (valor)

Valor: Corresponde al valor que se transformará en string o cadena.

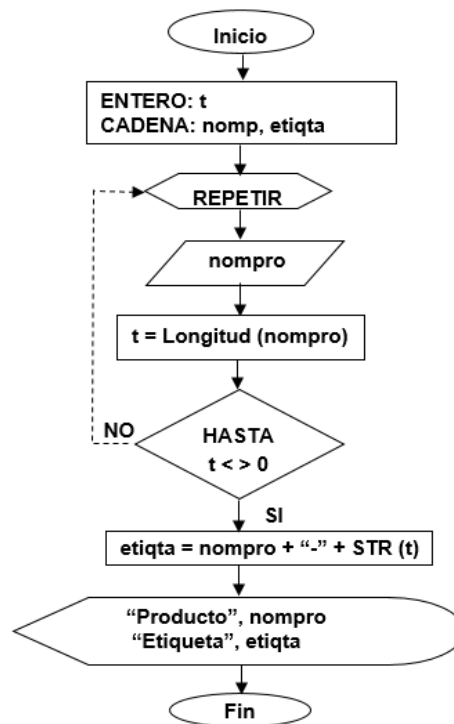
Variable: Almacena el valor convertido en *string*, por lo que su definición debe ser de tipo cadena.

Para realizar este tipo de conversión, considere:

- Cuando se convierten números en cadenas, se reserva siempre un espacio inicial para el signo del **número**. Si el número es positivo, la cadena devuelta contiene un espacio inicial y el signo **+** es implícito.
- La función **STR** sólo reconoce el **punto** (.) como un separador decimal válido.

Ejemplo 1: $X = \text{STR}(123)$ ➡ Retorna: " 123"
 Ejemplo 2: $X = \text{STR}(-250.80)$ ➡ Retorna: "-250.80"
 Ejemplo 3: $X = \text{STR}(320.001)$ ➡ Retorna: " 320.001"

Problema: Diagrama que permita generar etiquetas para productos de una empresa. Para ello, la etiqueta se genera concatenando: a) El nombre del producto, b) Un guion y c) La longitud del nombre del producto. Validar que la variable de ingreso del nombre del producto no esté vacía.



Problemas Propuestos utilizando diversas funciones para manipular cadenas

Problema 1: Crear un diagrama que permita el ingreso de una palabra. Como proceso envíe un mensaje de clasificar la palabra, según su longitud:

- Si tiene 1 a 4 caracteres: Palabra corta
- Si tiene 5 a 7 caracteres: Palabra media
- Si tiene 8 o más caracteres: Palabra larga

Problema 2: Diagrama que solicite ingresar una palabra (validar que la variable de ingreso no esté vacía) y un número **N** (validar que se encuentre entre 1 y 99, ambos inclusive). Se solicita mostrar la palabra **N veces de forma concatenada**. Ejemplo:

Entrada: hola, 3 Salida: holaholahola

Problema 3: Diagrama que permita usar una variable para ingresar una “palabra”. Como proceso, se pide obtener y mostrar la suma de los ordinales de los caracteres que conforman la palabra ingresada. Considere validar que la variable de ingreso deba contener entre 1 y 10 caracteres, ambos inclusive.

Problema 4: Diagrama que permita generar un código de matrícula para los estudiantes de la universidad. Este código se debe estructurar de la siguiente forma:

- La cédula de identidad
- Las 2 primeras letras **del nombre en mayúsculas**
- El **apellido**

Ejemplo: Cédula: 131876123-5 Nombre: Haziel Apellido: Sánchez

Resultado: 131876123-5HASánchez

Considerar la validación de: cédula, nombre y apellido (no deben estar vacías las variables de ingreso).

Problema 5: Diagrama que permita generar claves de acceso para un sistema informático. Para ello se debe ingresar: Nombre, Apellido y año de nacimiento (validar que las variables de nombre y apellido no estén vacías y que el año no debe ser menor a 1965). La generación de clave se hará siguiendo el siguiente formato concatenando:

- El nombre (en mayúsculas)
- Las 2 últimas letras del apellido (en mayúsculas)
- Los 2 últimos dígitos del año

Ejemplo: Nombre: Haziel Apellido: Sánchez Año Nacimiento: 2017

Resultado: HAZIELEZ17

Problema 6: Diagrama que permita ingresar una frase. Se pide “transformar la frase” en una contraseña. Para aquello, reemplace cada vocal de la frase (ya sea que esté en mayúscula o minúscula) con los siguientes valores:

A = 1, a = 2 E = 3, e = 4 I = 5, i = 6 O = 7, o = 8 U = 9, u = 0

Ejemplo: Hoy si hay clase H8y s6 h8y cl2s4

Considere que la frase no debe estar vacía y asuma que se excluyen los caracteres tildados.

Problema 7: Diagrama que permita ingresar el nombre completo de una persona (validar que el nombre este formado por 4 palabras). Se pide como proceso generar una abreviatura usando la primera letra de cada palabra del nombre.

Ejemplo: Nombre: Ian Alexandro Sánchez Molina

Resultado: I.A.S.M

Problema 8: Diagrama que solicite ingresar una palabra (validar que la variable de ingreso no esté vacía). Se solicita mostrar otra, donde cada palabra esté invertida, pero manteniendo el orden original de las palabras.

Ejemplo: Entrada: hola mundo Salida: aloh odnum

Problema 9: Diagrama que solicite ingresar una frase (validar que la variable de ingreso no esté vacía). Se solicita mostrar la palabra más larga (mayor número de caracteres) que exista en la frase.

Problema 10: Diagrama que solicite ingresar una frase y una palabra (validar que las variables de ingreso no estén vacías). Se solicita contar y mostrar cuantas aparece la palabra dentro de la frase ingresada.

Problema 11: Diagrama que solicite ingresar una frase (validar que la variable de ingreso no esté vacía). Se solicita mostrar otra, donde:

- Las letras mayúsculas se conviertan en minúsculas
- Las letras en minúsculas se conviertan en mayúsculas.

Considere para este problema usar las funciones CHR y ASC. Adicionalmente en la tabla de códigos ASCII busque la relación existente en los ORDINALES de las letras en mayúsculas y minúsculas para hacer la conversión solicitada.

Problema 12: Diagrama que solicite ingresar una palabra (validar que la variable de ingreso no esté vacía). Se solicita mostrar otra, donde cada letra se repita dos veces:

Ejemplo: Palabra: Playa Resultado: PPllaayyaa

Problema 13: Diagrama que solicite ingresar dos palabras (validar que las variables de ingreso no estén vacías y que las palabras tengan el mismo tamaño). Se solicita mostrar otra donde se intercalen los caracteres entre las dos palabras.

Ejemplo: Palabra1: Sol Palabra2: Luz Resultado: SLoulz

Problema 14: Diagrama que solicite ingresar una frase (validar que la variable de ingreso no esté vacía). Se solicita mostrar otra, donde la primera letra de cada palabra se muestre en mayúscula:

Ejemplo: Frase: La tarde en el campo Resultado: La Tarde En El Campo

Problema 15: Diagrama que solicite ingresar una cadena de caracteres (validar que la variable de ingreso no esté vacía). Se solicita comprimir la cadena ingresada, de tal manera que convierta secuencias de caracteres repetidos en el formato "cantidad+carácter".

Ejemplo: Cadena: AAABBBBCCDEEEEEE Resultado: 3A4B2C1D5E

Capítulo IV. Modularización y uso de Funciones



Objetivo General del Capítulo

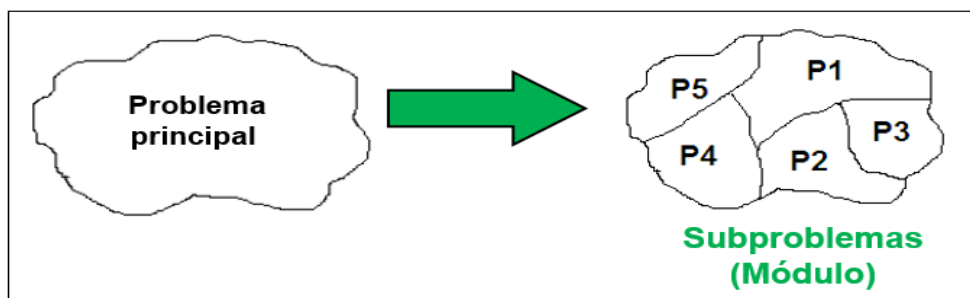
Desarrollar la comprensión del concepto de modularización como una estrategia de aprendizaje para estructurar y organizar problemas en partes más pequeñas y manejables que faciliten su comprensión, solución y desarrollo, mediante la introducción y uso módulos y funciones. Ello fortalecerá la creación de soluciones mediante módulos que mejoren su claridad, reutilización y facilidad de mantenimiento.

4.1 Modularización

Es una técnica que permite estructurar la solución de un problema en base a un diseño modular (modularización), que consiste en dividir el problema de forma lógica en otros subproblemas más pequeños llamados **módulos** (figura 17). De esta manera, en lugar de resolver un problema complejo y tedioso (problema principal), resolvemos otros más sencillos (subproblemas) y, a partir de ellos, llegamos a la solución del problema planteado.

Figura 17

Esquema de Modularización de un Problema



Nota. Fuente: Elaboración propia

Cada módulo representa una tarea específica dentro de la solución del problema que para efectos de formación, inicialmente se representarán mediante diagramas de flujo, para luego implementarse mediante funciones o procedimientos en un programa, tomando el nombre de **programación modular**.

4.1.1 Ventajas de la Modularización

- Disminuye la complejidad de los problemas a resolver, pues la solución se divide en módulos: partes o subtareas más simples o específicas. Dicha solución se organiza en forma jerárquica.
- Cuando existe una tarea que debe ejecutarse en más de una ocasión, la misma se transforma en un módulo y se la llama cuando sea necesario, ya sea para el mismo o diferentes programas (reutilización de código). Esto evita tener procesos repetitivos.
- Aumenta la claridad y fiabilidad de los programas.

- Facilita la depuración (modificaciones y correcciones) de programas, es decir, si un programa tiene inconsistencias (errores, defectos o fallas), solamente se localiza dicho módulo para su revisión.
- Fomenta la participación en grupos de desarrollo, permitiendo que varias personas puedan trabajar en la misma solución del problema, según su grado de complejidad y conocimiento.
- Facilita la ampliación del software mediante nuevos módulos.
- Disminuye el costo en el desarrollo de software.



En síntesis, la modularización busca **resolver problemas complejos dividiéndolos en problemas más pequeños**, cada uno implementado mediante un módulo que forma parte de la solución completa.

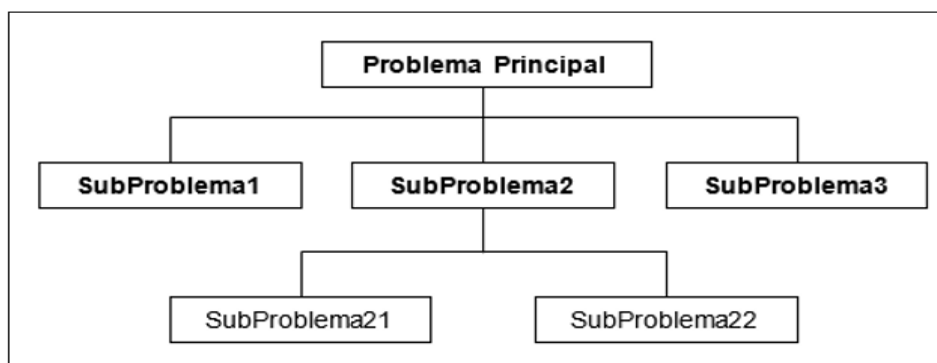
4.2 Diseño Top-Down

El diseño Top-Down o diseño descendente, es una técnica que consiste abordar un problema de lo general a lo particular, es decir, se comienza por una visión general de la solución y luego ir descomponiéndola en partes más pequeñas y específicas, para facilitar la comprensión y su solución.

En esta técnica (figura 18), primero se define el problema principal (o módulo principal) y luego ese módulo se divide en subproblemas (o submódulos), cada uno encargado de realizar una tarea específica, que al integrarse conforman la solución completa del problema. Si algún subproblema aún es complejo, puede a su vez subdividirse en otros subproblemas.

Figura 18

Esquema del Diseño Top-Down



Nota. Fuente: Elaboración propia



El diseño Top-Down se utiliza frecuentemente en la **programación modular**, ya que permite estructurar el programa de forma jerárquica y organizada, facilitando la comprensión del problema y la construcción de la solución paso a paso.

4.3 Elementos del Diseño Modular

Para lograr una adecuada estructuración del problema mediante esta técnica, es necesario considerar los dos siguientes elementos que definen la relación y organización del diseño.

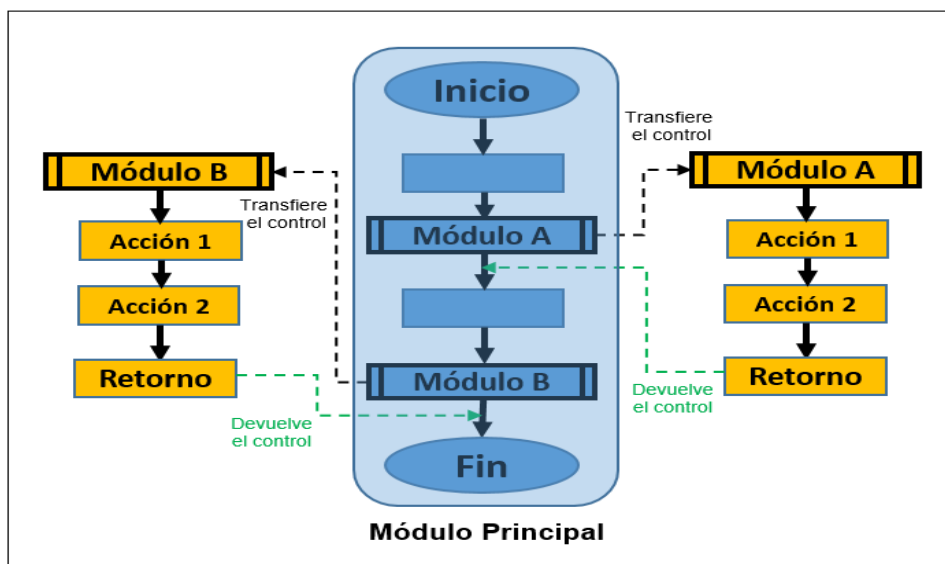
- Módulo Principal
- Módulos o submódulos

4.3.1 Módulo Principal

El módulo principal corresponde a la parte principal del problema y representa el punto de inicio de la solución dentro del diseño modular. Desde este módulo principal se coordinan y se llaman a ejecutar a los diferentes módulos o submódulos que forman parte de la solución (figura 19). Por lo tanto, el módulo principal controla el flujo general del proceso, organizando las diferentes tareas que deben ejecutarse para resolver el problema.

Figura 19

Esquema de un Diseño Modular



Nota. Fuente: Elaboración propia

Al diseñar el módulo principal, considere los siguientes aspectos:

- Dado que el módulo principal controla el flujo general del proceso, es el encargado de iniciar siempre la ejecución de la solución. En diagramación se lo reconoce porque tiene el símbolo de **inicio** y el símbolo de fin. Al codificar el diagrama, en programación por lo general lleva la palabra **Main**.
- Desde el módulo principal se organizan las llamadas a los diferentes módulos o submódulos que realizan tareas específicas (figura 4.x). Por lo tanto, se visualiza de forma general cómo se organiza y funciona la solución y cómo se produce la relación con los diferentes módulos.
- Al llamarse a un módulo o submódulo, el flujo o control del proceso pasa a este, y se ejecutan las acciones o instrucciones establecidas en él, y una vez acabada su ejecución, se retorna el control al punto siguiente donde se realizó la llamada.



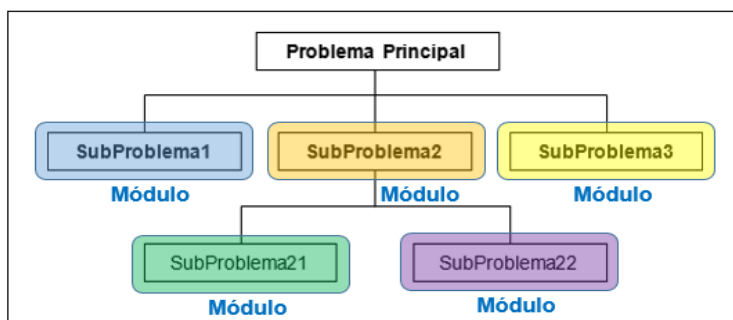
Cuando el módulo principal es codificado, es decir, cuando ya se programa, toma el nombre de **Programa Principal**.

4.3.2 Módulos o Submódulos

Un módulo es una sección separada del módulo principal, en el que se diseña y resuelve uno de los subproblemas en que se divide el problema original (figura 4.4). Por lo tanto, un módulo es autónomo ya que tiene y realiza una tarea o acción específica dentro de **todo** el proceso de solución. Al ser autónomo, un módulo se puede desarrollar, probar y mantener de forma independiente, lo que facilita su comprensión y la gestión del código (en programación) haciéndolo más legible.

Figura 20

Esquema de división de un problema mediante un Diseño Top-Down



Nota. Fuente: Elaboración propia

4.3.2.1 Consideraciones al utilizar los Módulos o Submódulos.

Al diseñar un **módulo o submódulo** dentro de un esquema modular, es necesario considerar los siguientes aspectos que permitan mantener una estructura clara y organizada:

- Todo módulo o submódulo debe tener una tarea específica, es decir, debe encargarse de una función o actividad dentro de la solución del problema. Ello dependerá del problema a resolver.
- Si una tarea se repite en varias partes de la solución, en ese caso, es recomendable crear un módulo que contenga esa tarea y pueda ser reutilizado las veces que sea necesario.
- Todo módulo debe tener un nombre representativo, que es definido por el Desarrollador y/o programador. Se recomienda sea descriptivo o nemotécnico.
- El nombre del módulo va encerrado en el siguiente símbolo en diagramación:



- En programación se debe revisar la documentación del Lenguaje de Programación a utilizar para definir un módulo. En algunos se usa la palabra PROCEDURE, DEF, FUNCTION, PUBLIC/PRIVATE, etc.
- El módulo debe finalizar con la palabra **Retorno** (en programación por lo general se usa: **Return**).
- Para llamar a un módulo desde el programa principal o desde cualquier otra ubicación, siempre se lo hará con el nombre con el que fue definido.
- Un módulo puede ser llamado o invocado las veces que sea necesario.
- Un módulo, al ser considerado un subproblema (en programación subprograma), puede tener sus propias declaraciones de variables.
- Los módulos también son conocidos como: Subproblemas, Subprogramas, Procedimientos, Funciones, Subrutinas, Rutinas, etc.

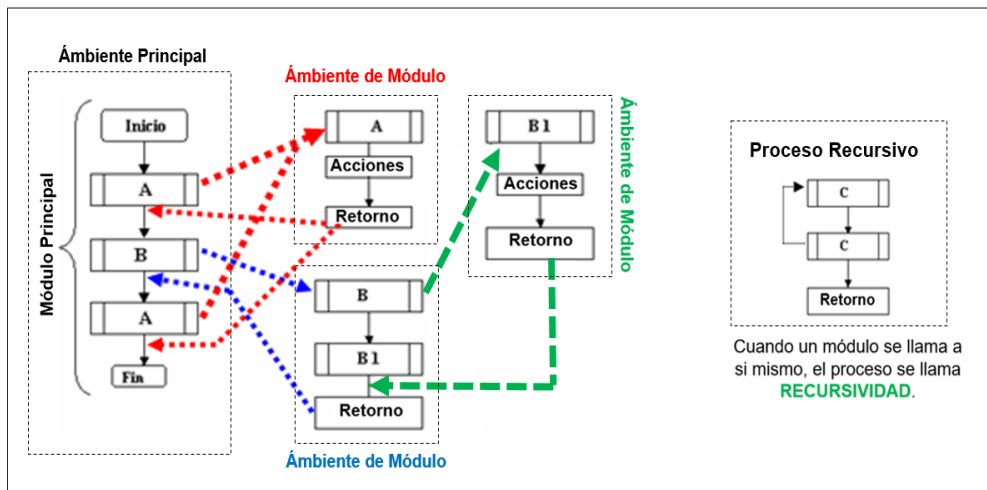
4.3.2.2 Llamada a Ejecución de un Módulo o Submódulo

La llamada a un módulo es el proceso mediante el cual se **“invoca a ejecutar un módulo”** y el flujo de control se transfiere a otro ambiente de trabajo (figura 21) que forma parte de la solución del problema. Para ello tenga en consideración lo siguiente:

- La llamada a ejecución de un módulo, por lo general se realiza desde del módulo principal (en programación será a través del programa principal) mediante el nombre con que fue declarado el módulo.
- Cuando se invoca un módulo concreto (a través de su nombre y parámetros), el control del flujo de proceso, pasará al ambiente del módulo.
- Dentro del módulo se realizan las tareas o acciones específicas, para lo cual fue creado el módulo.
- Al final, al encontrar la palabra **retorno** (o **Return**), se retorna el flujo de control a ejecutar el siguiente proceso que sigue a la llamada del módulo, que por lo general es al módulo principal o a otro sitio (puede ser otro módulo), según el esquema de diseño y solución.

