

# DESARROLLO PRÁCTICO DE LA PROGRAMACIÓN



ALGORITMOS

DIAGRAMAS DE  
FLUJO

PROGRAMACIÓN  
ESTRUCTURADA

LENGUAJE C

# CONTENIDO

INTRODUCCIÓN .....	1
CAPITULO I .....	2
<b>EL COMPUTADOR Y LOS ALGORITMOS</b> .....	2
<b>DEFINICIONES BÁSICAS</b> .....	2
¿Qué es el hardware? .....	2
¿Qué es el software? .....	3
¿Qué es el firmware?.....	3
<b>PREÁMBULO DE LA LÓGICA PROGRAMABLE Y LOS ALGORITMOS</b> .....	3
UNIDADES DE ENTRADA. ....	4
UNIDADES DE SALIDA. ....	4
UNIDAD CENTRAL DE PROCESOS.....	4
¿Qué es un programa de computadora? .....	4
¿Qué es la lógica? .....	5
<b>ALGORITMOS NATURALES</b> .....	5
<b>OPERADORES ARITMÉTICOS</b> .....	7
<b>PREÁMBULO A LAS CONDICIONES APLICADAS EN LA LÓGICA</b> .....	10
¿En qué consisten las preguntas o condiciones en la lógica? .....	10
<b>INTRODUCCIÓN A LOS PROCESOS REPETITIVOS</b> .....	13
¿Qué son los procesos repetitivos? .....	13
<b>RESUMEN</b> .....	17
CAPITULO II .....	18
<b>PSEUDOCÓDIGOS</b> .....	18
<b>PREÁMBULO DEL PSEUDOCODIGO EN LOS ALGORITMOS</b> .....	18
<b>VARIABLES</b> .....	19
<b>OPERADORES DE COMPARACIÓN Y LÓGICOS</b> .....	21
OPERADORES DE COMPARACIÓN .....	21
OPERADORES LÓGICOS.....	22
<b>USO DE CONDICIONES SIMPLES UTILIZANDO PSEUDOCÓDIGO</b> .....	24
<b>USO DE CONDICIONES MÚLTIPLES UTILIZANDO PSEUDOCÓDIGO</b> .....	26
<b>USO DE CONDICIONES DE CASO UTILIZANDO PSEUDOCÓDIGO</b> .....	28
<b>DEFINICIONES GENERALES DE CONTROL Y EVALUACIÓN DE RESULTADOS</b> .....	31
USO DE CONTADORES Y ACUMULADORES COMO TÉCNICAS DE CONTROL.....	31
PRUEBAS DE FUNCIONAMIENTO O PRUEBAS DE ESCRITORIO .....	32
<b>CICLOS REPETITIVOS</b> .....	32
USO DE LA TÉCNICA “IR A” PARA CONTROLAR PROCESOS REPETITIVOS .....	32
USO DE LA TÉCNICA “MIENTRAS” PARA CONTROLAR PROCESOS REPETITIVOS .....	33

USO DE LA TÉCNICA “PARA” COMO CONTROL A PROCESOS REPETITIVOS .....	35
USO DE LA TÉCNICA “REPETIR... HASTA QUE” PARA CONTROLAR PROCESOS REPETITIVOS .....	37
USO DE LA TÉCNICA HACER... MIENTRAS PARA CONTROLAR PROCESOS REPETITIVOS .....	39
RESUMEN .....	42
CAPITULO III .....	43
USO DE LOS DIAGRAMAS DE FLUJO PARA RESOLVER PROBLEMAS .....	43
PREÁMBULO DE LOS DIAGRAMAS DE FLUJO .....	43
CARACTERÍSTICAS A CONSIDERAR PARA CREAR UN DIAGRAMA DE FLUJO .....	45
FORMATO UTILIZADO PARA EXPRESAR CONDICIONES EN LOS DIAGRAMAS DE FLUJO .....	46
FORMATOS UTILIZADOS PARA EXPRESAR CICLOS REPETITIVOS EN LOS DIAGRAMAS DE FLUJO .....	49
FUNCIONES MATEMÁTICAS .....	50
EJERCICIOS CON CONDICIONES DE CASO Y CONTROL DE CICLOS REPETITIVOS .....	54
DIAGRAMAS DE FLUJO CON CONDICIONES DE CASO .....	57
PROCESOS REPETITIVOS CONTROLADOS CON “REPETIR ... HASTA QUE” .....	59
PROCESOS REPETITIVOS CONTROLADOS CON “HACER ... MIENTRAS” .....	60
PROCESOS REPETITIVOS CONTROLADOS CON CONTADORES AUTOMÁTICOS .....	62
RESUMEN .....	64
CAPITULO IV .....	64
LENGUAJE DE PROGAMACIÓN .....	64
PREÁMBULO A LOS LENGUAJES DE PROGRAMACIÓN .....	65
ESTRUCTURA DE UN PROGRAMA EN C. ....	66
CABECERA. ....	66
DIRECTIVAS DEL PREPROCESADOR.....	67
BLOQUES DE FUCIONES .....	69
CONTENIDO DE MEMORIA Y LÍMITES DE ALMACENAMIENTO .....	71
¿Qué contiene la memoria principal de la computadora?.....	71
¿Cómo se realizan los cálculos en la computadora? .....	72
¿Cómo está estructurada una función?.....	74
OPERADORES DE CONTROL UTILIZADOS EN LENGUAJE C .....	80
OPERADORES ARITMÉTICOS .....	80
OPERADORES DE COMPARACIÓN .....	81
OPERADORES LÓGICOS.....	81
USO DE LA DIRECTIVA #define .....	82
ASIGNACIONES CONDICIONADAS .....	83
ERRORES EXISTENTES AL ESCRIBIR UN PROGRAMA .....	84
ERRORES DE SINTAXIS .....	84
ERRORES DE EJECUCIÓN .....	84

ERRORES DE LÓGICA .....	85
ESTRUCTURAS DE CONTROL .....	86
ESTRUCTURA DE CONTROLES CONDICIONALES .....	86
ESTRUCTURA CONDICIONAL IF. ....	86
ESTRUCTURA DE CONTROL IF CON CONDICIONES MÚLTIPLES O ANIDADAS .....	87
ESTRUCTURA DE CONTROL CONDICIONAL DE CASO SWITCH.....	89
ESTRUCTURAS DE CONTROL DE PROCESOS REPETITIVOS.....	93
ESTRUCTURA WHILE. ....	93
ESTRUCTURA DO-WHILE.....	96
ESTRUCTURA for().....	98
RESUMEN .....	101
CAPITULO V .....	102
ADMINISTRACIÓN Y GESTIÓN DE ARREGLOS.....	102
USO DE CADENAS DE CARACTERES .....	102
INTRODUCCIÓN A LOS ARRAYS .....	109
VECTORES DE CADENAS DE CARACTERES .....	112
MÉTODOS DE ORDENAMIENTO APLICADOS A VECTORES .....	115
ALGORITMO DE ORDENAMIENTO APLICANDO EL MÉTODO BURBUJA.....	115
ALGORITMO DE ORDENAMIENTO APLICANDO EL MÉTODO DE SELECCIÓN .....	117
ALGORITMO DE ORDENAMIENTO APLICANDO EL MÉTODO POR INSERCIÓN .....	119
ARREGLOS BIDIMENSIONALES O MATRICES .....	122
RESUMEN .....	126
CAPITULO VI .....	127
USO DE LAS FUNCIONES .....	127
¿Por qué simplificar la programación? .....	127
¿Qué es una función? .....	128
PASO DE PARÁMETROS POR VALOR .....	130
PASO DE PARÁMETROS POR REFERENCIA .....	131
ARREGLOS COMO PASO DE PARÁMETROS .....	133
AMBITO DE UNA VARIABLE .....	138
CREACION DE LIBRERÍAS PROPIAS .....	140
RESUMEN .....	142
CAPITULO VII .....	143
ESTRUCTURAS DE DATOS PROPIOS .....	143
VARIABLES ESTRUCTURADAS .....	143
INICIALIZACIÓN DE UNA VARIABLE ESTRUCTURADA:.....	144
VARIABLES ESTRUCTURAS ANIDADAS .....	145

<b>CREACIÓN DE ARREGLOS UTILIZANDO VARIABLES ESTRUCTURADAS.....</b>	<b>147</b>
<b>PASO DE ESTRUCTURAS COMO PARÁMETROS EN LAS FUNCIONES .....</b>	<b>150</b>
<b>RESUMEN .....</b>	<b>151</b>
<b>CAPITULO VIII .....</b>	<b>151</b>
<b>GESTIÓN DE ARCHIVOS DIGITALES .....</b>	<b>151</b>
<b>ARCHIVOS .....</b>	<b>152</b>
<b>RESUMEN .....</b>	<b>161</b>

## INTRODUCCIÓN

Esta propuesta bibliográfica está desarrollada con la finalidad de que el lector pueda conocer, entender y aplicar las destrezas necesarias para crear y actualizar los diferentes programas de aplicabilidad utilizados por la computadora, para lograrlo se propone varios capítulos que deben ser leídos de forma secuencial desde el principio hasta el final de forma consecutiva, éstos capítulos se dividen en apartados y a su vez en temas de específico interés; por sus importancia los apartados de este libro incluyen objetivos básicos que se pretenden alcanzar con la lectura y el desarrollo de los diferentes ejercicios propuestos; considere que, por cada tema tratado, existe su explicación práctica que respalda la conceptualización propia de cada tema, además incluye ejercicios ejemplificadores identificados mediante un código único para que el lector lo pueda resolver y afianzar los objetivos planteados; en caso de que el lector tenga interrogantes sobre el planteamiento de una solución se puede analizar y hacer uso de las alternativas de solución que se plantean al final del libro.

Es importante destacar que el documento formula retos de desarrollo, cuya solución no se encuentra incluido en el libro pero sus modelos de solución, se basa en los ejercicios utilizados en la explicación los diferentes procedimientos lógicos de cada apartado; considerando estos retos, cada ejemplo tratado y propuesto en los diferentes temas, incluyen un análisis que facilita su entendimiento y desarrollo; los diferentes ejercicios analizados y explicado en su desarrollo, sirven de ejemplo y modelo para aplicarlos en problemas similares; esta documentación esta desarrollada con la finalidad de satisfacer la necesidad de adquirir de forma estratégica, los conocimientos técnicos básicos que un programador necesita para aplicar soluciones a cualquier tipo de programa, considere que el estudio de éste contenido se lo puede aplicar a estudiantes y personas en general que tengan el deseo de aprender a programar computadoras desde cero.

Así, el primer capítulo define la estructura básica de la computadora, la influencia del software sobre el hardware, plantea los procedimientos lógicos para organizar las ideas, aplica algoritmos naturales para definir instrucciones lógicas y plantear soluciones a problemas planteados, además define las diferentes simbología y su jerarquía aritmética para expresar cálculos; incluye condiciones para realizar determinadas acciones y explica la forma de trabajo de los diferentes controles sobre eventos repetitivos. En el segundo capítulo se plantea el uso de los pseudocódigos, variables, simbología para desarrollar operaciones lógicas y condicionales, condiciones simples, múltiples y de caso; se utilizan contadores y acumuladores y se aplican pruebas de funcionamiento a las diferentes soluciones, se conceptualiza las pruebas de escritorio y se aplican controles a los diferentes procesos repetitivos.

En el capítulo tres incluye el desarrollo de algoritmos mediante el uso de los diagramas de flujo y el capítulo cuatro introduce al programador en la importancia de utilizar un lenguaje de programación, define estructuras generales, elementos incluyentes de un programa, utiliza elementos aportados por otros programadores para realizar procesos de entrada y salida, explica los operadores de control, tipos de errores y las estructuras de control tanto condicionales como de repetición.

El quinto capítulo incluye los conocimientos de creación y administración de estructuras de datos expresada en arreglos unidimensionales y multidimensionales, comienza explicando las cadenas de caracteres, vectores, métodos de ordenamiento de vectores y el uso de las matrices. El capítulo seis explica la creación y administración de las funciones en general, explica el paso de parámetro por valor y el paso de parámetros por referencia, incluye la aplicación de arreglos como argumentos y parámetros, define el ámbito de una variable y la creación de librerías propias.

En el capítulo siete se explica la creación y aplicación de estructuras de datos propios, para la creación de variables de tipo registro o variables estructuradas y paso de datos estructurados como

argumentos a las funciones; el capítulo ocho se refiere a la gestión de archivos de texto y archivos con formatos binarios. Considere que los capítulos de este libro se desarrollaron con la finalidad de que el lector adquiriera la habilidad de desarrollar programas para aplicarlos a la computadora y generar nuevas funcionalidades a la misma.

## **CAPITULO I**

### **EL COMPUTADOR Y LOS ALGORITMOS**

Las computadoras son máquinas que resuelven los problemas mediante la ejecución de órdenes que la máquina obedece instrucción por instrucción de acuerdo a lo que el programador haya desarrollado, así, un programa es un documento lleno de instrucciones con capacidad de resolver las diferentes necesidades de cálculo o de automatización; para que una persona pueda desarrollar habilidades de programación aplicadas a las computadoras, necesita cumplir varias etapas de aprendizaje, que parten de la formalización del pensamiento lógico, aplica la identificación de problemas explícitos e implícitos y propone soluciones confiables.

En este capítulo se plantea la necesidad de identificar los diferentes elementos que componen un computador, observando su importancia y relación con el desarrollo de los diversos programas, así como la aplicación de los conceptos básicos que definen al pensamiento lógico; notará que se propone una redacción ordenada, que incluye una serie de secuencias de pasos naturales mediante el uso de la narración propia del ser humano para resolver problemas, se propone identificar y utilizar elementos, como el uso de textos simbólicos para realizar condiciones comparativas, que permitan seleccionar determinadas acciones en caso de cumplirse la condición y no otras; además se analiza y se aplica los diferentes recursos y técnicas que permiten controlar los procesos que se deben repetir para evitar redundancia en la escritura en los programas.

En esta etapa de mentalidad, el lector podrá utilizar como herramienta de desarrollo lógico, las técnicas de los algoritmos naturales, que permiten proponer acciones de solución a los diversos problemas planteados; en los problemas de cálculo se propone emplear los diferentes símbolos aritméticos, considerando su jerarquía y correcta utilización; es importante destacar que la revisión de este capítulo, le servirá como base de conocimiento para desarrollar programas con mayor complejidad y que requieren de los conocimientos descritos en éste capítulo, además cada apartado posee un objetivo de aprendizaje que permitirán entender los logros propuestos que se desean alcanzar en cada postulado.

**OBJETIVO DE APRENDIZAJE ESTE APARTADO:** El presente apartado le permitirá conocer que es un programa aplicado a la computadora, qué acciones se necesita de usted para desarrollar la lógica, qué herramientas puede utilizar para plasmar de forma inicial la lógica aplicada a un programa, cómo hace la computadora para realizar los cálculos y como aplicar la jerarquía de desarrollo en los diferentes cálculos matemáticos.

### **DEFINICIONES BÁSICAS**

Es importante entender, cómo funcionan todos los equipos electrónicos digitales que existen en el mercado en la actualidad, tales como: computadores, tabletas, teléfonos y electrodomésticos inteligentes, vehículos con controles electrónico, equipos de control de comunicaciones, entre muchos otros dispositivos que se ofrecen en la actualidad; considere que todos ellos tienen en común tres componentes principales: Hardware, Software y Firmware

#### **¿Qué es el hardware?**

Este término describe la parte física de cualquier equipo; se identifica por los componentes electrónicos, piezas de soporte, cubierta protectora, entre otros elementos, que se pueden observar y tocar; considere que estos equipos inteligentes tienen un elemento clave en común, este elemento

permite ampliar las posibilidades físicas y de utilidad a todo hardware electrónico, ha transformado realmente las posibilidades tecnológicas que se disfrutaban en la actualidad, el componente se llama microprocesador y es capaz de entender y cumplir las instrucciones dadas por el programador.

### ¿Qué es el software?

Este término describe la parte lógica de cualquier equipo, se constituye en la base que permite el funcionamiento del hardware, así también en aplicativos o programas de utilidad que poseen los diferentes dispositivos para ampliar la utilidad de cualquier equipo electrónico inteligente, otras manifestaciones del software, se da en los programas de configuración, programas de utilidad como los procesadores de texto, programas de entretenimiento como lectores de video, musicales, juegos, entre otros tipos de programas.

El software se constituye en los programas o las instrucciones dadas por el programador para que el equipo esté operando, en sí son las ideas plasmadas en ordenes inteligentes que el equipo obedece para administrar el hardware, es decir, es el componente importante que permite la existencia y utilidad extra que tiene todo equipo inteligente que utiliza un microprocesador.

### ¿Qué es el firmware?

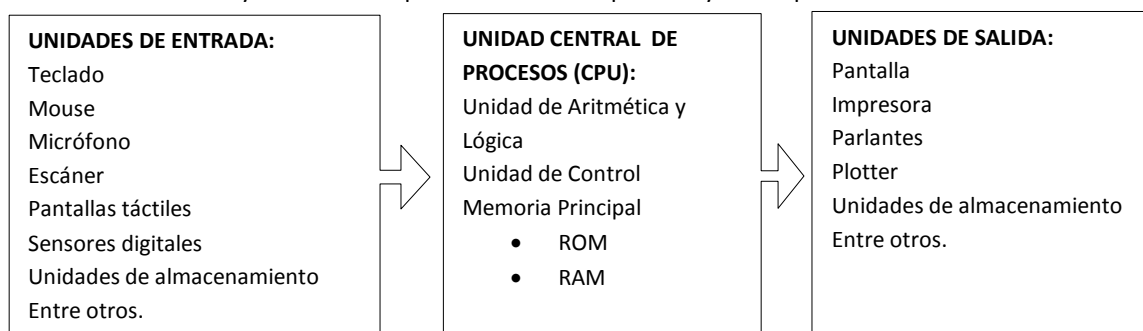
El firmware es una combinación entre el hardware y el software, son instrucciones o programas que tienen un propósito específico y que están contenidos en una memoria de solo lectura, estos programas permiten controlar los circuitos electrónicos de un dispositivo cualquiera, normalmente estos programas son desarrollados por el fabricante del componente.

Es importante entender que como futuro programador usted podrá realizar aplicaciones o programas que permitan administrar el hardware de cualquier dispositivo inteligente, podrá construir programas aplicativos para empresas de cualquier naturaleza comercial, aplicativos para áreas de investigación o programas de entretenimiento en general; estas afirmaciones están basadas en la utilidad importante que tiene la tecnología sobre la sociedad, y en las definiciones conceptuales que explican la utilidad imprescindible del software sobre el hardware; es importante resaltar que, en el campo comercial, los precios del software son mayores que los precios ofertados por el hardware.

## PREÁMBULO DE LA LÓGICA PROGRAMABLE Y LOS ALGORITMOS

Se considera al computador como un equipo electrónico digital multipropósito, esto quiere decir que se lo puede utilizar en todas las áreas del conocimiento humano, ya que puede desarrollar grandes cantidades de procesos, intercambios de datos, cálculos de mayor precisión entre otras posibilidades mediante el uso de los diferentes programas, considerando que, el computador entiende y cumple sin omitir instrucciones estipuladas por el programador, la estructura básica de un computador se detalla en la siguiente gráfica:

**Gráfica 1.** Elementos y secuencia de proceso de un computador y sus dispositivos interconectados



**Fuente:** Autores del libro



Los cuadros de la gráfica 1, detallan los diferentes orígenes o fuentes que proveen datos para que el procesador, aplique el programa correspondiente y cumpla los fines o propósitos para lo cual fue desarrollado, así:

### **UNIDADES DE ENTRADA.**

Son todos los equipos, periféricos, dispositivos o aparatos, que permiten la entrada de datos para que la Unidad Central de Proceso (CPU) pueda procesarlos y generar un resultado, estos dispositivos son utilizados para alimentar de información a los diferentes programas que ejecuta la computadora para darle aplicabilidad y funcionalidad al ordenador.

### **UNIDADES DE SALIDA.**

Son todos los equipos, periféricos, dispositivos o aparatos, que permiten la salida de datos procesados por la CPU, estos dispositivos son utilizados para mostrar o plasmar las acciones o la información que los diferentes programas gestionan junto al computador para exponer y evaluar resultados y su utilidad en el mercado y la ciencia.

### **UNIDAD CENTRAL DE PROCESOS CPU.**

Es la integración de un componente elemental del computador, está compuesta por la Unidad de Aritmética y Lógica (UAL) que sirve para realizar los procesos de comparación y los cálculos aritméticos; complementaria a la UAL es la Unidad de Control (UC) que sirve para controlar los flujos o circulación de datos, tanto de entrada, de proceso y de salida, con la firme intención de que los datos se trasladen de forma correcta; estos dos elementos se encuentran integrados en un solo Chip llamado microprocesador.

El otro elemento que compone la CPU es la memoria principal, este recurso importante del computador esta compuesta por dos partes principales: ROM Read Only Memory (Memoria de solo lectura), sirve para mantener almacenada la información física del computador (Configuración), para entender la utilidad de este recurso, considere que existen muchos dispositivos que se conectan a la CPU con diferentes proveedores y características propias de cada equipo y fabricante, esto complica el encendido del equipo ya que la CPU debe conocer con que elementos va a trabajar y cómo se comunicarán entre si, la memoria ROM evita tener que configurar el computador al indicarle que elementos tiene conectado cada vez que se enciende el ordenador, por su importancia la información no se borra, al contrario se mantiene aun cuando el computador se encuentra apagado.

El otro elemento importante de la memoria principal es la memoria RAM Random Access Memory (Memoria de lectura y escritura o de acceso aleatorio), se constituye en el complemento ideal para el microprocesador, se encarga de almacenar toda la información con la que está trabajando el computador, incluida parte de la información que posee la memoria ROM, cada instrucción de los programas que están ejecutando en el computador, es contenida antes y después de ser procesada por la CPU en la memoria RAM, es importante considerar que la información contenida, se borrará automáticamente, cada vez que finalice el programa o al momento de apagar el computador.

### **¿Qué es un programa de computadora?**

Se podría definir que un programa es una *secuencia de órdenes o instrucciones con un propósito lógico* que una computadora necesita para realizar una o varias tareas (resolver problemas), es decir son instrucciones dadas y almacenadas en forma de documentos que el computador ejecuta o cumple instrucción por instrucción hasta finalizar el documento.

Un programa es la respuesta de solución programable a un problema planteado, estas órdenes se las escribe en un lenguaje de programación, éste lenguaje define las reglas de escritura (sintaxis), la lista

de instrucciones disponibles y las normas de cooperación con el sistema operativo (Software principal que administra el computador), el hardware y otros lenguajes de programación.

### **¿Qué es la lógica?**

La lógica se constituye como un *pensamiento razonado que cumple un propósito*, es importante reconocer que los pensamientos son tan diversos como lo son los seres humanos, por lo tanto, para usted lo que es lógico, para otra persona no lo es; esta diversidad de pensamientos y de lógicas deben pulirse en el programador, de tal forma que genere resultados estandarizados (todos deben entenderlo), coordinarse y desarrollarse de forma que permita dar solución a una diversidad de problemas.

Así, un programa es el resultado de escribir instrucción por instrucción de forma secuencial la solución a un determinado problema, el programador con su lógica decidirá qué acciones debe hacer el computador para resolver un problema; para que el programador pueda desarrollar documentos de instrucciones, necesitara entender reglas y normas de redacción de instrucciones, para que el computador entienda cómo se resuelve el problema.

*El libro propone una cronología de aprendizaje*, que permitirá al lector conocer cómo resolver un problema mediante el desarrollo de una propuesta de pensamiento lógico; considere que la preparación de un programador requiere del cumplimiento de etapas que perfilan la mecánica y la aplicación de las diferentes metodologías orientadoras que facilitan el uso de herramientas desarrolladas para éste propósito, así se piensa cumplir con las siguientes etapas:

1. Uso de algoritmos naturales y pseudocodificados.
2. Uso de Diagramas de Flujo para el desarrollo de propuestas algorítmicas.
3. Uso de un Lenguaje de Programación para generar aplicaciones básicas.