Figura 21

Ambientes de Ejecución de un Módulo



Nota. Fuente: Elaboración propia

Problemas Resueltos con uso de Módulos

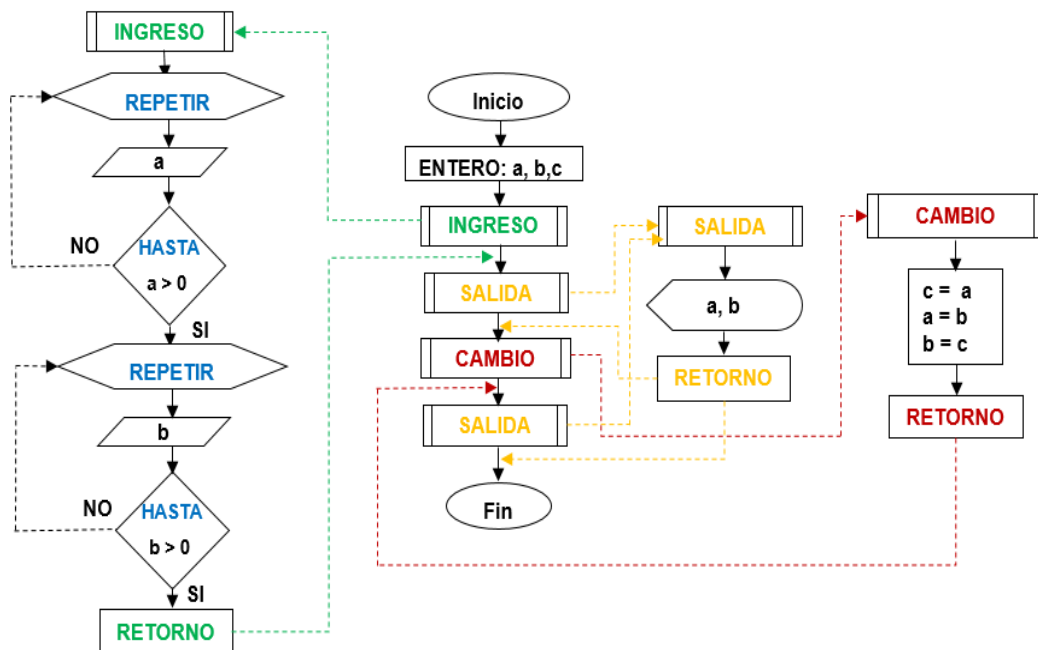
Problema 1: Diagrama que permita intercambiar valores entre dos (2) variables, cuyos valores son previamente ingresados (use las variables A y B). Usar los siguientes módulos:

- Un módulo que permita ingresar y validar los valores (deben ser positivos).
- Un módulo que permita el intercambio de valores entre las dos (2) variables ingresadas (no se debe perder ningún valor).
- Un módulo que muestre el contenido de las variables, antes y después del intercambio.

Módulo de ingreso y validación de valores.

Dado que el módulo principal controla el flujo general del proceso, es el encargado de **iniciar siempre** la ejecución de la solución.

Módulo de intercambio de valores entre las dos variables.



En este diagrama, el módulo llamado "salida", se invoca dos veces desde el módulo principal, para mostrar el contenido de las variables "antes y después" del intercambio. En este caso, como el proceso de impresión por pantalla es repetitivo, para evitar ese doble proceso, se lo ubico dentro de un solo módulo.



Este Pseudocódigo en formato del software **Pseint**, se utilizan las siguientes instrucciones:

1 El **Módulo Principal** se define con las palabras: **Proceso Principal** y **FinProceso**.

2 Los módulos o submódulos, se definen con las palabras: **SubProceso** y **FinSubProceso**, es equivalente a **Return**.

La definición del programa principal, por lo general va al principio del Pseudocódigo y al final la definición de los módulos. Sin embargo, esto queda a criterio de quien crea el código, puesto que también pueden ubicarse los módulos al principio y el módulo principal al final. Esto también es válido para el orden en que se definen los módulos.

// Módulo Principal

Proceso Principal

Definir a, b, c **Como Entero**

INGRESO

SALIDA

CAMBIO

SALIDA

FinProceso

// Módulo de Ingreso

SubProceso INGRESO

Repetir

Escribir "Ingrese Primer Valor"

Leer a

Hasta Que a > 0

Repetir

Escribir "Ingrese Segundo Valor"

Leer b

Hasta Que b > 0

FinSubProceso

// Módulo de Salida

SubProceso SALIDA

Escribir a, b

FinSubProceso

// Módulo de intercambio de valores

SubProceso CAMBIO

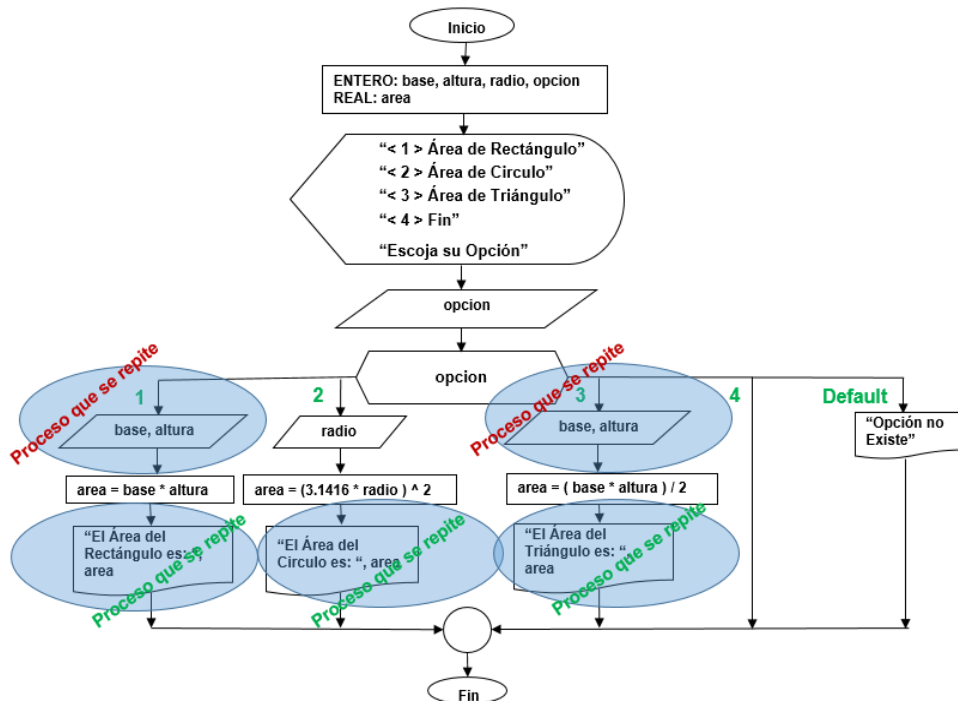
c = a

a = b

b = c

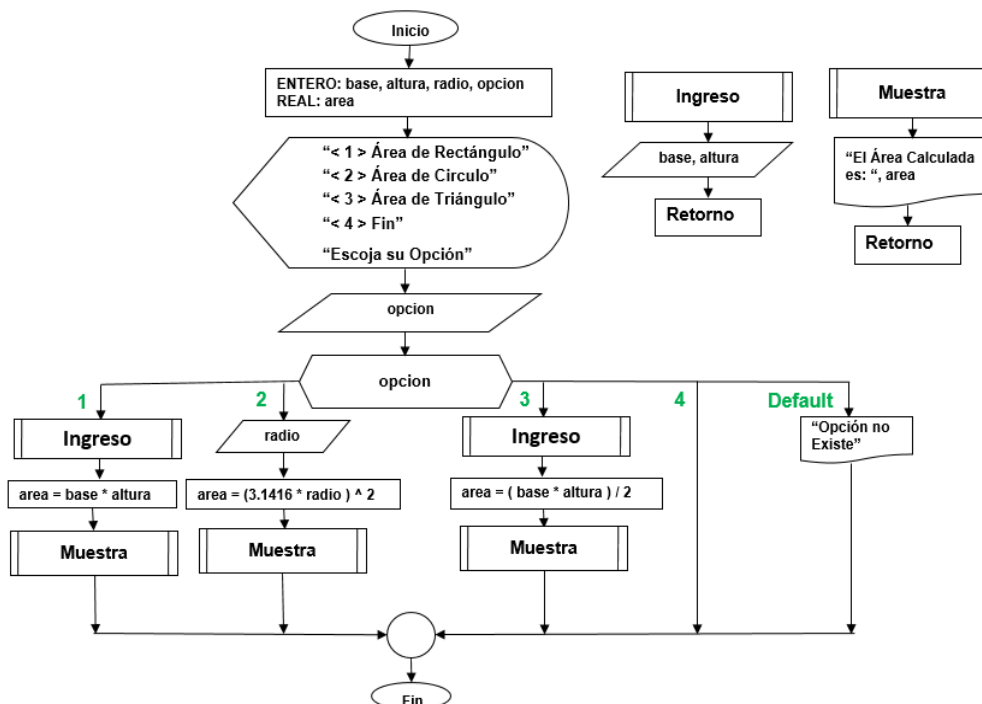
FinSubProceso

Problema 2: Diagrama que permita efectuar el cálculo del área de un: Rectángulo, Círculo y Triángulo según la opción seleccionada por el usuario a través de un menú. Además se deben ingresar los datos adicionales que se requieran para el cálculo del área según opción seleccionada.



En el diagrama se observa que hay dos procesos que se repiten:

1. El ingreso de Base y Altura.
2. La salida, donde se muestra el área calcula.



En este diagrama los dos procesos que se repiten se han ubicado en los módulos: **Ingreso** y **Muestra**. Los mismos que son invocados de acuerdo al flujo de control del diagrama.

Problema 3: El siguiente diagrama permite realizar un rol de pagos, en donde se ingresarán tantos empleados hasta que en el nombre de empleado se ingrese un espacio (" "). A continuación se muestra el gráfico del problema y al final del mismo los aspectos a considerar en cuanto a ingresos (* V representa ingreso y validación), los módulos que se solicitan, los cálculos a realizar según los datos de entrada (representados con la letra "C") y los totales que se requieren (representados por una línea con "x" en sus extremos).

Rol de Pagos

Nombre	* V Número Horas	* V Valor Hora	C Sueldo	C Iess	C Valor pagar
	40 ... 160	5 ... 15			
Totales	X—X	X—X	X—X	X—X	X—X

Procesar: Tantos empleados hasta ingresar espacio en el nombre.

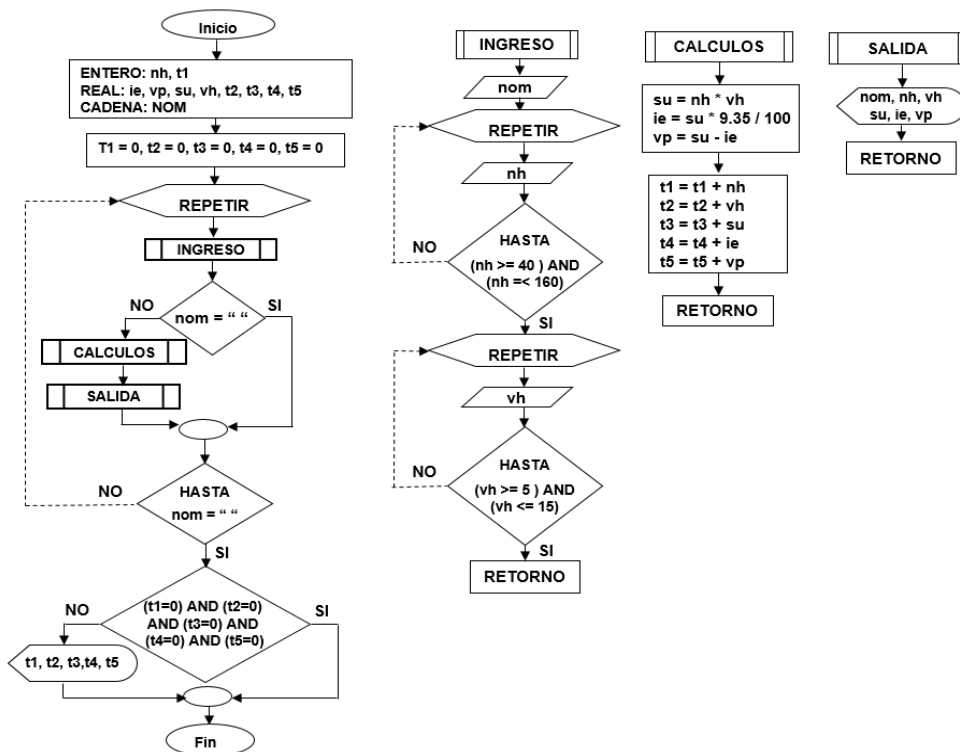
Usar un módulo para ingreso: * V

Usar un módulo para cálculos: C

Usar un módulo Mostrar: Los datos ingresados y calculados

Totales: X—X

Al final: Muestre totales según gráfico.



Esta condición, se establece en caso de que la primera vez al nombre se ingrese el espacio. Si los totales están con cero, no se deberían mostrar.

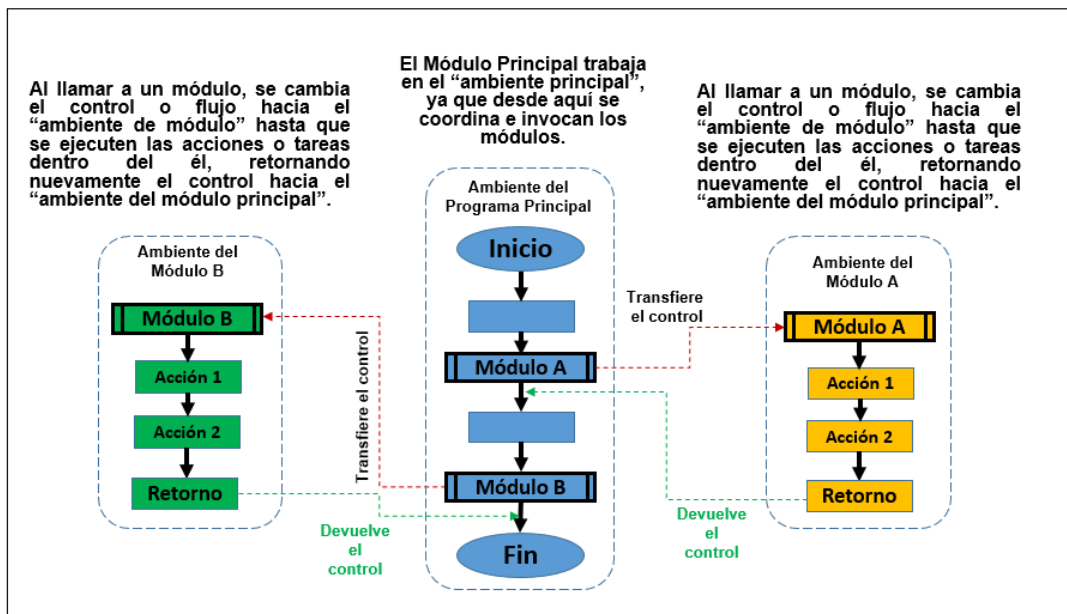
Este diagrama permite observar como un problema se puede dividir en subproblemas. Cada subproblema corresponde a un módulo que realiza una tarea específica como parte de la solución global del problema.

4.3.3 Ambientes de Trabajo en el Diseño Modular

En el diseño modular, los ambientes de trabajo se refieren al entorno (figura 22) en donde se **ejecutan las tareas o acciones** y en el lugar en donde se **definen y se utilizan las variables**. Por lo general, son dos tipos de ambientes: El ambiente principal y en ambiente de módulo. Estos ambientes determinan a que contenidos de variables se puede acceder o compartir, ya sea desde el módulo principal o desde un módulo o submódulo.

Figura 22

Ambientes de trabajo y control de flujo en el Diseño Modular



Nota. Fuente: Elaboración propia



En el diseño modular, comprender los ambientes de trabajo es fundamental para definir correctamente, cómo se comparte información entre el módulo principal y el resto de módulos.

4.3.4 Tipos de Variables Según su Ambiente de Trabajo

En el diseño modular, según el ambiente y/o alcance de trabajo y su lugar de definición o declaración, las variables se pueden clasificar en:

- Variables Globales
- Variables Locales

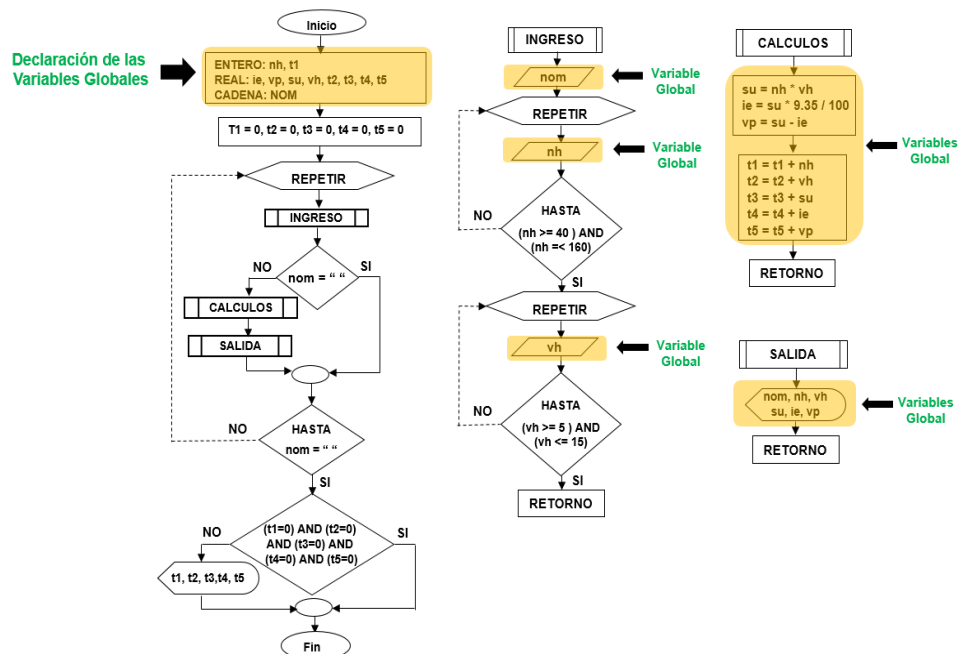
4.3.4.1 Variables Globales

Se declaran en el ambiente del módulo principal (programa principal en programación), por lo tanto, están disponibles para su uso en **TODO** el contexto del diseño modular (incluyendo el módulo principal y los submódulos), otorgándoles un alcance global.

Para utilizar este tipo de variables se debe tener en consideración los siguientes aspectos:

- Las variables globales generalmente se definen al inicio del módulo principal (o programa principal), antes de los módulos.
- Al ser accesible y modificables desde cualquier ambiente, las variables globales pueden ser utilizadas para almacenar datos que necesiten ser compartidos y utilizados por múltiples módulos.
- El uso excesivo de variables globales puede dificultar el mantenimiento y comprensión del flujo de proceso, ya que cualquier módulo puede modificar su valor. Por lo tanto, hay que tener claro qué módulo utiliza y modifica una variable global.

Ejemplo de variables locales: Problema para realizar un rol de pagos.



Las variables globales **mantienen su valor** durante todo el proceso de ejecución, a menos que sea modificado en algún módulo.

En el diseño modular, las variables locales permiten que cada módulo **trabaje con sus propios datos**, evitando interferencias con otros módulos del programa.

4.3.4.2 Variables Locales

Son aquellas variables declaradas dentro de un módulo específico y solo están disponibles dentro de ese módulo.

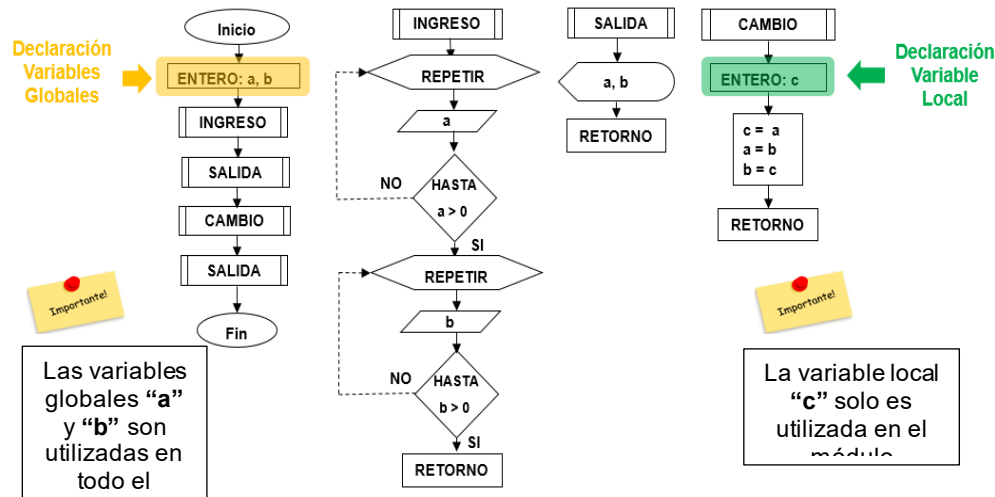
Para utilizar este tipo de variables se debe tener en consideración los siguientes aspectos:

- La declaración de una variable local, se hace igual que cualquier otra variable.
- Estas variables tienen un alcance local y no puede ser utilizada fuera del módulo en el que fue definida.
- Se utilizan para almacenar valores temporales o intermedios necesarios para la ejecución del módulo. Por lo tanto, cualquier cambio de valor que se haga a una variable local, sólo tiene efecto dentro del módulo que la utiliza.
- Una variable local es “temporal”, pues sólo existe mientras se está ejecutando el módulo (ambiente de módulo). Una vez que el módulo termina de ejecutarse, las variables locales se eliminan de la memoria.

Ejemplos de variables locales

Problema 1: Diagrama que permita intercambiar valores entre dos (2) variables, cuyos valores son previamente ingresados (use las variables A y B). Usar los siguientes módulos:

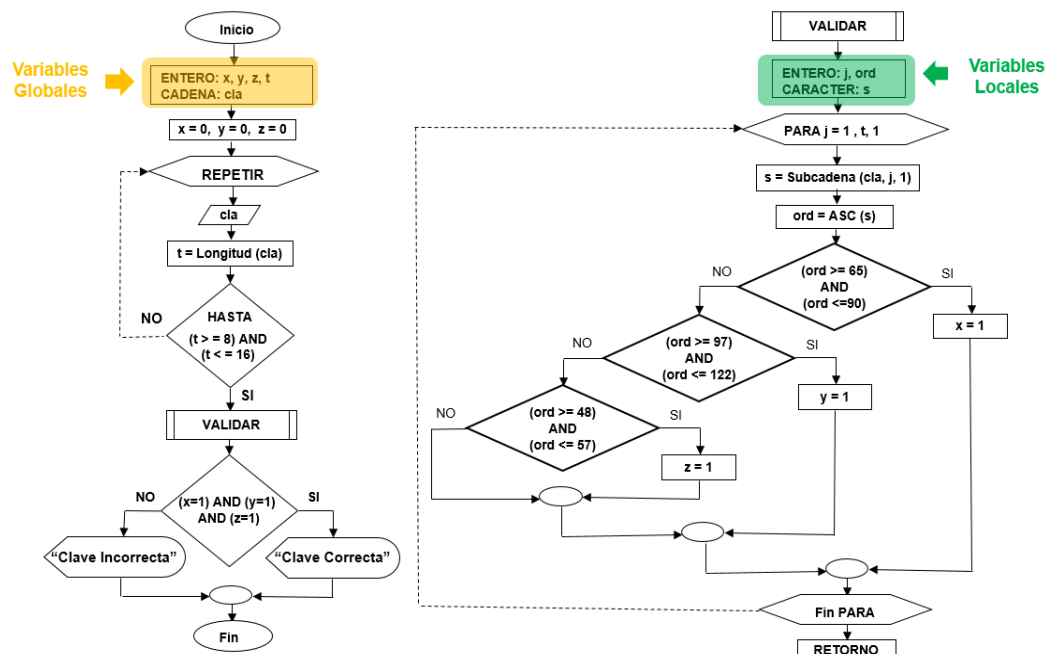
- Un módulo que permita ingresar y validar los valores (deben ser positivos).
- Un módulo que permita el intercambio de valores entre las dos (2) variables ingresadas (no se debe perder ningún valor).
- Un módulo que muestre el contenido de las variables, antes y después del intercambio.



Problema 2: Diagrama que permita ingresar una clave. Dicha clave debe cumplir los siguientes requerimientos:

- Tener mínimo 8 y máximo 16 caracteres.
- Utilizar un módulo para validar que: La clave tenga mínimo una letra en mayúsculas, una en minúsculas y mínimo un número.
-

Al final muestre si la clave ingresada fue o no correcta.



4.4 Parametrización

La parametrización es un mecanismo de comunicación, mediante el cual se envía datos entre el módulo principal (o programa principal) y los módulos o submódulos, para que puedan trabajar con ellos durante la ejecución de una tarea específica. Por lo tanto, es una forma de canalizar información entre los diversos ambientes de los elementos de un diseño modular.

La importancia de la parametrización radica en:

- Personalizar el comportamiento de un módulo o función.
- Intercambiar datos con uso de parámetros, entre los elementos del diseño modular.
- La reutilización de código en un ambiente de programación.
- Evitar el uso excesivo de variables globales.



Gracias a la parametrización, un mismo módulo puede **utilizarse varias veces con diferentes datos**, lo que aumenta la **flexibilidad y reutilización del código en programación**.

4.4.1 Parámetros

En programación, se trata de variables y/o valores, que proporciona el módulo principal (o programa principal) al módulo o función, o ser devuelto desde el módulo o función hasta el programa módulo principal. Es decir, cuando un módulo o función necesita trabajar con datos que provienen de otro módulo, esos datos se transmiten mediante parámetros. De esta manera, los módulos **“se comuniquen entre sí”** y realizan sus tareas utilizando los valores que reciben.

Existen dos formas de clasificar a los parámetros:

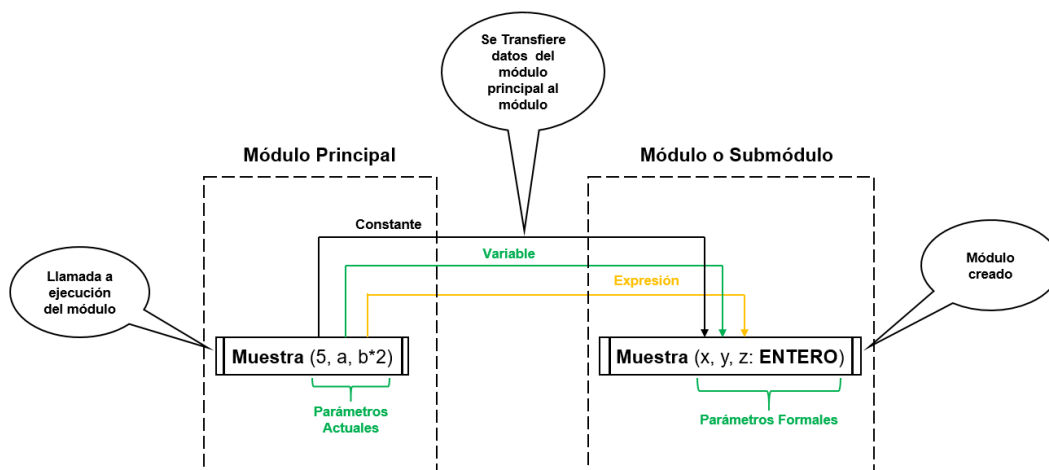
- a) Según su función en la llamada al módulo
- b) Según la forma de transmisión de datos al módulo

4.4.1.1 Clasificación de los Parámetros Según su Función en la Llamada al Módulo

Según su función en la llamada al módulo, los parámetros pueden clasificarse **Parámetros Actuales o Argumentos** y **parámetros formales**, los cuales permiten establecer la comunicación de información entre el módulo que realiza la llamada y el módulo que la recibe.

- Las variables definidas como parámetros formales, se declaran junto al nombre del módulo o función.
- Siempre van entre paréntesis y separados por comas cuando son varias.
- Los parámetros actuales en la llamada al módulo deben coincidir en número, orden y tipo con los parámetros formales de la declaración del procedimiento.
- La correspondencia entre los parámetros actuales y formales una vez que el procedimiento es llamado a ejecución se dará de izquierda a derecha.

En el siguiente ejemplo, se muestra como se envía datos a través de una constante, una variable y una expresión (que son los **parámetros actuales o argumentos**), hacia un módulo. Al crearse este módulo, se deben declarar tres (3) variables (**que son los parámetros formales**), donde se almacenarán los datos enviados desde el módulo principal.



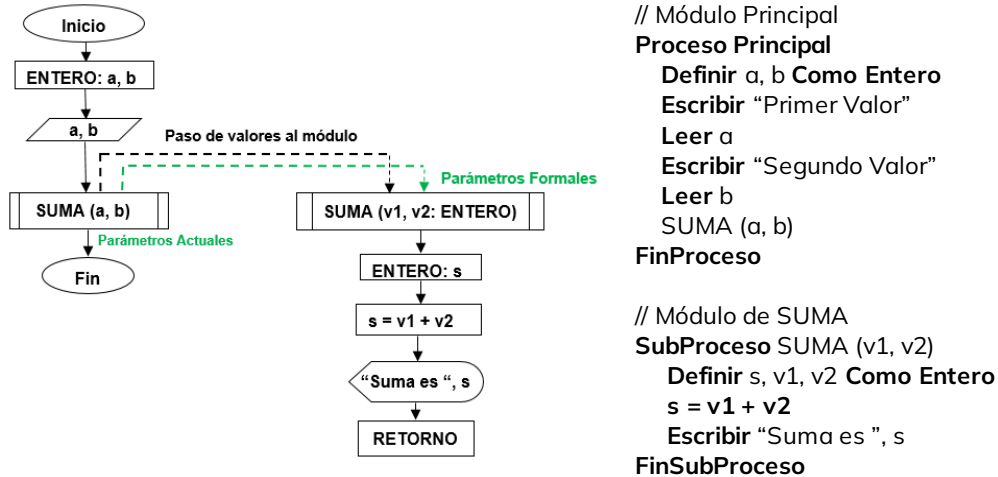
Los Parámetros Formales, actúan como espacios reservados para los valores que se pasarán cuando se llame al módulo. Dentro del módulo, se comportan como variables locales que pueden utilizarse para realizar cálculos y manipulaciones según el problema a resolver.



Cuando un módulo es llamado desde el módulo principal (o programa principal), los **parámetros actuales (argumentos)** envían los datos que serán recibidos por los **parámetros formales** del módulo.

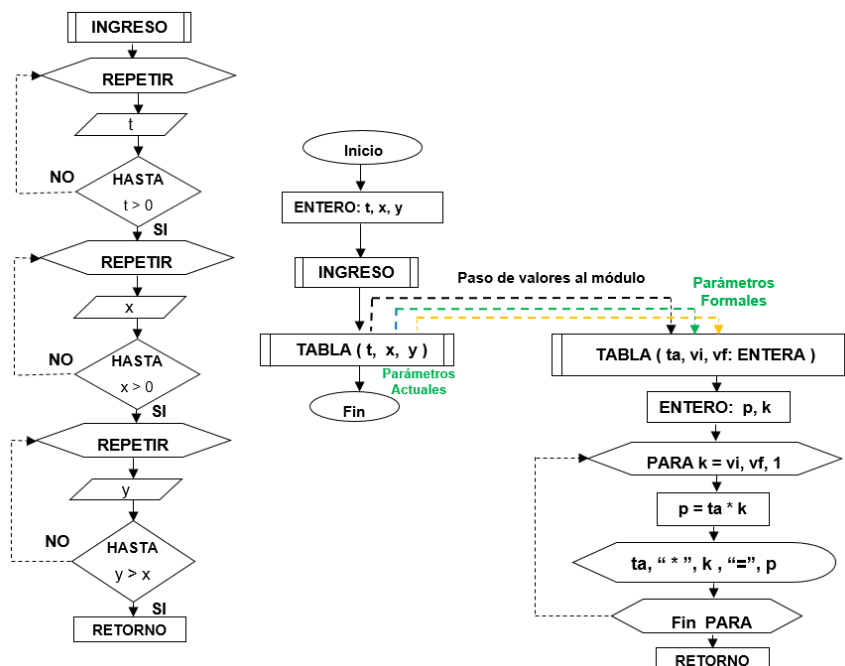
Problemas Resueltos con Módulos Parametrizados

Problema 1: Diagrama que utilizando un módulo, permita calcular la suma de dos valores ingresados en el módulo principal.



Problema 2: Diagrama que permita generar una tabla de multiplicar previamente ingresada (debe ser positiva). Dicha tabla debe ir desde un valor "X" hasta un valor "Y". Para ello use los siguientes módulos:

- Módulo de ingreso y validación (valores positivos)
- Módulo para generar la tabla, mismo que debe usar parámetros, para tomar valores desde el programa principal.




```

// Módulo Principal
Proceso Principal
    Definir t, x, y Como Entero
    INGRESO
    TABLA (t, x, y)
FinProceso

// Módulo INGRESO
SubProceso INGRESO
    Repetir
        Escribir "Ingrese Tabla"
        Leer t
        Hasta Que t > 0
    Repetir
        Escribir "Ingrese desde donde
inicia tabla"
        Leer x
        Hasta Que x > 0
    Repetir
        Escribir "Ingrese Hasta donde va
la tabla"
        Leer y
        Hasta Que y > x
FinSubProceso

// Módulo TABLA
SubProceso TABLA (ta, vi, vf)
    Definir ta, vi, vf, p, k Como Entero
    Para k ← vi Hasta vf Con Paso 1
        p = ta * k
        Escribir ta, "*", k, "=", p
    Fin Para
FinSubProceso

```



El diagrama de flujo anterior, fue codificado en Pseudocódigo, utilizando el software PSeInt.

En el software PSeInt, la definición del **tipo de dato de los parámetros formales** no se especifica en la cabecera del SubProceso, por lo que estas variables se definen, como cualquier otra variable.

En programación y dependiendo del tipo de lenguaje de utilizar, el tipo de dato de los parámetros formales se **definen dentro de la cabecera del módulo**, tal como se muestra en el diagrama.

4.4.1.2 Clasificación de los Parámetros Según la forma de Transmisión de Datos al Módulo

Los parámetros, según la forma en que los datos son transmitidos al módulo o función, se clasifican principalmente en los dos siguientes grupos, lo que determina cómo se utilizan y si las modificaciones realizadas dentro del módulo afectan o no a los datos originales.

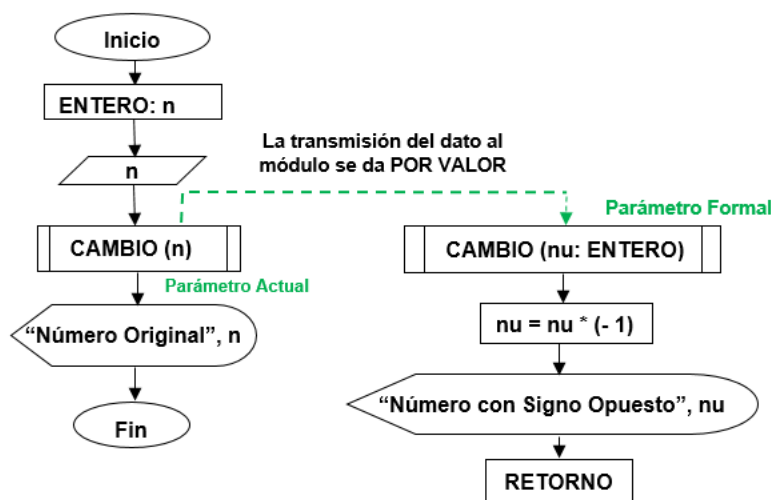
- c) Parámetros por Valor
- d) Parámetros por Referencia

4.4.1.2.1 Parámetros por Valor

Los parámetros por valor, es un mecanismo de transmisión de datos donde se envía desde el módulo principal (o programa principal) una copia del valor de los parámetros de argumento al módulo o función. El módulo o función trabaja con esa copia, por lo que **cualquier modificación realizada dentro del módulo no afecta al valor original** de la(s) variable(s) de argumento. Por lo tanto, se utilizan cuando el módulo **solo necesita utilizar el o los valores recibido(s)**.

En todos los problemas explicados anteriormente con **módulos y parámetros**, se ha utilizado como forma de transmisión de datos **parámetros por valor**, ya que los valores de las variables utilizadas como parámetros de argumentos y enviadas a un módulo no han sufrido variaciones en su valor.

Ejemplo: Diagrama que utilizando un módulo, permita el cambio de signo, obteniendo su opuesto, mismo que debe ser mostrado en el módulo. En el módulo principal ingrese el número y muéstrelo con su signo original.



Como se observa en la prueba del diagrama de flujo, el valor ingresado a la variable “n” es 5 que es el parámetro formal o argumento, que se transfiere como copia al módulo y es recibido por el parámetro formal “nu” con 5, el cual si cambia si cambia al realizar el proceso, pero el valor de “n” no cambia una vez que flujo retorna al módulo principal.

Ambiente Módulo Principal

n	Salida
5	Número Original 5

Ambiente del Módulo

nu	Salida
~~5~~	
- 5	Número con Signo Opuesto - 5

```

// Módulo Principal
Proceso Principal
  Definir n Como Entero
  Escribir "Ingrese Número"
  Leer n
  CAMBIO (n)
  Escribir "Número Original ", n
FinProceso

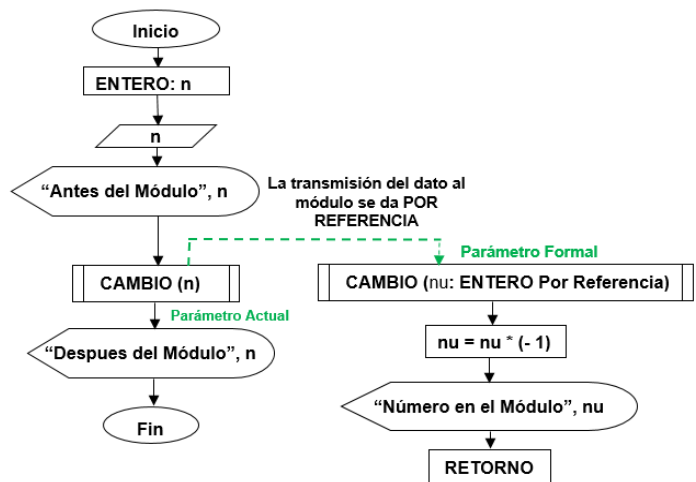
// Módulo de Cambio de signo
SubProceso CAMBIO (nu)
  Definir nu Como Entero
  nu = nu * (-1)
  Escribir "Número con Signo Opuesto ", nu
FinSubProceso

```

4.4.1.2.2 Parámetros por Referencia

Los parámetros por referencia, es un mecanismo de transmisión de datos donde se envía desde el módulo principal (o programa principal) la referencia o dirección de memoria de una o más variables como parámetros de argumento a un módulo o función, en lugar de una copia de valores. Por lo tanto, **los cambios realizados dentro del módulo sí afectan directamente a la(s) variable(s) original(es)** que fueron de argumento al módulo.

Ejemplo: Diagrama que utilizando un módulo, permita el cambio de signo, de un valor ingresado en el módulo principal. Se utilizará como forma de transmisión de datos, parámetros por referencia. El número de debe ser mostrado en los dos ambientes de trabajo.



Como se observa en la prueba del diagrama de flujo, el valor de 5 ingresado a la variable "n" es el parámetro formal o argumento, se transfiere como referencia al módulo y es recibido por el parámetro formal "nu" con 5. Dentro del módulo al realizar el proceso, cambia el valor de "nu" a -5, lo que afecta también el valor de "n" que también cambia a -5, ya que la transmisión del dato fue por referencia (dirección de memoria).

Ambiente Módulo Principal

n	Salida
~~-5~~	Antes del Módulo 5
- 5	Después del Módulo - 5

Ambiente del Módulo

nu	Salida
~~-5~~	
- 5	Número en el Módulo - 5

```

// Módulo Principal
Proceso Principal
  Definir n Como Entero
  Escribir "Ingrese Número"
  Leer n
  Escribir "Antes del Módulo", n
  CAMBIO (n)
  Escribir "Después del Módulo", n
FinProceso

// Módulo de Cambio de signo
SubProceso CAMBIO (nu Por Referencia)
  Definir nu Como Entero
  nu = nu * (-1)
  Escribir "Número en el Módulo", nu
FinSubProceso
  
```



Cuando un parámetro se pasa por referencia, cualquier modificación realizada dentro del módulo se refleja también en la variable de argumento del módulo principal

Problemas Resueltos con Módulos Parametrizados (paso por valor)

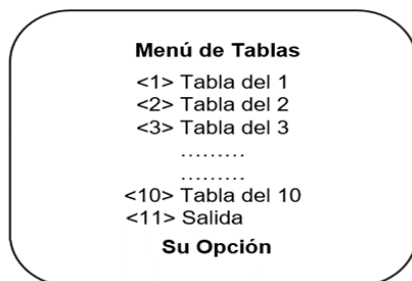
Problema 1: Diseñe un diagrama que calcule la **suma de los números entre dos valores** ingresados utilizando un módulo. El ingreso de los números se hará también mediante un módulo, donde se validen que sean positivos y que el último número no sea mayor al primero.

Problema 2: Diagrama que permita crear un módulo para **invertir los dígitos de un número entero y positivo** mediante la manipulación de cadenas. El dígito se ingresa y valida en módulo principal. Ejemplo:

Número ingresado = 7896

Número invertido = 6987

Problema 3: Diagrama que permita, **mediante un menú, enviar como opciones las 10 primeras tablas de multiplicar**. Se pide utilizar un módulo (con parámetro) para generar cualquiera de las tablas del 1 al 12. Considere que el proceso sólo termina cuando se escoja la opción de salida, en caso contrario el proceso de selección es repetitivo.



Problema 4: Diagrama que permita ingresar una frase (validar que su número de caracteres debe estar entre 3 y 70 ambos inclusive). Se pide utilizar un módulo con parámetro (pasar la frase) que permita **“centrar” y mostrar dicha frase** en una línea de la pantalla que tiene 80 columnas.

Problema 5: Diagrama que permita ingresar una frase (validar que no esté vacía). Se pide utilizar un módulo con parámetro (pasar la frase) para **transformar la frase en una clave**. Para ello reemplace cada vocal de la frase (ya sea que esté en

mayúscula o minúscula) con los siguientes valores (considere excluir los caracteres tildados):

A – a = 1 E – e = 2 I – i = 3 O – o = 4 U – u = 5

Ejemplo: Hoy si hay clase H4y s3 h1y cl1s2

4.5 Funciones

En el diseño modular, un problema se divide en subproblemas que mediante módulos ejecutan tareas específicas como parte de la solución. Sin embargo, en muchas situaciones no solo se requiere realizar una operación, sino también obtener un resultado que pueda ser utilizado posteriormente. Para este propósito se utilizan las funciones.

En programación, una **función es un tipo especial de módulo** o procedimiento, que consiste en un **bloque de código** que realiza una tarea específica y que **devuelve un único valor** al ambiente en que fue llamada. Por lo tanto, una función recibe como argumento uno o varios valores y devuelve un único resultado.

A diferencia de un módulo o submódulo, cuya función principal es ejecutar un grupo de instrucciones, una función siempre retorna un valor como resultado después de realizar una operación. Esta característica la diferencia esencialmente de un módulo o procedimiento.

4.5.1 Clasificación de las Funciones

Según su origen de creación, las funciones se clasifican en los dos siguientes tipos:

- a) Funciones Predefinidas en los Lenguajes de Programación.
- b) Funciones Definidas o Creadas por el Desarrollador / Programador.

4.5.2 Funciones Predefinidas en los Lenguajes de Programación

Todo Lenguaje de Programación incorpora **librerías** que contienen funciones estandarizadas para simplificar la tarea de programación. Estas funciones ya están desarrolladas y realizan tareas específicas, por lo que el desarrollador / programador debe:

- Conocer las diferentes funciones incorporadas al lenguaje de programación a utilizar, revisando la documentación respectiva.
- Usar la función que se ajuste a la tarea que va realizar en el programa.