Es importante considerar que las dos primeras etapas, son básicamente de aprendizaje de conocimientos y técnicas que consisten en aplicar técnicas de control para resolver problemas propuestos, una vez finalizadas y definidas las habilidades de aplicación de las diferentes técnicas y procedimientos lógicos, se estará en capacidad de hacer uso directo de un lenguaje de programación.

### **ALGORITMOS NATURALES**

A esta técnica se la define como el conjunto de pasos redactados en un lenguaje natural, que tiene una secuencia ordenada de instrucciones, que permiten alcanzar un determinado objetivo. El término secuencial, implica que de forma ordenada, se debe cumplir una instrucción después de otra, esto quiere decir que, las instrucciones se deben redactar en un orden obligatorio, por ejemplo, supongamos que a usted le piden redactar los pasos para sumar dos números de forma ordenada, una posible solución al problema sería el siguiente algoritmo:

#### **Algoritmo 1:**

*Iniciamos*

1. Recibir los dos números que desea sumar
2. Sumar los dos números (como este proceso es un cálculo, considere que todo cálculo *siempre generará un resultado*)
3. Entregar/escribir o mostrar el resultado

*Finalizamos*

Observe, que en el ejemplo los pasos están numerados, esto definen el orden o la secuencia lógica de cómo sumar dos números; si se analizan los pasos detallados, no sería lógico “entregar/escribir o mostrar el resultado” antes de realizar el cálculo “verdad”; considere que, el aplicar la lógica

programable para resolver problemas cotidianos, consistirá en detallar y ordenar eventos que normalmente omitimos por la práctica directa y sobre entendida de los pormenores; expertos en ésta área definen a la lógica algorítmica, como los pasos normales que aprendimos cuando atravesamos la niñez, en la cual “una cosa lleva o va después de otra” para cumplir una tarea, ¿Será por esto que a los niños se les hace más fácil comprender esto?, y que como una técnica propuesta, necesitamos recordar para aplicar instrucciones ordenadas, que el computador entenderá para realizar procesos adecuados en la resolución de problemas.

Ahora considere aplicar un ejemplo lógico de la vida real, en la que se desea un algoritmo para ponerse cualquier camisa; una posible solución pormenorizada podría ser:

#### **Algoritmo 2:**

*Iniciamos*

1. Dirigirnos al lugar donde tenemos la camisa.
2. Si su ubicación es el closet o el cajonero, entonces se abre y coger la camisa.
3. Si la camisa tiene los botones abrochados, entonces se debe desabrochar los botones.
4. Abrir la camisa.
5. Meter uno de los brazos por su manga correspondiente.
6. Meter el otro brazo por la otra manga.
7. Acomodar la camisa a su tronco.
8. Abrochar (botón a botón) y ajustarla a su cuerpo.

*Finalizamos.*

Este algoritmo propone la solución en ocho pasos lógicos, pero de seguro usted podría optar por resolverlo de otra forma en más o en menos pasos, ya que las soluciones dependen del *pensamiento lógico* de cada ser humano y su realidad cercana. Un principio básico que debe tener claro, es que las soluciones lógicas no son únicas, esto quiere decir que pueden tener diferentes formas de solución y que estará sujeta al pensamiento lógico y a la realidad de cada ser humano.

Este libro propone estandarizar (entendible para todos) los pensamientos lógicos, empezará aplicando algoritmos de tres pasos, que consisten en:

- 1.-Recibir/obtener/pedir datos,
- 2.-Procesarlos y
- 3.-Entregar/escribir o mostrar el resultado solicitado por el problema.

Esta propuesta de algoritmos estandarizados son ideales para resolver problemas de cálculo, tome en cuenta que siempre se piden los datos necesarios para ser procesados o calculados, que para mostrar los resultados, es importante conocer cómo se aplican los cálculos y cómo se resuelven; es decir, si el programador no posee los conocimientos necesarios de como se aplican los cálculos, será difícil desarrollar una propuesta de algoritmo que ofresca una solución a dicho problema, ante este tipo de dificultades, se propone desarrollar un análisis, que permita conocer la forma de aplicar las instrucciones necesarias y los procedimientos adecuados antes de desarrollar una propuesta de algoritmo.

Para ilustrar lo detallado hasta ahora, considere el siguiente ejemplo; supóngase que se le pide el algoritmo que calcule el área de un rectángulo, parte del análisis consistiría en conocer cómo de encuentra el resultado, si existe o no una fórmula adecuada, para este caso, los cálculos se aplican mediante la fórmula:

$$a = b \times h$$

Donde (a) representa el resultado denominada “área”, (b) representa el valor de “base” y (h) representa el valor de “altura”, fíjese que el cálculo de la fórmula consiste en una simple multiplicación; para resolver la fórmula y encontrar el área, necesitará recibir los valores de base y

de altura; aplicar el cálculo y obtener el resultado; por lo tanto la propuesta algorítmica para resolver el problema, quedaría de la siguiente forma:

### Algoritmo 3:

Iniciamos

1. Recibir los valores de base y altura
2. Multiplicar base por altura y obtener el resultado
3. Entregar o escribir el resultado

Finalizamos

Al desarrollar una propuesta algorítmica, es importante realizar un análisis que muestre los detalles técnicos, para esto se plantea responder las siguientes interrogantes:

*¿Qué datos se necesitaron para realizar el proceso?*

Los datos que se necesitan, son aquellos que se aplican en la fórmula, es decir los valores de b (Base) y h (Altura), fíjese que no se incluye pedir el valor de “a” (Área) ya que en la formula, ésta representa el resultado de la multiplicación.

*¿Qué resultados se muestra?*

Para este ejemplo el resultado consistiría en el área, que se obtiene de multiplicar base por altura, este proceso lógico simplemente aplica la fórmula paso a paso hasta obtener el resultado.

La propuesta algorítmica de estandarización de los tres pasos lógicos, aplicados a resolver problemas de cálculo, obliga al lector a conocer y aplicar los diferentes símbolos, que identifican los cálculos y su formulación aritmética.

## OPERADORES ARITMÉTICOS Y SUS JERARQUÍAS

Básicamente los operadores aritméticos son símbolos que el programador debe utilizar para realizar operaciones aritméticas, como sumas, restas, multiplicaciones, divisiones entre otras, es importante conocer que las operaciones se realizan de *izquierda a derecha* respetando la jerarquía de cada operación; la siguiente tabla define los símbolos aritméticos y su jerarquía de cálculo:

**TABLA 1.** Símbolos aritméticos y sus jerarquías operacionales de cálculo

JERARQUÍA	OPERADOR	USO
1 <sup>ero</sup>	()	Permite agrupar cálculos que se realizarán primero
2 <sup>do</sup>	**, ^	Para cálculos de potencia o exponenciación
3 <sup>ero</sup>	*	Para cálculos de multiplicación, se encarga de devolver el producto
	/	Para cálculo de división, se encarga de devolver el cociente
4 <sup>to</sup>	MOD	Para cálculo de división, se encarga de devolver el residuo
5 <sup>to</sup>	+	Para cálculos de suma
	-	Para cálculos de resta

**Fuente:** Autores del libro

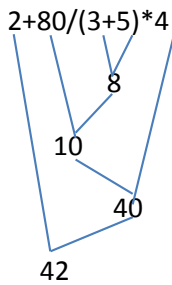
Es importante recordar una vez más, que las operaciones aritméticas se resuelven de izquierda a derecha, respetando la jerarquía descrita en la tabla 1, esto significa que primero se resolverá lo que está entre paréntesis, después se resolverán los cálculos que tienen potencias, después las multiplicaciones y divisiones con la misma jerarquía, después la división de módulo (devuelve el residuo de la división), y por último se resolveran las sumas y restas con la misma jerarquía.

Cómo detalle destacable, observe que en esta tabla se utiliza nueva simbología, las mismas que reemplazan algunos signos aritméticos como la multiplicación y la división; para describir la importancia de la jerarquía considere el siguiente ejemplo de cálculo:

$$2 + 2 * 2$$

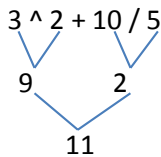
Normalmente se podría pensar que el resultado es (8) pero eso es incorrecto, el resultado respetando la jerarquía es 6, ya que primero por jerarquía se realiza la multiplicación  $2*2$  y luego se realiza la suma  $2+4$ ; los siguientes ejercicios, son ejemplos que aplican jerarquía con éstos operadores:

- ✓ Ejemplo de agrupamiento de cálculos con paréntesis ():



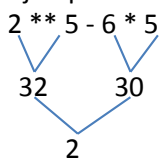
El ejemplo tiene 4 operaciones de cálculos aplicando jerarquía: el primero es la suma que está entre paréntesis  $3+5$ , el segundo cálculo es la división  $80/8$  ya que está ubicada a la izquierda con mayor jerarquía, el tercer cálculo es la multiplicación  $10 * 4$  y el último cálculo es la suma  $2+40$

- ✓ Ejemplo utilizando potencia:

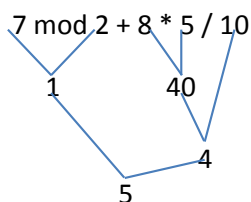


En este ejemplo se tienen tres cálculos, el primero es la potencia  $3^2=3*3$ , el segundo cálculo es la división  $10/5$  y el tercer cálculo es la suma  $9 + 2$

- ✓ Ejemplo utilizando la potencia, también se puede expresar con el doble asterisco (\*\*)



- ✓ Ejemplo utilizando la división con el signo / y con MOD:



Sí tenemos la división: 
$$\begin{array}{r} 7 \overline{) 2} \\ (1) \quad 3 \end{array}$$

El cociente es 3 y el residuo es 1, el primer cálculo obtiene el residuo de dividir 7 para 2, en el segundo cálculo realiza la multiplicación  $8*5$  y después la división  $40/10$ , esto ya que los cálculos son de izquierda a derecha, y por último se realiza la suma  $1+4$ .

Considerado los ejemplos descritos y teniendo en cuenta la importancia de las jerarquías aplicadas a las operaciones aritméticas, suponga que le han solicitado un algoritmo, que calcule el promedio de tres notas de un estudiante cualquiera, el análisis previo consistiría en aplicar la suma de las notas y el resultado dividirlo para tres, por lo tanto se aplicaría la siguiente fórmula:

$$\text{Promedio} = (\text{nota1} + \text{nota2} + \text{nota3}) / 3$$

Al desarrollar una propuesta de algoritmo, quedaría de la siguiente forma:

#### Algoritmo 4:

*Iniciamos*

1. Recibir los valores que corresponden a las tres notas: nota1, nota2 y nota3
2. Calcular: Sumando las tres notas y el resultado se lo divide para 3
3. Entregar o escribir el resultado

*Finalizamos*

Este algoritmo ejemplificado, define lo importante de establecer la secuencia y jerarquía de los cálculos, ya que al no especificar los detalles como el agrupamiento, se podría incurrir en un error, por ejemplo: "Promedio =  $\text{nota1} + \text{nota2} + \text{nota3} / 3$ ", al aplicar el cálculo de esta forma, solo la nota3 se dividiría para 3 y después se sumarían las otras dos notas, lo cual no está correcto. Considere las siguientes interrogantes técnicas para describir la propuesta del algoritmo:

*¿Qué datos se necesitaron para realizar el proceso?*

Para éste proceso se necesitó los valores de las tres notas del estudiante.

*¿Qué resultado se muestra?*

Como resultado el algoritmo mostrará el promedio de las tres notas.

En ejemplos anteriores, se han planteado problemas donde el proceso es solicitado de forma explícita (describiendo el cálculo que se aplicará), pero es recomendable aplicar ejemplos que permitan desarrollar un análisis, dando la posibilidad de razonar las diferentes formas de resolver un problema. Para ejemplificar este concepto, se propone elaborar un algoritmo que permite calcular el salario de un jornalero que gana por horas de trabajo.

Análisis: Si una persona trabaja 8 horas y se le paga 3 dólares por cada hora, significa que como salario recibirá 24 dólares, este resultado se obtuvo mediante la multiplicación de 8 por 3, ya que se paga 3 dólares por cada hora de trabajo y laboró 8 horas; una propuesta de solución algorítmica al problema quedaría de la siguiente forma:

#### **Algoritmo 5:**

*Iniciamos*

1. Recibir el número de horas trabajadas y el valor que se le paga por cada hora
2. Multiplicamos horas trabajadas por el valor de cada hora
3. Entregar o escribir el resultado de la multiplicación, es decir el salario del jornalero

*Finalizamos*

Importante: Todos los temas del libro proponen actividades de refuerzo, estas permiten al lector realizar ejercicios de práctica, aplicando los temas tratados en cada apartado con la finalidad de identificar debilidades de comprensión y/o reafirmar los conocimientos adquiridos, como una actividad de aprendizaje se recomienda desarrollarlos y en caso de que se presente dificultades o confusión, se puede recurrir a la verificación del desarrollo del ejercicio que se encuentra en el solucionario del libro; por otro lado también se incluyen actividades extras, son retos de desarrollo a problemas planteados que el programador debe realizar por su propia cuenta, ya que las soluciones de los mismos no se encuentran incluidos en el libro; por último se presentan actividades de autoevaluación que permitirán al programador realizar una evaluación de forma personal, estas actividades se diseñaron con la finalidad de medir el nivel de comprensión logrado en cada una de las actividades propuestas.

#### **Actividades de refuerzo (AR):**

- AR1. Desarrollar los siguientes ejercicios de jerarquía de cálculos:  $2 * 2 + 2 ^ 2 - (2 + 2 \text{ Mod } 2)$        $7 * 5 - 5 + (3 ^ 3 / 3)$
- AR2. Modifique el algoritmo 1 de forma que, en vez de realizar la suma, realice la resta, considere que no se debe de dar como resultado un valor negativo.
- AR3. Modifique el algoritmo 2 de forma que, elabore los pasos para ponerse el pantalón.
- AR4. Modifique el algoritmo 3 de forma que, calcule el área de un cuadrado.
- AR5. Modifique el algoritmo 4 de forma que, calcule el promedio de las siguientes notas: Deber, lección, aporte, cuaderno al día y examen.
- AR6. Modifique el algoritmo 5 de forma que, calcule el salario más un bono por aniversario de la empresa.

#### **ACTIVIDADES EXTRAS:**

- A1. Desarrolle un algoritmo natural que permita calcular el cubo de un número
- A2. Desarrolle un algoritmo natural que permita calcular el área de un triángulo
- A3. Desarrolle un algoritmo natural que permita calcular el área de una circunferencia.
- A4. Desarrolle un algoritmo natural que permita calcular la edad en años de una persona sin considerar que ha o no cumplido con la fecha de nacimiento
- A5. Desarrolle un algoritmo natural que permita calcular el número de byte que tiene un archivo de texto cuyo tamaño está en megabyte.

### AUTOEVALUACIÓN:

- E1. Elabore una lista de dificultades presentadas al desarrollar las actividades propuestas y las actividades extras.
- E2. Proponga un ejercicio con su respectiva respuesta.

**OBJETIVO DE APRENDIZAJE PARA EL SIGUIENTE APARTADO:** Conocer las diferentes posibilidades de crear preguntas o condiciones y entender que instrucciones se pueden dar como respuesta a las preguntas planteadas.

### PREÁMBULO A LAS CONDICIONES APLICADAS EN LA LÓGICA

En la mayoría de los problemas algorítmicos que se proponen, existirá la posibilidad de incluir condiciones que permitirán al algoritmo plantear instrucciones que cumplan con acciones dependiendo de una condición planteada, esta capacidad es propia de los algoritmos y de los programas que el computador utiliza para resolver problemas ejecutando instrucción por instrucción.

Los problemas algorítmicos desarrollados y propuestos hasta el momento, se basan en *solicitar datos, procesarlos y entregar sus resultados*; considere que, de estas tres etapas planteadas para desarrollar algoritmos, la etapa de *proceso*, solo está siendo utilizada para realizar únicamente cálculos, éste material bibliográfico propone profundizar el conocimiento e incluir instrucciones más elaboradas e inteligentes, que se logre en usted la aplicación natural de condiciones a sus propuestas de solución; en esta etapa del aprendizaje se incluirá en los algoritmos, el uso de las comparaciones para expresar condiciones o preguntas lógicas.

#### ¿En qué consisten las preguntas o condiciones en la lógica?

En su forma más básica, una condición es un requisito solicitado, que debe cumplirse para realizar una determinada acción; en si una condición incluye necesariamente el uso de comparaciones, y las comparaciones permitirán dependiendo de la respuesta, aplicar instrucciones específicas para determinados casos de solución, en el transcurso de su preparación, se presentarán problemas en donde la solución, requerirá del uso de comparaciones, como programador se debe conocer y aplicar las diferentes formas de realizar preguntas; la siguiente lista, muestra las únicas posibilidades que se pueden utilizar para realizar las comparaciones:

- ✓ Se puede comparar sí, un dato es *mayor que* otro dato
- ✓ Se puede comparar sí, un dato es *menor que* otro dato
- ✓ Se puede comparar sí, un dato es *mayor o igual que* otro dato
- ✓ Se puede comparar sí, un dato es *menor o igual que* otro dato
- ✓ Se puede comparar sí, un dato es *igual que* otro dato
- ✓ Se puede comparar sí, un dato es *diferente que* otro dato

Un dato puede ser un valor o un texto que tiene su relativa importancia para resolver un problema.

En los algoritmos, el uso de una comparación tiene una de dos posibilidades de respuesta: VERDADERO o FALSO; en caso de aplicar comparaciones, las instrucciones que se deben desarrollar como respuesta en verdadero son obligatorias, y las instrucciones que se deben hacer como respuesta a falso son opcionales, es decir, si usted desea las incluye o no, esto dependerá de la conveniencia del programador. Es importante destacar que, en una comparación, solo se tomará una de las dos alternativas como respuesta, y jamás tomará ambas al mismo tiempo; al momento de realizar una propuesta algorítmica es recomendable desarrollar las instrucciones para ambos eventos (por verdadero y por falso) ya que puede existir la eventualidad de tomar cualquiera de ellas.

El siguiente ejemplo, muestra la aplicación de una condición en el algoritmo y las instrucciones a cumplir dependiendo de la respuesta a la comparación, para entenderlo, suponga que se necesita un algoritmo que compare dos valores diferentes y muestre como resultado solo el valor mayor:

### Algoritmo 6:

*Iniciamos*

1. Recibir el primer y el segundo número
2. *Preguntamos*, si el primer número *es mayor que* el segundo, (En caso de ser *verdadero*) *Entonces*, se muestra el primer número como mayor. *Caso contrario* (En caso de ser *falso*), se muestra el segundo número como mayor.

*Finalizamos*

Primero, note que el algoritmo no tiene cálculos en la etapa de proceso, y segundo, no existe el tercer paso habitual de los algoritmos naturales anteriores (Mostrar el resultado), ya que se encuentra incluido dentro del proceso del segundo paso (como respuesta a la condición, tanto por verdadero como por falso).

Sí cumplimos las instrucciones dadas por el algoritmo, podrá notar que las dos acciones, de verdadero y falso, no se ejecutarán al mismo tiempo, ya que solo una de las dos respuestas, mostrará al mayor de los dos números; aplicado las interrogantes técnicas, tenemos:

*¿Qué datos entran a ser procesados?*

Dos números diferentes cualesquiera.

*¿Qué resultados muestra?*

Aplicando la comparación de los dos números, muestra solo un número, el mayor de los dos valores

*¿Qué condiciones se presentan en el proceso?*

Se presenta solo una condición, la que decide cual es el número mayor.

Considere el siguiente ejemplo que muestra el uso de la comparación al desarrollar un algoritmo que teniendo un valor cualquiera verifica si es un número positivo o un número negativo.

Análisis: Un número se lo considera positivo si su valor es mayor que cero, por consiguiente es negativo cuando su valor es menor que cero, así, solo se necesita comparar el valor, preguntando si es mayor que cero para mostrar un mensaje indicando que es positivo o caso contrario mostrar el mensaje indicando que es negativo.

### Algoritmo 7:

*Iniciamos*

1. Recibir el número
2. *Preguntamos* si el valor es *menor que* cero, (En caso de ser *verdadero*) *Entonces* se muestra un mensaje que dice "Numero Negativo", *Caso contrario* (En caso de ser *falso*) se muestra un mensaje que dice "Numero Positivo".

*Finalizamos*

En este ejemplo el resultado es un mensaje que especifica la naturaleza positiva o negativa del número. Considere las siguientes interrogantes técnicas:

*¿Qué datos entran a ser procesados?*

El único dato es un número cualquiera

*¿Qué resultados muestra?*

Solo un mensaje que indica si el número es positivo o si es negativo

*¿Qué condiciones se presentan en el proceso?*

Se presenta solo una condición, la que decide si el número es negativo al ser *menor que* cero o positivo en caso contrario.

Para seguir con las ejemplificaciones, el siguiente algoritmo muestra como calcular el salario de un jornalero que trabaja por horas en el día, además si ha trabajado más de 8 horas, se le aumentará 10 dólares adicionales a su salario, como recompensa a su trabajo extra:



### Algoritmo 8:

*Iniciamos*

1. Recibir el número de horas trabajadas y el valor que se le paga por hora
2. Multiplicamos horas trabajadas por el valor de cada hora
3. *Preguntamos*, si el número de horas trabajadas es *mayor que 8*, (En caso de ser *verdadero*) *Entonces*, se le sumará 10 dólares al salario calculado
4. Entregar/Escribir o mostrar el salario del jornalero.

*Finalizamos*

Observe que en este ejemplo no muestra una acción por la alternativa de *falso* a la pregunta o comparación, esto se debe a que el planteamiento del problema solo pedía adicionar 10 dólares en caso de cumplir la condición, considerando que el jornalero haya trabajado más de 8 horas.

*¿Qué datos entran a ser procesados?*

El algoritmo solicita dos datos, el número de horas que trabajo y el valor que se le pagará por cada hora

*¿Qué resultados muestra?*

El algoritmo mostrará como resultado el producto de multiplicar horas trabajadas por el valor de la hora, si cumple con la condición, mostrará el salario incremento en 10 dólares, siempre y cuando el número de horas sea mayor que 8.

*¿Qué condiciones se presentan en el proceso?*

El algoritmo utiliza una condición, que tiene solo una alternativa como respuesta a la pregunta, compara si el número de horas trabajadas es mayor que 8 horas para agregar dinero a su salario; no utiliza la alternativa por falso ya que la condición no lo amerita; considere que la propuesta presentada puede tener otras alternativas de solución, esto dependerá de la lógica que aplique el programador al desarrollar su propuesta de solución.

### **Actividades de refuerzo (AR):**

- AR7. Modifique el algoritmo 6 de forma que pida la base y altura de 2 terrenos calcule el área y muestre el terreno que tiene más metros cuadrados.
- AR8. Modifique el algoritmo 7 de forma que verifique si un deudor al realizar su pago queda o no en deuda, considere que el pago jamás será mayor a la deuda.
- AR9. Modifique el algoritmo 8 de forma que calcule el salario, considere que por ley nadie puede ganar menos de 30 dólares, por lo que es necesario cumplir con lo dispuesto por las autoridades, por lo tanto si el salario es menor a 30, entonces debe tomar como sueldo 30 dólares.