Las funciones dentro de los lenguajes de programación por lo general se clasifican en:

4.5.2.1 Funciones numéricas

Para realizar tareas matemáticas complejas, tales como: raíz cuadrada, logarítmicas, exponenciales, trigonométricas, etc. Ejemplos:

ABS - SQR - SIN - COS - etc.

4.5.2.2 Funciones para uso de Cadenas

Permiten facilitar la tarea en el tratamiento de cadenas o textos, tales como: Cálculo de longitud de una cadena, subcadenas, inserción, eliminación, etc. Ejemplos:

LEN - LEFT - CONCAT - MID - UPPER - LOWER – etc.

4.5.3 Beneficios de las Funciones Predefinidas en los Lenguajes de Programación

Las funciones predefinidas son “herramientas incorporadas” que han sido optimizadas y que permiten realizar operaciones comunes en una aplicación sin necesidad de programarlas. Por lo tanto, constituyen un recurso fundamental en la práctica de la programación por los siguientes beneficios:

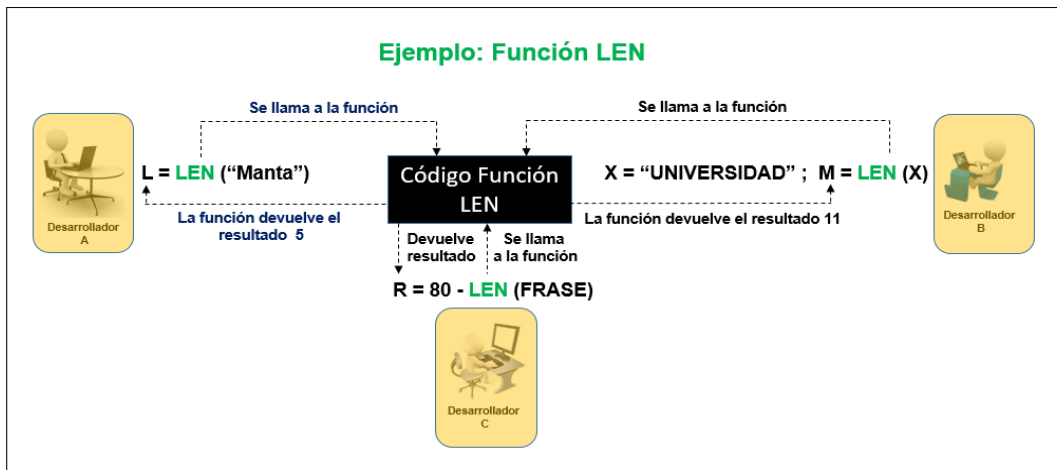
- Están definidas en el lenguaje o entorno de desarrollo y no necesitan ser programadas.
- Facilitan la resolución de problemas, simplificando determinadas tareas de programación y el ahorro de recursos.
- Contribuyen a escribir programas más claros, eficientes y confiables.
- Al encapsular una tarea específica en una función, dicha función puede ser invocada varias veces, en un mismo o diferente programa. De esta forma se aplica y promueve uno de los principios fundamentales de la programación moderna: la **reutilización de código**.

La figura 23 ilustra el funcionamiento de una función predefinida dentro de un lenguaje de programación (**función LEN**) y evidencia el principio de **reutilización y encapsulamiento** de código que ofrecen las funciones en la programación. La

figura muestra cómo una función predefinida puede ser utilizada por diferentes desarrolladores, sin necesidad de conocer las instrucciones internas de su implementación.

Figura 23

Ejemplo de Reutilización y Encapsulamiento de código de una función predefinida



Nota. Fuente: Elaboración propia

En el ejemplo gráfico se utiliza la función **LEN**, cuyo propósito es **calcular el tamaño o longitud** de una cadena de caracteres. Los desarrolladores A, B y C llaman a la misma función desde sus programas, enviando distintos datos como argumentos de entrada (en forma de constante, variable o expresión). La función procesa la información y **devuelve el resultado correspondiente**, por ejemplo, la cantidad de caracteres de la cadena recibida.

De esta manera, se ilustra como una función actúa como un **módulo reutilizable**, que puede ser invocado múltiples veces por distintos programas, proporcionando resultados según los datos que recibe.

4.5.4 Funciones Definidas o Creadas por el Desarrollador

Son aquellas funciones planificadas, diseñadas y programadas por el propio desarrollador para realizar una tarea específica dentro de un programa y que pueda ser compartida a otras soluciones. Estas funciones se crean cuando se necesita resolver un problema particular o implementar una operación que no está disponible de forma directa en el lenguaje o en el entorno de desarrollo.

Al diseñar una función considere:

- Establecer una tarea específica para la función. Esto facilita su comprensión y reutilización.
- Toda función debe tener un nombre asignado por el desarrollador, mismo que debe ser descriptivo en relación a la tarea que va a realizar. Ejemplos: calcularPotencia, calcularPromedio, intercambioValores, etc.
- Determinar adecuadamente los parámetros en cuanto a número y tipos de datos que recibirá para realizar correctamente la tarea específica.
- Definir adecuadamente el valor de resultado de la función, ya que este debe ser coherente con la operación realizada y fácil de utilizar en el módulo principal.

Para su construcción considere:

- En diagramación para definir una función se utilizará la palabra “**Funcion**”. En programación se deberá consultar el manual del lenguaje de programación a usar. Por ejemplo: En Python se usa “**def**”.
- Al definir la función, se ubica el nombre de la función y entre paréntesis, se deberá también definir las variables que actuarán como parámetros formales (separados por “,” si son varios) que permitirán receptar los valores enviados como argumentos al llamar la función. Ejemplo: Funcion SUMA (a, b)
- Los argumentos (o valores) que necesitará la función, deben ser enviados cuando se llama la función (solo se llama con el nombre de la función) y lo decide el desarrollador, según la tarea que va realizar la misma. Ejemplo: SUMA (5, 10)
- La función devuelve un valor como resultado, mismo que se retorna al usar la palabra **Retorno**, seguido de la variable con el resultado de retorno (consulte el manual del lenguaje de programación). Ejemplo: **Retorno** resultado

En resumen, para definir una función considere el siguiente esquema:

Funcion **NombreFunción** (parámetro [, parámetro1])

Acción o tarea de la función

Retorno **variable_resultado**

Es importante considerar, que el tipo de variable que va en **Retorno**, debe ser del mismo tipo que variable que recepa el resultado en la llamada al función.

4.5.4.1 Beneficios de las Funciones Creadas por el Desarrollador

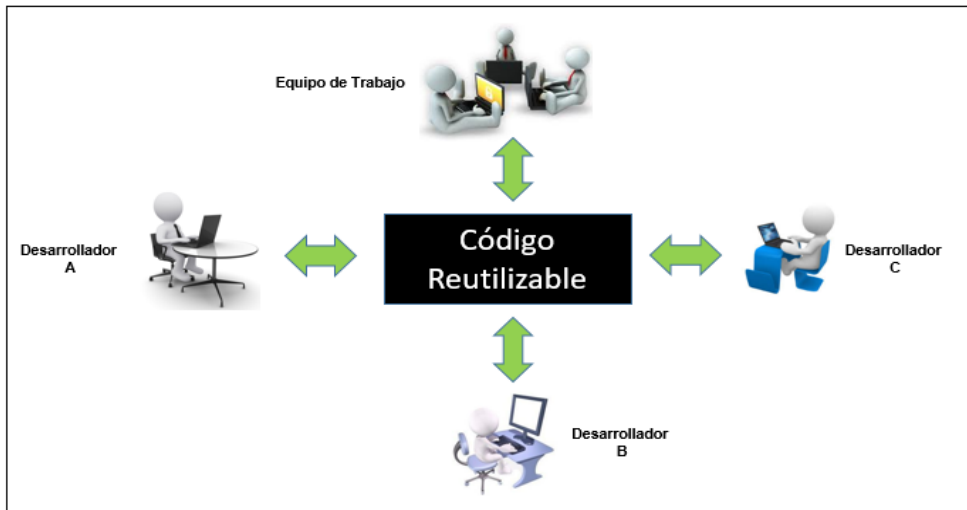
Al crear funciones un desarrollador, adapta tareas específicas a las necesidades particulares de un problema y que posiblemente se necesiten a otras soluciones, más aún, cuando el software es creado en equipos de trabajo. Algunos beneficios son:

- La reutilización de código en programación.
- Simplificar la tarea de programación al aplicar el modularidad.
- Facilidad de mantenimiento y facilidad de pruebas.
- Mejora la eficiencia y calidad del desarrollo de software.
- Aplica la Abstracción (oculta los detalles internos de su implementación).

En relación al beneficio principal de las funciones creadas por el desarrollador que es la **reutilización de código**, la figura 24 muestra cómo el código reutilizable puede ser utilizado por varios desarrolladores o miembros de un equipo de trabajo. En el centro se encuentra el módulo o conjunto de instrucciones reutilizables, al cual pueden acceder diferentes desarrolladores (A, B, C y el equipo de trabajo).

Figura 24

Ilustración de código reutilizable en el desarrollo de software

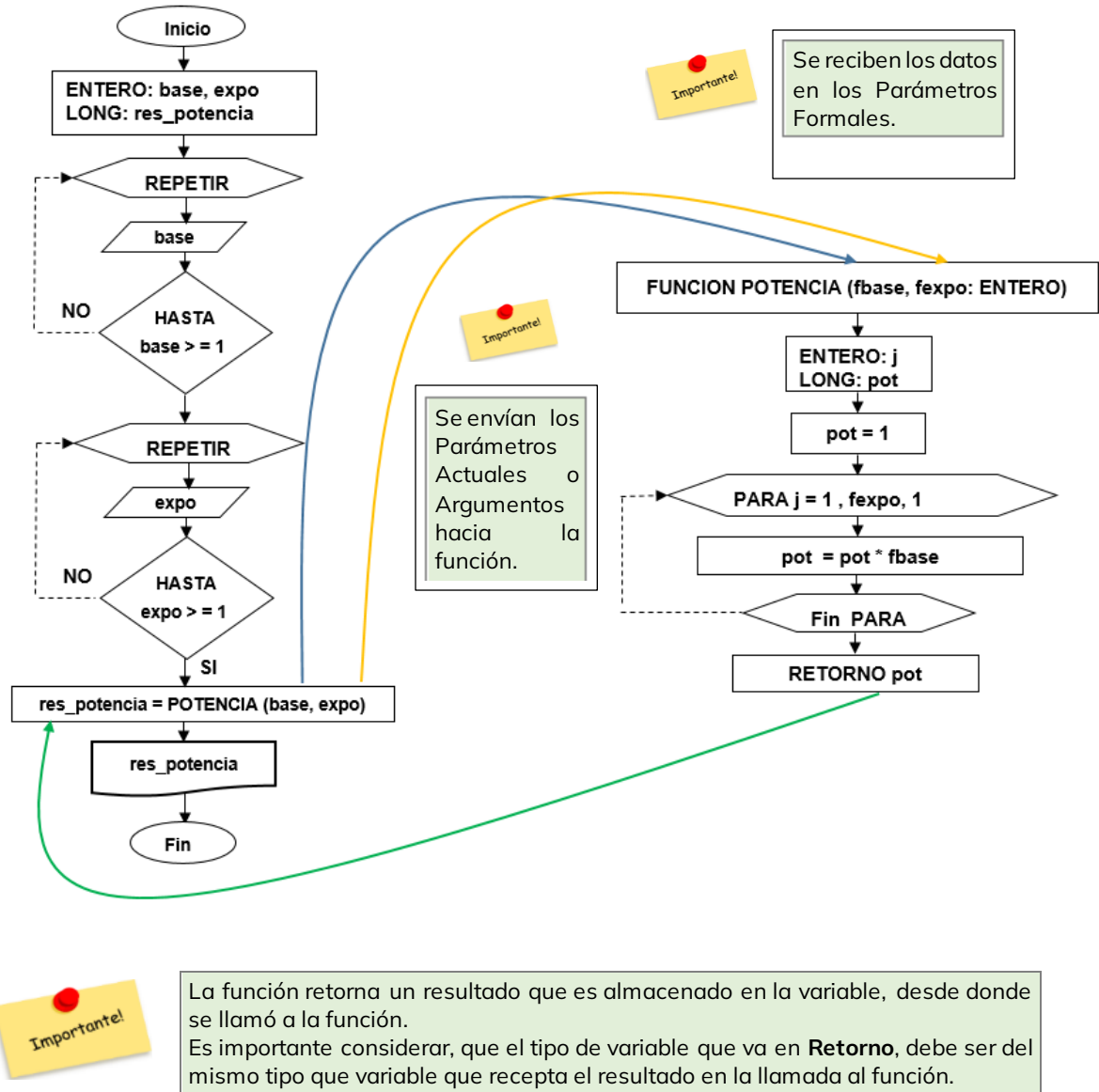


Nota. Fuente: Elaboración propia

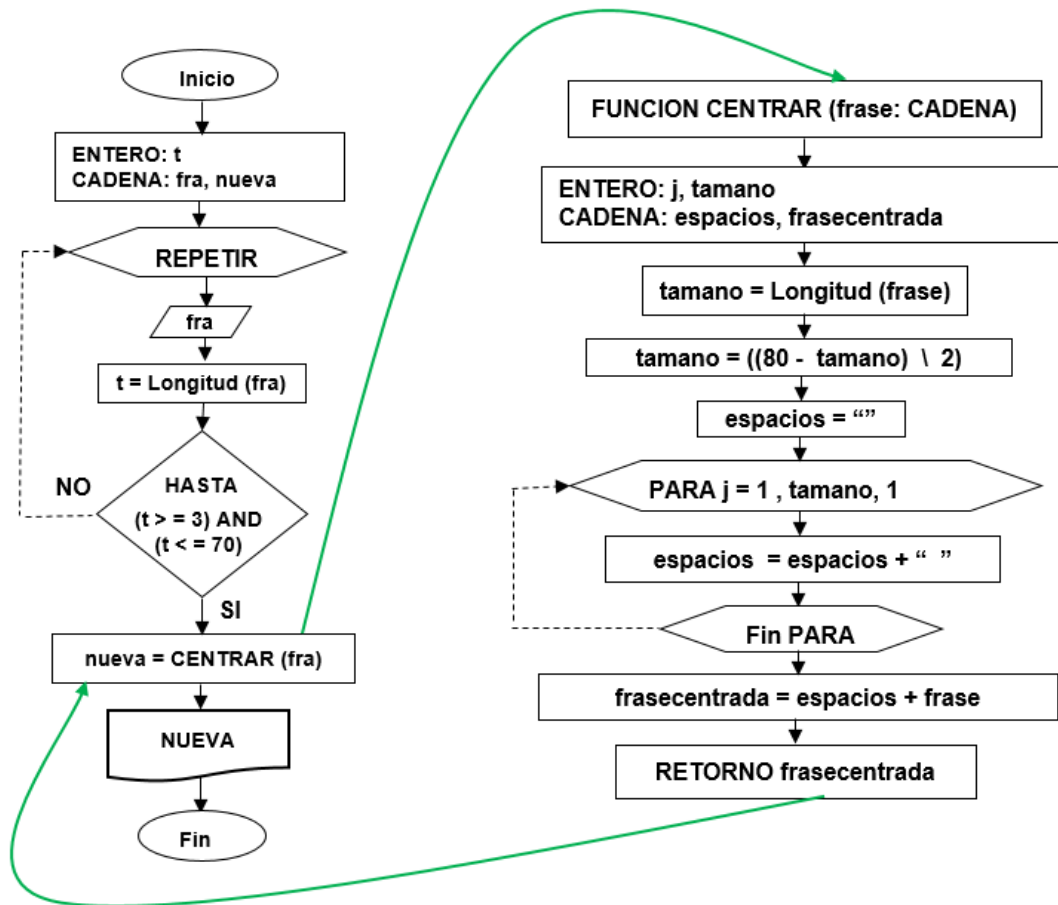
Cada desarrollador tiene **acceso a utilizar el mismo código** dentro de sus propios programas, **evitando tener que volver a programar las mismas soluciones**. De esta manera, la reutilización de código permite compartir soluciones, mejorar la eficiencia del desarrollo, ahorro de recursos y tener una mejor organización en los proyectos de software.

Problemas Resueltos con Funciones Creadas por el Desarrollador

Problema 1: Diagrama que permite crear una función para “calcular la potencia” de un valor (se valida en el módulo principal que la base y exponente sean positivos).



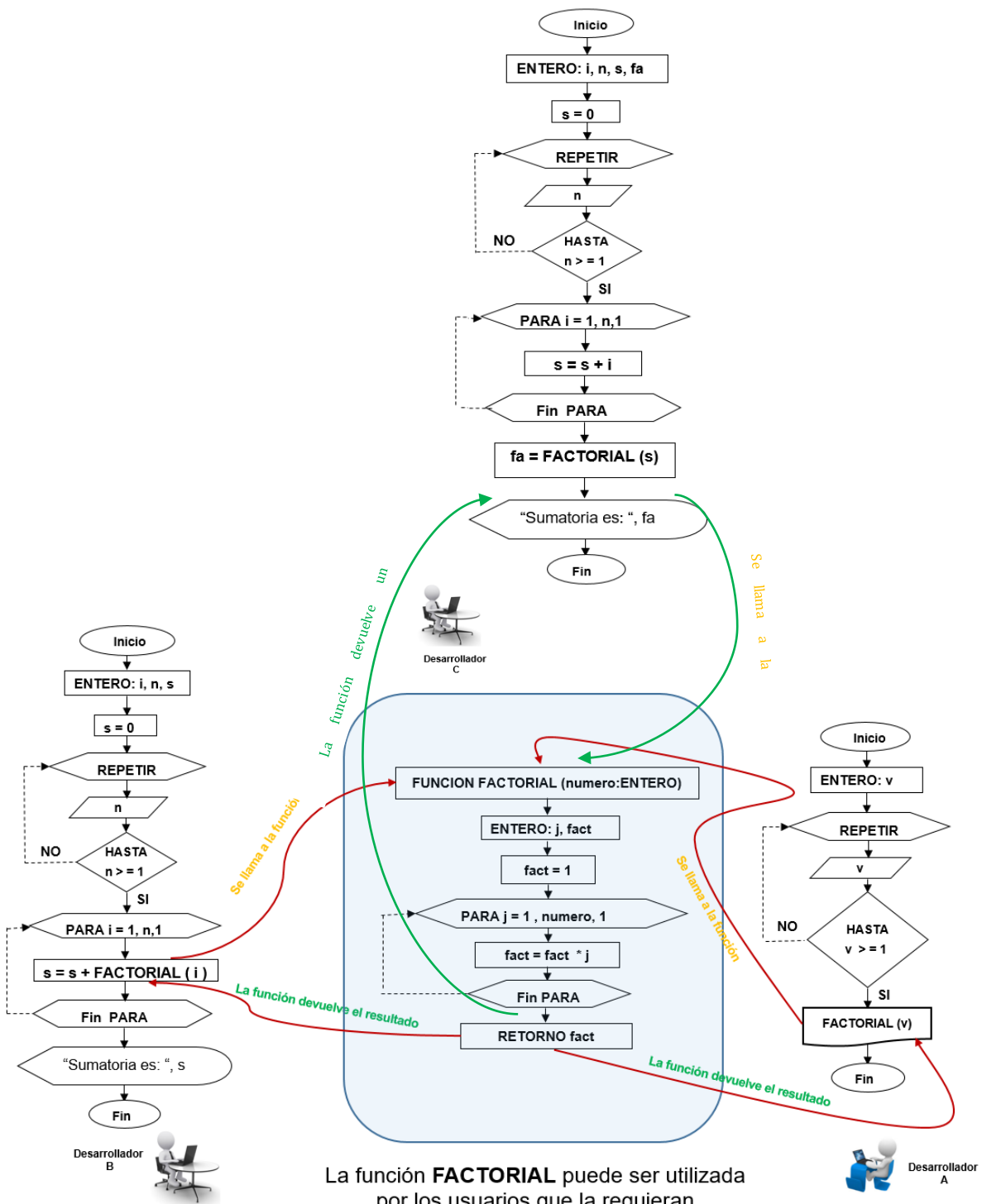
Problema 2: Diagrama que permite crear una función para “centrar” una frase (se valida que no esté vacía y tenga entre 3 y 70 caracteres, ambos inclusive) en una pantalla de 80 columnas.



En la función se utiliza un ciclo “PARA” en donde se concatenan espacios que determinan el margen de espaciado. A este margen se concatena también la frase ingresada. Este resultado es el que se devuelve al módulo principal donde se muestra la frase ya “centrada”.

Problema 3: En el siguiente diagrama se crea una función para “calcular el factorial (!)” de un valor. En este problema se ilustra, cómo al tener una función ya creada, se puede tener acceso a la misma desde diversas soluciones, en este caso representadas por diagramas. El acceso se realiza a través de tres problemas diferentes:

- El desarrollador A: Calcula factorial de un valor ingresado $v!$
- El desarrollador B: Calcula la suma de la siguiente serie $S = 1! + 2! + 3! + \dots + N!$
- El desarrollador C: Calcula la suma de la siguiente serie $S = (1 + 2 + 3 + \dots + N)!$



La función **FACTORIAL** puede ser utilizada por los usuarios que la requieran

Problemas Propuestos con Funciones Creadas por el Desarrollador

Problema 1: Diseñe un diagrama que lea un número entero y utilice una función para calcular su cuadrado. Dicho resultado se debe mostrar en el módulo principal.

Problema 2: Elabore un diagrama que permita ingresar dos números enteros y determine cuál de ellos es el mayor. La comparación deberá realizarse mediante una función que reciba ambos valores y retorne el número mayor al módulo principal.

Problema 3: Diagrama que permita ingresar dos números y calcule los resultados de las operaciones básicas. Para ello, el programa debe utilizar funciones independientes llamadas **Sumar**, **Restar**, **Multiplicar** y **Dividir**, cada una encargada de realizar la operación correspondiente. Considere que los resultados de estas operaciones se muestran en el módulo principal.

Problema 4: Diagrama que permita crear una FUNCIÓN con argumento para “transformar un tiempo en segundo” y que dicho valor sea retornado al programa principal. Para ello, debe ingresar el tiempo en: horas (validar 0 – 23), minutos (validar 0-59) y segundo (validar 0-59) en el módulo principal.

Problema 5: Diagrama que diseñe una función que reciba el peso (en kg) y la altura (en metros) de una persona. La función debe calcular el índice de masa corporal ($\text{peso} / \text{altura}^2$). En el módulo principal debe evaluar el resultado e indicar la categoría: "Bajo peso", "Normal", "Sobrepeso" u "Obesidad".

Problema 6: Diagrama que permita ingresar desde el programa principal una frase y un carácter de tipo texto. Validar desde este ambiente que la variable donde se ingresa la frase no este vacía y que el carácter ingresado solo sea alfabético (A...Z... a...z). Se solicita crear una función que devuelva al programa módulo cuántas palabras de la frase comienzan con la letra ingresada.

Problema 7: Diagrama que permita crear dos funciones. Una para transformar kilómetros a metros y la segunda para transformar kilómetros a decímetros.

Considere ingresar los kilómetros en el módulo principal, considerando que dicho valor sea positivo.

Problema 8: Diagrama que permita crear una función, cuya tarea será verificar si un valor ingresado desde el programa principal es POSITIVO y de tipo INTEGER. Si no lo es no deberá salir de la función. En el programa módulo se deberá mostrar el mensaje “Número Correcto”.

Problema 9: Diseñe un diagrama que permita ingresar tres calificaciones de un estudiante. El programa debe calcular el promedio mediante una función llamada **CalcularPromedio** y luego determinar, utilizando otra función llamada **DeterminarEstado**, si el estudiante **aprueba o reprueba**, considerando que aprueba con promedio mayor o igual a 7.

Problema 10: Diagrama que permita crear una función, cuya tarea será contar el número de palabras que contengan una frase previamente ingresada. Adicionalmente use otra función para validar que la variable donde se ingrese la frase no esté vacía.

Problema 11: Diagrama que permita ingresar desde el módulo principal una frase. Se solicita lo siguiente:

- Crear una función que mediante programación “elimine” los posibles espacios a la derecha de la frase ingresada.
- En el programa principal debe mostrar la nueva frase.

Considere validar en el programa principal que la variable donde se ingresa la frase debe tener mínimo dos palabras.

Problema 12: Diagrama que permita crear una función que reciba el monto total de una compra y el nivel de membresía del cliente (1, 2 o 3). La función debe aplicar descuentos escalonados: Nivel 1 (5 %), Nivel 2 (10 %), Nivel 3 (15 %). Si el nivel ingresado no es 1, 2 o 3, la función debe asumir Nivel 1. La función debe retornar el monto final a pagar después del descuento. El módulo principal mostrará el ahorro total y el precio final.

Problema 13: Diagrama que tenga una función en la que se determine el costo de envío de un paquete. Recibe el peso en kg y el código de la zona (1, 2 o 3). El costo base es \$5. Por cada kg, se suma \$2. Si la zona es 2, se multiplica el total por 1.5. Si la zona es 3, se multiplica por 2. Si el peso es mayor a 50kg, se suma un recargo fijo de \$20 al final. La función retorna el costo total al módulo principal para mostrarlo.

Problema 14: Diagrama que crea una función que reciba la edad de una persona. La función debe retornar una cadena de texto indicando la etapa de vida: "Niño" (0-12), "Adolescente" (13-17), "Adulto" (18-59), "Adulto Mayor" (60+). Si la edad es negativa o mayor a 120, la función debe retornar la cadena "Edad inválida". El módulo principal recibe la cadena y la muestra en forma de una oración completa.

Problema 15: Diagrama que permita en el ingreso de un valor positivo (validar en el programa principal). Se solicita crear una función para determinar mediante un mensaje de retorno al programa principal si el valor ingresado es o no un **NÚMERO PERFECTO**. Un número es PERFECTO, cuando la suma de sus divisores exactos (excepto él mismo) coincide con el número.

Problema 16: Diagrama que permita ingresar desde el programa principal cualquier frase. Se solicita crear una función que "cuente" el número de veces que en dicha frase aparece la subcadena "La" o "la" en forma individual o entre palabra. La función debe devolver al programa principal el respectivo contador para su visualización. Ejemplos:

Las calles están engalanadas → 2 veces

Tienes las palabras escritas en la pizarra → 3 veces

Validar en el módulo principal que la variable donde se ingresa la frase debe tener mínimo dos (2) palabras.

Problema 17: Diagrama que permita el ingreso de una frase cualquiera. Se solicita crear una función para visualizar dentro de ella cada palabra de la frase una debajo

de otra, seguida cada palabra del número de caracteres que la componen. Al módulo principal retorne y muestre la suma de los caracteres de cada palabra.

Ejemplo: Hoy es jueves

Hoy	3 Caracteres
es	2 Caracteres
jueves	6 Caracteres

La suma de los caracteres de cada palabra es: **11**

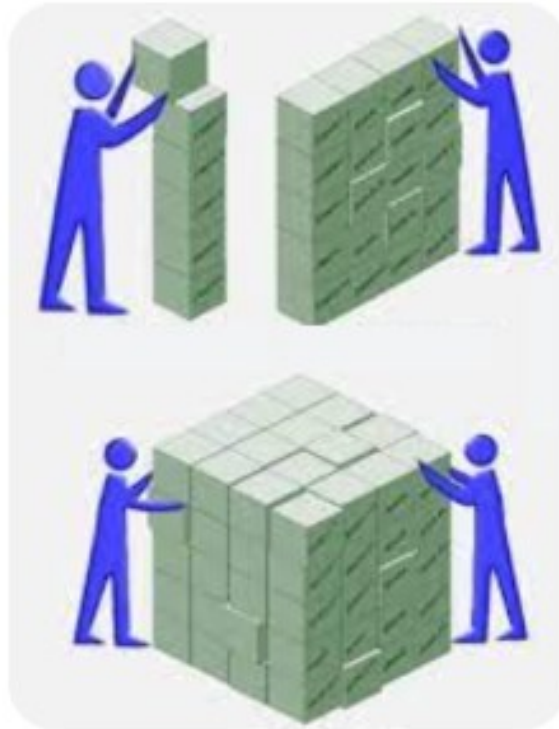
Validar en el programa principal que la variable donde se ingresa la frase debe tener mínimo dos (2) palabras.

Problema 18: Diagrama que permita el ingreso de un número entero y positivo entre 10 y 99 (validar en el módulo principal). Crear una función que calcule y retorne para mostrar en el programa principal el factorial de cada dígito del número ingresado.

Problema 19: Diagrama que diseñe una función que reciba tres valores numéricos representando las longitudes de los lados de un triángulo. La función debe determinar primero si es posible formar un triángulo con esas medidas (la suma de dos lados debe ser siempre mayor al tercero). Si no es posible, retorna 0. Si es posible, la función debe retornar un código entero: 1 si es equilátero (tres lados iguales), 2 si es isósceles (dos lados iguales) y 3 si es escaleno (todos diferentes). El módulo principal recibirá este código e mostrará el nombre del tipo de triángulo correspondiente.

Problema 20: Diagrama que diseñe una función que reciba un número entero y positivo. La función debe descomponer ese número y sumar todos sus dígitos individuales (ej: si recibe 2026, debe retornar $2+0+2+6 = 10$). El valor retornado debe ser mostrado por el módulo principal junto con el número original ingresado.

Capítulo V. Vectores y Matrices: Introducción a los Arreglos



Objetivo General del Capítulo

Desarrollar la capacidad de organizar y manipular conjuntos de datos mediante el uso de arreglos unidimensionales y bidimensionales (vectores y matrices), comprendiendo su estructura de almacenamiento, acceso mediante índices y su aplicación en la resolución de problemas mediante algoritmos.

5.1 Introducción a los Arreglos

En programación, a medida que los problemas se vuelven más grandes y complejos, surge la necesidad de trabajar con mayores cantidades de datos. En estos casos, ya no es suficiente utilizar una sola variable para almacenar información, sino que es necesario manejar una colección o conjunto de valores que pertenecientes a un mismo tipo de dato permitan su procesamiento.

Por ejemplo, si queremos almacenar los nombres de 5 estudiantes y sus respectivas calificaciones de un curso de programación, sería poco práctico realizar lo siguiente:

nombre1 = "Mónica"	nota1 = 10
nombre2 = "Ian"	nota2 = 8
nombre3 = "Hazel"	nota3 = 9
nombre4 = "Zaid"	nota4 = 10
nombre5 = "Karen"	nota5 = 7

Si a lo anterior, se solicita agregar nuevos estudiantes y realizar operaciones matemáticas con las calificaciones, el proceso se vuelve aún más complicado por la cantidad de variables que se utilizarían en el procesamiento de toda esta información. Para solucionar estos tipos de problemas, los lenguajes de programación ofrecen una forma de estructura de almacenamiento de datos llamada **arreglo**.

Un arreglo (o **array**) es una estructura de datos que **almacena una colección de valores de un mismo tipo de dato en una variable**. Es decir, no es que se alojen los datos en una variable al mismo tiempo, sino que se tienen una serie de contenedores que conforman el arreglo y en cada uno de esos contenedores se guardan datos distintos, pero bajo un mismo nombre de variable. Cada valor dentro del arreglo ocupa una posición específica que puede identificarse mediante un **índice**.

En la figura 25, se puede observar cómo se almacena en memoria la información anterior de manera tradicional (con 10 variables) y el almacenamiento utilizando arreglos (solo 2 variables).

Figura 25

Ejemplo con Almacenamiento Ordinario y Almacenamiento Dimensionado

Almacenamiento Tradicional

Una variable solo puede tener en su interior un solo valor a la vez.

nombre1	Mónica	10	nota1
nombre2	Ian	8	nota2
nombre3	Haziel	9	nota3
nombre4	Zaid	10	nota4
nombre5	Karen	7	nota5

En memoria hay 10 variables
Normales u Ordinarias

Almacenamiento con Arreglos

Una variable puede tener varios valores en su interior al mismo tiempo.

		nombre		nota		
índice	0	Mónica		10	0	índice
	1	Ian		8	1	
	2	Haziel		9	2	
	3	Zaid		10	3	
	4	Karen		7	4	

En memoria hay 2 variables
DIMENSIONADAS

Nota. Fuente: Elaboración propia

En el **almacenamiento tradicional**, cada dato se guarda en **una variable diferente**. Por ejemplo, para guardar cinco nombres y cinco notas se necesitan diez variables independientes (nombre1, nombre2, nota1, nota2, etc.). Esto provoca que se utilicen una mayor cantidad de variables y que el manejo de los datos sea menos organizado.

En cambio, en el **almacenamiento con arreglos**, una sola variable puede contener **varios valores del mismo tipo organizados mediante índices**. En el ejemplo, los nombres se almacenan en un arreglo llamado **nombre** y las notas en otro arreglo llamado **nota**, donde cada posición se identifica con un índice que comienza en 0. De esta manera, los mismos datos se almacenan utilizando **solo dos variables dimensionadas**, lo que permite una organización más eficiente y facilita el acceso y manejo de la información.

En los arreglos, el índice representa la posición de cada elemento dentro del arreglo. En muchos lenguajes de programación, la numeración de los índices comienza en **0**. Por ejemplo: nombre (3) = Haziel, nombre (0) = Mónica y nota (4) = 7.



Los arreglos se emplean para almacenar múltiples valores en una sola variable, frente a las variables normales u ordinarias que sólo pueden almacenar un valor por variable. Esto facilita organizar grandes cantidades de información de manera eficiente.

5.2 Tipos de arreglos

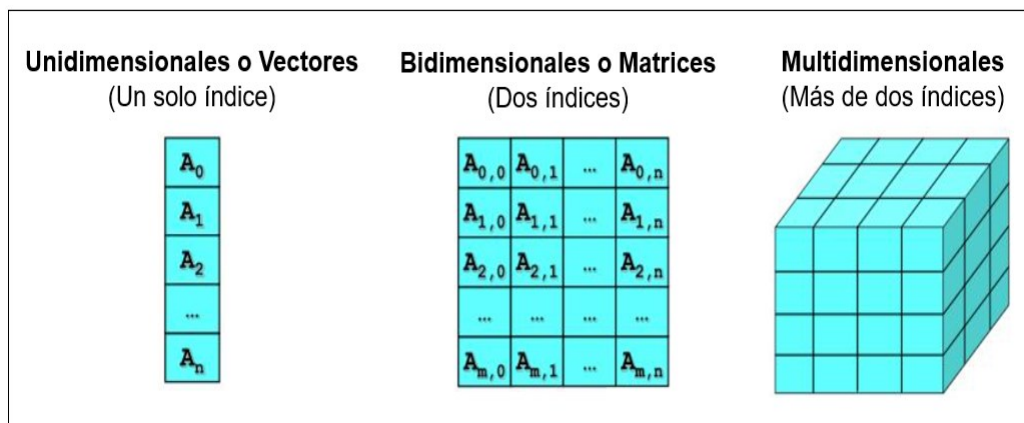
Dependiendo de la forma en que los datos se organizan, los arreglos se clasifican de acuerdo al número de dimensiones que tienen:

- Arreglos Unidimensionales o Vectores (un sólo índice)
- Arreglos Bidimensionales o Matrices (dos índices)
- Arreglos Multidimensionales (más de dos índices)

La dimensión se refiere al número de índices (figura 26) necesarios para acceder a un elemento dentro del arreglo. Para los propósitos de aprendizaje de este libro, se estudiarán los dos primeros tipos de arreglos.

Figura 26

Tipos de Arreglos según su Dimensión



Nota. Fuente: Elaboración propia

5.2.1 Arreglos Unidimensionales o Vectores

Los **Arreglos Unidimensionales** también llamados “**Vectores**”, es el tipo más simple de arreglo y consiste en una colección lineal de datos del mismo tipo y que se pueden referenciar con un mismo nombre. Se caracterizan porque:

- Son estructuras de datos estáticas, ya que hay que declarar su tamaño o límite antes de utilizarlos. Es decir, debe determinarse cuál será el número máximo de elementos que podrán formar parte del vector.
- Contienen una sola dimensión, decir, utilizan **un solo índice** que indica la posición de un dato dentro del arreglo.

- Permite almacenar en un mismo nombre de variable varios valores al mismo tiempo.
- Los datos en el vector son homogéneos, es decir, tienen el mismo tipo de datos.
- El acceso a sus datos se realiza indicando el nombre del vector y su índice, este último dependiendo del lenguaje de programación va entre “()” o entre “[]” o entre “{ }”. Por lo tanto, es importante revisar la documentación del lenguaje a utilizar.

5.2.1.1 Creación de Arreglos Unidimensionales o Vectores

Para crear un vector dependerá del entorno de programación a utilizar. A nivel de diagramación, se utilizará la palabra “**ARRAY**” de la siguiente forma:

- a) Definir el tipo de dato y la variable como vector, es decir el nombre del arreglo.

Tipo de Dato **Array**: *Nombre Vector*

Nombre Vector: Hace referencia a la(s) variable(s) que se va(n) a dimensionar. Puede ser numérica o alfanumérica, según la información a guardar en el vector. Si son varios, separar con coma (,). Por ejemplo:

Entero Array: edad **Entero Array:** sueldos **Cadena Array:** nombres

- b) Establecer el número de elementos que va(n) tener la(s) variable(s) dimensionada(s).

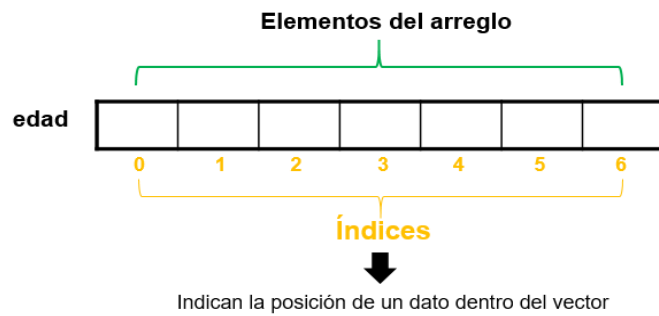
Nombre Vector [*Tamaño del vector*]

Tamaño del Vector: Es un valor numérico que se puede representar por una constante, el valor de una variable o el resultado de una expresión aritmética. A través de este valor, se indica la dimensión o el número de elementos que va a contener el vector. Por ejemplo: Crear el vector **edad** para **7** elementos.

Entero Array: edad

edad [6]

En forma ilustrativa se crearía el vector de la siguiente forma:



Esto indica que se crea un vector llamado **edad** con 7 elementos. Dado que el vector se ha creado, sus elementos están vacíos. Un aspecto muy importante, es que la cada elemento tiene una posición identificada con un índice que arranca numerado desde cero, así **el primer elemento** se accede a la posición con **índice [0]** y el último elemento tendrá el índice de **tamaño del vector menos 1**. Esto se conoce como **"indexación base-cero (0)"**.

5.2.1.2 ¿Cómo Trabajar con los Elementos de un Vector?

Para ingresar, acceder o modificar un dato en un vector, **siempre** se hace referencia al **nombre del vector y su índice**. Este índice, debe estar encerrado entre "()" o entre "[]" o entre "{ }", lo que depende del lenguaje de programación. Por ejemplo:

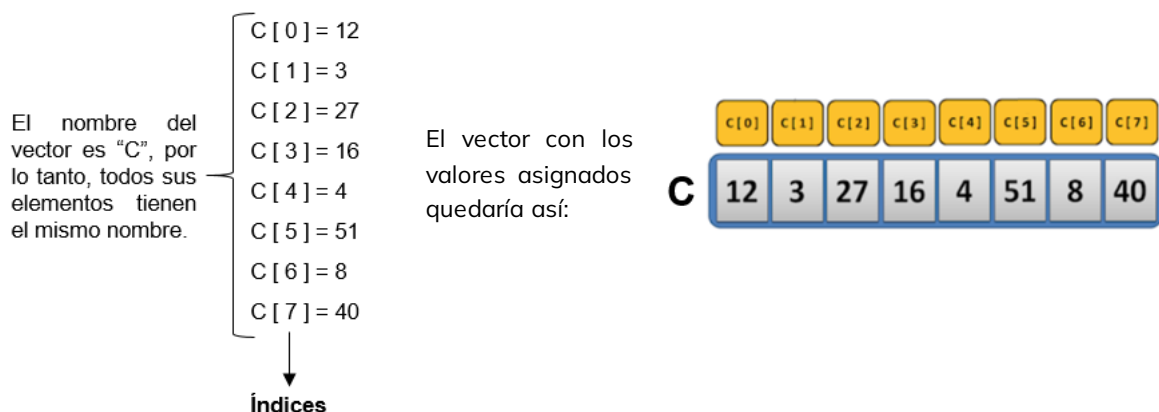
Entero Array: c

`c [8]`

Se crea el vector...



Si queremos **llenar el vector asignando** los siguientes valores, se haría de la siguiente forma:



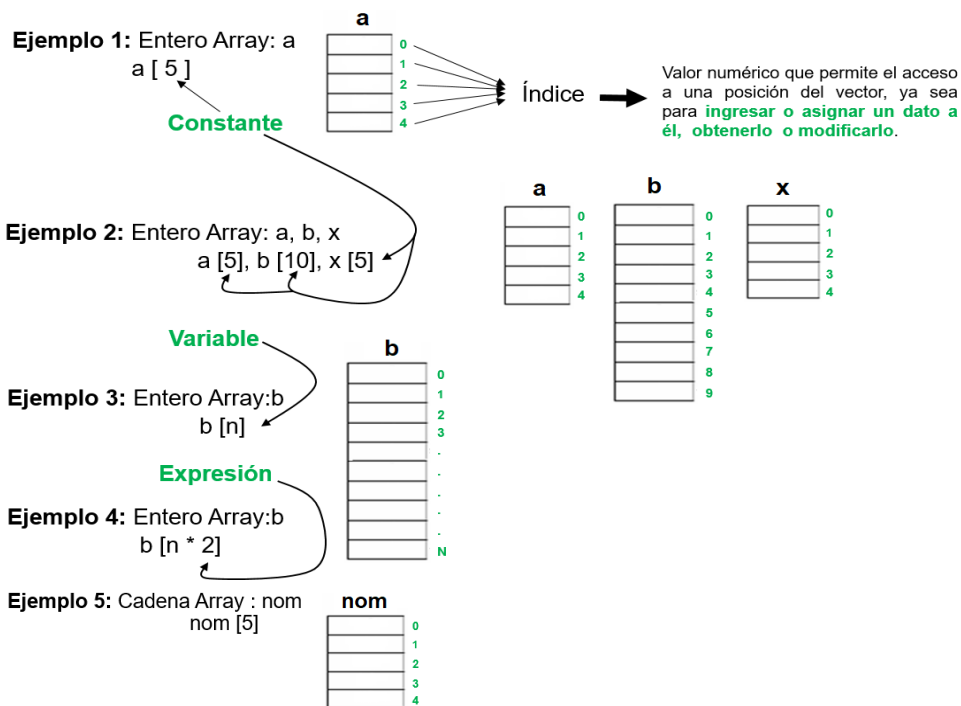
Si queremos tener **Acceso a los elementos** del vector, por ejemplo:

- Si deseo acceder al valor **16**: Será con **C [3]**
- Si deseo acceder al valor **40**: Será con **C [7]**

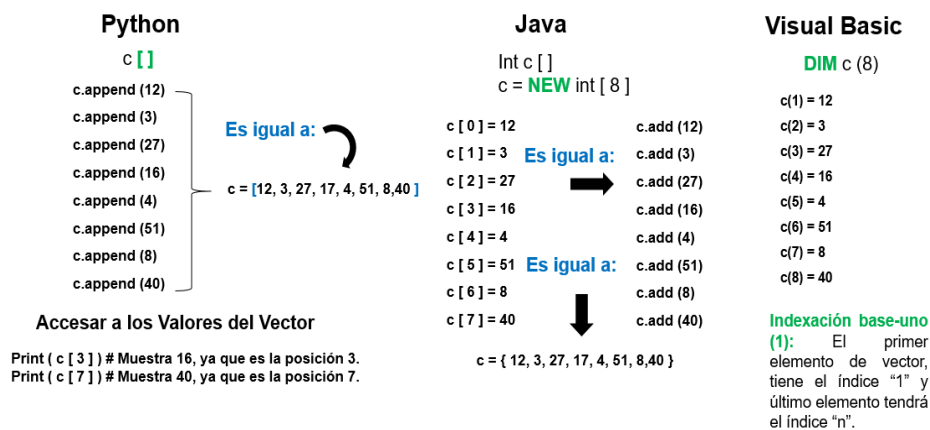
Si queremos **Modificar un elemento** del vector, por ejemplo:

- Cambiar el valor **8**, por **25**: Será con **C [6] = 25**

Ejemplos de Creación de Vectores



A continuación unos ejemplos de creación y asignación de valores en diferentes lenguajes de programación:



Cada Lenguaje de Programación utiliza distintas formas para crear y asignar valores a un vector. Se recomienda hacer uso de la documentación del lenguaje prevista para ello.