### **ACTIVIDADES EXTRAS**

- A6. Conocer si al restar 2 números el resultado es positivo o negativo.
- A7. Conocer si un número tiene un dígito o es de más de un dígito.
- A8. Conocer si una persona es o no mayor de edad.
- A9. Conocer si una persona llega o no puntual a su trabajo, considerando que la hora de entrada es a las 8:00
- A10. Conocer si un número es o no par

### **AUTOEVALUACIÓN:**

- E3. Elabore un párrafo con sus palabras, que describa que actividades se pueden hacer utilizando condiciones mediante la computadora.
- E4. Proponga un ejercicio conforme a lo entendido que incluya condiciones y desarrolle su solución.

**OBJETIVO DE APRENDIZAJE PARA EL SIGUIENTE APARTADO:** Conocer y aplicar técnicas que permitan controlar procesos que se pueden repetir, evitando el exceso de escrituras similares.

## INTRODUCCIÓN A LOS PROCESOS REPETITIVOS

Cuando se estudia y se propone soluciones algorítmicas, notará que se demandará desarrollar soluciones a problemas, cuyas instrucciones de solución se repiten una y otra vez, a este proceso se lo conoce como *ciclos* o *bucles repetitivos*, por ejemplo, suponga que se le pide calcular el salario de 20 trabajadores que ganan por horas de trabajo al día, al desarrollar la solución algorítmica, se verá en la necesidad de repetir el mismo proceso 20 veces, es decir por cada trabajador se debe recibir el número de horas trabajadas, el valor que se le paga por hora, después se debe calcular el respectivo salario para posteriormente mostrarlo.

Para evitar la desproporción de escribir todas estas instrucciones algorítmicas, se ha desarrollado técnicas que utilizan los programadores para controlar y simplificar la redundante escritura de los mismos pasos.

### ¿Qué son los procesos repetitivos?

Como ya se ha indicado, en algunos casos para resolver un algoritmo, se necesitará repetir uno o varios procesos un determinado número de veces, a esto se le denomina *procesos repetitivos*; para lograr controlar este recurso, usted debe considerar, qué procesos necesitan repetirse y como vigilarlos para no exceder el número de repeticiones y provocar ciclos infinitos; para ilustrar de forma más detallada las oportunidades que ofrece esta técnica, considere el siguiente ejemplo algorítmico que pide los pasos para mostrar un nombre cinco veces, el siguiente algoritmo propone una solución sin utilizar las técnicas de control de procesos repetitivos, este ejemplo se lo utilizará como modelo para mostrar la eficiencia que ofrece el recurso de control de procesos repetitivos:

#### Algoritmo 9:

*Iniciamos*

1. Recibir el Nombre
2. Escribir el Nombre “por primera vez”
3. Escribir el Nombre “por segunda vez”
4. Escribir el Nombre “por tercera vez”
5. Escribir el Nombre “por cuarta vez”
6. Escribir el Nombre “por quinta vez”

*Finalizamos*

La solución propuesta de esta forma funciona, porque cumple con dar solución al problema planteado (Mostrar cinco veces un nombre), pero notará que existe un exceso de instrucciones que se repiten lo que lo hace poco estético y monótono de hacer, si se propone mostrar el mismo nombre 30 veces.

Considere las mismas interrogantes técnicas planteadas en las otras ejemplificaciones:

*¿Qué datos entran a ser procesados?*

El algoritmo solicita un nombre cualquiera

*¿Qué resultados muestra?*

Mostrará 5 mensajes repetitivos el nombre solicitado.

*¿Qué condiciones se presentan en el proceso?*

Ninguna.

Para que un proceso repetitivo se cumpla, necesitará por ahora incluir obligatoriamente una *condición o pregunta*, esta le permitirá controlar, las veces que se desee repetir un determinado proceso, así mismo necesitará tener claro donde iniciará y donde finalizará los procesos que desea repetir con la finalidad de ubicar las instrucciones de forma estratégica. Para ejemplificar estas nociones, el siguiente algoritmo, propone una solución al problema anterior (Mostrar cinco veces un nombre), aplicando el control de repetición, mediante el uso de una de muchas técnicas de control

de procesos repetitivos, esta técnica de control utiliza el conteo para definir el inicio y la finalización de las repeticiones, la propuesta algorítmica de solución al problema quedaría de la siguiente forma:

#### **Algoritmo 10:**

*Iniciamos*

1. Recibir el nombre
2. Comenzamos a *contar* y empezamos en **1**
3. Mostrar el "Nombre"
4. Como ya se mostró, *contamos* uno más
5. Preguntamos: sí al *contar*, el resultado es *MENOR O IGUAL QUE 5 Entonces*, repita desde el paso *número 3*, caso contrario finalizamos.

*Finalizamos*

Es importante considerar, que el control de la repetición, dependerá de la comparación o pregunta que usted proponga, para el ejemplo, ésta volverá a repetir desde el paso número 3, siempre y cuando, el conteo no llegue a ser mayor que cinco, de no cumplirse esta condición el algoritmo finalizará. Considere las siguientes interrogantes técnicas para detallar la solución algorítmica:

*¿Qué datos entran a ser procesados?*

El algoritmo solicita un nombre cualquiera

*¿Qué resultados muestra?*

El algoritmo mostrará cinco mensajes repetitivos, y cada uno de ellos con el nombre solicitado.

*¿Qué condiciones se presentan en el proceso?*

El algoritmo verifica si el conteo es menor o igual que cinco, para permitir o no la repetición desde el paso número 3.

Ahora, analizando y concluyendo con las dos últimas soluciones algorítmicas, observe que el primer algoritmo, propone mostrar el nombre en cinco líneas diferentes, que, hasta cierto punto cumple con el propósito planteado, pero sería preocupante aplicar esta lógica para hacer un algoritmo que muestre en 100 líneas el mismo resultado ¿Lo haría usted?; la segunda propuesta algorítmica, utiliza como técnica de control una condición y un proceso, básicamente consiste en realizar conteos, para controlar las veces que se desea repetir un determinado proceso, esta técnica permite que la instrucción que muestra al nombre (paso número tres), se repita cuantas veces el programador quiera, esto siempre y cuando el contador sea *menor o igual que* el número de veces que desea repetir el proceso; para definir la ventaja del segundo algoritmo sobre el primero, considere la misma suposición anterior ¿qué pasaría con la técnica de control, si se desea mostrar 100 veces el nombre y no solo 5?, aplicando esta técnica, solo se reemplazaría el valor de 5, por el valor de 100, específicamente, al momento de plantear la pregunta en el paso número cinco, y lo más notable, no se tendría que modificar las otras instrucciones dadas.

Considere ahora el siguiente ejemplo, se trata de un algoritmo que aplica el control, a los procesos repetitivos mediante el uso de contadores, la propuesta algorítmica muestra los pasos lógicos para escribir los números pares entre el número 1 y el número 100, para lograrlo, primero considere el siguiente análisis:

Los números pares son aquellos que empiezan en 2 y se incrementan de 2 en 2, por ejemplo para resolver el problema propuesto empezaría así: 2, 4, 6, 8, 10, 12, 14, 16, consecutivamente hasta el 100, es decir se empieza en 2 y se repite mientras el conteo de control sea menor o igual que 100, ahora para plasmar lo analizado examine la siguiente propuesta:

#### **Algoritmo 11:**

*Iniciamos*

1. Comenzamos a *contar* y empezamos en 2
2. Escribir/Mostrar el *conteo*
3. Como ya se mostró, le sumamos 2 más al *conteo*

4. *Preguntamos*: sí al sumar el conteo, el resultado es *MENOR O IGUAL QUE 100*, Entonces, repita desde el paso número 2, *Caso contrario* se finaliza.

*Finalizamos*

Como un factor importante a tomar en cuenta, observe que el proceso de conteo, no necesariamente se hace de uno en uno, sino que podrá variar a conveniencia de la solución propuesta.

*¿Qué datos entran a ser procesados?*

Ninguno, solo se controla el proceso repetitivo mediante un conteo que inicia con un valor de dos y se incrementará de dos en dos.

*¿Qué resultados muestra?*

Mostrará los números pares uno por uno, empezando en 2 y terminando en 100.

*¿Qué condiciones se presentan en el proceso?*

Utiliza un proceso de control, que permite realizar repeticiones consecutivas; aplica una condición que verifica si el conteo es menor o igual que 100 para repetir o no, desde el paso número dos.

Note que las palabras *contar* y *conteo* están destacadas en el algoritmo, esto se debe porque tienen el propósito de definir y quedar en claro que se refieren al mismo proceso de control, la lógica de repetición, empieza en el paso número dos y finaliza en el paso número cuatro, el *control de los procesos de repetición* inicia cuando se le asigna el valor inicial de dos al conteo; el proceso continúa cuando se muestra el conteo e incrementa su valor en dos; finaliza cuando evalúa la comparación, la misma definirá, si repite o no las instrucciones desde el paso número dos.

Para el siguiente ejercicio, se plantea un algoritmo que permita resolver, el cálculo de 10 áreas de terrenos rectangulares, para cada terreno se debe recibir, las medidas de largo y de ancho de cada uno de los 10 terrenos.

Análisis: considere que al ser terrenos rectangulares se aplica la fórmula ( $a = b \times h$ ), el algoritmo para resolver el problema, necesita calcular por cada terreno el área "a"; Por lo tanto, por cada repetición se necesitará recibir, los valores de base "b" y altura "h" o largo y ancho de cada terreno.

#### **Algoritmo 12:**

*Iniciamos*

1. Comenzamos a contar los terrenos y empezamos en 1
2. Pedimos los valores de ancho y largo del terreno
3. Calculamos el área multiplicando el largo por ancho del terreno
4. Mostramos el resultado de la multiplicación que equivale al área del terreno
5. Como ya lo calculamos, le sumamos 1 más al conteo de terrenos
6. *Preguntamos*: sí al sumar el conteo, el resultado es *MENOR O IGUAL QUE 10* Entonces repita desde el paso número 2, *Caso contrario* se finaliza

*Finalizamos*

Considere las siguientes preguntas técnicas, que permiten estandarizar el criterio lógico utilizado en el algoritmo:

*¿Qué datos entran a ser procesados?*

Por cada repetición se pide el largo y ancho de cada terreno

*¿Qué resultados muestra?*

Por cada repetición mostrará el área de cada terreno.

*¿Qué condiciones se presentan en el proceso?*

La condición incluida en el algoritmo controla el proceso de repetición, verifica si el conteo es menor o igual que 10, para repetir desde el paso número dos o finalizar.

El siguiente ejercicio desarrolla un algoritmo que calcula el vuelto o cambio de 43 socios que han consumido alimentos en un restaurante del club, para desarrollar el cálculo del cambio o vuelto, la cajera necesita un algoritmo, que por cada socio, tome el valor de su consumo y el dinero para pagar,

considere que el dinero del pago, siempre será mayor o igual que el total del consumo, por lo que la lógica del ejercicio, debe mostrar, por cada cliente el cambio a devolver por el pago de lo consumido.

Análisis: asumiendo que usted es uno de los socios y ha consumido 23 dólares en alimentos, al momento de pagar usted entrega 30 dólares en billetes, considerando que el cálculo es una resta ( $30 - 23$ ) tendría como resultado siete dólares de cambio, note que el pago, siempre será igual o mayor que el consumo, por lo que es recomendable restarle al valor del pago, el valor del consumo, para evitar que le dé un resultado negativo; la propuesta algorítmica quedaría de la siguiente forma:

### **Algoritmo 13:**

*Iniciamos*

1. Comenzamos a *contar* los clientes y empezamos en 1
2. Pedimos los valores de *consumo* y *pago* de dicho alimento
3. Calculamos el cambio *restando* el pago menos el consumo
4. Mostramos el *resultado de la resta*, que equivale al cambio a devolver
5. Como ya lo calculamos le sumamos 1 más al *conteo*
6. *Preguntamos*: si al sumar el conteo, el resultado es *MENOR O IGUAL QUE 43* Entonces repita desde el paso número 2, *Caso contrario* se finaliza

*Finalizamos*

Notará que, en todos los ejercicios desarrollados, el conteo equivale a controlar el número de veces que desea repetir un proceso, así se podrán contar terrenos, socios, empleados, ventas, entre muchos otros ejemplos; la codición que se presenta, siempre definirá si se repite o no, desde un determinado paso, y los conteos pueden darse a conveniencia del que propone una solución. Considerando el ejemplo anterior, observe las siguientes interrogantes técnicas que complementan los análisis de todo algoritmo:

*¿Qué datos entran a ser procesados?*

Por cada repetición se pide el valor total de lo consumido y el pago de dicho consumo

*¿Qué resultados muestra?*

Por cada repetición, mostrará el cambio o vuelto de cada socio.

*¿Qué condiciones se presentan en el proceso?*

La condición incluida en el algoritmo, controla el proceso de repetición, verificando si el conteo es menor o igual que 43, para repetir desde el paso número 2 o finalizar.

Es importante entender que en estos tipos de ejercicios planteados y resueltos en este apartado siempre incurren en lo mismo, se utiliza un conteo para controlar los procesos repetitivos y una condición que define la acción a tomar; esta técnica de control, propone mediante el uso de condiciones, la repetición desde un paso determinado o la finalización de la misma, esto quiere decir que necesariamente, usted utilizará condiciones o preguntas para controlar la secuencia de acciones que se realizarán en las diferentes soluciones algorítmicas que usted proponga.

Complementario a lo revisado en el apartado, es importante entender que existen otros tipos de ejercicios que incluyen controles de repetición y que necesariamente no utilizará el conteo como técnica de control, pero incluyen otros conocimientos que se tratarán poco a poco más adelante en el libro; para cumplir con las nuevas propuestas de control lógico, es necesario incluir conocimientos de estandarización en la narración; éstos se tratan en el siguiente apartado, por ahora se propone desarrollar ejercicios de fortalecimiento, en los que se necesita aplicar los temas tratados.

### **Actividades de refuerzo (AR):**

AR10. Modifique el algoritmo 11, de forma que muestre la lista de números, entre el 1 y el 100 pero solo los números múltiplos de 5.

AR11. Modifique el algoritmo 12, de forma que, calcule el área de cada uno de los 25 redondeles que existen en la ciudad.

AR12. Desarrolle un algoritmo que calcule el IVA (12%), de 15 productos que se consumen en un restaurante, para el cálculo se debe recibir solo el valor de cada producto.

### **AUTOEVALUACIÓN:**

- E5. Elabore un cuadro o una gráfica que explique cómo funciona la lógica en los procesos repetitivos.
- E6. Proponga un ejercicio conforme a lo entendido que incluya procesos repetitivos y desarrolle su solución algorítmica.

### **RESUMEN**

Las computadoras necesitan de los programas para realizar acciones de diversas utilidades, los programas son instrucciones que el sistema operativo convierte en órdenes para que la CPU obedezca una a una sin omitirlas; para que un programa pueda ejecutarse, sus instrucciones obligatoriamente deben contenerse en la memoria RAM, los programas son el producto de aplicar algoritmos que detallan la forma en que deben funcionar, los algoritmos son ideas ordenadas que permiten resolver los diferentes problemas mediante la aplicación de pasos ordenados y sucesivos, es decir plantea soluciones paso a paso, cuya calidad dependerá del pensamiento lógico del programador; el programador necesita conocer y aprender de forma inicial, como ordenar las secuencias de instrucciones, con la finalidad de desarrollar propuestas de solución; la redacción de los algoritmos naturales se desarrollan con instrucciones propias de la realidad de cada persona; se propone de forma inicial, aplicar una mecánica de tres pasos: recibir los datos que serán utilizados para encontrar una solución; procesarlos, es decir describir como se obtiene el resultado y por último mostrar el resultado esperado; además el proceso incluye el uso de condiciones, que básicamente consiste en aplicar comparaciones, solo se pueden aplicar seis comparaciones “Mayor que, Menor que, Mayor o Igual que, Menor o Igual que, Igual que y Diferente que” que sirven para crear una condición; las condiciones permiten dividir las acciones en dos alternativas bien definidas: verdadero o falso, cada una de ellas con acciones que se realizarán dependiendo de la respuesta; aplicando condiciones se puede controlar procesos repetitivos que indican desde donde repetir o simplemente no volver hacerlo, los procesos repetitivos reducen las instrucciones redundantes.

## CAPITULO II

### PSEUDOCÓDIGOS

Este capítulo propone el empleo de reglas y palabras, que denotan de forma simple, una acción precisa, en vez de las narraciones explicativas que se utilizaron en los algoritmos naturales del capítulo I; en este proceso, se aspira lograr la estandarización de la escritura algorítmica, al aplicar procedimientos y reglas específicas, en el desarrollo de propuestas de solución a problemas planteados; el uso de palabras pseudocodificadas (palabras claves que denotan acción y es común para todos los entendidos), minimizan el exceso de escritura que normalmente se aplica en la narración natural de los algoritmos, el pseudocódigo propone la utilización de instrucciones únicas y claras, que el programador conoce y es de común utilización con otros entendidos.

El uso de pseudocódigos, es una técnica que exige de los algoritmos, reglas muy similares a los utilizados por los lenguajes de programación, por ejemplo identifica de forma clara las instrucciones únicas y con acciones precisas, aplica límites a las instrucciones y a los controles de proceso en general, utiliza variables que representan datos de forma específica y pueden ser utilizados para reemplazar resultados en los diferentes procesos, aplica simbología especial para desarrollar cálculos, comparaciones y operaciones lógicas para grupos de comparaciones; en sí un algoritmo a partir de este capítulo solo tendrá instrucciones definidas en palabras únicas que denotan una acción precisa, variables que representan valores y datos, simbología especial para realizar cálculos, comparaciones y operaciones lógicas, en sí cualquier otro texto escrito en el algoritmo debe encerrarse entre comillas como mensajes, ya que estos textos pueden ser confundidos como nombres de variables o instrucciones pseudocodificadas.

Este capítulo expone los elementos necesarios, para definir variables y su utilidad en los diferentes algoritmos, explica y ejemplifica los símbolos utilizados para construir condiciones, hace uso del planteamiento de condiciones simples en los algoritmos así como las condiciones múltiples en la resolución de problemas, instruye el uso y el control de los diferentes ciclos repetitivos, y aplica las validaciones con la finalidad de obtener datos confiables.

**OBJETIVO DE APRENDIZAJE DEL SIGUIENTE APARTADO:** Conocer en que consiste la estandarización en la redacción de los algoritmos así como aplicar reglas para crear y gestionar variables.

### PREÁMBULO DEL PSEUDOCODIGO EN LOS ALGORITMOS

Cuando se presentan problemas que tienen mayor dificultad, notará que los algoritmos escritos de forma natural se vuelven más extensos y difíciles de entender, esto es un problema ya que existe la necesidad de proponer una solución, que represente un medio fácil de entender y de seguir al aplicar las diversas acciones presentadas en los algoritmos, considere que al momento de redactar soluciones más extensas, se presenta la posibilidad de no ser entendible, esto se hace evidente, ya que al expresar una idea que dé una solución, la terminología que se esté utilizando no sea la misma que domine el lector que desea aplicar la solución, así la exposición de ideas que denoten una solución comprensible para todos, sería muy complicado aplicarla de forma natural, recuerde que este tipo de redacción no se puede dar por la diversidad de pensamientos que existe entre las diferentes personas, es decir “lo que está claro para usted, para otra persona no lo estará”.

Ante esta dificultad, los promotores del pensamiento lógico aplicados a la programación realizaron acuerdos que permitieren estandarizar la narrativa mediante el uso de reglas y palabras únicas denominada redacción “pseudocodificadas”, esta técnica permite expresar una acción sin la necesidad de redactar la idea o la acción de forma natural, pseudocódigo es una palabra compuesta que está dividida en *pseudo* que significa *supuesto* o común para todos, y la palabra *código* cuyo significado expresa normativas o aplicación de reglas, en sí las palabras pseudocodificadas son palabras que suponen una acción y están sometidas a reglas de redacción.

Como propuesta pedagógica de aprendizaje, el libro propone aplicar de forma superficial las reglas estandarizadas de escritura pseudocodificada, para esto considere las siguientes reglas de redacción:

- Identificar y utilizar palabras claves que identificarían una determinada acción, por ejemplo: la palabra “*Recibir*”, identificaría la acción de solicitar datos que se necesitan para procesarlos, para mostrar resultados de cálculos, mensajes, o cualquier dato o información se puede utilizar las palabras “*Mostrar o Escribir*”.
- los cálculos propuestos deben realizarse directamente utilizando valores constantes y/o *variables* sin la necesidad de redactarlos o explicarlos mediante la redacción natural.
- Todas las instrucciones que de detallan en la redacción de algoritmo deben terminar en punto y coma (;)
- Cuando las instrucciones tienen varios datos separados se pueden unir o concatenar en una sola acción mediante el uso de la coma (,).
- Se puede aplicar controles de procesos mediante el uso de instrucciones o grupo de palabras únicas que no se deben utilizar como nombres de variables, por ejemplo: el uso coordinado de las palabras “*Sí, entonces, caso contrario*”, permitiría aplicar en los algoritmos las condiciones lógicas de comparación, los procesos de control en ciclos repetitivos puede aplicar técnicas como: “*Ir a*”, “*repetir ... hasta que*”, “*Mientras ... hacer*”, “*Para ... hacer*”, “*Hacer ... mientras*”, cada uno de éstos controles se explicarán uno a uno en éste capítulo.

Es importante destacar que los algoritmos propuestos en este capítulo no definen la redacción formal de un lenguaje de programación en cuanto a la definición de encabezados, secciones, declaraciones de variables y uso de comentarios, ya que esta propuesta bibliográfica pretende desarrollar de forma inicial el pensamiento y mecánica lógica en la redacción de algoritmos básicos que definan las pautas esenciales en la creación de programas aplicados a la computadora.

## VARIABLES

En su forma más básica, una variable es un nombre que representa un dato, este nombre es utilizado en el programa como sinónimo de dicho valor y podrá utilizar el nombre como si se tratara del mismo dato, un ejemplo de nombres de variables podría ser: una nota (N, nota, n1, etc.), una edad (e, ed, edad, etc.), el resultado de una operación aritmética (a, area, salario, suma, resta, etc.), entre muchas otras posibilidades, observe que el término *variable* tiene su significado en que, el nombre utilizado para representar un dato puede cambiar de valor a conveniencia de los procesos que se realizan en un programa, para ilustrar la definición de una variable, considere el ejemplo del primer algoritmo que consistía en sumar dos números, se podría indicar que el primer número tiene como nombre “A” o “numero1” o “N1” o “Num1” o cualquier nombre, considerando que para el programador representa dicho valor; así se puede utilizar cualquier nombre para representar cualquier valor o dato que se necesite para desarrollar un determinado proceso cualquiera.

Es importante destacar que, en un programa no puede existir dos variables con el mismo nombre, los nombres de variable no pueden contener como parte del nombre el espacio es decir “Horas trabajadas” no se puede considerar como nombre de variable porque contiene el espacio, lo correcto sería “HorasTrabajadas”, tampoco está permitido que los nombres de variables empiecen con un número o un símbolo o que incluya en cualquier parte del nombre símbolos no permitidos, por ejemplo “1A”, “mes&dia”, lo correcto sería “A1” y “mesydia”.