5.2.1.3 Consideraciones al Utilizar Vectores

Al utilizar los vectores o arreglos unidimensionales, ya sea en programación o en la elaboración de algoritmos mediante diagramas de flujo, es importante tener en cuenta ciertos aspectos que permiten utilizarlos de manera correcta y eficiente. Estas consideraciones ayudan a evitar errores comunes y facilitan el manejo de los datos almacenados en el vector.

- a) Antes de utilizar un vector, es necesario definir el tamaño del vector, es decir, el número máximo de posiciones disponibles para almacenar datos. Este proceso se conoce como **dimensionamiento**. En este aspecto, el tamaño del vector debe ajustarse a la **cantidad de datos reales** que se almacenan, esto evitar ocupar **memoria innecesariamente**.
- b) Es recomendable asignar **nombres claros y representativos** a los vectores, de manera que reflejen el tipo de información que almacenan. Ejemplos:

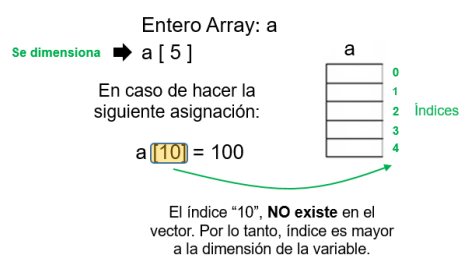
Entero Array: notas	Cadena Array: ciudad	Entero Array: sueldos
notas [20]	ciudad [10]	sueldos [5]

- c) Una variable al ser dimensionada, siempre debe ser tratada como tal. Es decir, **siempre** debe ir seguida de: “()” o “[]” o “{ }” (según el lenguaje de programación) y dentro de ellos el índice que hace referencia a la posición del valor dentro del vector. Ejemplos:

Entero Array: notas	Cadena Array: ciudad
notas [20]	ciudad [10]
notas [0] = 1, notas [5] = 10	ciudad [0] = “Manta”, ciudad [10] = “Quito”

- d) El valor de un índice, **nunca** deberá ser mayor al tamaño o dimensión que tenga el vector. Por lo tanto, es importante no acceder a posiciones **fuera del rango del vector**, ya que esto puede provocar errores en el programa.

Ejemplo:

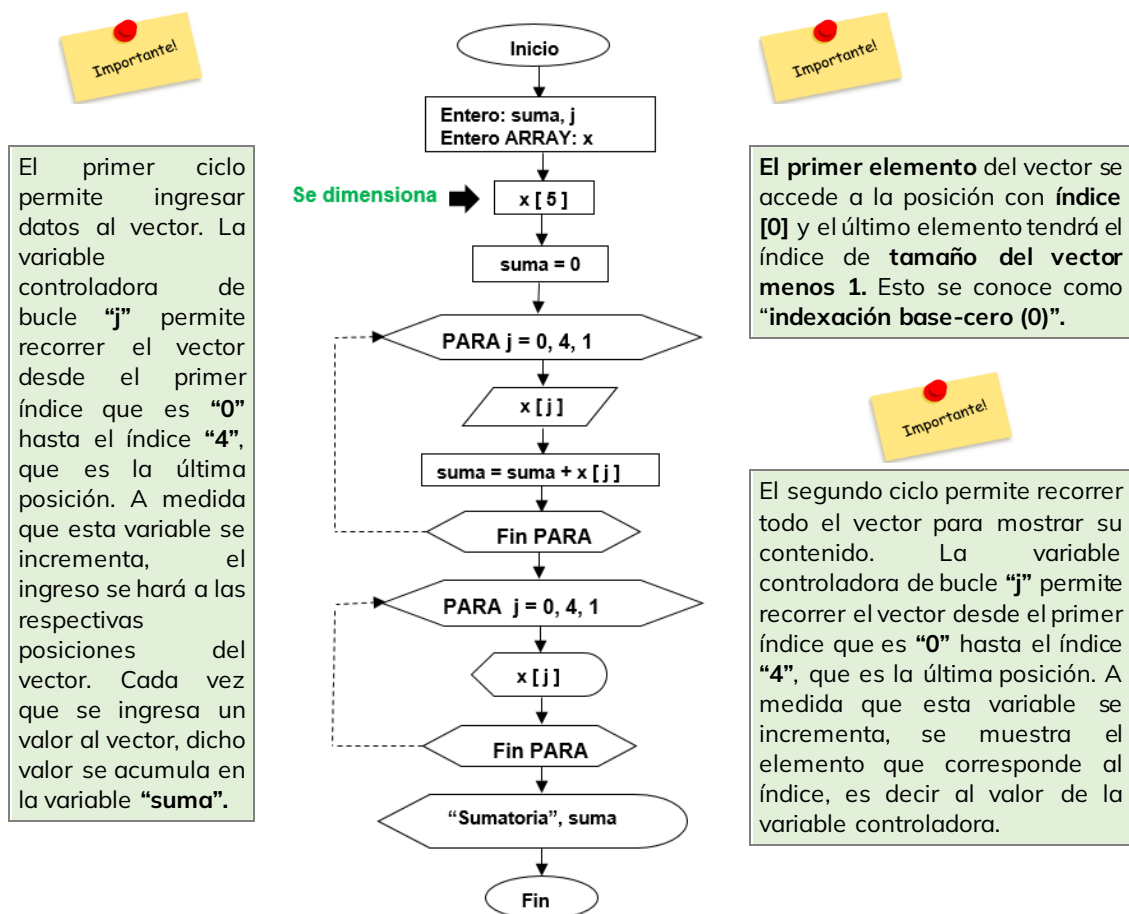


- e) Los elementos o valores contenidos en un vector deben ser del mismo tipo de datos, con que se definió el vector.
- f) Para recorrer en todos los elementos del vector, generalmente se utilizan estructuras repetitivas o ciclos. Las variables controladoras de bucle, serán aquellas que representen a los “índices”, lo que permite recorrer cada posición de forma ordenada del vector.

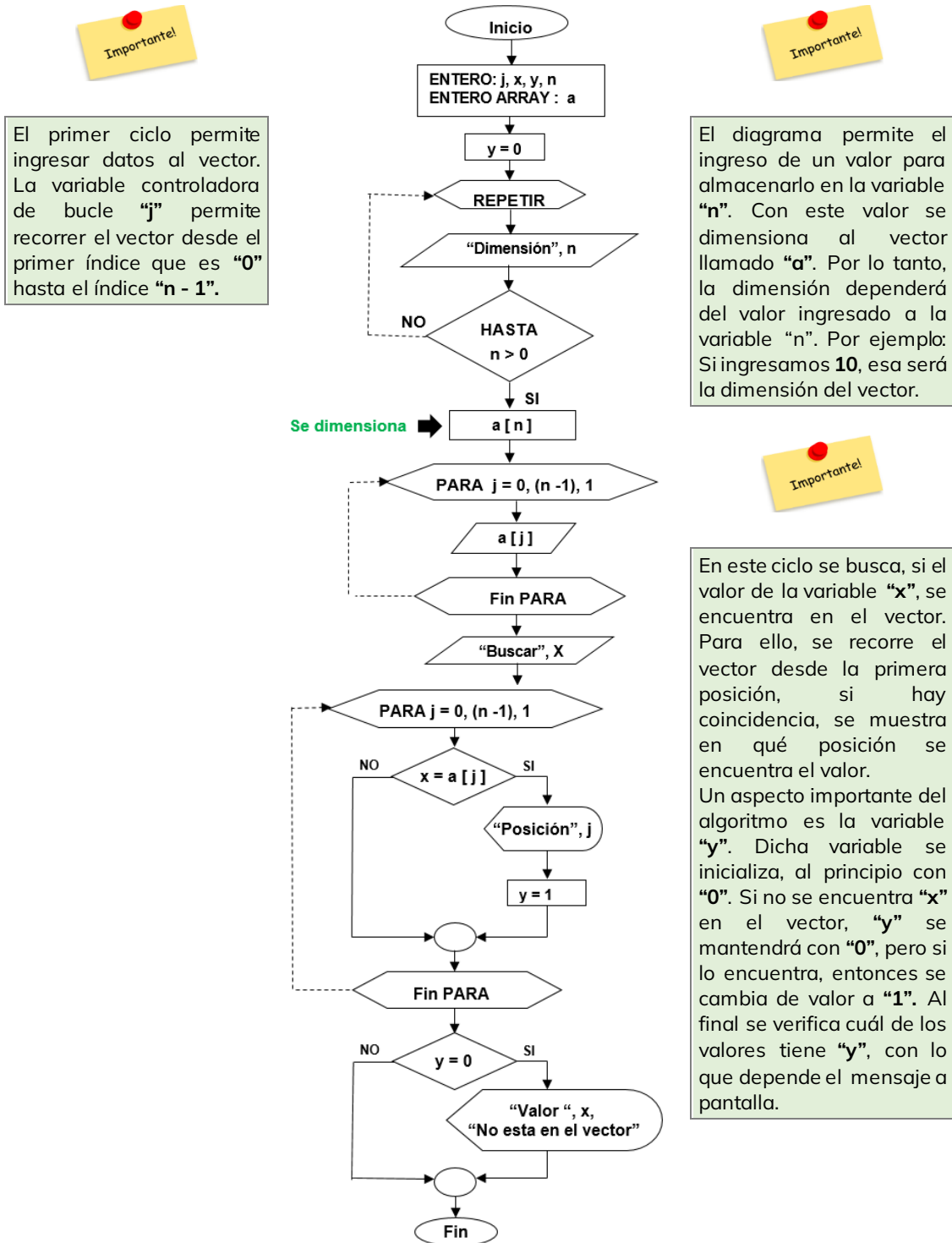
Problemas Resueltos con Uso de Vectores

Problema 1: Crear un diagrama que permita la utilización de un vector con 5 elementos, con este vector se pide lo siguiente:

- a) Ingrese información al vector.
- b) Obtenga y muestre la sumatoria de los valores almacenados en el vector.
- c) Muestre los valores del vector.

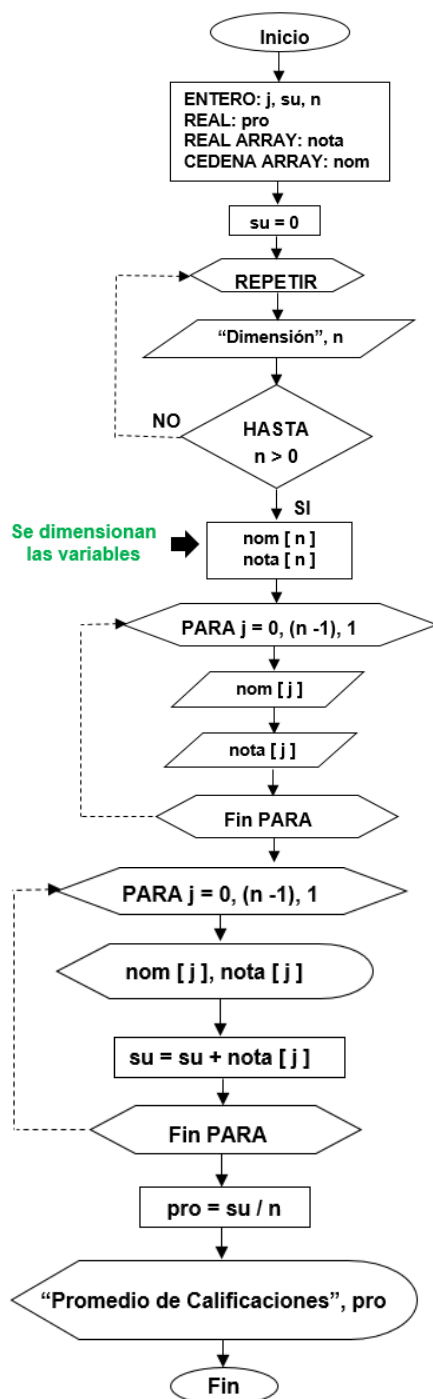


Problema 2: Diagrama que permita ingresar información a un vector llamado "A", el cual contiene N elementos (debe ser positivo). Como proceso se pide ingresar información al vector y verificar si un valor X se encuentra en dicho vector. En caso de encontrarse muestre en qué posición del vector se encuentra.



Problema 3: Diagrama que permita crear dos vector para almacenar los nombres y las calificaciones de N cantidad de estudiantes (los vectores serán dimensionados para N). Como proceso se pide:

- Ingresar información a los dos vectores.
- Calcular y mostrar el promedio aritmético de las calificaciones almacenadas en el vector correspondiente.
- Muestre a cada estudiante con su respectiva calificación.

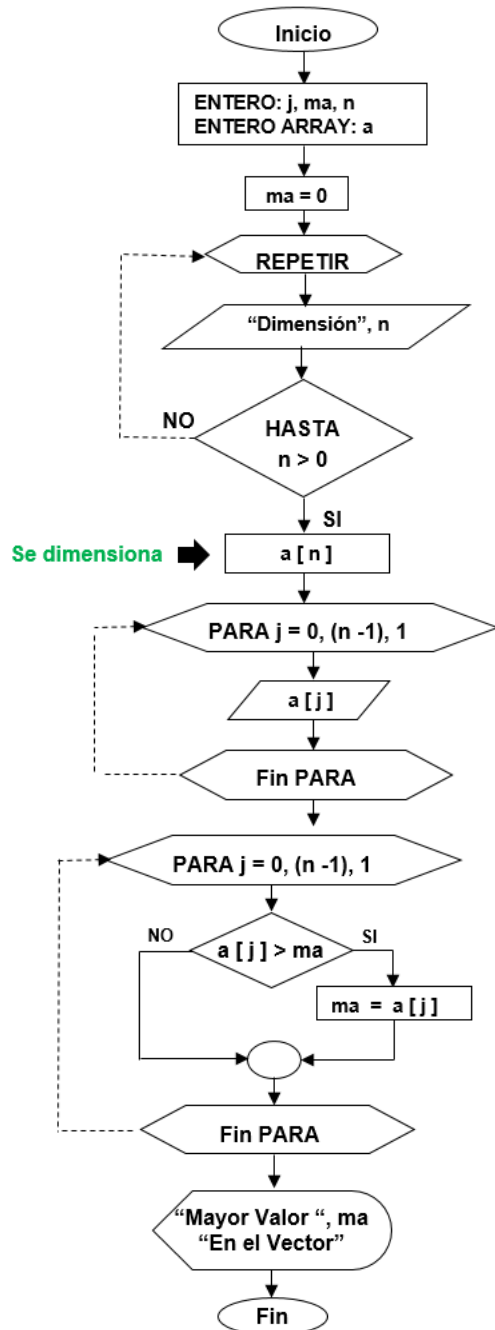


El diagrama permite el ingreso de un valor para almacenarlo en la variable "n". Con este valor se dimensionan los vectores llamados "nom" y "nota". Por lo tanto, la dimensión dependerá del valor ingresado a la variable "n". Por ejemplo: Si ingresamos 10, esa será la dimensión de los vectores.

Este primer ciclo permite ingresar los datos de nombre y calificaciones a los dos vectores. La variable controladora de bucle "j" permite recorrer el vector desde el primer índice que es "0" hasta el índice "n - 1". En cada posición se ingresa la información.

El segundo ciclo permite recorrer todas las posiciones de los dos vectores para mostrar sus contenidos. La variable controladora de bucle "j" permite recorrer los vectores desde el primer índice que es "0" hasta el índice "n - 1", que es la última posición. A medida que esta variable se incrementa, se muestran los elementos de ambos vectores y adicionalmente se acumulan las calificaciones del vector "nota", para al final calcular y mostrar el promedio.

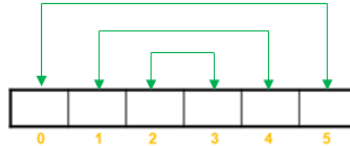
Problema 4: Diagrama que permita ingresar información a un vector llamado “A”, el cual contiene N elementos (debe ser positivo). Como proceso se pide ingresar información al vector y encontrar y mostrar el **mayor valor ingresado en él**.



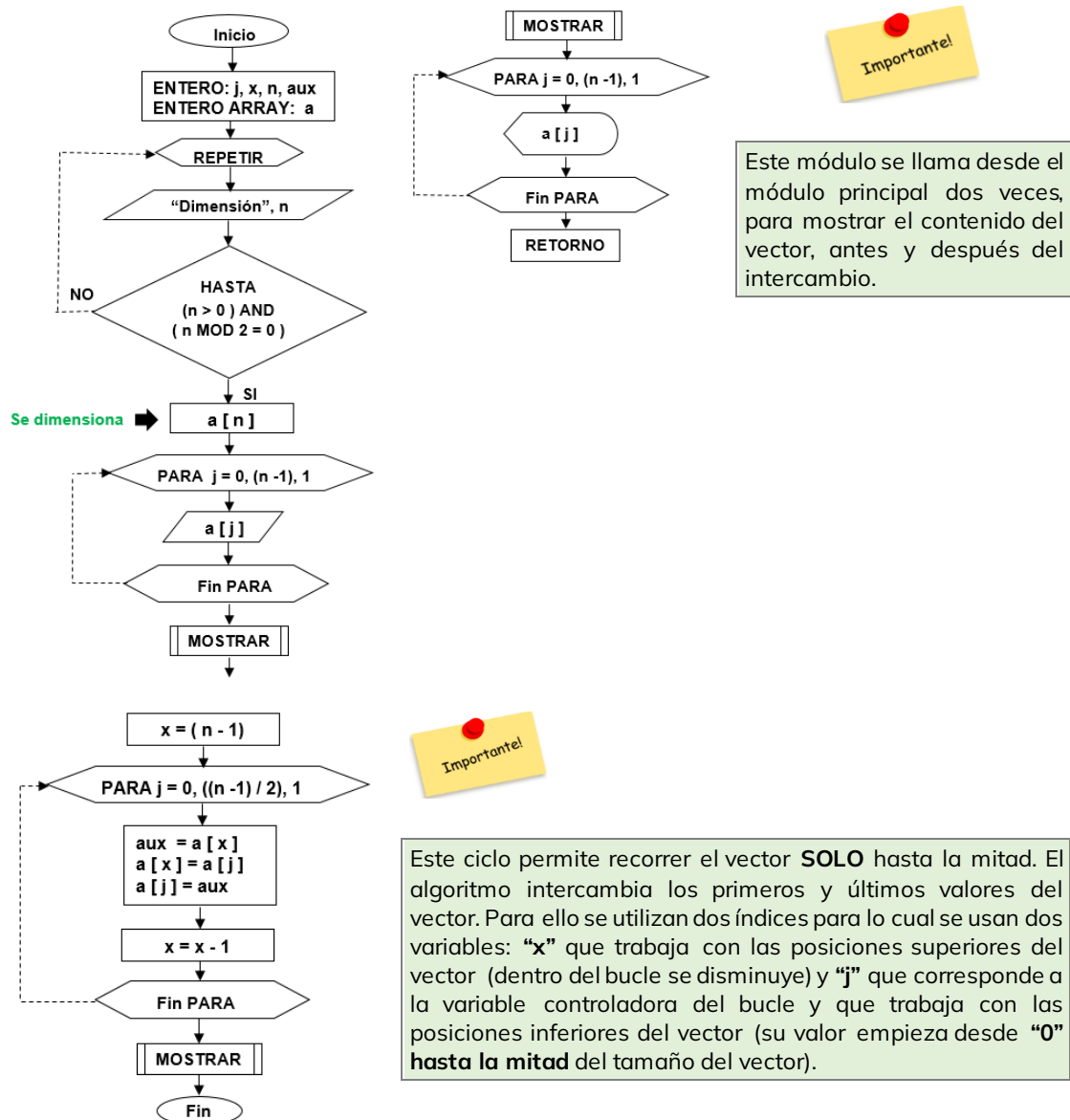
Este ciclo recorre todo el vector. El algoritmo consiste en preguntar si cada valor del vector es mayor al contenido de la variable “ma”. En caso de cumplirse, entonces se toma el valor del vector y se guarda en la variable “ma”. Esta búsqueda se repite hasta buscar en todo el vector.

Problema 5: Diagrama que permita ingresar información a un vector llamado “A”, el cual contiene N elementos (debe ser positivo y valor PAR). Como proceso se pide lo siguiente:

- Ingresar información al vector
- Intercambiar los valores del vector, entre las primeras y últimas posiciones.



- Utilizar un módulo para mostrar el contenido del vector antes y después del intercambio.



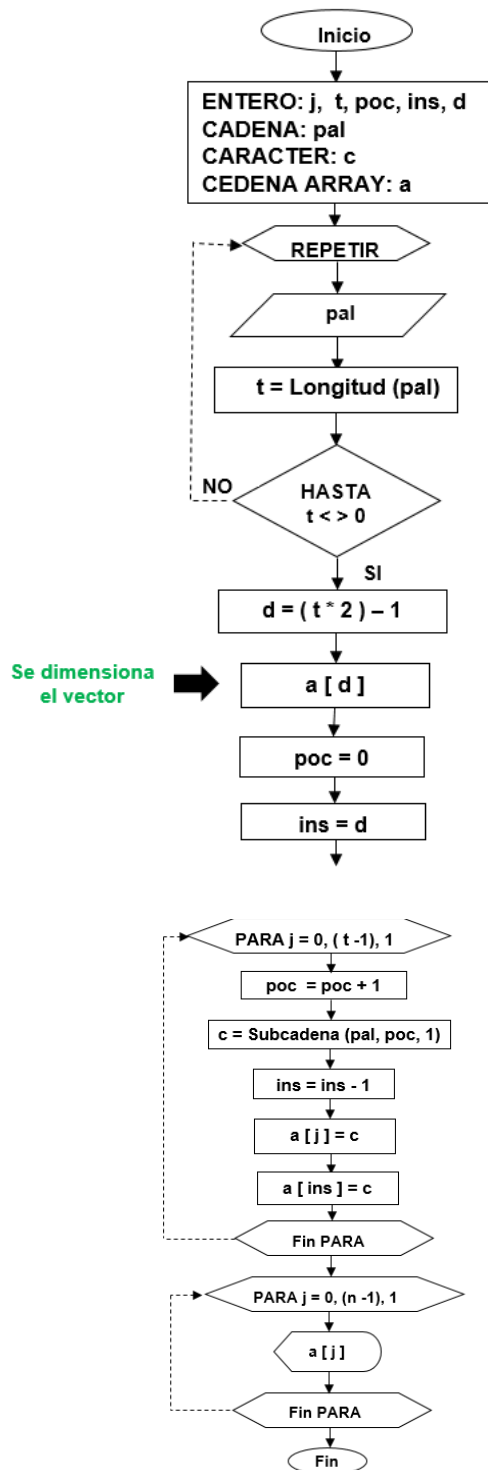
Problema 6: Diagrama que permita crear un vector tipo “cadena” llamado “A”.

Como proceso se pide lo siguiente:

- Ingresar una palabra (validar que la variable de ingreso no este vacía).
- Dimensionar el vector llamado “A” de acuerdo a lo siguiente: Longitud de la “palabra” ingresada multiplicado por 2 y restado en 1.
- Llenar el vector con los caracteres de la “palabra” ingresada de tal manera que se lea por igual de izquierda a derecha y viceversa.

Ejemplo: Si la palabra ingresada es **Sol**. La dimensión del vector será de 5 (Longitud de la “palabra” que es 3, multiplicado por 2 y restado en 1). El vector quedaría:

S	o	l	o	S
0	1	2	3	4



El algoritmo utiliza una serie de variables para asignar cada carácter de la palabra ingresada al vector.

d = Dimensión del vector, según lo solicitado.

poc = Es un contador, que permite obtener la posición de cada carácter de la palabra ingresada.

c = Almacena cada carácter de la palabra.

ins = Contador descendente, **que indicará el índice de las posiciones superiores** del vector, es por eso que inicia con el valor de la variable “d”.
j = Es la variable controladora del bucle y **que indicará el índice de las posiciones inferiores** del vector.

Estas variables son las utilizadas en el bucle, mismo que permite asignar los caracteres al vector de acuerdo al enunciado del problema.

Problema 7: Diagrama que permita **mediante un menú** efectuar el procesamiento de información correspondiente a productos. Las opciones del menú son las siguientes:

- 1: Ingreso de Productos
- 2: Consulta Producto por Código
- 3: Genera Listado de los Productos
- 4: Listado de Productos para Reposición
- 5: Salida

Para las primeras cuatro opciones se usan los siguientes módulos:

Módulo de ingreso: En este módulo se ingresará la siguiente información en vectores, dimensionados para N elementos:

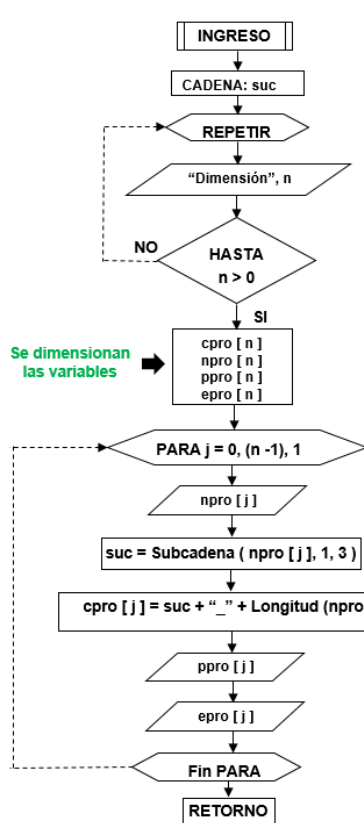
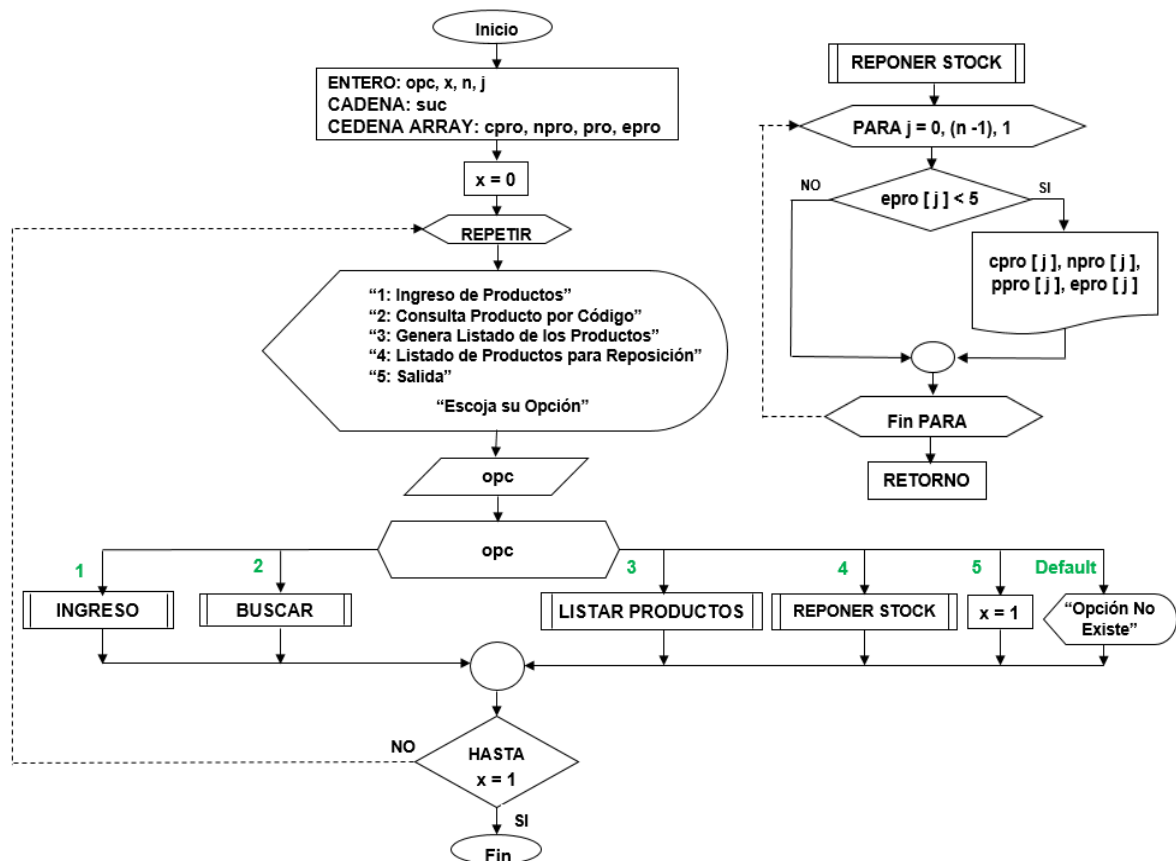
- Código del Producto (variable cpro): No se ingresa, se genera con los 3 primeros caracteres del nombre del producto, más “_” y más el tamaño en caracteres del nombre del producto.
- Nombre del Producto (variable npro)
- Precio del Producto (variable ppro)
- Existencia del Producto (variable epro)

Módulo de Consulta Por Producto: En este módulo se puede consultar un producto específico mediante el ingreso del código respectivo. En caso de no existir, se envía un mensaje de “producto no existe”.

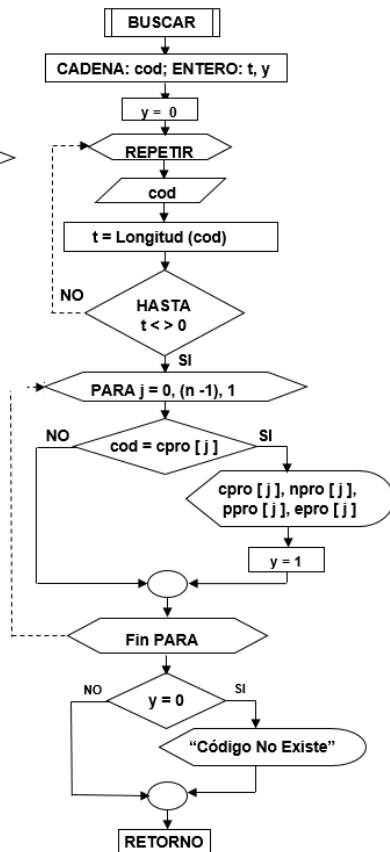
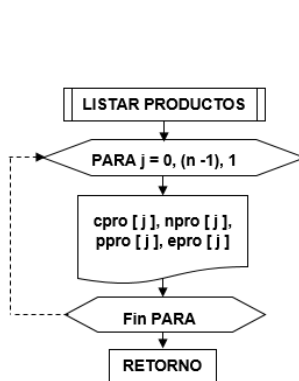
Módulo de Listado de Productos: En este módulo se muestra el contenido de todos los productos almacenados en los vectores.

Módulo de Listado de Productos para Reposición: En este módulo se muestra solo aquellos productos que necesitan reposición de existencia. A manera de ejemplo, se consideran solo los productos cuya existencia sea menor a 5.

El diagrama solo permite finalizar, cuando se escoja la opción 5 (salida), de no ser así, se seguirá mostrando el menú.



Se dimensionan las variables



Problemas Propuestos con Uso de Vectores

Problema 1: Diagrama que permita ingresar 20 números a un vector. Se solicita mostrar primero todos los números guardados en las posiciones pares y posteriormente los de las impares.

Problema 2: Diagrama que permita asignar los N primeros números a un vector. Al final muestre el contenido del vector en forma inversa a su asignación.

Problema 3: Diagrama que permita crear un vector con N elementos (debe ser positivo mayor a 2 y valor PAR). Como proceso se pide calcular y mostrar la suma del primer número introducido con el último, el segundo con el penúltimo, el tercero con el antepenúltimo, y así sucesivamente.

Problema 4: Diagrama que permita ingresar información a un vector llamado "A", el cual contiene N elementos (debe ser positivo). Como proceso se pide ingresar información al vector y encontrar y mostrar el **menor valor ingresado en él**.

Problema 5: Crear un vector con N números (debe ser positivo). Como proceso ingrese información al vector. Calcular y mostrar la media aritmética de los valores que estén en las posiciones impares.

Problema 6: Crear un vector con N números (debe ser positivo). Como proceso ingrese información al vector. Calcular el promedio de sus valores e indicar cuantos elementos del vector son mayores que el promedio y cuantos son menores e iguales.

Problema 7: Diagrama que permita crear tres vectores dimensionados para N elementos (debe ser positivo). Como proceso ingrese datos a dos vectores, calcule la suma de estos valores guardándolos en un tercer vector. Mostrar por pantalla el contenido de los tres vectores.

Problema 8: Crear un vector con N números (debe ser positivo). Como proceso se solicita:

- a) Ingresar información al vector
- b) Ordenar los valores del vector en forma ascendente (**use el método de burbuja**)
- c) Usar un módulo para mostrar el contenido del vector antes y después de su ordenamiento.

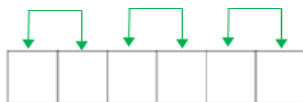
Problema 9: Diseñar un diagrama que calcule y almacene en un arreglo los 20 primeros números pares posteriores a un número dado N (realice la validación respectiva). Posteriormente proceda a mostrar el contenido del vector.

Problema 10: Diseñar un diagrama que permita crear tres vectores para ingresar los nombres, edades y tallas de N atletas (validar que sea positivo). Como proceso se solicita

- a) Ingresar información a los tres vectores de acuerdo a lo solicitado.
- b) Calcular y mostrar el promedio de las edades y de las tallas.
- c) Mostrar cuantos atletas son menores de edad (menores de 18 años) y cuantos atletas son mayores de edad.
- d) Mostrar los nombres de los atletas que son más altos que el promedio.

Problema 11: Diagrama que permita ingresar información a un vector llamado "A", el cual contiene N elementos (debe ser positivo mayor a 2 y valor PAR). Como proceso se pide lo siguiente:

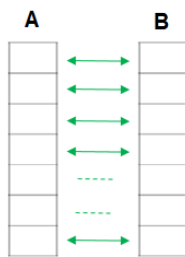
- a) Ingresar información al vector
- b) Intercambiar los valores del vector, de acuerdo al siguiente esquema:



- c) Utilizar un módulo para mostrar el contenido del vector antes y después del intercambio.

Problema 12: Diagrama que permita ingresar información a dos vectores llamados "A" y "B", los cuales serán dimensionados para N elementos (debe ser positivo). Como proceso se pide lo siguiente:

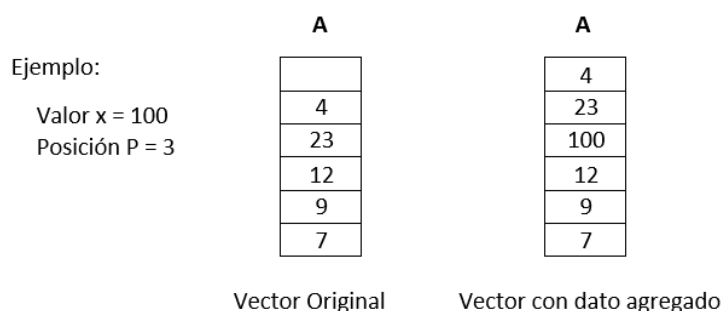
- Ingresar información a los dos vectores
- Intercambiar los valores de los vectores, de acuerdo al siguiente esquema:



- Utilizar un módulo para mostrar el contenido de ambos vectores antes y después del intercambio.

Problema 13: Diagrama que permita crear un vector con N elementos (debe ser positivo), se pide como proceso lo siguiente:

- Ingreso de información al vector desde la posición 1 hasta la última posición (en la posición 0 no se ingresa).
- Insertar un valor X (debe ser ingresado) en una posición P (debe ser ingresado) en el vector, de tal manera que el resto de datos del vector se deberán desplazar hacia las posiciones inferiores.
- Muestre el contenido del vector.



Problema 14: Se solicita el ingreso de un valor cuyo rango debe estar entre 1 y 99, ambos inclusive, como proceso se pide transformar el valor ingresado en letras (desglosar cantidad y aplicar concatenación con los valores de los tres vectores). Para ello, se hará el uso de los siguientes tres vectores de tipo cadena, cuya información será asignada así:

A	B	C
uno	once	diez
dos	doce	veinte
tres	trece	treinta
cuatro	catorce	cuarenta
cinco	quince	cincuenta
seis		sesenta
siete	diez y seis	setenta
ocho	diez y siete	ochenta
nueve	diez y ocho	noventa

Ejemplos: 14 **Catorce** 78 **Setenta y Ocho** 50 **Cincuenta**

Problema 15: Realizar una agenda para guardar la información de N estudiantes (debe ser positivo) de los cuales se toman los siguientes datos como vectores:

- Nombre
- Apellido
- Cédula
- Ciudad de Origen
- Carrera

Se pide que se disponga de un menú de opciones que muestre por pantalla:

- 1: Ingreso de datos a los vectores
- 2: Consulta por Apellido
- 3: Consulta por Ciudad
- 4: Consulta por Cédula
- 5: Consulta por Carrera
- 6: Salida

Para las primeras cinco opciones utilice módulos, tomando como referencia el problema resuelto No. 7 de vectores.

El diagrama solo permite finalizar, cuando se escoja la opción 5 (salida), de no ser así, se seguirá mostrando el menú.

5.2.2 Arreglos Bidimensionales (Matrices)

Los arreglos bidimensionales, conocidos como matrices, son estructuras de datos que permiten almacenar un conjunto de datos del mismo tipo organizados en filas y columnas (figura 27). A diferencia de los vectores que utilizan un solo índice para acceder a sus elementos, las matrices requieren **dos índices**: uno para identificar la **fila** y otro para identificar la **columna** donde se encuentra el dato.

Figura 27

Representación de un Arreglo Bidimensional (matriz)

	Columna 0	Columna 1	Columna 2	Columna 3
Fila 0				
Fila 1				
Fila 2				

Nota. Fuente: Elaboración propia

5.2.2.1 Creación de Arreglos Bidimensionales o Matrices

Para crear un arreglo bidimensional dependerá del entorno de programación a utilizar. A nivel de diagramación, se utilizará la palabra “**ARRAY**” de la siguiente forma:

- Definir el tipo de dato y la variable como matriz, es decir el nombre del arreglo.

Tipo de Dato **Array**: *Nombre Matriz*

Nombre Matriz: Hace referencia a la(s) variable(s) que se va(n) a dimensionar. Puede ser numérica o alfanumérica, según la información a guardar en la matriz. Si son varias, separar con coma (,). Por ejemplos:

Entero: Array: mat, nota **Cadena Array:** nombres

- b) Establecer el número de **Filas** y **Columnas** que va(n) tener la(s) variable(s) dimensionada(s).

Nombre Matriz [**Número filas**] [**Número columnas**]

Número Filas y Columnas: Son valores numéricos que se puede representar por una constante, el valor de una variable o el resultado de una expresión aritmética. A través de estos valores, se indica la dimensión o el número de elementos que va a contener la matriz. Por ejemplo: Crear una matriz **mat** con 3 filas y 4 columnas.

Entero Array: mat

mat [3] [4]

En forma ilustrativa se crearía la matriz de la siguiente forma:

	Columna 0	Columna 1	Columna 2	Columna 3
Fila 0	Mat (0,0)	Mat (0,1)	Mat (0,2)	Mat (0,3)
Fila 1	Mat (1,0)	Mat (1,1)	Mat (1,2)	Mat (1,3)
Fila 2	Mat (2,0)	Mat (2,1)	Mat (2,2)	Mat (2,3)

Mat

Esto indica que se crea la matriz para contener 12 elementos. Cada elemento se identifica con el nombre de la matriz y su posición con la fila y la columna. Un aspecto muy importante, es que la primera fila y la primera columna empiezan con “0”. Por ejemplo, para guardar el valor “50” en la matriz “mat” en la posición: fila 1 y columna 2, se lo haría de la siguiente forma:

	Columna 0	Columna 1	Columna 2	Columna 3
Fila 0				
Fila 1			50	
Fila 2				

mat [1] [2] = 50

5.2.2.2 Consideraciones al Utilizar Matrices

Las matrices son ampliamente utilizadas en programación para manejar conjuntos de datos relacionados, en forma de registros de información o tablas numéricas. De ahí, que es importante utilizarlas de forma eficiente en un algoritmo representado por un diagrama o en programa, especialmente cuando se requiere manejar grandes cantidades de información estructurada. Considere los siguientes aspectos a tenerse en cuenta:

- Al ser estructuras de datos estáticas al igual que un vector, para su creación debe tener un **tamaño previamente definido del número de filas y columnas** que va a tener la matriz.
- Usan sus índices para acceder a los datos de la matriz. El primero representa la **Fila** y el segundo la **Columna**.
- Se utilizan paréntesis “()” o corchetes “[]” según el entorno de desarrollo a utilizar, para indicar la fila y la columna. Por ejemplo, para hacer referencia al elemento de la fila1 y columna de un arreglo bidimensional llamado “matriz”, podría ser: matriz (1, 2) o matriz [1] [2].
- Los datos en una matriz, se almacenan bajo un mismo nombre de variable, por lo tanto, tienen el mismo tipo de datos.
- Al trabajar con una matriz, siempre se hace referencia a su nombre y a la posición (establecida por dos índices: la fila y la columna) de los datos a procesar.
- El **acceso a sus datos**, puede hacerse en **forma directa**, indicando solo sus índices lo que permite obtener o modificar un valor sin necesidad de recorrer toda la matriz.
- Para trabajar con todos los elementos de una matriz normalmente se utilizan **dos ciclos anidados** (generalmente **ciclos for**), uno para recorrer las **filas (ciclo externo)** y otro para recorrer las **columnas (ciclo interno)**.