Antes de realizar varios ejemplos, es necesario explicar que es una *asignación de datos o asignación de valores*; básicamente consiste en conocer que las variables pueden intercambiar valores y/o textos, por consiguiente también pueden recibir valores como el resultado de un cálculo, para lograr conseguir este propósito es necesario utilizar el signo igual (=) y cumplir con la siguiente regla de asignación:

*Destino = Origen;*

Esto significa que la variable destino siempre se ubicará a la izquierda del igual (=), y el contenido a pasar siempre se ubicará a la derecha del signo igual (=), así al tener una variable llamada “*Sueldo*” y desear pasarle un valor de 340, lo correcto sería: *Sueldo = 340;*



De igual forma el origen puede ser el resultado de un cálculo cualquiera, para ilustrar esta definición asuma que la variable “*Sueldo*” tomará un valor que consiste en el resultado de calcular 40 horas de trabajo con un pago de 8 dólares la hora, así tendríamos:  $Sueldo = 40 * 8$ ;

Las asignaciones y las variables están íntimamente ligadas entre sí, esto se debe a que los valores, resultados de cálculos, entre otros, deben estar contenidas o almacenadas en variables y la forma de transferir datos o valores de un lugar a otro es mediante la asignación, para ilustrar el uso de variables y de las asignaciones, considere los siguiente tres algoritmos, el primero se encarga de sumar dos número cualquiera, el segundo calcula el promedio de tres notas y el tercero calcula el área de un rectángulo, estos algoritmos ya se realizaron anteriormente sin el uso de variables:

#### **ALGORITMO 14**

Secuencia lógica que calcula la suma de 2 números:

*Iniciamos*

1. Recibir N1 , N2;
2. Suma=N1 + N2;
3. Escribir Suma;

*Finalizamos*

*EXPLICACIÓN:* N1 representa uno de los dos números, N2 representa el segundo número, Suma representa el valor que como resultado de la operación de la suma contiene el resultado de sumar N1 y N2, considere que el tercer paso se especifica el resultado a entregar mediante la variable Suma

#### **ALGORITMO 15**

Secuencia lógica que calcula el promedio de 3 notas:

*Iniciamos*

1. Recibir Nota1, Nota2, Nota3;
2. Promedio=(Nota1+Nota2+Nota3)/3;
3. Escribir Promedio;

*Finalizamos*

*EXPLICACIÓN:* Las variables Nota1, Nota2 y Nota3 representan las 3 notas involucradas en el cálculo del promedio, la variable Promedio contendrá el resultado de ésta operación aritmética, y será utilizada para mostrar el resultado en el tercer paso

#### **ALGORITMO 16**

Este algoritmo muestra como calcula el área de un rectángulo

*Iniciamos*

1. Recibir Base, Altura;
2. Area= Base \* Altura;
3. Escribir Area;

*Finalizamos*

*EXPLICACIÓN:* Las variables Base y Altura representan los valores utilizados en la formula ( $a=b \times h$ ) para calcular el área de un rectángulo, la variable Area contendrá el resultado de ésta operación aritmética, y será utilizada para mostrar el resultado en el tercer paso

#### **Actividades de refuerzo (AR):**

AR13. Desarrolle un algoritmo que permita calcular el pago de un electricista que trabaja por horas, considere que por ser feriado al valor de la hora se le debe incrementar el 50%.

AR14. Se necesita un algoritmo que transforme la distancia de kilómetros a metros en una ruta entre dos poblaciones, el algoritmo debe Recibir el dato en kilómetros y mostrar su equivalencia en metros.

AR15. Desarrolle un algoritmo que calcule el interés simple del pago de una tarjeta de crédito por un adelanto en efectivo.

## ACTIVIDADES EXTRAS:

- A11. Como calcular el 10% de una cantidad
- A12. Como calcular el perímetro de un triángulo
- A13. Como calcular la longitud de una circunferencia.
- A14. Como calcular el número de diagonales que puede tener un polígono
- A15. Como calcular el Interés simple de un préstamo a pagar en un año.

**OBJETIVO DE APRENDIZAJE EN ESTE BLOQUE:** Conocer cómo se crean las condiciones, su estructura, simbología, resultados y control de acciones como respuesta a cada posibilidad lógica, así como desarrollar soluciones más elaboradas que permitan integrar condiciones para determinadas acciones.

## OPERADORES DE COMPARACIÓN Y LÓGICOS

Estos operadores permiten el control de las condiciones aplicadas a la toma de decisiones, son símbolos que permiten al programador realizar procesos de cálculo lógicos aplicados a las comparaciones que solo pueden tener uno de dos resultados (Verdadero o Falso), por su naturaleza práctica se clasifican en operadores de comparación y operadores lógicos de unión de comparaciones, parte de la familia de los operadores son los aritméticos que se estuvieron tratando en los apartados anteriores, las dos clasificaciones restantes se detallan a continuación:

### OPERADORES DE COMPARACIÓN

Estos símbolos especiales son utilizados para exponer una pregunta, las preguntas se pueden realizar a valores constantes como "10, 5.2, -18, o cualquier otro valor", así también a resultados de un cálculo, a contenidos de variables o a la combinación de éstos, considere la siguiente tabla que muestra los símbolos a utilizar:

**TABLA 2.** Símbolos utilizados como operadores de comparación

Operador de comparación	SIGNIFICADO
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>	Diferente que
=	Igual que

Es importante destacar que estos operadores de comparación devuelven un resultado por pregunta o condición y podrá estar entre *verdadero* o *falso*.

**Fuente:** Autores del libro

En apartados anteriores se han tratado las comparaciones expuestas de forma narrada, a partir de ahora las comparaciones utilizarán estos símbolos para realizar condiciones, aquí algunos ejemplos de comparaciones con valores constantes, observe cómo se utilizan estos operadores de comparación y el resultado que arrojan:

**Tabla 3.** Ejercicios de comparación y resultados obtenidos

<b>29 &gt; 38</b>	<b>Esto devuelve como resultado <i>falso</i>, ya que 29 no es mayor que 38.</b>
<b>"C" &lt; "H"</b>	Al comparar estas dos letras, devuelve como resultado <b>verdadero</b> , ya que las letras se ordenan de forma ascendente, así la letra A tiene un valor inferior a Z que tiene un valor máximo entre todas las letras del alfabeto.
Sí A tiene 1, y B tiene -12 <b>A &lt; B</b>	Al comparar las variables A y B, da como resultado <b>falso</b> ya que B tiene un número negativo.
Sí X tiene 100 y K tiene -10 <b>X+K&gt;=90</b>	Al realizar la operación $X(100)+K(-10)>=90$ la respuesta es <b>verdadero</b> ya que al sumar los valores da 90.

**Fuente:** Autores del libro

Es importante entender que el uso de los operadores de comparación provoca matemáticamente un cálculo binario (ceros y unos), que dará como resultado el valor de 1 (uno) que significa verdadero o el valor de 0 (cero) que significa falso.

## OPERADORES LÓGICOS

Estos operadores permiten unir varias comparaciones en una sola condición, en sí devuelve una sola respuesta entre “Falso o Verdadero”, el uso de esta herramienta se basa en la aplicación lógica de cómo unir varias comparaciones en una sola condición, para entender el propósito de cómo funciona la unión de comparaciones considere las siguientes tablas de la verdad “AND (y), OR (o)” y un inversor de respuesta “NOT (no)”.

### OPERADOR AND

Este operador exige que *todas* las preguntas sean verdaderas para que su respuesta sea verdadera, por ejemplo: suponga que desea cobrar un *cheque*, las dos condiciones básicas que le permitiría cobrar el cheque sería: tener el cheque (y) tener la identificación del cobrador, si usted tiene el cheque y no su identificación no podrá cobrarlo (En término lógico el resultado sería falso) ya que no cumple con las dos condiciones básicas, bajo esta premisa se plantea la siguiente tabla asumiendo que se tienen dos comparaciones:

**Tabla 4.** Cuadro de la verdad utilizando el operador lógico AND  
Comparación 1 AND Comparación 2

Resultado de la comparación 1	Resultado de la comparación 2	Resultado definitivo
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso
Falso	Verdadero	Falso
Falso	Falso	Falso

**Fuente:** Autores del libro

Matemáticamente el uso de un operador lógico AND es definida como una multiplicación, por ejemplo considere que el resultado de la comparación 1 es verdadero (1) y de la comparación 2 es falso (0), matemáticamente es  $1*0$  el resultado es (0) que significa falso.

### OPERADOR OR

El uso de este operador exige que al menos *una* de las comparaciones sea verdadera para que su respuesta sea verdadera, por ejemplo: suponga que una persona presenta temperatura alta de fiebre, para bajarla tiene dos alternativas, bañarse o tratarla con medicamento, es decir que cualquiera de las dos opciones le permitirá estabilizar la temperatura, bajo esta premisa se plantea la siguiente tabla asumiendo que se tienen dos comparaciones:

**Tabla 5.** Cuadro de la verdad utilizando el operador lógico OR  
Comparación 1 OR Comparación 2

Resultado de la comparación 1	Resultado de la comparación 2	Resultado definitivo
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Falso	Verdadero	Verdadero
Falso	Falso	Falso

**Fuente:** Autores del libro

Al observar la tabla, notará que al menos una de las comparaciones debe tener como resultado verdadero para que la respuesta lógica de las dos comparaciones sea verdadera.

Matemáticamente el uso de un operador lógico OR es definida como una suma, por ejemplo considere que el resultado de la comparación 1 es verdadero (1) y de la comparación 2 es falso (0), matemáticamente es  $1+0$  el resultado es (1) que significa verdadero.

### OPERADOR INVERSOR NOT

Este operador invierte el resultado de una condición, es decir, si como respuesta es *verdadero*, éste operador lo convierte en *falso* y viceversa, bajo esta premisa se plantea la siguiente tabla:

**Tabla 6.** Cuadro de la verdad utilizando el operador lógico NOT  
NOT Condición

Resultado de la condición	Al aplicar Not el resultado definitivo es
Verdadero	Falso
Falso	Verdadero

**Fuente:** Autores del libro

### Ejercicios con expresiones lógicas

Los siguientes ejercicios permitirán identificar los resultados al usar los operadores descritos anteriormente, asumiendo que se cuentan con las siguientes variables y sus respectivos valores:  $W = 20$   $X = 17$   $Y = 25$   $Z = 10$

EXPRESIÓN	RESPUESTA
$(X > Z) \text{ And } (W < Y)$	
$\begin{array}{cc} 17 & 10 \\ & \swarrow \searrow \\ & V \end{array}$ $\begin{array}{cc} 20 & 25 \\ & \swarrow \searrow \\ & V \end{array}$	Verdadero
$(X < W) \text{ Or } (Z > Y)$	
$\begin{array}{cc} 17 & 20 \\ & \swarrow \searrow \\ & V \end{array}$ $\begin{array}{cc} 10 & 25 \\ & \swarrow \searrow \\ & F \end{array}$	Verdadero
$\text{Not } (X < W) \text{ Or } (Z > Y)$	
$\begin{array}{cc} & 17 & 20 \\ & \swarrow \searrow \\ & V \end{array}$ $\begin{array}{cc} 10 & 25 \\ & \swarrow \searrow \\ & F \end{array}$	Falso
	F

### Actividades de refuerzo (AR):

AR16. Asumiendo que se tienen las siguientes variables  $A = 53$   $B = 44$   $C = 37$   $D = 71$  desarrolle la siguiente propuesta:  $((A > D) \text{ And } (C < B)) \text{ Or } (\text{Not}(A = C) \text{ AND } (D > B))$ .

AR17. Asumiendo que se tienen los siguientes valores  $A = 11$   $B = 16$   $C = 27$   $D = 11$  desarrolle la siguiente propuesta:  $(\text{Not}(A = D) \text{ And } (C < B) \text{ And } (D < C)) \text{ Or } \text{Not}((A = C) \text{ AND } (C > A))$ .

**OBJETIVO DE APRENDIZAJE PARA EL SIGUIENTE APARTADO:** Afirmar los Conocimientos de aplicar preguntas o condiciones simples y múltiples en la solución de problemas utilizando la simbología apropiada incluyendo reglas y posibles acciones de respuesta en herramientas como las condiciones de caso.

### USO DE CONDICIONES SIMPLES UTILIZANDO PSEUDOCÓDIGO

Una condición simple se aplica cuando se incluye en un algoritmo solo una condición o una comparación en el desarrollo de una solución a un problema determinado, al aplicar esta condición, se define la acción o acciones a tomar entre dos alternativas posibles, estas alternativas se producen como respuesta a una comparación y estará dada como las acciones a tomar en caso de ser verdadero o acciones a tomar en caso de ser falso, recuerde que en un algoritmo al aplicar cualquier comparación, ésta solo incluye los símbolos estudiados como  $(>)$  mayor qué,  $(<)$  menor qué,  $(\geq)$  mayor o igual qué,  $(\leq)$  menor o igual qué,  $(=)$  igual qué y  $(<>)$  diferente qué.

Para ilustrar el uso de condiciones simples, se aplicarán ejemplos con algoritmos que incluyen preguntas en la lógica, el siguiente algoritmo compara dos números cualquiera y muestra como resultado solo el número mayor:

#### Algoritmo 17:

*Iniciamos*

1. Recibir Num1 , Num2;
2. Sí Num1 > Num2 Entonces Escribir Num1;  
Caso contrario Escribir Num2;

*Finalizamos*

Observe que la instrucción compuesta por “Sí”, “Entonces” y “Caso contrario” utilizada para realizar comparaciones, se cierra automáticamente con el uso del punto y coma de la respuesta por verdadero o por la respuesta del falso (caso contrario).

*¿Qué datos entran a ser procesados?*

El algoritmo solicita dos números cualquiera, uno es contenido en la variable Num1 y el otro es el contenido en la variable Num2

*¿Qué resultados muestra?*

Mostrará el mayor de los dos números.

*¿Qué condiciones se presentan en el proceso?*

Verifica si el valor de la primera variable Num1 es mayor que el valor de la segunda variable Num2.

La aplicación de pseudocódigo disminuye la redacción y simplifica su comprensión el ejemplo utiliza la palabra *Entonces* para indicar las acciones que se realizan en caso de ser verdadero y después de las palabras *Caso contrario* describe las acciones a tomar en caso de que el resultado de la pregunta sea falsa.

El siguiente ejemplo aplica un algoritmo, que teniendo un valor cualquiera, verifica si es un número positivo o un número negativo, considere que un número es positivo si su valor es mayor que cero, a su vez es negativo cuando su valor es menor que cero.

#### **Algoritmo 18:**

*Iniciamos*

1. Recibir N;
2. Sí N >= 0 Entonces Escribir “Número Positivo”;  
Caso contrario Escribir “Número Negativo”;

*Finalizamos*

*¿Qué datos entran a ser procesados?*

El algoritmo solicita un número cualquiera y es contenido en la variable N

*¿Qué resultados muestra?*

Mostrará un mensaje entre “Numero Positivo” o “Número Negativo”.

*¿Qué condiciones se presentan en el proceso?*

Verifica si el valor de la variable Num1 es mayor que 0 para mostrar el mensaje respectivo.

El siguiente algoritmo muestra como calcular el salario de un jornalero que trabaja por horas, además si ha trabajado más de 8 horas se le dará 10 dólares adicionales como recompensa a su trabajo extra:

#### **Algoritmo 19:**

*Iniciamos*

1. Recibir NHT , VH;
2. *Salario* = NHT \* VH;
3. Sí NHT > 8 Entonces *Salario* = *Salario* + 10;
4. Mostrar *Salario*;

*Finalizamos*

Observe que cuando el número de horas trabajadas (NHT) cumple con la condición, se produce un cálculo especial que se explicará con mayor detalle más adelante en éste capítulo, fíjese que, al valor que representa la variable *Salario* se le suma 10 y el resultado es contenido en la misma variable *Salario*.

*¿Qué datos entran a ser procesados?*

El algoritmo recibe dos valores, uno representa el número de horas trabajadas NHT y el otro representa el valor de la hora VH

*¿Qué resultados muestra?*

Mostrará el *salario* con o sin el valor agregado por trabajar más de 8 horas.

*¿Qué condiciones se presentan en el proceso?*

Verifica si el valor de la variable NHT es mayor que ocho para agregar al salario 10 dólares más como recompensa.

Considere que en éste ejemplo no se utiliza el *caso contrario*, lo que significa que el valor de recompensa se agregaría solo cuando el número de horas trabajadas (NHT) cumpla con la condición ( $NHT > 8$ ), el último paso, sin importar que cumpla o no con la condición mostrará el salario calculado.

### USO DE CONDICIONES MÚLTIPLES UTILIZANDO PSEUDOCÓDICO

En algunos problemas se encontrará con la necesidad de utilizar múltiples condiciones, para esto usted puede hacer uso de las comparaciones simples pero en grupo y/o de forma consecutiva, por ejemplo, se plantea la necesidad de un algoritmo que permita registrar una cantidad positiva menor a 10000, el algoritmo antes de finalizar debe mostrar el número de dígitos que posee.

Como un análisis simple, se podría indicar:

- ✓ Los números de un dígito son los que comprende de 0 a 9
- ✓ Las cantidades de dos dígitos son los que comprende desde 10 a 99
- ✓ Las cantidades de tres dígitos son los que comprende desde 100 a 999
- ✓ Las cantidades de cuatro dígitos son los que comprende desde 1000 a 9999

Sí aplicamos condiciones múltiples en la resolución de este problema tendríamos:

#### Algoritmo 20:

*Iniciamos*

1. Recibir *Cant*;
2. Sí *Cant* > 9999 Entonces Escribir "Cantidad tiene más de 4 dígitos";
3. Caso contrario Sí *Cant* > 999 Entonces Escribir "Cantidad tiene 4 dígitos";
4. Caso contrario Sí *Cant* > 99 Entonces Escribir "Cantidad tiene 3 dígitos";
5. Caso contrario Sí *Cant* > 9 Entonces Escribir "Cantidad tiene 2 dígitos";
6. Caso contrario Escribir "Número tiene 1 dígito";

*Finalizamos*

*¿Qué datos entran a ser procesados?*

El algoritmo solicita una cantidad cualquiera contenida en la variable *Cant*

*¿Qué resultados muestra?*

Mostrará el número de dígitos que tiene la cantidad.

*¿Qué condiciones se presentan en el proceso?*

Se presentan tres condiciones o preguntas, la primera condición verifica si la cantidad es mayor que 999, según el límite establecido en el problema, significa que la cantidad tendría cuatro dígitos, en caso contrario se aplica la segunda condición, que verifica si la cantidad es mayor que 99, en cuyo caso la cantidad tendría tres dígitos, entienda que para que se verifique la segunda condición la primera debe ser falsa, la tercera condición se aplica si la segunda es falsa, verifica si la cantidad es mayor que 9, si es verdadero mostrará que la cantidad tiene dos dígitos, caso contrario mostrará que la cantidad tiene un dígito.

En este ejemplo existen cuatro alternativas de acción que mostrará el número de dígitos, pero solo una de las respuesta por verdadero se ejecutará cuando se cumpla la condición en cualquiera de las

preguntas; analizando la primera pregunta, si la respuesta es verdadero no se evaluarán las dos preguntas restantes ya que pertenecen al caso contrario o falso de la primera pregunta, esta lógica es mecánica y se aplica así sucesivamente para las otras preguntas.

Observe el siguiente ejemplo, el algoritmo pide dos números, con ellos ofrece la posibilidad de realizar una de las siguientes operaciones: 1 suma, 2 resta, 3 multiplicación y 4 división.

#### Algoritmo 21:

*Iniciamos*

1. Recibir *num1, num2*;
2. Mostrar “Escriba el número de opción: (1) suma, (2) resta, (3) multiplicación, (4) división”;
3. Recibir *Opción*;
4. Sí *Opción = 1 Entonces* *Result=num1+num2*;
5. Caso contrario Sí *Opción = 2 Entonces* *Result=num1-num2*;
6. Caso contrario Sí *Opción = 3 Entonces* *Result=num1\*num2*;
7. Caso contrario Sí *Opción = 4 Entonces* *Result=num1/num2*;
8. Caso contrario *Result=0*;
9. Mostrar “El Resultado es:”, *Result*;

*Finalizamos*

*¿Qué datos entran a ser procesados?*

El algoritmo inicialmente solicita dos cantidades cualquiera, después solicita escribir el número de opción para realizar el cálculo.

*¿Qué resultados muestra?*

Mostrará el resultado de uno de los cuatro posibles cálculos.

*¿Qué condiciones se presentan en el proceso?*

Presenta cuatro condiciones o preguntas, la primera condición verifica si el cálculo deseado es la suma (*Opción=1*), por el caso contrario se aplica la segunda condición o pregunta, verifica si el cálculo deseado es la resta (*Opción=2*), si no es así, el caso contrario de esta pregunta es para aplicar la tercera condición, verifica si el cálculo deseado es la multiplicación (*Opción=3*), al no ser verdadero, el caso contrario de ésta aplica la cuarta y última condición, verifica si el cálculo deseado es la división (*Opción=4*), al no ser verdadero ninguna de las alternativas, aplica el último caso contrario y define el resultado como cero.

*Considere el siguiente Problema:* Una empresa textil por temporada contrata empleados ocasionales, éstos empleados se clasifican en cinco categorías de responsabilidades de la cual dependerá su pago, cada responsabilidad categorizada tiene un valor diferente de pago por cada hora de trabajo, observe la descripción que se muestra en el cuadro:

Se necesita un algoritmo que permita pagar por sus horas de trabajo a un empleado de la empresa textil, para esto se deberá ingresar el código de la categoría y el número de horas trabajadas, la lógica deberá mostrar el valor a cancelar al empleado.

CATEGORÍA	VALOR HORA TRABAJO
1	\$ 12.78
2	\$ 10.25
3	\$ 8.78
4	\$ 6.00
5	\$ 4.80

#### Algoritmo 22:

*Iniciamos*

1. Recibir *Categoría, Horas\_Trab*;
2. Sí *Categoría = 1 Entonces* *Sueldo= Horas\_Trab \* 12.78* ;
3. Caso contrario Sí *Categoría = 2 Entonces* *Sueldo= Horas\_Trab \* 10.25*;
4. Caso contrario Sí *Categoría = 3 Entonces* *Sueldo= Horas\_Trab \* 8.78*;



5. Caso contrario Si *Categoría* = 4 Entonces Sueldo= Horas\_Trab \* 6.00;
  6. Caso contrario Sueldo= Horas\_Trab \* 4.80;
  7. Mostrar "El sueldo es:", *Sueldo*;
- Finalizamos*

*¿Qué datos entran a ser procesados?*

El algoritmo solicita dos valores, el primer valor representa la *categoría* y el otro representa las horas trabajadas *Horas\_Trab*.

*¿Qué resultados muestra?*

Mostrará el sueldo.

*¿Qué condiciones se presentan en el proceso?*

Presenta cuatro condiciones o preguntas, la primera condición verifica si la categoría es la primera (*Categoría*=1) que significa que el valor de la hora es \$12.78, caso contrario se aplica la segunda condición o pregunta, verifica si la categoría es la segunda (*Categoría*=2) lo que significa que el valor de la hora es \$10.25, en caso de que la condición no se cumpla, aplica el caso contrario para chequear la tercera condición, ésta verifica si la categoría es la tercera (*Categoría*=3) lo que significa que el valor de la hora es \$8.78, en el caso contrario de ésta, aplica la cuarta y última condición que verifica si la categoría escogida es la cuarta (*Categoría*=4) lo que significa que el valor de la hora es \$6.00, por el caso contrario a esta última condición asumirá que la categoría es la quinta sin necesidad de preguntarla, que significa que el valor de la hora es \$4.80.

### **Actividades de refuerzo (AR):**

- AR18. Modifique el algoritmo 20 de forma que muestre el número de dígitos de una cantidad que se encuentra limitada entre 1 y 1'000.000.
- AR19. Por seguridad los clientes de varios negocios transfieren por la red un código compuesto en dos partes el número de día en la semana y la hora, que identifica la transferencia de dinero de la compañía de seguridad, elabore un algoritmo que identifique el día en letras considerando que el primer día de la semana es domingo (1) y el último día es sábado (7).
- AR20. Desarrolle un algoritmo que transforme de un dígito decimal a un número romano.