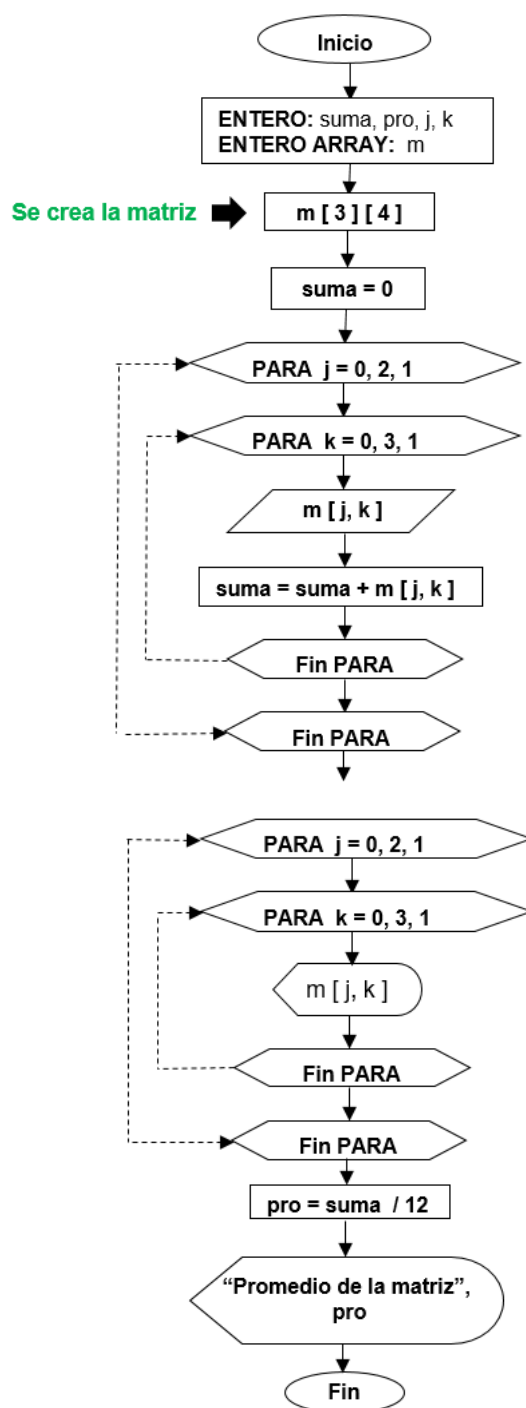


- Los índices de las matrices tanto de las filas como de las columnas empiezan en 0 y terminan en tamaño fila-1 y tamaño columna-1 respectivamente.
- Cada Lenguaje de Programación utiliza distintas formas para crear matrices. Se recomienda hacer uso de la documentación del lenguaje prevista para ello.

Problemas Resueltos con Matrices

Problema 1: Crear un diagrama que permita la utilización de una matriz llamada "M" de tres filas y cuatro columnas. Se solicita lo siguiente:

- Ingresar valores a la matriz.
- Calcular y mostrar el promedio aritmético de valores en la matriz.
- Mostrar los valores de la matriz.



Importante!

Estos ciclos anidados, permiten el ingreso de valores a la matriz. El ciclo externo "j" hará referencia a las filas y el ciclo interno "k" a las columnas. Recordar que los ciclos anidados se cierran de adentro hacia fuera, por lo tanto, la primera vez "j" empieza con "0", de igual forma "k" va desde "0" hasta 3, al cumplirse, salta al ciclo externo de "j" a 1, y "k" vuelve a empezar de "0" a "3", y así sucesivamente hasta llenar toda la matriz.

Para calcular el promedio de la matriz, dentro de estos ciclos se acumula cada valor ingresado.

Importante!

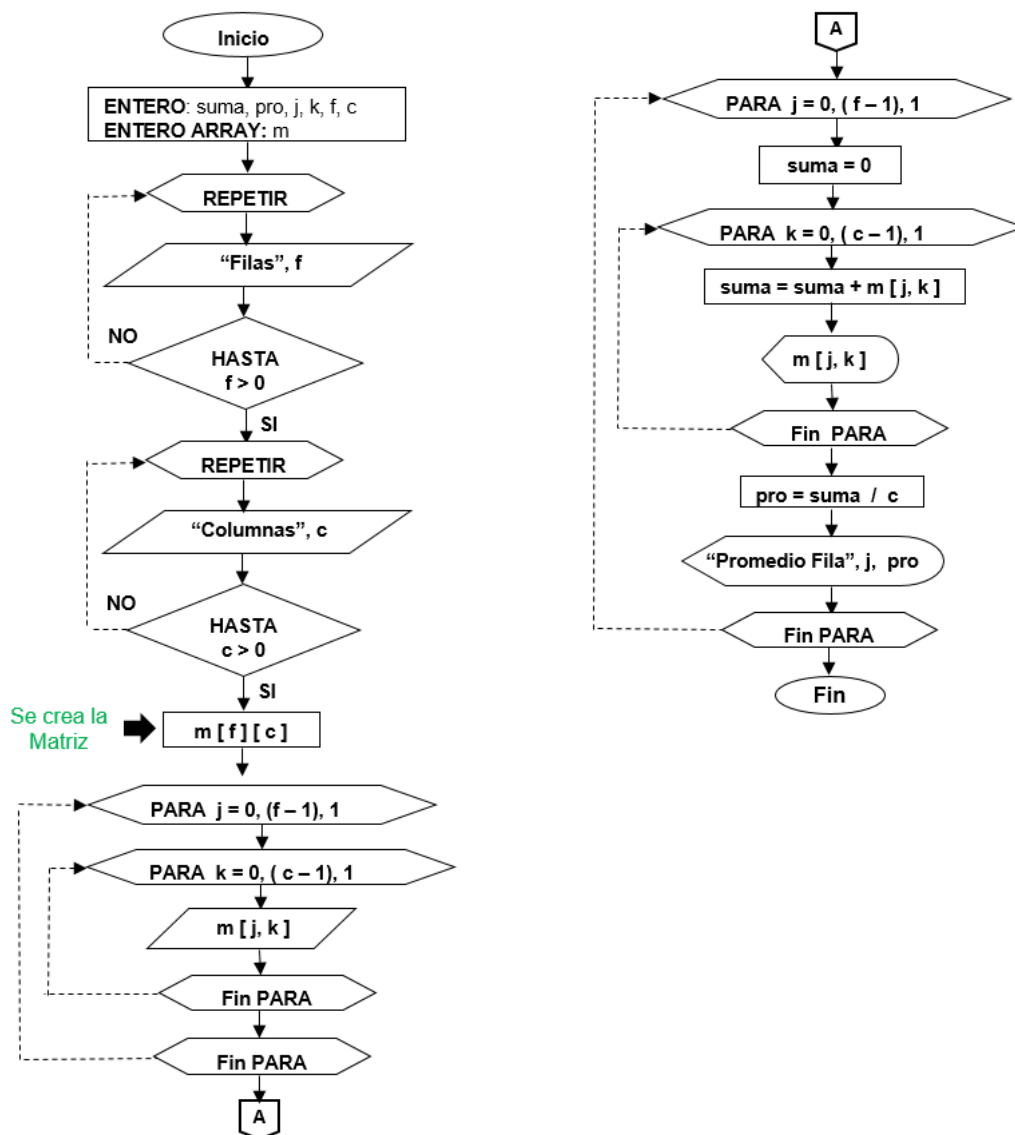
Estos ciclos anidados, permiten recorrer toda la matriz, permitiendo mostrar sus valores. Al final calcula el promedio de sus valores.

Problema 2: Diagrama que utilice una matriz llamada "M" de "F" filas y "C" columnas (ingresar y validar con valores positivos). Se solicita lo siguiente:

- Ingresar valores a la matriz.
- Calcular y mostrar el promedio aritmético de valores en la matriz por FILAS.

x	x	x	x	→ Promedio
x	x	x	x	→ Promedio
x	x	x	x	→ Promedio

- Mostrar los valores de la matriz.



Problema 3: Crear un diagrama que permita la utilización de una matriz “cuadrada” llamada “M” con N filas y N columnas (validar que sea positivo). Se solicita lo siguiente:

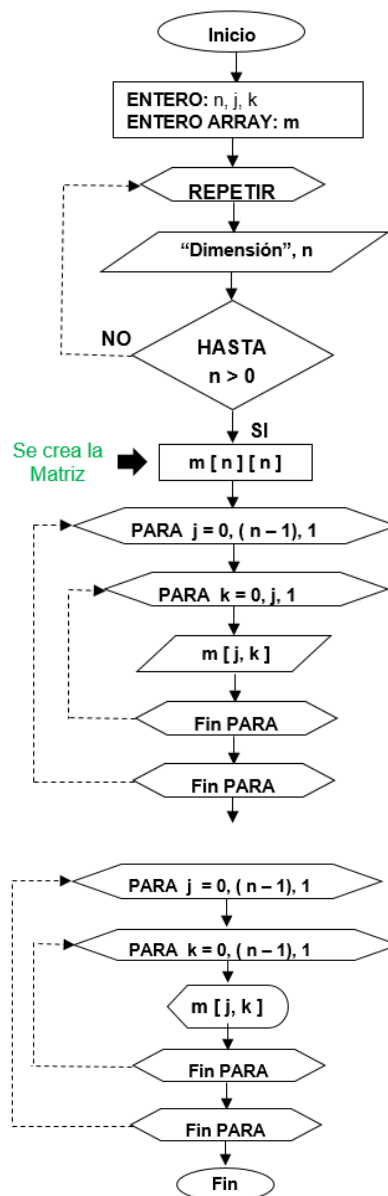
- a) Ingresar valores a la matriz solo de su diagonal principal hacia abajo.

x			
x	x		
x	x	x	
x	x	x	x

- b) Mostrar los valores de la matriz, incluyendo las posiciones sin valores ingresados.



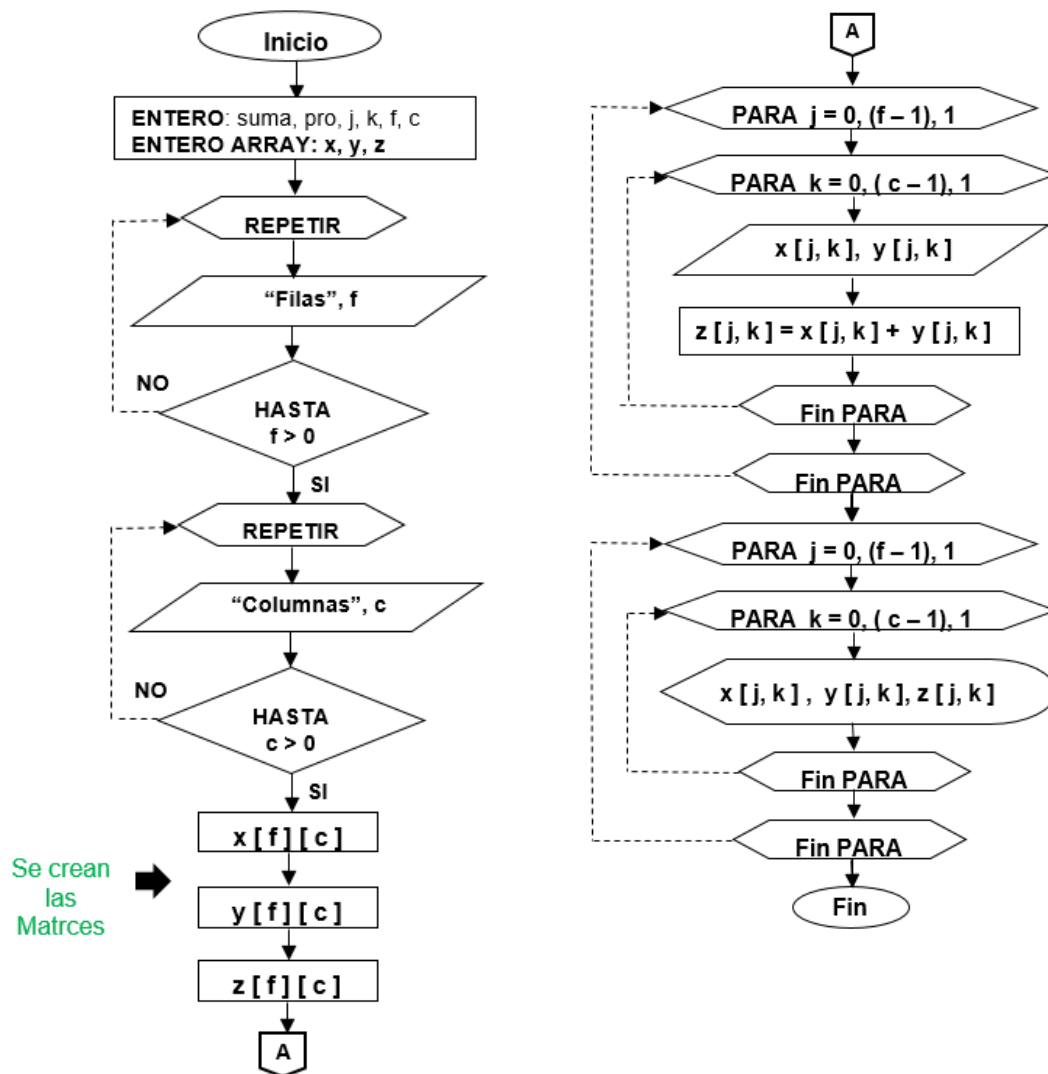
La matriz se dimensiona para el mismo valor de la variable “n”. Al realizar la dimensión de esta forma, se crea una matriz cuadrada (misma filas, mismas columnas).



Estos ciclos anidados, permiten el ingreso de valores a la matriz. El algoritmo determina que el ingreso se realice a igual número de fila igual número de columna.

Problema 4: Diagrama que utilizando tres matrices (X, Y y Z) dimensionadas para “F” filas y “C” columnas (ingresar y validar con valores positivos). Se realiza lo siguiente:

- Ingresar valores a solo a las matrices “X” y “Y”.
- La matriz “Z”, será el resultado de la **suma de valores de las matrices “X” y “Y”**.
- Mostrar los valores de las tres matrices.



Problemas Propuestos con uso de Matrices

Problema 1: Diagrama que permita crear una matriz de 2 filas y 4 columnas. Se solicita lo siguiente:

- a) Ingresar valores a la matriz.
- b) Asignar los valores de la matriz a otra matriz (opuesta), de la siguiente forma:

15	50	1	27
80	7	20	5

15	80
50	7
1	20
27	5

- c) Muestre por pantalla las dos matrices

Problema 2: Diagrama que utilice una matriz llamada “M” de “F” filas y “C” columnas (ingresar y validar con valores positivos). Se solicita lo siguiente:

- a) Ingresar valores a la matriz.
- b) Contar y mostrar cuantos valores dentro de la matriz son “pares”.

Problema 3: Diagrama que utilice una matriz llamada “M” de “F” filas y “C” columnas (ingresar y validar con valores positivos). Se solicita lo siguiente:

- a) Ingresar valores a la matriz.
- b) Ingresar un número “K”, y generar una nueva matriz donde cada elemento de la matriz ingresada sea **multiplicado por K** (multiplicación por escalar).

Problema 4: Diagrama que utilice una matriz llamada “M” de “F” filas y “C” columnas (ingresar y validar con valores positivos). Se solicita lo siguiente:

- a) Llenar la matriz de tal forma que cada elemento de la misma contenga el valor de su propia fila.
- b) Muestre el contenido de la matriz.

Problema 5: Crear un diagrama que permita la utilización de una matriz “cuadrada” llamada “M” con N filas y N columnas (validar que sea positivo). Se solicita lo siguiente:

- a) Ingresar valores a la matriz **solo** a su diagonal principal.

x			
	x		
		x	
			x

- b) Muestre los valores de la matriz, incluyendo las posiciones sin valores ingresados.

Problema 6: Crear un diagrama que permita la utilización de una matriz “cuadrada” llamada “**M**” con N filas y N columnas (validar que sea positivo). Se solicita lo siguiente:

- a) Ingresar valores a la matriz desde su “**diagonal principal**”, hacia **arriba**.

x	x	x	x
	x	x	x
		x	x
			x

- b) Muestre los valores de la matriz, solo en las posiciones donde se ingresó.

Problema 7: Crear un diagrama que permita la utilización de una matriz “cuadrada” llamada “**M**” con N filas y N columnas (validar que sea positivo). Se solicita lo siguiente:

- a) Asignar valores de “1” en su “**diagonal secundaria**”, asignar “0” por encima de la diagonal y “2” por debajo de la diagonal.

0	0	0	1
0	0	1	2
0	1	2	2
1	2	2	2

- b) Muestre los valores de la matriz.

Problema 8: Diagrama que utilice una matriz llamada “**M**” de “**F**” filas y “**C**” columnas (ingresar y validar con valores positivos). Se solicita lo siguiente:

- a) Llenar la matriz (relleno zig-zag) con los primeros números (total de elementos de la matriz), de acuerdo al flujo siguiente:

1	2	3	4
8	7	6	5
9	10	11	12
16	15	14	13

- b) Mostrar los valores de la matriz.

Problema 9: Diagrama que utilice una matriz llamada “**M**” de “**F**” filas y “**C**” columnas (ingresar y validar con valores positivos). Se solicita lo siguiente:

- Ingresar valores a la matriz.
- Buscar y mostrar en qué posición de la matriz se encuentra un valor “**X**” previamente ingresado.

Considere qué en el caso de no encontrar el valor en la matriz, se envíe el mensaje: “Valor no se encuentra en la Matriz”.

Problema 10: Diagrama que utilice una matriz llamada “**M**” de “**F**” filas y “**C**” columnas (ingresar y validar con valores positivos). Se solicita lo siguiente:

- Ingresar valores a la matriz.
- El diagrama debe solicitar un **valor** que represente cualquier número de fila de la matriz (validar en base al valor que tenga “**F**”)
- Se debe buscar y mostrar el mayor valor ingresado en la fila ingresada

Problema 11: Diagrama que utilice una matriz llamada “**M**” de “**F**” filas y “**C**” columnas (ingresar y validar con valores positivos). Se solicita lo siguiente:

- Ingresar valores a la matriz.
- Buscar y mostrar en que posiciones de la matriz, se encuentran el mayor y el menor valor ingresados.

Problema 12: Crear un diagrama que permita la utilización de una matriz “**cuadrada**” llamada “**M**” con N filas y N columnas (validar que sea positivo). Se solicita lo siguiente:

- Ingresar valores a la matriz.
- intercambie los valores de la fila “**x**” con la fila “**y**”, siendo “**x**” y “**y**” dos valores ingresados (validar).
- Utilice un módulo para mostrar la matriz antes y después del intercambio.

Problema 13: Diagrama que utilice una matriz llamada “**M**” de “**F**” filas y “**C**” columnas (ingresar y validar con valores positivos). Se solicita lo siguiente:

- a) Ingresar valores a la matriz.
- b) Se pide determinar y mostrar que fila de la matriz tiene la mayor suma de elementos

Problema 14: Diagrama que utilice una matriz llamada “**M**” de “**F**” filas y “**C**” columnas (ingresar y validar con valores positivos). Se solicita lo siguiente:

- a) Ingresar valores a la matriz.
- b) Se pide calcular la posición del **punto de silla de matriz** (si es que existe en la matriz). Se dice que una matriz tiene un punto de silla **si alguna posición de la matriz es el menor valor de su fila y a la vez el mayor de su columna.**

Problema 15: Diagrama que registre las calificaciones de estudiantes en varias asignaturas usando una matriz donde: **filas** (ingresar) = estudiantes y **columnas** (ingresar) = asignaturas. El diagrama debe:

- a) Calcular y mostrar el promedio por estudiante
- b) Calcular y mostrar el promedio por asignaturas
- c) Calcular y mostrar el promedio general.

Bibliografía¹

- Cadena, C. E. C. (2022). *Algoritmos, diagramas de flujo y pseudocódigo: Los fundamentos de la programación*. Carlos Ernesto Custodio Cadena.
- Cairo Battistutti, O. (2019). *Metodología de la programación: algoritmos, diagramas de flujo y programas*. Alfaomega.
- Duque, D. F., et al. (2022). *Algoritmos y programación en pseudocódigo*. Universidad Santiago de Cali.
- González Guerra, L. H., Pérez Murueta, P. Ó. y de la Cueva Hernández, V. M. (2021). *Algoritmos: Análisis, diseño e implementación*. Ediciones Tecnológico de Monterrey.
- Joyanes Aguilar, L. (2020). *Fundamentos de programación: algoritmos, estructuras de datos y objetos* (5.ª ed.). McGraw-Hill.
- López Román, L. S. (2021). *Metodología de la programación: Algoritmos, diagramas de flujo y pseudocódigo* (4.ª ed.). Alfaomega.
- Mancilla Herrera, A (2014). *Diseño y construcción de algoritmos*.
- Pes, C. (2021). *Pseudocódigo para principiantes: teoría, ejemplos y ejercicios resueltos*. Independently published.
- Pinales Delgado, F. J. & Velázquez Amador, C. E. (2018). *Algoritmos resueltos con diagramas de flujo y pseudocódigo*. Alfaomega.
- Tamayo Silva, F. M. (2023). *Fundamentos de lógica de programación*. Editorial Académica Española / Amazon.

¹ Este libro se basa principalmente en la experiencia docente del autor, complementada con referencias clásicas de la literatura en ciencias de la computación, las cuales están disponibles en repositorios académicos y editoriales especializadas.

Autor

Ing. Winther Abel Molina Loor, Mg.



Docente universitario con amplia trayectoria en educación superior y formación tecnológica, con estudios de posgrado realizados en la Universidad Técnica Federico Santa María de Chile, donde obtuvo el título de Magíster Ejecutivo en Informática de Gestión y Nuevas Tecnologías, complementando su formación con diplomados y programas de especialización en Tecnologías de la Información.

Posee formación de tercer nivel en la Universidad Laica Eloy Alfaro de Manabí, obteniendo los títulos de Ingeniero en Sistemas, Analista de Sistemas, Tecnólogo en Computación Administrativa y Licenciado en Ciencias de la Educación.

Cuenta con experiencia docente en educación superior desde 1989 y en educación secundaria desde 1986, desarrollando actividades académicas en asignaturas relacionadas al área de los algoritmos y la programación, así como funciones administrativas y de gestión universitaria como Director de Carreras en la Facultad de Ciencias Informáticas de la ULEAM.

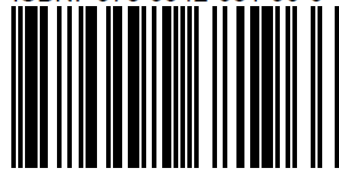
Es autor de artículos relacionados a las tecnologías y participante de varios congresos. Su experiencia profesional incluye la dirección de trabajos de titulación, así como el desarrollo de proyectos vinculados al campo de las Tecnologías de la Información y la Comunicación (TIC), con especial interés en la programación, la formación tecnológica y la innovación educativa.



Uleam

UNIVERSIDAD LAICA
ELOY ALFARO DE MANABÍ

ISBN: 978-9942-681-96-6



9789942681966



Todos los derechos reservados
Prohibida su venta
2026



<https://libros.uleam.edu.ec/>