### **USO DE CONDICIONES DE CASO UTILIZANDO PSEUDOCÓDIGO**

Las condiciones de caso, es una técnica empleada para realizar comparaciones de igualdad con varios posibles casos, su estructura condicional está desarrollada para realizar varias acciones por cada caso de forma independiente a cada posible caso, esta técnica emplea condiciones de igualdad al comparar el contenido de una variable con una lista de opciones, su aplicación entre otras posibilidades reduce y facilita la escritura al reemplazar algunas posibilidades de condiciones múltiples, la herramienta empieza con la instrucción "Según" y finaliza con "Fin Según", cada "Caso" puede incorporar una opción o una lista de opciones separados por comas (,) a continuación se escribe dos puntos (:) y seguido se escriben las instrucciones que desee, al coincidir con un caso se realizarán las instrucciones expuestas en éste y no realizará las instrucciones de los otros casos.

Para ilustrar ésta definición considere el siguiente ejemplo, se trata de un algoritmo que permite escoger un cálculo entre cuatro posibles opciones, el algoritmo a ejemplifica es el mismo utilizado en la explicación de las condiciones múltiples, pide dos números, con ellos ofrece la posibilidad de realizar una de las siguientes operaciones: 1 suma, 2 resta, 3 multiplicación y 4 división.

#### **Algoritmo 23:**

*Iniciamos*

1. Recibir *num1*, *num2*;
2. Mostrar "Escriba el número de opción: (1) suma, (2) resta, (3) multiplicación, (4) división";
3. Recibir *Opción*;
4. Según *Opción* Hacer

- 4.1. Caso 1: Result=num1+num2;
  - 4.2. Caso 2: Result=num1-num2;
  - 4.3. Caso 3: Result=num1\*num2;
  - 4.4. Caso 4: Result=num1/num2;
  - 4.5. Caso contrario Result=0;
  - 4.6. Fin según;
  5. Mostrar "El Resultado es:", Result;
- Finalizamos*

Observe que la instrucción de control "Según... hacer" se cierra con la instrucción "Fin según".

*¿Qué datos entran a ser procesados?*

El algoritmo inicialmente solicita dos cantidades cualquiera, después solicita recibir el número de la opción para resolver el cálculo deseado.

*¿Qué resultados muestra?*

Mostrará el resultado de una de los cuatro posibles cálculos.

*¿Qué condiciones se presentan en el proceso?*

Presenta una sola condición y cuatro preguntas, dependiendo del valor que contiene la variable *Opción* lo compara con cada caso (chequeando si es igual), *si el caso es uno* realiza la operación y la variable *Result* tomará la suma de los dos números, al no ser el primer caso, compara con el segundo caso, verifica *si el valor es dos* realizará la operación y la variable *Result* tomará la resta de los dos números, al no ser el segundo caso, compara con el tercer caso, verifica *si el valor es tres* después realizará la operación y la variable *Result* tomará la multiplicación de los dos números, al no ser el tercer caso, compara en el cuarto caso, verifica *si el valor es cuatro*, acción seguida realizará la operación y la variable *Result* tomará la división de los dos números, por último si ningún caso se cumple tomará el caso contrario y el valor de la variable *Result* será cero.

El siguiente ejemplo modifica el algoritmo 22 que calcula el promedio de un empleado que trabaja en una textilera cuyo sueldo depende de la categoría de responsabilidad asignada, aplicamos la técnica de condiciones de caso tendríamos el siguiente resultado:

#### **Algoritmo 24:**

*Iniciamos*

1. Recibir *Categoría*, *Horas\_Trab*;
2. Según Categoría Hacer
  - 2.1. Caso 1: Sueldo= Horas\_Trab \* 12.78;
  - 2.2. Caso 2: Sueldo= Horas\_Trab \* 10.25;
  - 2.3. Caso 3: Sueldo= Horas\_Trab \* 8.78;
  - 2.4. Caso 4: Sueldo= Horas\_Trab \* 6.00;
  - 2.5. Caso contrario Sueldo= Horas\_Trab \* 4.80;
  - 2.6. Fin según;
3. Mostrar "El sueldo es:", *Sueldo*;

*Finalizamos*

*¿Qué datos entran a ser procesados?*

El algoritmo solicita dos datos, uno representa la *categoría* y el otro representa las horas trabajadas *Horas\_Trab*.

*¿Qué resultados muestra?*

Mostrará el sueldo.

*¿Qué condiciones se presentan en el proceso?*

Presenta una condición y cuatro comparaciones, la condición verifica el valor de la categoría, funciona verificando si coincide con el primer caso es decir chequea si cumple la pregunta implícita (*Categoría=1*) lo que significa que el valor de la hora es \$12.78 y lo multiplica por el valor de la variable *Horas\_Trab*, si no se cumple el caso, verifica el segundo caso o comparación, chequea si la categoría

es la segunda (*Categoría=2*) que significa que el valor de la hora es \$10.25 y lo multiplica por el valor de la variable *Horas\_Trab*, si no se cumple este caso verifica si es el tercer caso (*Categoría=3*) que significa que el valor de la hora es \$8.78 y lo multiplica por el valor de la variable *Horas\_Trab*, si no se este caso verifica el cuarto chequeo, si la categoría es la cuarta (*Categoría=4*) que significa que el valor de la hora es \$6.00 y lo multiplica por el valor de la variable *Horas\_Trab*, por el caso contrario asumirá que la categoría es la quinta sin necesidad de verificar el caso, que significa que el valor de la hora es \$4.80 y lo multiplica por el valor de la variable *Horas\_Trab*.

*Considere el siguiente problema:* Desarrolle un algoritmo que muestre en mensaje el número de días que posee un mes cualquiera, para esto se debe recibir el número correspondiente del mes.

Análisis: los meses que tienen 31 días son: enero (1), marzo (3), mayo (5), julio (7), agosto (8), octubre (10) y diciembre (12); y los meses que tienen 30 días son: abril (4), junio (6), septiembre (9) y noviembre (11) y el mes de febrero (2) que tiene entre 28 y 29 días.

#### **Algoritmo 25:**

*Iniciamos*

1. Recibir *mes*;
2. Según mes Hacer
  - 2.1. Caso 1,3,5,7,8,10,12: Mostrar “Tiene 31 días”;
  - 2.2. Caso 4,6,9,11: Mostrar “Tiene 30 días”;
  - 2.3. Caso contrario Mostrar “Tiene 28 días”;
  - 2.4. Fin según;

*Finalizamos*

*¿Qué datos entran a ser procesados?*

El algoritmo solicita el mes de forma numérica, y contiene el valor en el variable *mes*.

*¿Qué resultados muestra?*

Mostrará un mensaje que indica el número de días que tiene el *mes*.

*¿Qué condiciones se presentan en el proceso?*

Presenta una condición y dos preguntas, la condición permite chequear el valor de la variable *mes*, verifica si dicho valor coincide con cualquiera de la lista del primer caso, de ser así mostraría el mensaje de 31 días, en caso de no estar en la primera lista, verifica la lista del segundo caso, si uno de la lista coincide se mostrará el mensaje de 30 días, en caso de no coincidir en la lista se ejecutaría el caso contrario donde asumiría que el mes tiene 28 días.

#### **Actividades de refuerzo (AR):**

AR21. Desarrolle un algoritmo que muestre la actividad de un estudiante dependiendo de la hora en un día con formato de 24 horas, las actividades se detallan en el siguiente cuadro:

HORAS	ACTIVIDADES
Desde las 6:00 hasta las 8:00	Desayuno
Desde las 10:00 hasta las 11:00	Estudios prácticos
Desde las 12:00 hasta las 13:00	Almuerzo
Desde las 15:00 hasta las 19:00	Clases en el aula
Desde las 20:00 hasta las 21:00	Merienda
Otras horas	Descanso

AR22. Desarrolle un algoritmo que clasifique los vehículos que no sean maquinarias de construcción o producción por el número de llantas, considere el siguiente cuadro:

NUMERO DE LLANTAS	TIPO DE VEHÍCULO
2	Personal
4	Familiar
6	Carga mediana
10 O MÁS	Carga pesada

**OBJETIVO DE APRENDIZAJE EN ESTE BLOQUE:** Conocer y desarrollar las habilidades para implementar y controlar los diferentes procesos repetitivos, utiliza las herramientas necesarias para comprobar la funcionalidad de la lógica aplicada en la resolución de los problemas.

## DEFINICIONES GENERALES DE CONTROL Y EVALUACIÓN DE RESULTADOS

La aplicación de la lógica en los diferentes programas requiere de técnicas que ayuden a controlar los procesos que requieren ser repetidos, así como entender la lógica aplicada para resolver los diferentes problemas, las siguientes definiciones explican el uso del conteo para controlar los procesos repetitivos y las pruebas de escritorio para entender la lógica aplicada en los diferentes programas.

## USO DE CONTADORES Y ACUMULADORES COMO TÉCNICAS DE CONTROL

El uso de un contador es una técnica que permite incrementar de forma consecutiva el contenido de una variable de forma que se pueden generar secuencias, por ejemplo suponga que tenemos las variables “*turno*”, “*paso*” y “*tique*”, considere que estas variables se les asigna un valor inicial, al incrementarlas, sus valores iniciales cambiarían de la siguiente forma:

### Ejemplo 1

- 1.- turno = 1;
- 2.- turno = turno + 1;

### Ejemplo 2

- 1.- paso = 1;
- 2.- paso = paso + 2;

### Ejemplo 3

- 1.- tique = 0;
- 2.- tique = tique + 5;

Al momento de realizar el paso dos de cada ejemplo, sus contenidos variarán de la siguiente forma: el contenido de la variable “*turno*” será de 2, el valor de la variable “*paso*” será de 3 y el valor de la variable “*tique*” será de 5. Si se repite la misma línea, el contenido variará de la siguiente forma:

- ✓ “*turno*” que actualmente vale 2 al incrementarse en 1, el resultado se almacenará en la misma variable quedando en 3
- ✓ “*paso*” que actualmente vale 3 al incrementarse en 2, el resultado se almacenará en la misma variable quedando en 5
- ✓ “*tique*” que actualmente vale 5 al incrementarse en 5, el resultado se almacenará en la misma variable quedando en 10

Este proceso al hacerlo repetitivo, el contenido de la variable se incrementará de forma sucesiva, es importante recalcar que también funciona con la misma utilidad al decrementarse, multiplicarse o dividirse, considere los siguientes ejemplos:

### Ejemplo 1

- 1.- turno = 100;
- 2.- turno = turno – 1;

### Ejemplo 2

- 1.- paso = 2;
- 2.- paso = paso \* 2;

### Ejemplo 3

- 1.- tique = 136;
- 2.- tique = tique / 2;

Como complemento se considera el término *acumulador* que técnicamente es como un contador, la diferencia radica en la cantidad utilizada en la operación, es decir un contador realiza los cálculos con

un valor fijo, ya sea de uno en uno o de dos en dos o utilizando cualquier otro número pero de forma fija; el *acumulador* se incrementa utilizando diferentes valores, el uso de acumuladores se lo evidenciará en los siguientes algoritmos que se utilizarán como ejemplo.

## PRUEBAS DE FUNCIONAMIENTO O PRUEBAS DE ESCRITORIO

El uso de pseudocódigo aplicado a la lógica de los algoritmos, ayuda mucho en el análisis de la solución aplicada a los problemas, pero a pesar de la facilidad que se presenta en ésta técnica, existen soluciones complejas que necesitan de la experimentación o la prueba escrita que dé seguimiento de las instrucciones dadas en los algoritmos una por una, para esto se utilizan datos reales o ficticios para entender su funcionamiento y lógica aplicada a solución, ante esta situación se crea una tabla, en la cual las columnas son las variables y las filas los contenidos que ellas toman conforme a la secuencia aplicada en la lógica paso a paso.

El éste libro se mostrarán cuadros de pruebas de funcionamiento o pruebas de escritorio en los problemas complejos para su mayor entendimiento, así mismo en las actividades extras y de refuerzo se recomienda la utilización de los mismos.

## CICLOS REPETITIVOS

Para aplicar la metodología de control para procesos que deben repetirse un número determinado de veces, existen varias técnicas o reglas de control que son aplicados por todos los programadores, entre ellos los más utilizados son:

- Ir a
- Mientras... Hacer
- Para... Hacer
- Repetir ... hasta que
- Hacer ... mientras

Cada una de estas técnicas posee reglas que permiten controlar las veces que uno o varios procesos se repetirán, considerando la ejemplificación de cada una se explicarán de forma separada.

## USO DE LA TÉCNICA “IR A” PARA CONTROLAR PROCESOS REPETITIVOS

Esta técnica consiste controlar los procesos repetitivos mediante la indicación de paso a repetir desde una determinada instrucción, para controlar que cumpla la repetición y evitar que se creen ciclos de repetición infinitos, se utilizan las condiciones de forma obligatoria que permiten o no que el proceso de repetición se cumpla, considere el siguiente pseudocódigo que muestra el algoritmo que escribe un nombre cinco veces, a un costado del algoritmo observe la prueba de funcionamiento, la cual muestra a secuencia numerada entre paréntesis:

### Algoritmo 26:

*Iniciamos*

1. Contador = 1;
2. Recibir *Nombre*;
3. Escribir *Nombre*;
4. Contador = Contador + 1;
5. Sí Contador <= 5 *Entonces* Ir a paso número 3;

*Finalizamos*

Valor de la variable *Nombre*: (2) “**Jesús**”

Condición de repetición	Se muestra	Valor de la variable <i>Contador</i>
		(1) <b>1</b>
	(3) Jesús	(4) <b>2</b> = 1 + 1
(5) <b>2</b> <= 5 (V)	(6) Jesús	(7) <b>3</b> = 2 + 1
(8) <b>3</b> <= 5 (V)	(9) Jesús	(10) <b>4</b> = 3 + 1
(11) <b>4</b> <= 5 (V)	(12) Jesús	(13) <b>5</b> = 4 + 1
(14) <b>5</b> <= 5 (V)	(15) Jesús	(16) <b>6</b> = 5 + 1
(17) <b>6</b> <= 5 (F)		

Note que el control de repetición está dado por la variable *Contador* que toma un valor inicial de 1, después se le suma 1 y es contenido el resultado en la misma variable, cada vez que se repite desde el paso tres, el resultado del incremento se vuelve a contener en la misma variable *Contador*, la condición ubicada en el paso cinco comprueba que la variable *Contador* sea menor o igual a 5 para volver a repetir desde el paso número 3.

La condición encierra entre paréntesis la respuesta a la comparación, (V) significa que el resultado dio verdadero y (F) significa que dio falso, en la columna (Valor de la variable *Contador*) los valores resaltados son los valores que toma la variable en cada repetición.

*¿Qué datos entran a ser procesados?*

El algoritmo solicita un nombre cualquiera, como ejemplo de la prueba de funcionamiento se considera el nombre “Jesús”

*¿Qué resultados muestra?*

Mostrará cinco veces el contenido de la variable nombre para el caso “Jesús”.

*¿Qué condiciones se presentan en el proceso?*

Controla el proceso de repetición verificando si el contenido de la variable *Contador* es menor o igual que 5 para volver a repetir desde el paso número tres.

Otro ejemplo de pseudocódigo consiste en un algoritmo que muestra, entre el 1 y el 100 los números impares, como análisis considere que el *contador* debe empezar en 1 y se incrementara de 2 en 2, por ejemplo 1, 3, 5, 7, 9, 11, 13, 15, y así sucesivamente hasta el 99

#### Algoritmo 27:

*Iniciamos*

1. Contador=1;
2. Escribir *Contador*;
3. *Contador* = *Contador* + 2;
4. Sí *Contador* < 100 Entonces Ir a paso número 2;

*Finalizamos*

Este ejemplo es muy similar al algoritmo anterior, la diferencia está en que el contador se incrementa de 2 en 2, considere la siguiente prueba de funcionamiento para comprobar su solución:

Condición de repetición	Se muestra	Valor de la Variable <i>Contador</i>
		(1) <b>1</b>
	(2) 1	(3) <b>3</b> = 1 + 2
(4) <b>3</b> < 100 (V)	(5) 3	(6) <b>5</b> = 3 + 2
(7) <b>5</b> < 100 (V)	(8) 5	(9) <b>7</b> = 5 + 2
(10) <b>7</b> < 100 (V)	(11) 7	(12) <b>9</b> = 7 + 2
(13) <b>9</b> < 100 (V)	(14) 9	(15) <b>11</b> = 9 + 2
.	.	.
.	.	.
<b>99</b> < 100 (V)	99	<b>101</b> =99+2
<b>101</b> < 100 (F)		

#### USO DE LA TÉCNICA “MIENTRAS” PARA CONTROLAR PROCESOS REPETITIVOS

Esta técnica consiste en realizar ciclos repetitivos, primero evaluando la condición de control y acto seguido realizando el ciclo repetitivo *mientras la respuesta a la condición sea verdadera*, este proceso de repetición se mantendrá en ejecución hasta que la condición se haga falsa.

Para ilustrar la posibilidad de controlar procesos repetitivos mediante esta técnica, considere el siguiente ejemplo que muestra la lógica para extraer los impares del 1 al 100, pero utiliza la técnica *Mientras* para controlar el mismo proceso:

**Algoritmo 28:***Iniciamos*

1. Contador=1;
2. **Mientras** Contador < 100 **Hacer**
3.     Escribir *Contador*;
4.     *Contador* = *Contador* + 2;
5. **Fin mientras** y retorno al paso número 2;

*Finalizamos*

En este ejemplo se repiten los pasos tres y cuatro mientras la condición dada sea verdadera (considerando que el Contador sea menor que 100); Es importante entender que, al momento de que la condición se haga falsa, terminará el algoritmo cumpliendo con el propósito planteado.

El siguiente algoritmo plantea la necesidad de mostrar cuantos billetes de 5 dólares se necesitan para dar de cambio o vuelto en la compra de un producto cualquiera, para resolver el problema considere que se pedirá solo el valor del cambio o vuelto; el algoritmo mostrará el número de billetes de 5 dólares:

**Algoritmo 29:***Iniciamos*

1. Recibir *Cambio* ;
2. Contador = 0;
3. **Mientras** Cambio >= 5 **Hacer**
4.     Contador = Contador + 1;
5.     Cambio = Cambio – 5;
6. **Fin mientras** y retorno al paso 3;
7. Escribir *Contador*;

*Finalizamos*

Pregunta	Valor de la Variable <i>Contador</i>	Valor de la variable <i>Cambio</i>
	(2) <b>0</b>	(1) 23
(3) 23>=5 (V)	(4) <b>1</b>	(5) 18
(6) 18>=5 (V)	(7) <b>2</b>	(8) 13
(9) 13>=5 (V)	(10) <b>3</b>	(11) 8
(12) 8>=5 (V)	(13) <b>4</b>	(14) 3
(15) 3>=5 (F)		
Muestra lo que contiene <i>Contador</i> : 4		

Lo primero que pide el algoritmo es el valor de *Cambio*, asuma que se le da 23, el siguiente paso es asignarle cero a la variable *Contador* ya que esta variable contará el número de billetes, el tercer paso condiciona la repetición del proceso de la siguiente forma “mientras 23>=5 hará” *Contador* que vale 0 le suma 1 y *Cambio* que vale 23 le resta 5; vuelve a la condición de repetición, esta vez preguntará “Mientras 18>=5 Hacer”, recuerde que la variable *Cambio* ahora tiene 18, volverá a sumarle 1 a *Contador* y a restarle 5 a *Cambio*, este proceso se repetirá una y otra vez hasta que *Cambio* solo tendrá 3, y la respuesta a la condición de control será Falsa ya que 3 no es mayor ni igual que 5, entonces pasará al paso 7, el cual mostrará lo que contiene la variable *Contador*.

**Actividades de refuerzo (AR):**

- AR23. Desarrolle un algoritmo que permita registrar las edades de dos niños, se requiere encontrar la diferencia en años de las dos edades sin el uso del signo menos (-).
- AR24. Se necesita un algoritmo que calcule el valor a pagar de una compra de 10 productos, la lógica debe solicitar el valor del producto y la cantidad comprada, antes de finalizar se debe mostrar el total de la compra.
- AR25. Desarrolle un algoritmo que permita calcular el promedio general de un curso de 15 estudiantes, por cada estudiante se debe Recibir su promedio.

### ACTIVIDADES EXTRAS:

- A16. Teniendo 10 edades mostrar cuantas de las edades son mayores de edad.
- A17. Como resolver el cálculo de la potencia de un número cualquiera
- A18. Como calcular el promedio de sueldo de un vendedor durante el año pasado.
- A19. Como calcular la multiplicación de 2 números sin utilizar el signo de multiplicación.
- A20. En un curso de 30 estudiantes se desea elegir a un presidente entre dos candidatos, usted debe crear el algoritmo que permita recabar los votos, contarlos y mostrar el ganador por mayoría simple.

### USO DE LA TÉCNICA “PARA” COMO CONTROL A PROCESOS REPETITIVOS

A esta técnica también se la conoce como “control de procesos repetitivos con contadores automáticos”, considere que hasta ahora se han desarrollado ejercicios con procesos repetitivos controlados mediante el uso de condiciones, estas deben utilizar un contador que normalmente usted debe incrementarlo para limitar el número de repeticiones, la técnica del “*para*” o “*desde*” es utilizada en gran frecuencia por los programadores ya que solo se debe definir los límites de inicio y fin sin la necesidad de que usted especifique el incremento del contador ya que esto lo hace de forma automática, para ilustrar estas definiciones observe el siguiente ejemplo, se trata de un algoritmo que muestra cinco veces un nombre:

#### Algoritmo 30:

*Iniciamos*

1. Recibir *Nombre*;
2. **Para** *Contador* = 1 hasta 5 **Hacer**
3.       Escribir *Nombre*;
4. *Fin Para* y retorno al paso número 2;

*Finalizamos*

Note que la variable *Contador* toma un valor inicial de 1 y su incremento llegará hasta 5, el incremento será automático por cada repetición, esta herramienta disminuye las instrucciones en la lógica y facilita su comprensión en ejercicios más complejos.

El siguiente ejercicio muestra los valores que toma el contador cuando se solicita mostrar los números del 1 al 100

#### Algoritmo 31:

*Iniciamos*

1. **Para** *Contador* = 1 hasta 100 **Hacer**
2.       Escribir *Contador*;
3. *Fin Para* y retorno al paso número 1;

*Finalizamos*

En este ejemplo el contenido de la variable *Contador* empezará en 1 y se incrementará de uno en uno por cada repetición, es decir por cada repetición se escribirá el valor que tenga la variable *Contador* dando como resultado la lista de números desde el 1 hasta el 100.

*¿Qué datos entran a ser procesados?*

Para este ejemplo ninguno, ya que solo mostrará la lista desde el 1 hasta el 100.

*¿Qué resultados muestra?*

Mostrará una lista de números autoincrementados de uno en uno y que será tomada por la variable *Contador* desde el 1 hasta el 100.

Para ejemplificar el uso de esta técnica con procesos calculables, considere el siguiente algoritmo que calcula el promedio general de un curso de 7 estudiantes, por cada estudiante se debe solicitar su promedio individual:



**Algoritmo 32:***Iniciamos*

1.  $Acumulado = 0$ ;
2. **Para**  $Cont = 1$  hasta 7 **Hacer**
3.     Recibir *Promedio*;
4.      $Acumulado = Acumulado + Promedio$ ;
5. *Fin Para* y retorno al paso número 2;
6.  $PromedioGeneral = Acumulado / 7$ ;
7. Escribir  $PromedioGeneral$ ;

*Finalizamos*

Valor de la Variable <b>Cont</b>	Valor de la Variable <b>Promedio</b>	Valor de la Variable <b>Acumulado</b>
		(1) <b>0</b>
(2) <b>1</b>	(3) <b>8</b>	(4) <b>8</b> = 0 + 8
(5) <b>2</b>	(6) <b>7.5</b>	(7) <b>15.5</b> = 8 + 7.5
(8) <b>3</b>	(9) <b>9</b>	(10) <b>24.5</b> = 15.5 + 9
(11) <b>4</b>	(12) <b>10</b>	(13) <b>34.5</b> = 24.5 + 10
(14) <b>5</b>	(15) <b>8.4</b>	(16) <b>42.9</b> = 34.5 + 8.4
(17) <b>6</b>	(18) <b>10</b>	(19) <b>52.9</b> = 42.9 + 10
(20) <b>7</b>	(21) <b>9.5</b>	(22) <b>62.4</b> = 52.9 + 9.5
Variable	<b>PromedioGeneral</b>	(23) <b>8.91</b> = 62.4 / 7

En este ejercicio se utiliza un acumulador que permite sumar los promedios de los siete estudiantes, el algoritmo antes de finalizar utiliza la suma del acumulador para dividirlo entre 7 y obtener el promedio general del curso.

*¿Qué datos entran a ser procesados?*

Inicialmente *Acumulado* toma el valor de cero y por cada repetición se solicita el *Promedio* de cada estudiante, una vez finalizado el programa se habrán ingresado siete promedios, uno por uno.

*¿Qué resultados muestra?*

Antes de finalizar el algoritmo mostrará el promedio general del curso al dividir la suma de todos los promedios contenidos en la variable *Acumulado* para siete

*¿Qué condiciones se presentan en el proceso?*

Ninguna, ya que la técnica ejerce un control automático de repetición.

Para ejemplificar el uso de la técnica *Para* en algoritmos que incluyen condiciones, considere la modificación del algoritmo 32, de forma que muestre a más del promedio general, el número de estudiantes que tienen un promedio individual entre 9 y 10.

**Algoritmo 33:***Iniciamos*

1.  $Acumulado = 0$ ;
2.  $Est = 0$ ;
3. **Para**  $Cont = 1$  hasta 7 **Hacer**
4.     Recibir *Promedio*;
5.      $Acumulado = Acumulado + Promedio$ ;
6.     **Sí**  $Promedio \geq 9$  **Entonces**  $Est = Est + 1$ ;
7. *Fin Para* y retorno al paso número 2;
8.  $PromedioGeneral = Acumulado / 7$ ;
9. Escribir *Est*, " tienen un promedio entre 9 y 10 ";
10. Escribir  $PromedioGeneral$ ;

*Finalizamos*

En este ejemplo a diferencia del algoritmo 32, utiliza la variable *Est* que contará el número de estudiantes que cumplen con la condición ( $Promedio \geq 9$ ), para lograrlo utiliza la técnica del contador.

*¿Qué datos entran a ser procesados?*

Inicialmente las variables *Acumulado* y *Est* toman el valor de cero antes de realizar los ciclos repetitivos, por cada repetición se solicita el *Promedio* de cada estudiante, se acumula la sumatoria de todos los promedios y evalúa si cumple la condición *mayor o igual que nueve* para realizar el conteo de la variable *Est*.

*¿Qué resultados muestra?*

Antes de finalizar mostrará el número de estudiantes que tendrán un promedio entre 9 y 10, además muestra el promedio general del curso al dividir la suma de todos los promedios contenidos en la variable *Acumulador* para siete.

*¿Qué condiciones se presentan en el proceso?*

La única condición es la que evalúa si el promedio del estudiante cumple la condición para contarlo entre los que tienen de nueve a diez como promedio individual.

#### **Actividades de refuerzo (AR):**

AR26. Desarrolle un algoritmo que permita la resta de 2 números sin el signo menos (-) utilizando el ciclo de repetición PARA.

AR27. Se necesita un algoritmo que muestre el número de estudiantes aprobados y reprobados de un curso de 30 estudiantes, la lógica debe solicitar el promedio de cada estudiante, se considera que un estudiante ha aprobado cuando tiene una nota mayor o igual que 8, antes de finalizar se debe mostrar el total de estudiantes aprobados y el total de estudiantes reprobados.

AR28. Desarrolle un algoritmo que permita mostrar los primeros 35 números pares.

#### **ACTIVIDADES EXTRAS:**

A21. Teniendo 10 edades mostrar cuantas de las edades son pares y cuantas son impares.

A22. Se necesita un algoritmo que muestre el resultado de los 12 primeros números de la siguiente serie:  $2 + 5 + 8 + 11 + 14 + \dots$

A23. Desarrolle un algoritmo que permita sumar los primeros 20 impares.

#### **USO DE LA TÉCNICA “REPETIR... HASTA QUE” PARA CONTROLAR PROCESOS REPETITIVOS**

Todas las técnicas de control de repetición tienen un propósito, facilitar el desarrollo de la lógica para controlar ciclos repetitivos, esta técnica de control de repetición facilita *la validación de datos*, consiste en repetir determinadas acciones en la cual se incluye el recibir datos, hasta que el dato solicitado o procesado cumpla con la condición del problema, es decir, suponiendo que se está solicitando el *mes* de nacimiento de una persona, si se recibe un valor entre 1 y 12 no hay problema, pero que pasaría si se recibe un valor mayor como el 16 u otro; lo más seguro es que presente error, no de lógica o mal planteada la solución algorítmica sino un error de proceso al encontrar algún resultado no esperado por un dato mal recibido; cuando usted solicita un dato, no existe la garantía de que se cumpla con lo solicitado en dicho proceso, ante esta situación es recomendable *validar* el recibir o ingreso de datos.

Para ilustrar el propósito principal de ésta técnica, considere el siguiente ejemplo planteado como un problema: se necesita un algoritmo que permita recibir dos números positivos, el algoritmo deberá mostrar los números ordenados de forma descendente:

Análisis: cuando se recibe un valor positivo, significa que dicho valor debe ser mayor que cero, por lo que es necesario validar los dos números a recibir, ya que se pueden recibir valores negativos y no cumplir con lo solicitado en el problema, además el problema pide mostrar los dos números ordenados de forma descendente, esto significa que el mayor irá primero y después el menor; como solución al problema considere la siguiente propuesta algorítmica:

**Algoritmo 34:**

*Iniciamos*

1. **Repetir**
2.     Recibir A;
3.     **Hasta que**  $A > 0$  ;
4.     **Repetir**
5.         Recibir B;
6.     **Hasta que**  $B > 0$ ;
7.     **Sí**  $A > B$  *Entonces* Escribir A, B;
8.     **Caso contrario** Escribir B, A;

*Finalizamos*

En el ejemplo se presentan dos validaciones, una para cada variable, esto le permitirá garantizar al algoritmo que los valores a recibir sean positivos, considere que cada validación expresa: “ repetirá hasta que el contenido de la variable sea mayor que cero”.

*¿Qué datos entran a ser procesados?*

Mediante el uso de la validación, se recibirá dos valores mayores que cero.

*¿Qué resultados muestra?*

Mostrará ordenado de forma descendente los dos números recibidos.

*¿Qué condiciones se presentan en el proceso?*

La primera y la segunda condición están dadas para validar los datos recibidos, por cada valor recibido repetirán hasta que, el valor de la variable A y el valor de variable B sean mayores que cero; la tercera condición chequeará si el contenido de la variable A es mayor que el contenido de la variable B con el propósito de mostrar primero el contenido de la variable A y después el contenido de la variable B, caso contrario mostrará primero el contenido de la variable B y después el contenido de la variable A.

El siguiente ejemplo plantea el problema: Desarrollar un algoritmo que permita encontrar la diferencia entre dos números sin utilizar el signo de sustracción (-), el algoritmo debe validar que el minuendo sea mayor que el sustraendo.

**Algoritmo 35:**

*Iniciamos*

1. Recibir *minuendo*;
2. **Repetir**
3.     Recibir *sustraendo*;
4.     **Hasta que**  $sustraendo < minuendo$ ;
5.     Resul=0;
6.     **Para** Cont = *sustraendo* hasta *minuendo* **Hacer**
7.         Resul = Resul + 1;
8.     **Fin Para** y retorno al paso número 6;
9.     Escribir Resul;

*Finalizamos*

Observe que la validación es más detallada, ya que al recibir el *sustraendo*, este debe ser menor que el *minuendo*, esto con la finalidad de cumplir con la condición propuesta para resolver el problema, una vez validados los valores a recibir se realiza un recorrido que va desde el número menor hasta el número mayor, esto permitirá encontrar la diferencia entre dos números, o dicho de otra forma encontrar el resultado de la resta.

*¿Qué datos entran a ser procesados?*

Dos valores, primero pide un número cualquiera y lo contiene en la variable *minuendo*, el segundo valor contenido en la variable *sustraendo*, este valor está condicionado a ser menor que el contenido de la variable *minuendo*.

*¿Qué resultados muestra?*

Mostrará el número de lugares a recorrer entre un número menor y un número mayor

*¿Qué condiciones se presentan en el proceso?*

La condición presentada es para validar el ingreso del segundo número.

En el siguiente problema se plantea la necesidad de un algoritmo que permita el ingreso de 100 billetes en dólares americanos, el algoritmo debe validar que los billetes sean de 1, 2, 5, 10 y 20, considere que no se aceptan billetes de 50 y 100, el algoritmo debe mostrar la cantidad recibida:

#### **Algoritmo 36:**

*Iniciamos*

1. Total=0;
2. **Para** Cont = 1 hasta 100 **Hacer**
3.     **Repetir**
4.         Recibir billete;
5.     **Hasta que** billete=1 Or billete=2 Or billete=5 Or billete=10 Or billete=20;
6.     Total = Total + billete;
7. *Fin Para* y retorno al paso número 2;
8. Escribir Total;

*Finalizamos*

En la solución planteada al problema se presenta una validación que utiliza varias comparaciones unidas con el operador lógico OR, recuerde que, si al menos una de las comparaciones es verdadera el resultado es verdadero, es decir asumiendo que se asigna 50 a la variable billete, al realizar las comparaciones, ninguna es verdadera por lo tanto el resultado es falso y volverá a repetir hasta que al menos una de ellas sea verdadera.

*¿Qué datos entran a ser procesados?*

100 billetes validados entre 1, 2, 5, 10 y 20

*¿Qué resultados muestra?*

El total de dinero ingresado, no el total de billetes

*¿Qué condiciones se presentan en el proceso?*

La única condición propuesta es para validar el ingreso de cada billete.

#### **USO DE LA TÉCNICA HACER... MIENTRAS PARA CONTROLAR PROCESOS REPETITIVOS**

Esta técnica de control de repetición facilita *la validación de datos* igual a la ofrecida en la técnica "*Repetir... Hasta que*", es decir consiste en repetir determinadas acciones *Mientras* el dato solicitado o procesado cumpla con la condición del problema (mientras sea verdadera la respuesta a la condición), considerando el mismo ejemplo utilizado en la explicación del control "*Repetir hasta que*", suponga que se necesita el número del *mes* de nacimiento de una persona para realizar una determinada acción, si se recibe un valor entre 1 y 12 no hay problema, pero que pasaría si se recibe un valor negativo o un valor mayor como el 16 u otra cantidad que no está entre 1 y 12, de seguro el resultado esperado nunca se mostrará, considere la siguiente validación que resuelve el problema planteado utilizando los dos controles "*Repetir... hasta que*" y "*Hacer... mientras*" para diferenciar sus potencialidades:

## VALIDACIÓN UTILIZANDO REPETIR HASTA QUE

1. ....
2. **Repetir**
3.     Recibir *mes*;
4. **Hasta que** *mes* >= 1 AND *mes* <= 12;
5. ....

## VALIDACIÓN UTILIZANDO HACER MIENTRAS

1. ....
2. **Hacer**
3.     Recibir *mes*;
4. **Mientras** *mes* < 1 OR *mes* > 12;
5. ....

Ambas técnicas cumplen con el mismo propósito, la diferencia radica en que la técnica “*Repetir... Hasta que*” realiza una nueva repetición siempre y cuando la condición sea falsa y sale del ciclo repetitivo cuando la condición sea verdadera, en cambio la técnica “*Hacer... Mientras*” realiza una nueva repetición siempre y cuando la condición sea verdadera y sale del ciclo repetitivo cuando sea falsa.

Para ilustrar el propósito principal de la técnica “*Hacer... mientras*”, se aplicarán los mismos ejercicios planteados y explicados en la técnica “*Repetir... Hasta que*”:

El primero de los problemas propone la necesidad de un algoritmo que permita recibir dos números positivos, el algoritmo deberá mostrar los números ordenados de forma descendente:

Análisis: cuando se recibe un valor positivo, significa que dicho valor debe ser mayor que cero, por lo que es necesario validar el proceso de recibir de los dos números, ya que por ocurrencia de los usuarios se pueden recibir valores negativos y no cumplir con lo solicitado en el problema, además el problema pide mostrar los dos números ordenados de forma descendente, esto significa que el mayor irá primero y después el menor; como solución al problema considere la siguiente propuesta algorítmica utilizando la técnica de control “*Hacer ... Mientras*”:

**Algoritmo 37:**

*Iniciamos*

1. **Hacer**
2.     Recibir *A*;
3. **Mientras** *A* <= 0;
4. **Hacer**
5.     Recibir *B*;
6. **Mientras** *B* <= 0;
7.     *Sí* *A* > *B* Entonces Escribir *A*, *B*;
8.     *Caso contrario* Escribir *B*, *A*;

*Finalizamos*

En el ejemplo se presentan dos validaciones, una para cada variable, esto le permitirá garantizar al algoritmo que los valores a recibir son positivos, considere que cada validación expresa, que repetirá mientras el contenido de la variable sea menor o igual que cero.

*¿Qué datos entran a ser procesados?*

Mediante el uso de la validación, se recibirá dos valores mayores que cero.

*¿Qué resultados muestra?*

Mostrará ordenado de forma descendente los dos números recibidos.

*¿Qué condiciones se presentan en el proceso?*

La primera y la segunda condición están dadas para validar los datos recibidos, por cada valor recibido repetirán mientras el valor de la variable *A* y el valor de variable *B* sean menores o iguales que cero; la tercera condición chequeará si el contenido de la variable *A* es mayor que el contenido de la variable *B* con el propósito de mostrar primero el contenido de la variable *A* y después el contenido de la variable *B*, caso contrario mostrará primero el contenido de la variable *B* y después el contenido de la variable *A*.

El siguiente ejemplo plantea el problema: Desarrollar un algoritmo que permita encontrar la diferencia entre dos números sin utilizar el signo de sustracción (-), el algoritmo debe validar que el minuendo sea mayor que el sustraendo.

#### Algoritmo 38:

*Iniciamos*

1. Recibir *minuendo*;
2. **Hacer**
3.     Recibir *sustraendo*;
4.     **Mientras** *sustraendo* >= *minuendo*;
5.     Resul=0;
6.     **Para** Cont = *sustraendo* hasta *minuendo* **Hacer**
7.         Resul = Resul + 1;
8.     **Fin Para** y retorno al paso número 6;
9.     Escribir Resul;

*Finalizamos*

*¿Qué datos entran a ser procesados?*

Dos valores, primero pide un número cualquiera y lo contiene en la variable *minuendo*, el segundo valor contenido en la variable *sustraendo*, este valor está condicionado a ser menor que el contenido de la variable *minuendo*.

*¿Qué resultados muestra?*

Mostrará el número de lugares a recorrer entre un número menor y un número mayor

*¿Qué condiciones se presentan en el proceso?*

La condición presentada es para validar el ingreso del segundo número.

En el siguiente problema se plantea la necesidad de un algoritmo que permita el ingreso de 100 billetes en dólares americanos, el algoritmo debe validar que los billetes sean de 1, 2, 5, 10 y 20, considere que no se aceptan billetes de 50 y 100, el algoritmo debe mostrar la cantidad recibida:

#### Algoritmo 39:

*Iniciamos*

1. Total=0;
2. **Para** Cont = 1 hasta 100 **Hacer**
3.     **Hacer**
4.         Recibir billete;
5.         **Mientras** billete<>1 AND billete<>2 AND billete<>5 AND billete<>10 AND billete<>20;
6.         Total = Total + billete;
7.     **Fin Para** y retorno al paso número 2;
8.     Escribir Total;

*Finalizamos*

En la solución planteada al problema se presenta una validación que utiliza varias comparaciones unidas con el operador lógico AND, recuerde que, todas las comparaciones deben ser verdaderas el resultado sea verdadero, es decir asumiendo que se asigna 50 a la variable billete, al realizar las comparaciones, todas son verdaderas por lo tanto el resultado es verdadero y volverá a repetir mientras todas las comparaciones sean verdaderas.

*¿Qué datos entran a ser procesados?*

100 billetes validados entre 1, 2, 5, 10 y 20

*¿Qué resultados muestra?*

El total de dinero ingresado, no el total de billetes

*¿Qué condiciones se presentan en el proceso?*

La única condición propuesta es para validar el ingreso de cada billete.

#### **Actividades de refuerzo (AR):**

AR29. Modifique el algoritmo # 39 de forma que muestre el número total de billetes de cada denominación.

AR30. Desarrolle un algoritmo que valide un número entre 1 y 10, la lógica debe mostrarlo en romano.

AR31. Desarrolle un algoritmo que valide un número de 2 cifras, muestre si es par o impar.

#### **ACTIVIDADES EXTRAS:**

A24. Se necesita un algoritmo que permita el ingreso de un número positivo múltiplo de tres, el algoritmo antes de finalizar debe mostrar si es o no par.

A25. Desarrolle un algoritmo que permita el ingreso de dos números de un solo dígito cada uno, considere que el segundo número debe ser mayor que el primero, el algoritmo antes de finalizar debe mostrar la media de los dos números.

A26. Desarrolle un algoritmo que permita el ingreso de 10 números pares de dos dígitos, el algoritmo antes de finalizar debe mostrar la sumatoria de los 10 número.

#### **RESUMEN**

El uso de pseudocódigo disminuye la redacción natural de los algoritmos y facilita la aplicación de las instrucciones para definir las acciones apropiadas que resuelven el problema, las palabras pseudocodificadas son únicas y describen la acción que se desea aplicar con la instrucción, así los valores a procesar o datos en general son representados en los algoritmos con el uso de nombres llamados variables, las variables son palabras únicas cuyo contenido puede cambiar a conveniencia de la solución algorítmica, los nombres de variables pueden ser utilizados para recibir y mostrar datos, aplicarlos en cálculos y generar condiciones mediante la aplicación de comparaciones, aplicando algoritmos pseudocodificados las soluciones expresadas solo contienen palabras únicas propias del pseudocódigo, nombres de variables y valores o cantidades definidas como constantes, se aplican simbologías de comparación y de operaciones lógicas para unir varias comparaciones con la finalidad de generar condiciones más complejas, los controles aplicados a las condiciones simples y las condiciones múltiples permiten definir acciones específicas a realizar entre varias posibilidades o alternativas, el uso de pseudocódigo ofrece la posibilidad de incluir condiciones de caso que simplifican las comparaciones de igualdad, esta técnica puede reemplazar a las condiciones múltiples en comparaciones de igualdad, la aplicación de contadores y acumuladores son técnicas que permiten incrementar o decrementar el contenido de las variables en sí misma, la implementación y el control de procesos repetitivos se logra mediante el uso de variables que permiten el conteo y la aplicación de una condición que evalúa si se repite o no los procesos, mediante el uso de controles como “repetir...hasta que” y “hacer... mientras” se logra la aplicación de procesos de validación que asegura el proceso mediante el uso de datos solicitados.

**IMPORTANTE** Cuando los algoritmos son extensos se hace difícil entender la lógica, para ello se recurre a los dibujos (Diagramas) que facilitan la interpretación visual de la lógica y disminuye la escritura, esta técnica muestra la secuencia lógica con el uso de flechas (*Flujo*), el siguiente apartado lo introduce al uso de los diagramas de flujo.

## CAPITULO III

### USO DE LOS DIAGRAMAS DE FLUJO PARA RESOLVER PROBLEMAS

Cuando los algoritmos proponen soluciones complejas o extensas, resulta difícil entender o dar a entender los procedimientos lógicos que resuelven el problema, este capítulo ofrece la posibilidad de desarrollar soluciones algorítmicas mediante el uso de graficas geométricas llamadas diagramas de flujo, esta técnica consiste en representar gráficamente las diversas instrucciones dadas por el programador en los algoritmos.

Para hacer comprensibles los algoritmos mediante el uso de los diagramas de flujo, el capítulo incluye la explicación y la ejemplificación de los diferentes símbolos gráficos normalizados como el rombo, el rectángulo, la circunferencia, flechas direccionales, entre otros; la simbología gráfica utilizada en los diagramas de flujo son descritos conforme a las estructuras de control que representan, así se realizan diagramas de flujo que detallan algoritmos que aplican condiciones, estructuras de control para ciclos repetitivos, validaciones, entre otras herramientas de control.

La estructura de éste capítulo ejemplifica los diagramas de flujo utilizando como base inicial los algoritmos básicos de tres pasos que detallan la resolución de problemas de cálculos, propone y explica el uso de algoritmos que aplican condiciones simple, condiciones múltiples y condiciones de casos, continua desarrollando los diagramas de flujo que aplican algoritmos de control de procesos repetitivos como los utilizados al aplicar pseudocódigos en el capítulo II, es decir define los procedimientos mecánicos y lógicos de las estructuras como: *Hacer...Mientras, Repetir...Hasta que, Mientras y Para.*

**OBJETIVO DE APRENDIZAJE DEL SIGUIENTE APARTADO:** Conocer en que consiste la diagramación de la lógica, entender la secuencia lógica de los pasos de forma gráfica, aplicar la simbología recomendada y desarrollar ejercicios más complejos con la ayuda de los elementos gráficos.

### PREÁMBULO DE LOS DIAGRAMAS DE FLUJO



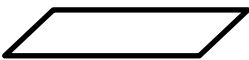





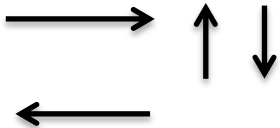
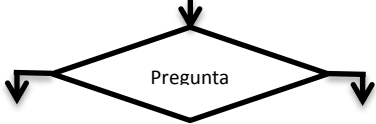
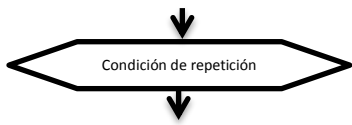
El diagrama de flujo es una técnica que permite plantear una propuesta algorítmica a los problemas propuestos, reemplaza las palabras utilizadas en el algoritmo como Recibir, mostrar, Escribir, calcular, entre otros, por *dibujos geométricos únicos*, que expresan de forma gráfica las soluciones algorítmicas, cada paso lógico se expresa con una figura, cada figura tiene sus propias reglas y los datos a ser procesados son contenidos en *variables*, las numeraciones utilizadas en los algoritmos que indican el orden de proceso, se reemplazan por *flechas* que indican la secuencia del siguiente proceso.

El diagrama de flujo es una técnica que ofrece una descripción visual de la lógica, funciona mostrando el orden y secuencia de forma más simple, facilita la rápida comprensión de cada actividad sin la necesidad de leer o entender la redacción y su relación con las demás gráficas.

Existe una variedad de gráficas simbólicas que permiten representar a los diferentes procesos y dispositivos que utiliza la computadora, por ejemplo para graficar el proceso de recibir datos, hay varias posibilidades, entre ellas utilizar la gráfica del teclado, de la tarjetas perforadas, de la cintas magnéticas, u otro dispositivo que grafique la acción de proveer datos a la computadora, así el proceso de mostrar o escribir datos y/o resultados, se puede representar con el símbolo de la pantalla o la impresora, entre otras posibilidades actuales; por tal razón este material bibliográfico, utilizará los símbolos más representativos como metodología de enseñanza, el siguiente cuadro muestra los símbolos gráficos y su propósito:



**TABLA 7.** Símbolos gráficos utilizados en el diagrama de flujo

SÍMBOLO GRÁFICO	PROPÓSITO O MODO DE EMPLEO
	<i>Inicio/Fin</i> , es utilizado para indicar donde inicia y donde finaliza el diagrama de flujo
	<i>Proceso</i> , es utilizado para escribir cálculos y para pasar un valor o asignar valores
	Representa recibir datos desde cualquier dispositivo, en su interior se escribirán variables, es decir nombres que representan valores y que pueden cambiar conforme a la lógica aplicada
	Representa Recibir datos desde una tarjeta perforada, para el caso es igual que el gráfico anterior
	Representa Recibir datos de forma manual desde el teclado, en sí es igual que los dos gráficos anteriores
	Conector, ésta gráfica permite conectar secuencias
	Esta gráfica representa mostrar datos, mensajes o resultados por pantalla
	Esta gráfica representa mostrar datos, mensajes o resultados por impresora
	Las flechas representan la secuencia del siguiente paso
	Esta gráfica permite realizar preguntas para definir acciones en base a dos posibilidades (Verdadero o Falso) de respuesta a la pregunta.
	Esta gráfica es utilizada para representar controles que gestionan los procesos repetitivos, en su interior se escriben, la o las condiciones que se aplican para las nuevas repeticiones.

**Fuente:** Autores del libro

## CARACTERÍSTICAS A CONSIDERAR PARA CREAR UN DIAGRAMA DE FLUJO

- ✓ El sentido de orientación y comprensión lógica de desarrollo es similar a los algoritmos narrados y pseudocodificados, de arriba hacia abajo, pero por su naturaleza gráfica también es permitido hacerlo de izquierda a derecha.
- ✓ La unión entre un proceso y otro “es decir entre un símbolo y otro símbolo” obligatoriamente debe ser mediante el uso de la flecha que indica el sentido y orden de procesos, es recomendable no utilizar flechas diagonales, ni cruzadas entre sí, para la unión de varias flechas se recomienda el uso de conectores.
- ✓ Es recomendable que no exista flechas sin conexión, ya que no tendría sentido apuntar a un paso inexistente.
- ✓ En general las gráficas pueden tener varios accesos de entrada y una sola salida, a excepción del símbolo de inicio/fin y los símbolos que aplican condiciones.

Para explicar el uso de cómo se utilizan los diagramas de flujo, considere el primer algoritmo explicado en el apartado de anterior, el cual le piden los pasos para sumar dos números, el diagrama de flujo quedaría así:

**Diagrama 1:**

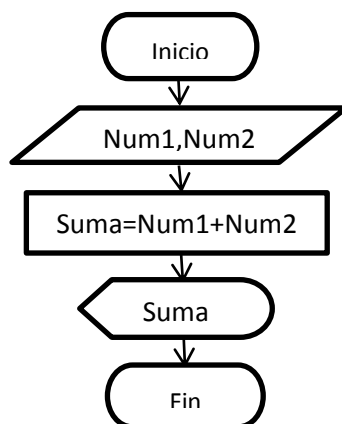
### ALGORITMO

*Iniciamos*

1. Recibir N1, N2;
2.  $\text{Suma} = \text{N1} + \text{N2}$ ;
3. Escribir Suma;

*Finalizamos*

### DIAGRAMA DE FLUJO



*¿Qué datos entran a ser procesados?*

Dos número cualquiera

*¿Qué resultados muestra?*

La suma de los dos números

*¿Qué condiciones se presentan en el proceso?*

ninguna

Observe que la lógica aplicada en la gráfica indica que los valores a recibir *Num1* y *Num2* pueden ser tomados desde cualquier dispositivo de entrada, y el resultado contenido en la variable *Suma*, será mostrado o escrito utilizando la pantalla.

Para el siguiente ejemplo se aplica la lógica para calcular el promedio de tres notas, observe que para la mayor comprensión de la lógica mediante el uso de los diagramas, se muestra el algoritmo pseudocodificado y su respectivo diagrama de flujo:

**Diagrama 2:**

*Iniciamos*

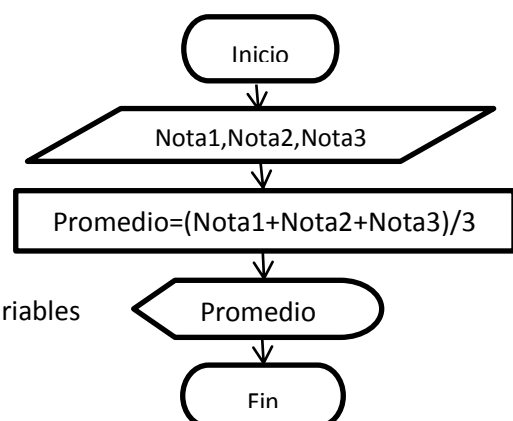
1. Recibir Nota1, Nota2, Nota3;
2.  $\text{Promedio} = (\text{Nota1} + \text{Nota2} + \text{Nota3}) / 3$ ;
3. Escribir Promedio;

*Finalizamos*

*¿Qué datos entran a ser procesados?*

Se reciben tres notas cualesquiera, contenidas en las variables Nota1, Nota2 y Nota3

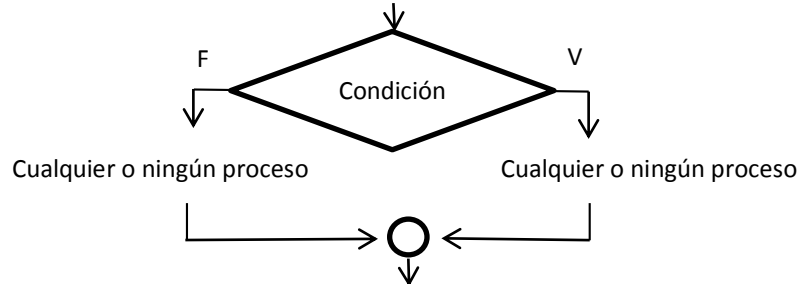
*¿Qué resultados muestra?*



El promedio de las tres notas

### FORMATO UTILIZADO PARA EXPRESAR CONDICIONES EN LOS DIAGRAMAS DE FLUJO

Las condiciones utilizan un rombo en cuyo interior debe incluir una condición, la figura debe tener una entrada y dos salidas que después de realizar los procesos cualesquiera se unen en un conector para continuar con el siguiente proceso, cada salida debe estar etiquetada con un verdadero (V, SI, S) o con un falso (F, NO, N), su formato es la siguiente:



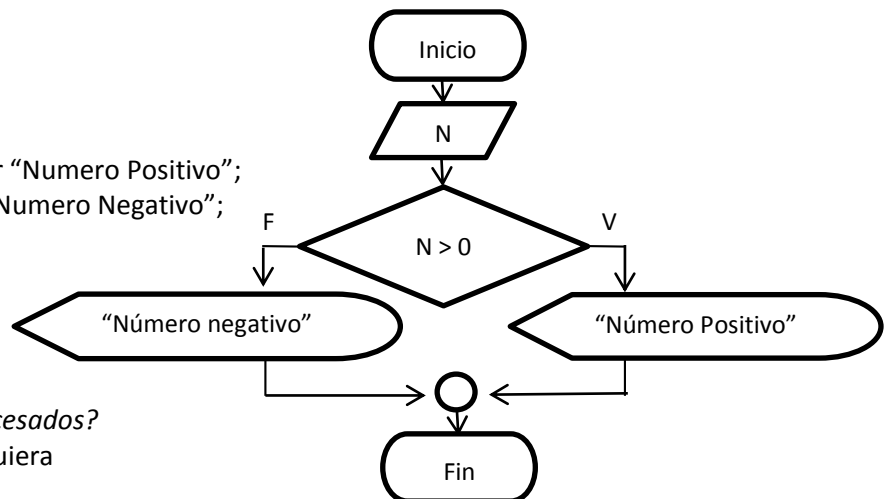
En el siguiente ejemplo se aplica el algoritmo y el diagrama de flujo para verificar si un valor cualquiera es un número positivo o un número negativo, recuerde que un número es positivo, si su valor es mayor que cero, a su vez es negativo cuando su valor es menor que cero, para este ejemplo se mostrarán mensajes alusivos a la respuesta de la condición, observe que los mensajes siempre estarán encerrados ente comillas (""):

#### Diagrama 3:

*Iniciamos*

1. Recibir N;
2. Sí  $N > 0$  Entonces Escribir "Numero Positivo";  
Caso contrario Escribir "Numero Negativo";

*Finalizamos*



*¿Qué datos entran a ser procesados?*

Se recibirá un número cualquiera

*¿Qué resultados muestra?*

Un mensaje indicando si el número es positivo o negativo

*¿Qué condiciones se presentan en el proceso?*

Compara si el número recibido es mayor que cero.

El ejemplo del diagrama de flujo, se muestra el uso del símbolo de condición (El rombo), en cuyo interior se escriben las comparaciones deseadas, por normativa de uso, las puntas izquierda y derecha se utilizan para graficar las acciones a realizar por verdadero y/o por falso, es importante indicar que, a la derecha se grafican las acciones por verdadero y a la izquierda las acciones por falso, pero esto no es una camisa de fuerza, ya que se lo puede hacer conforme a la comprensión del programador; la condición obligatoriamente separa las acciones de verdadero y de falso, pero la secuencia lógica normaliza que deben unirse para continuar con la cadena hasta finalizar el programa, las acciones por verdadero y falso se deben encontrar en un conector y del conector la flecha al siguiente proceso lógico.

Considere el siguiente ejemplo, que incluye una condición en su lógica para comparar dos números cualesquiera y mostrar como resultado solo el mayor:

**Diagrama 4:**

*Iniciamos*

1. Recibir *Num1*, *Num2*
2. Sí *Num1* > *Num2* Entonces Escribir *Num1*;  
Caso contrario Escribir *Num2*;

*Finalizamos*

*¿Qué datos entran a ser procesados?*

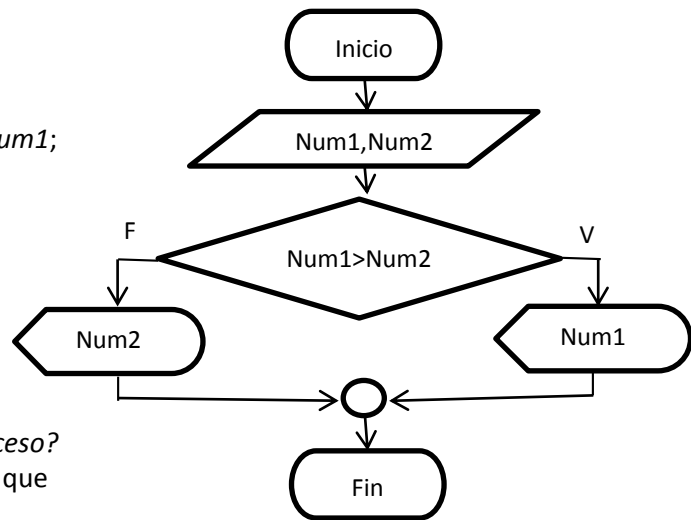
Dos números cualesquiera

*¿Qué resultados muestra?*

El mayor de los dos números

*¿Qué condiciones se presentan en el proceso?*

Compara si uno de los número es mayor que el otro.

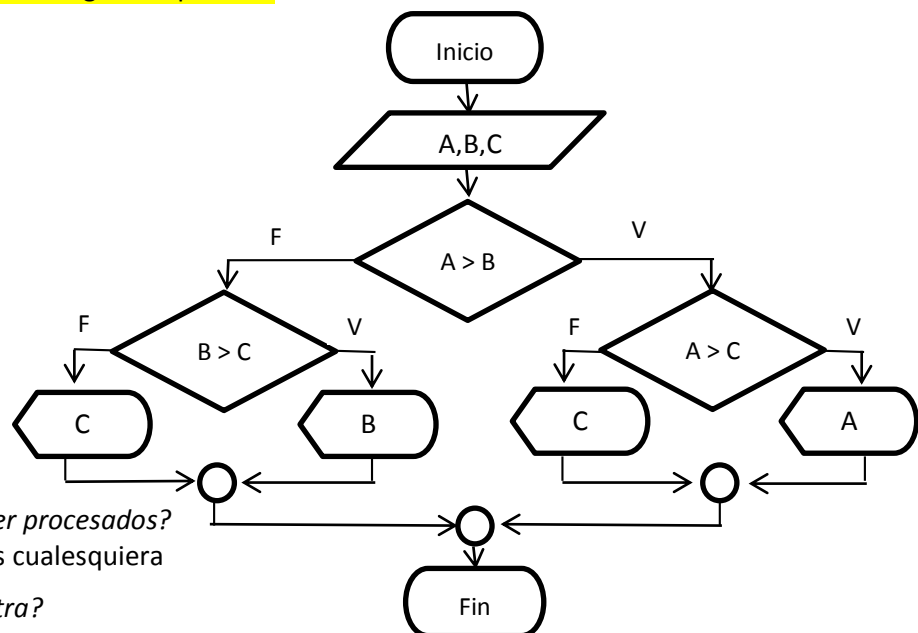


La siguiente tabla es una prueba de funcionamiento o prueba de escritorio, ésta es el resultado de hacer paso a paso lo descrito en la lógica algorítmica, tiene la finalidad de mostrar el funcionamiento de la solución, considere que se recibe cualquier valor para las variables *Num1* y *Num2*, la prueba de escritorio o de funcionamiento debe mostrar el mayor de los dos números:

Valor de la Variable <i>Num1</i>	Valor de la Variable <i>Num2</i>	Condición	Resultado a mostrar como mayor
(1) 15	(2) 58	(3) 15 > 58 (F)	(4) 58

El siguiente ejercicio sería complicado desarrollarlo en un algoritmo narrado, se trata de un diagrama de flujo que solicita tres números cualesquiera y como resultado mostrará el mayor, es importante destacar que cuando se presentan problemas complejos, el uso de los diagramas de flujo facilita la comprensión y simplifica la lógica al aplicarlo:

**Diagrama 5:**



*¿Qué datos entran a ser procesados?*

Se recibe tres números cualesquiera

*¿Qué resultados muestra?*

El mayor de los tres números

*¿Qué condiciones se presentan en el proceso?*

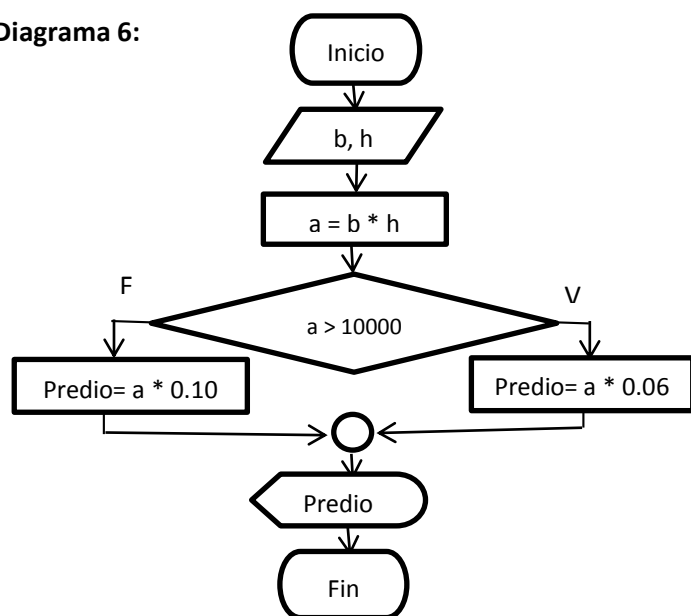
El diagrama presenta tres condiciones, en la primera condición compara dos variables y dependiendo de la respuesta se aplica la respuesta por verdadero o falso, sin importar que dirección toma, las comparaciones utilizadas a la izquierda o a la derecha encontrará el mayor definitivo de los tres valores recibidos.

En este ejemplo las variables A,B y C representan tres valores diferentes, el orden de comparación usted puede decidirlo, para el caso expuesto, primero pregunta si el valor de la variable A es mayor que el valor de la variable B, al ser verdadera la respuesta, vuelve a preguntar si el valor de la variable A es mayor que el valor de la variable C, al ser verdadera la respuesta muestra como mayor el valor de la variable A, caso contrario es mayor el valor de la variable C (No necesita comparar el valor de la variable B con el valor de la variable C porque el valor de A era mayor que el valor de B); si retornamos a la primera pregunta ( $A > B$ ) y la respuesta es *falso*, significa que el valor de la variable B es mayor que el valor de la variable A, por lo tanto pregunta si el valor de B es mayor que el valor de C, al ser verdadera la respuesta muestra el valor de B como Mayor, caso contrario muestra el valor de C como mayor (No es necesario comparar el valor de C con el valor de A porque el valor de B era mayor).

El siguiente ejercicio plantea la necesidad de desarrollar un diagrama de flujo que permita calcular el área de un terreno rectangular para el pago de los predios, considere que el municipio cobra el impuesto por metro cuadrado del terreno, para este año se ha fijado un costo de 0,10 Ctv. por cada  $m^2$ , la alcaldía ha decidido realizar un descuento de 4 Ctv. para aquellos terrenos que exceden los 10000  $m^2$ .

Análisis: para saber cuántos metros cuadrado tiene un terreno rectangular se aplica la fórmula ( $a = b \times h$ ), lo que implica que se debe solicitar el número de metros que tiene el terreno de largo y ancho, el resultado de dicha multiplicación debe comparar si el terreno excede o no los 10000  $m^2$  para definir el costo a pagar, la propuesta es la siguiente:

**Diagrama 6:**



*¿Qué datos entran a ser procesados?*

Dos valores que corresponden a *largo* y *ancho* de un terreno, éstos de representan por las variables *b* y *h*

*¿Qué resultados muestra?*

Total que debe pagar en los predios

*¿Qué condiciones se presentan en el proceso?*

Compara si el total de  $m^2$  es mayor que 10000, de ser así multiplica el total por 0.06 reduciendo los 4 Ctv. del descuento, caso contrario se multiplicará el total de  $m^2$  por 10 Ctv. normales.

Considere el siguiente problema: El Banco del Estado, ha designado para cada cuenta de las diferentes alcaldías, dos números claves, el primero para los alcaldes y segundo para los jefes financieros, los números combinados (Primero el del alcalde y después el del jefe financiero) permiten tener acceso a la cuenta del municipio para realizar transferencias; por equivocación del departamento de informática pide primero el número del jefe financiero y después el del alcalde, elabore un diagrama de flujo que intercambie los valores ingresados para enviarlos en el orden dispuesto por el Banco del Estado.

Este ejercicio muestra cómo podemos intercambiar valores entre variables, para lograrlo se plantea el siguiente análisis: Suponga que el número secreto del alcalde es 123 y que el número secreto del jefe financiero es 456, además suponga que la variable utilizada por el alcalde es *NSA* (Número Secreto del Alcalde), y la variable del jefe financiero es *NSJF* (Número Secreto del Jefe Financiero) y que los valores asignados son:  $NSA=456$ ,  $NSJF=123$ , asignados de forma incorrecta, si se hace que  $NSA=NSJF$ , entonces *NSA* estaría correcto, pero se perdería el número secreto del jefe financiero, ya que ambas variables tendrían 123, si se hace lo contrario tendríamos el mismo resultado, para evitar

esto se necesita de una variable *auxiliar* que mantenga uno de los numero a salvo mientras se intercambian, por ejemplo:

$Aux = NSJF$  esto significa que tanto  $Aux$  como  $NSJF$  tienen 123

$NSJF = NSA$  esto significa que tanto  $NSJF$  y  $NSA$  tienen 456

$NSA = Aux$  esto significa que  $NSA$  tomará 123

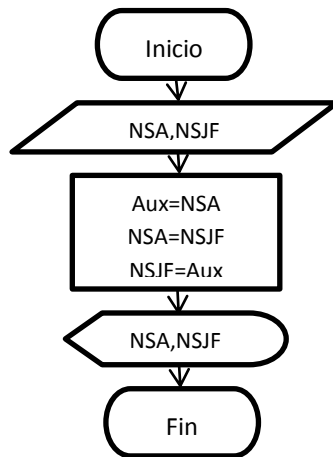
Cuando finaliza estas tres asignaciones  $NSA$  tiene el número secreto del alcalde 123 y  $NSJF$  tiene el número secreto del jefe financiero 456.

#### Diagrama 7:

*Iniciamos*

1. Recibir  $NSA$ ;
2. Recibir  $NSJF$ ;
3.  $Aux = NSA$ ;
4.  $NSA = NSJF$ ;
5.  $NSJF = Aux$ ;
6. Escribir  $NSA, NSJF$ ;

*Finalizamos*



*¿Qué datos entran a ser procesados?*

Dos números cualquiera

*¿Qué resultados muestra?*

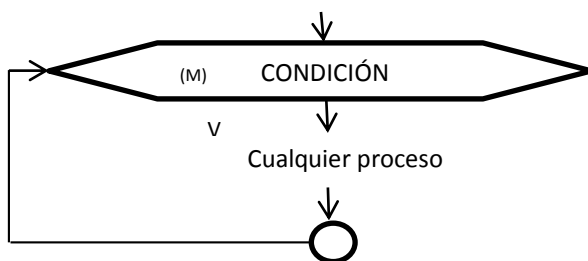
Cada variable con el número de la otra variable

*¿Qué condiciones se presentan en el proceso?*

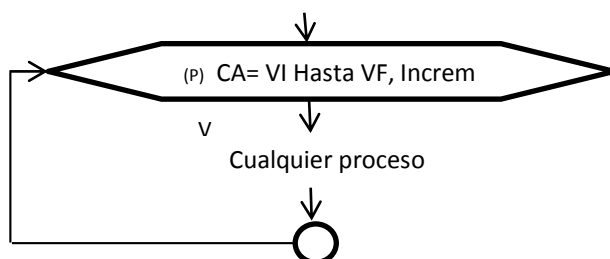
Ninguna

### FORMATOS UTILIZADOS PARA EXPRESAR CICLOS REPETITIVOS EN LOS DIAGRAMAS DE FLUJO

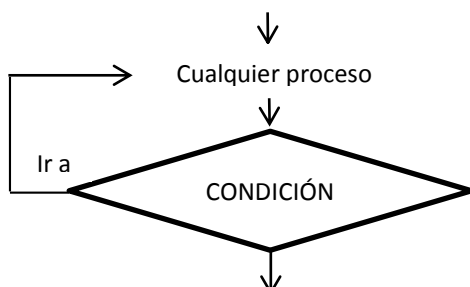
Los ciclos repetitivos estudiados hasta el momento son:



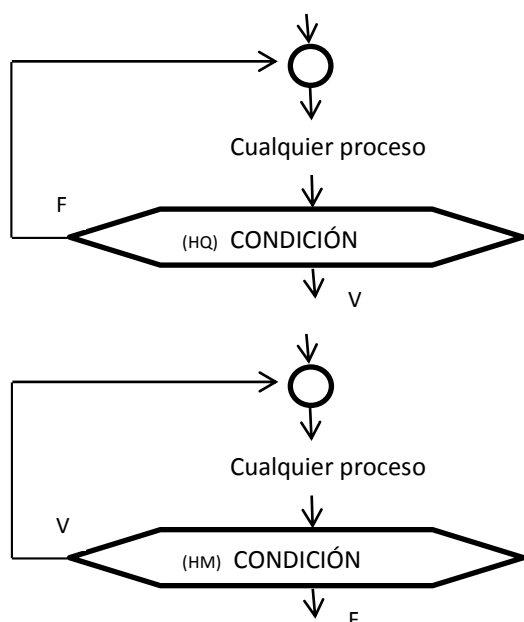
Formato del control *mientras* (M), tiene una entrada que parte de la condición de repetición y una salida que sigue después del conector límite que indica hasta donde se realizan los procesos repetitivos, los ciclos se cumplen mientras la condición es verdadera.



Formato del control *Para* (P), tiene una entrada que parte del control de repetición y una salida que sigue después del conector límite que indica hasta donde se realizan los procesos repetitivos, los ciclos se cumple de forma automática, donde el contador automático CA tomará los valores desde el valor inicial VI hasta el valor final VF, existe la posibilidad de incrementar escribiendo un valor positivo en *Increm* o decrementarse si utiliza un valor negativo.



Formato del control *Ir a*, tiene una entrada que parte desde cualquier proceso, obligatoriamente incluye una condición que evaluará si se realiza o no la repetición desde un determinado proceso, la salida es una de las alternativa de respuesta a la condición.

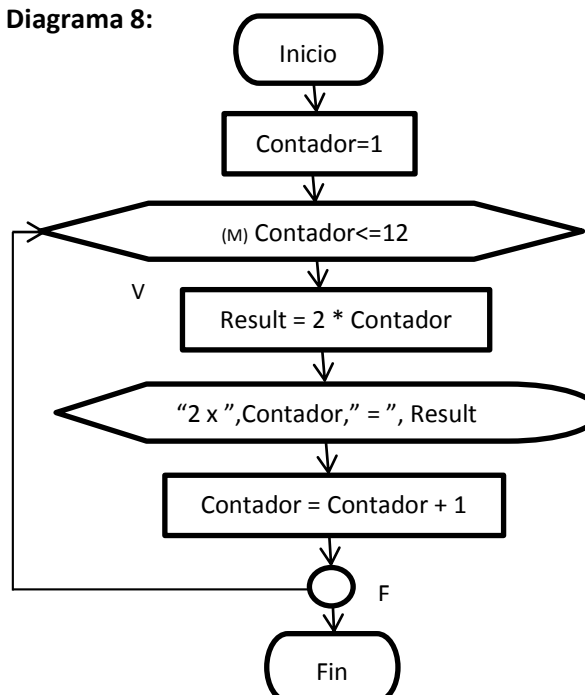


Formato del control *Repetir... Hasta que (HQ)*, tiene una entrada que parte desde el conector hasta la condición de repetición, su base se fundamenta en realizar los procesos repetitivos hasta que la condición sea verdadera.

Formato del control *Hacer... Mientras (HM)*, tiene una entrada que parte desde el conector hasta la condición de repetición, su base se fundamenta en realizar los procesos repetitivos mientras la condición sea verdadera.

El siguiente diagrama de flujo, muestra los pasos lógicos para escribir la tabla de multiplicar del dos:

**Diagrama 8:**



Condición de repetición	Valor de la Variable <b>Result</b>	Se muestra por pantalla	Valor de la Variable <b>Contador</b>
			(1) <b>1</b>
(2) <b>1</b> <= 12 (V)	(3) <b>2</b> = 2 * 1	(4) 2 x 1 = 2	(5) <b>2</b> = 1 + 1
(6) <b>2</b> <= 12 (V)	(7) <b>4</b> = 2 * 2	(8) 2 x 2 = 4	(9) <b>3</b> = 2 + 1
(10) <b>3</b> <= 12 (V)	(11) <b>6</b> = 2 * 3	(12) 2 x 3 = 6	(13) <b>4</b> = 3 + 1
(14) <b>4</b> <= 12 (V)	(15) <b>8</b> = 2 * 4	(16) 2 x 4 = 8	(17) <b>5</b> = 4 + 1
(18) <b>5</b> <= 12 (V)	(19) <b>10</b> = 2 * 5	(20) 2 x 5 = 10	(21) <b>6</b> = 5 + 1
(22) <b>6</b> <= 12 (V)	(23) <b>12</b> = 2 * 6	(24) 2 x 6 = 12	(25) <b>7</b> = 6 + 1
(26) <b>7</b> <= 12 (V)	(27) <b>14</b> = 2 * 7	(28) 2 x 7 = 14	(29) <b>8</b> = 7 + 1
(30) <b>8</b> <= 12 (V)	(31) <b>16</b> = 2 * 8	(32) 2 x 8 = 16	(33) <b>9</b> = 8 + 1
(34) <b>9</b> <= 12 (V)	(35) <b>18</b> = 2 * 9	(36) 2 x 9 = 18	(37) <b>10</b> = 9 + 1
(38) <b>10</b> <= 12 (V)	(39) <b>20</b> = 2 * 10	(40) 2 x 10 = 20	(41) <b>11</b> = 10 + 1
(42) <b>11</b> <= 12 (V)	(43) <b>22</b> = 2 * 11	(44) 2 x 11 = 22	(45) <b>12</b> = 11 + 1
(46) <b>12</b> <= 12 (V)	(47) <b>24</b> = 2 * 12	(48) 2 x 12 = 24	(49) <b>13</b> = 12 + 1
(50) <b>13</b> <= 12 (F)	Fin del programa		

Observe que el proceso de control a los ciclos repetitivos se ejecuta mientras el *Contador* sea menor o igual que 12 (V), una vez que la condición por el proceso sea falsa el programa continuará después del conector (F).

### Actividades de refuerzo (AR):

AR32. Modifique el diagrama 8 de forma que muestre las primeras 5 tablas de multiplicar.

AR33. Desarrolle un diagrama de flujo que cuente el número de dígitos de una cantidad cualquiera.

### **FUNCIONES MATEMÁTICAS**

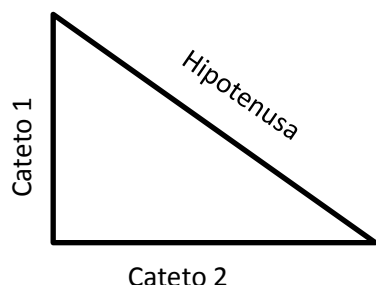
Las técnicas revisadas de desarrollo de la lógica de programación como los algoritmos, diagramas de flujo y los lenguajes de programación proporcionan instrucciones con propósitos específicos que devuelven un resultado y son conocidas como funciones, entre las más populares se tienen: Abs(Num) que devuelve el valor absoluto de cualquier número, Arctan(Num) devuelve el arco tangente de un número, Cos(Num) devuelve el coseno de un número, Sen(Num) devuelve el seno de un número, entre otras; estas instrucciones pueden realizar diferentes tipos de procesos

suministrándoles textos, valores o datos combinados, en esta etapa de aprendizaje se utilizarán funciones de cálculo matemáticos que realizan cálculos como la raíz cuadrada “sqrt()”, note que se trata de un nombre seguidos de paréntesis, funciona pasándole un valor o el contenido de una variable dentro del paréntesis, por ejemplo *sqrt(9)*, al realizar este proceso la función devolverá como resultado 3; Supongamos que asignamos 25 a una variable X=25, al aplicar la raíz cuadrada del valor de X tendríamos:

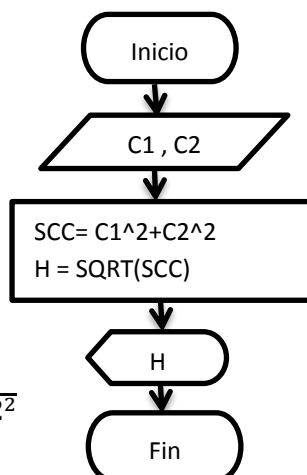
*sqrt(X)* al realizar esta instrucción la función devolverá como resultado 5.

Para ejemplificar el uso de la función *sqrt()*, el siguiente diagrama de flujo resuelve la hipotenusa de un triángulo:

**Diagrama 9:**



La fórmula es:  $H = \sqrt{C1^2 + C2^2}$



*¿Qué datos entran a ser procesados?*  
Dos números cualquiera que representan los catetos

*¿Qué resultados muestra?*  
La hipotenusa

*¿Qué condiciones se presentan en el proceso?*  
Ninguna

Observe que la fórmula para calcular la hipotenusa expresa que se suman los cuadrados de los catetos y posterior a esto se calcula la raíz cuadrada de la suma, el diagrama de flujo propuesto realiza lo estipulado en la formula, pero se pudo simplificar el proceso de la siguiente forma:  $H = \text{SQRT}(C1^2+C2^2)$ , ya que la función también puede recibir un cálculo como valor a resolver, en cuyo caso el computador resuelve primero el cálculo y después pasa el resultado a la función para encontrar la raíz cuadrada.

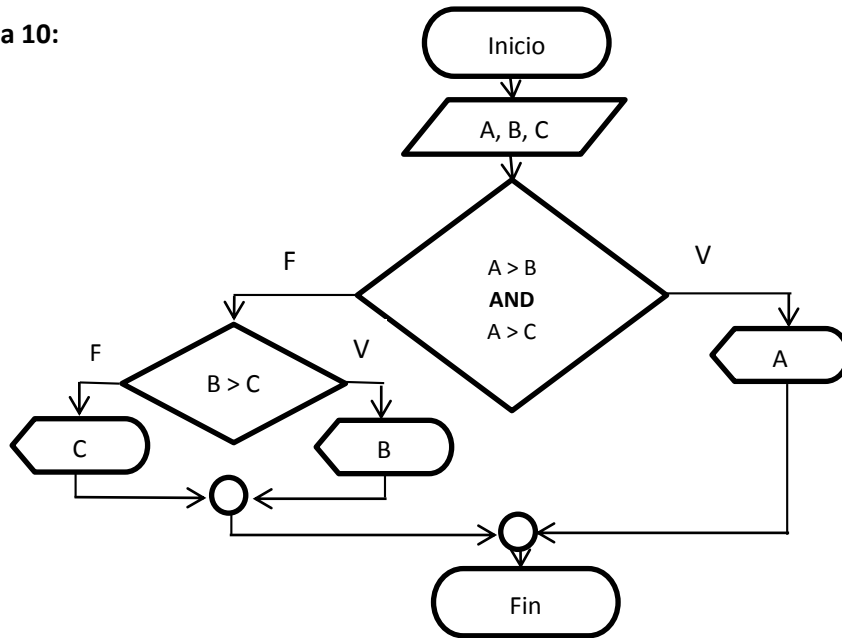
### **Actividades de refuerzo (AR):**

- AR34. Desarrolle un diagrama de flujo que permita mostrar los primeros 20 números múltiplos de 7.
- AR35. Se necesita un diagrama de flujo que permita al departamento de práctica pre profesional mostrar el número de estudiantes que están listo a recibir la certificación, cada estudiante debe realizar 3 prácticas, una vez realizadas se le entrega su certificación, elabore un diagrama que permita a un grupo de 30 estudiantes mostrar el total de certificados a entregar y el porcentaje en relación al total de estudiantes.
- AR36. Desarrolle un diagrama de flujo que muestre 5 veces 5, 4 veces 4, 3 veces 3 y así sucesivamente hasta el 1.



Para el siguiente ejercicio considere que se necesita un diagrama de flujo que reciba tres números, el programa debe mostrar el mayor de los tres:

**Diagrama 10:**



El diagrama propuesto como solución utiliza el operador lógico AND que permite simplificar y disminuir el número de condiciones para encontrar el mayor de los tres valores.

*¿Qué datos entran a ser procesados?*

Recibe tres números cualesquiera, contenidos en las variables A, B y C.

*¿Qué resultados muestra?*

Antes de finalizar mostrará el valor mayor de las tres cantidades.

*¿Qué condiciones se presentan en el proceso?*

Aplica 2 condiciones, la primera chequea si el contenido de la variable A es mayor a los contenidos de la variable B y C, al ser verdadero muestra el valor de A como mayor, caso contrario A queda descartado ya que el mayor puede estar entre las variables B o C, de hecho esa es la segunda condición, chequea si el valor de B es mayor que C para mostrar al mayor.

#### **Actividad de refuerzo (AR):**

AR37. Desarrolle un diagrama de flujo que encuentre el mayor y el menor entre tres números sin utilizar operadores lógicos.

Considere al siguiente ejemplo como un problema a resolver: La distribuidora ABC se encarga de distribuir alimentos, medicina, vituallas, bisutería y artículos para el hogar, ha recibido como política del estado, un comunicado que establece el porcentaje de impuesto a la venta de ciertos productos categorizados, la empresa para dar cumplimiento codificó las categorías en las siguientes mercaderías con su correspondiente porcentaje de impuesto:

CATEGORÍA	MERCADERÍA	PORCENTAJE DE IMPUESTO
1	Medicina	0%
2	Lácteos	5%
3	Bisutería	10%
4	Limpieza	8%
5	Panadería	0%
6	Vituallas	8%
7	Electrodomésticos	10%
9	Cárnicos	5%

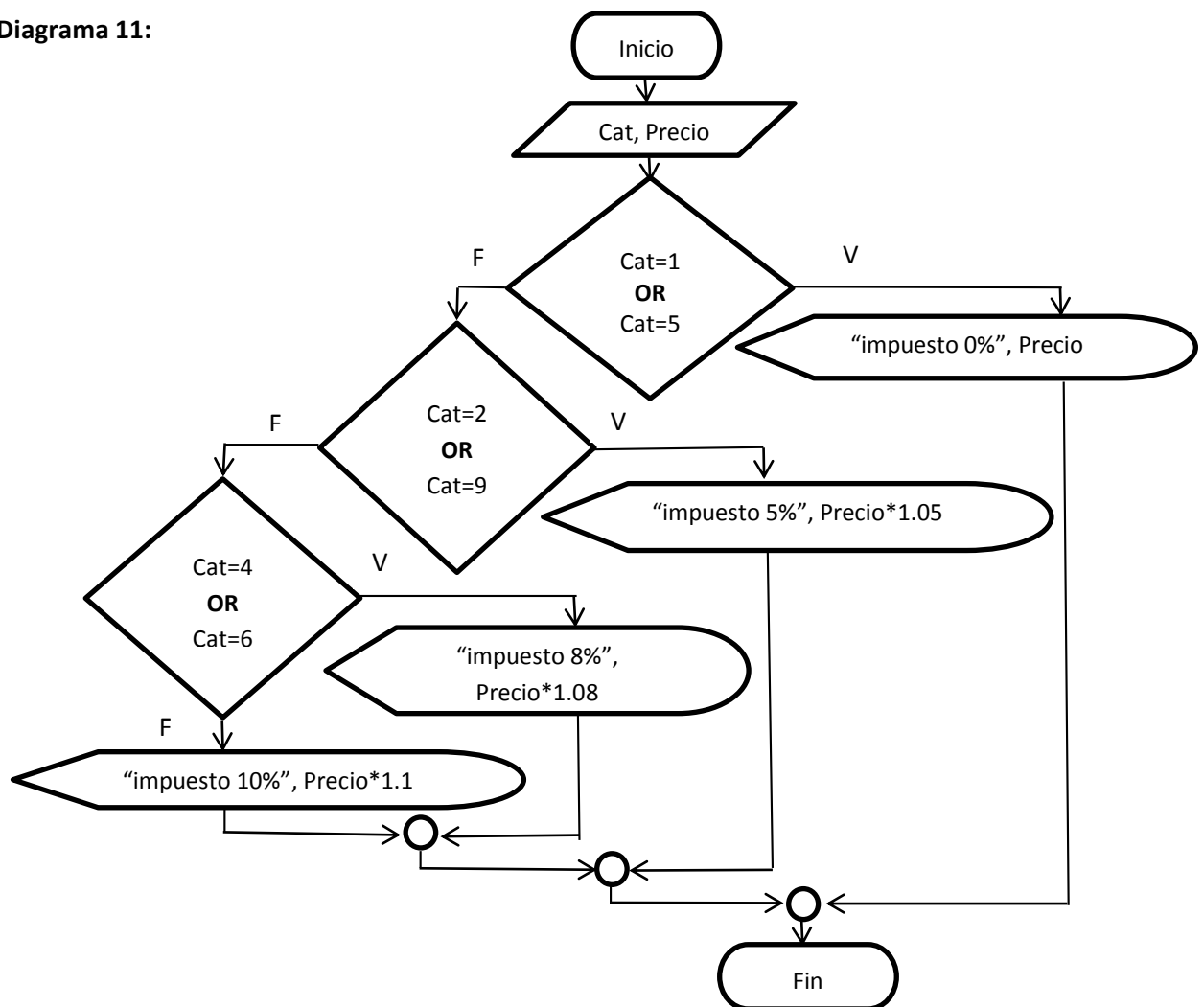
Desarrolle un diagrama de flujo que solicite la categoría y el precio de un producto cualquiera, el programa deberá mostrar el porcentaje de impuesto y su cálculo correspondiente.

Análisis: el código de la categoría define el porcentaje a incrementar en el precio del producto, el cálculo de incremento porcentual se lo podría aplicar de la siguiente forma: asumiendo que se ha comprado un producto de categoría 2 cuyo precio es \$ 12.00, a pagar sería  $12 + (12 * 5 / 100)$  lo que significa  $12 + 0.60$  dando un total de \$ 12.60.

Éste cálculo se lo puede simplificar de la siguiente forma  $12 * 1.05$ , considere que todo número multiplicado por la unidad es igual a la misma cantidad y el 0.05 es equivalente al 5 por ciento ( $5 / 100$ ), al resolver  $12 * 1.05$  el resultado es \$ 12.60 igual que el método anterior, queda a su criterio que método le resulta más entendible para su aplicación.

Observe que hay categorías que se repiten en cuanto al porcentaje de impuesto, es decir lácteos y cárnicos tienen el mismo 5%, bisuterías y electrodomésticos 10%, así también las otras categorías, para simplificar la propuesta a resolver se utiliza el operador lógico OR en las comparaciones, de forma que si una de las comparaciones coincide se realizará el mismo cálculo.

**Diagrama 11:**



*¿Qué datos entran a ser procesados?*

Se reciben dos números cualquiera, uno representa la categoría *Cat* y el otro representa el *Precio*.

*¿Qué resultados muestra?*

Antes de finalizar mostrará el precio incrementado de acuerdo a la categoría.

*¿Qué condiciones se presentan en el proceso?*

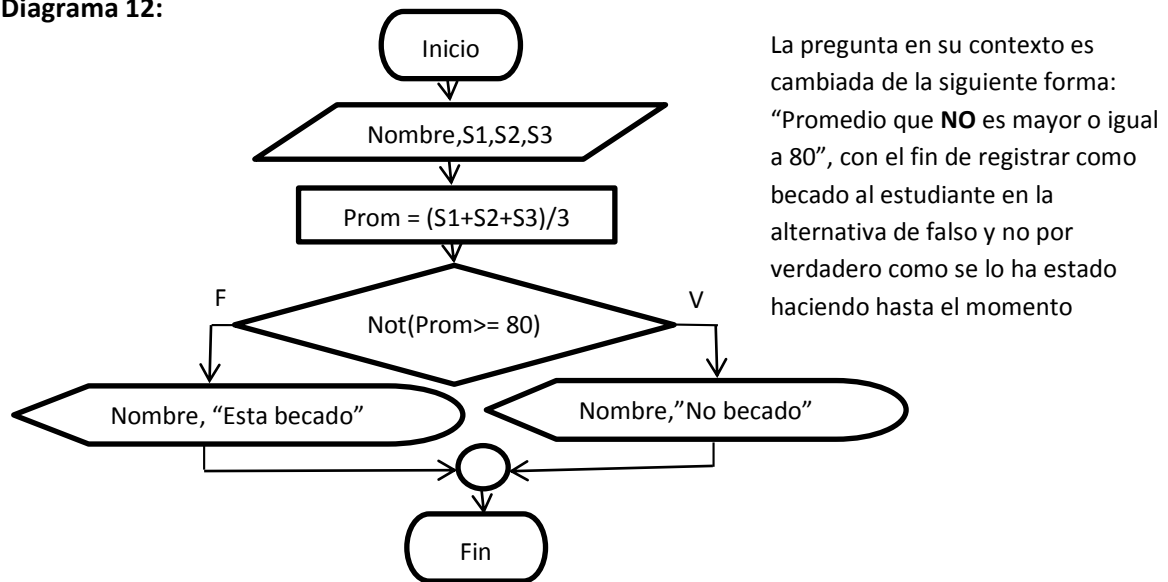
Se aplican tres condiciones, la primera verifica si la categoría ingresada es igual a 1 o 5, que para el caso no tiene incremento y o se realiza el cálculo del impuesto, la segunda condición verifica si la

categoría es igual a 2 o 9, para aplicar el incremento del 5%, la tercera condición verifica si la categoría es igual a 4 o 6 para incrementar en el precio el 8% de impuesto; y si el resultado es falso a la última comparación, asume que la categoría esta entre 3 o 7 para incrementar el 10%.

Suponga que se le propone resolver el siguiente problema: La Universidad está becando a los estudiantes que tengan un promedio mayor o igual a 80 en sus sumatorias finales, entienda que se reciben notas de las tres mejores materias que el estudiante aprobó, para esto el programa debe recibir el nombre del estudiante y su respectivas sumatorias, antes de finalizar debe mostrar un mensaje indicando si tiene o no la beca.

Análisis: Las notas finales en la Universidad es la sumatorias de todas las actividades realizadas durante el periodo lectivo con un máximo de 100 puntos y un mínimo de 70 puntos para aprobar la materia, se necesita Recibir tres sumatorias que son el equivalente a las tres mejores sumatorias recibidas de las materias aprobadas o las más altas en calificaciones, se deben promediar y verificar si el resultado es mayor o igual a 80 para obtener la beca.

**Diagrama 12:**

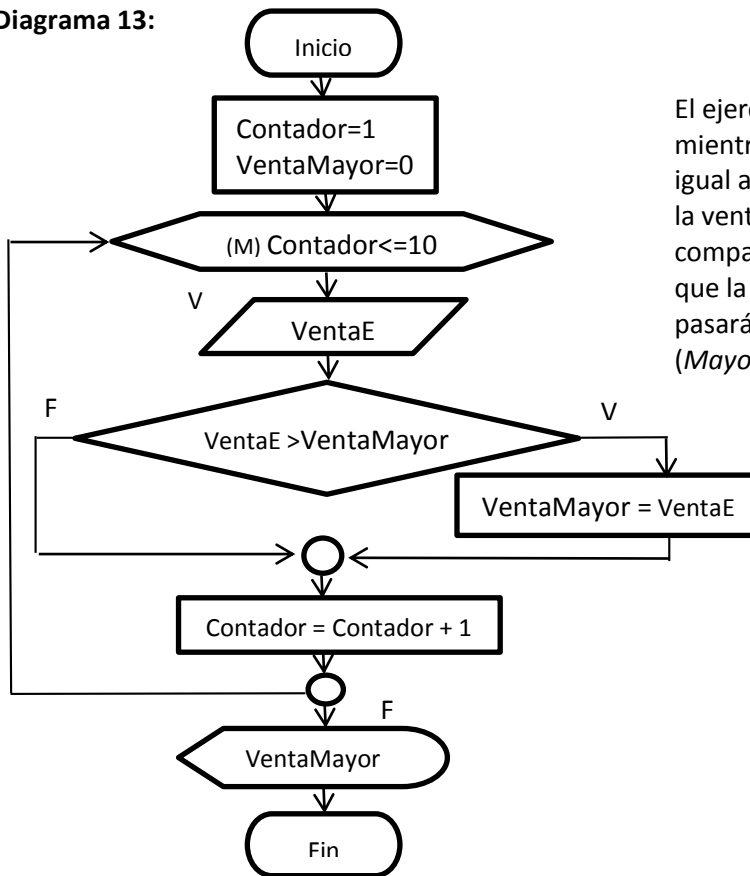


## EJERCICIOS CON CONDICIONES DE CASO Y CONTROL DE CICLOS REPETITIVOS

Para ejemplificar el control de ciclos repetitivos utilizando diagramas de flujo considere el siguiente problema: Una empresa que se dedica a la venta de vehículos cuenta con 10 empleados, todos han vendido y no se igualan en ventas, se necesita un diagrama de flujo que permita encontrar al empleado que más venta registra.

Análisis: Considere que se necesita una variable contador para controlar el ingreso de los 10 empleados, otra variable que contenga la cantidad en ventas de cada uno de los 10 vendedores, se necesita una variable que contenga el mayor de todos los ingresos de ventas, para lograrlo como estrategia se dará como valor inicial el cero, ya que el primer número que se ingrese como cantidad en ventas será mayor que cero y de ahí en adelante se comparará con las nuevas ventas de los otros empleados y solo será mayor el que cumpla con la condición de ser mayor.

Diagrama 13:

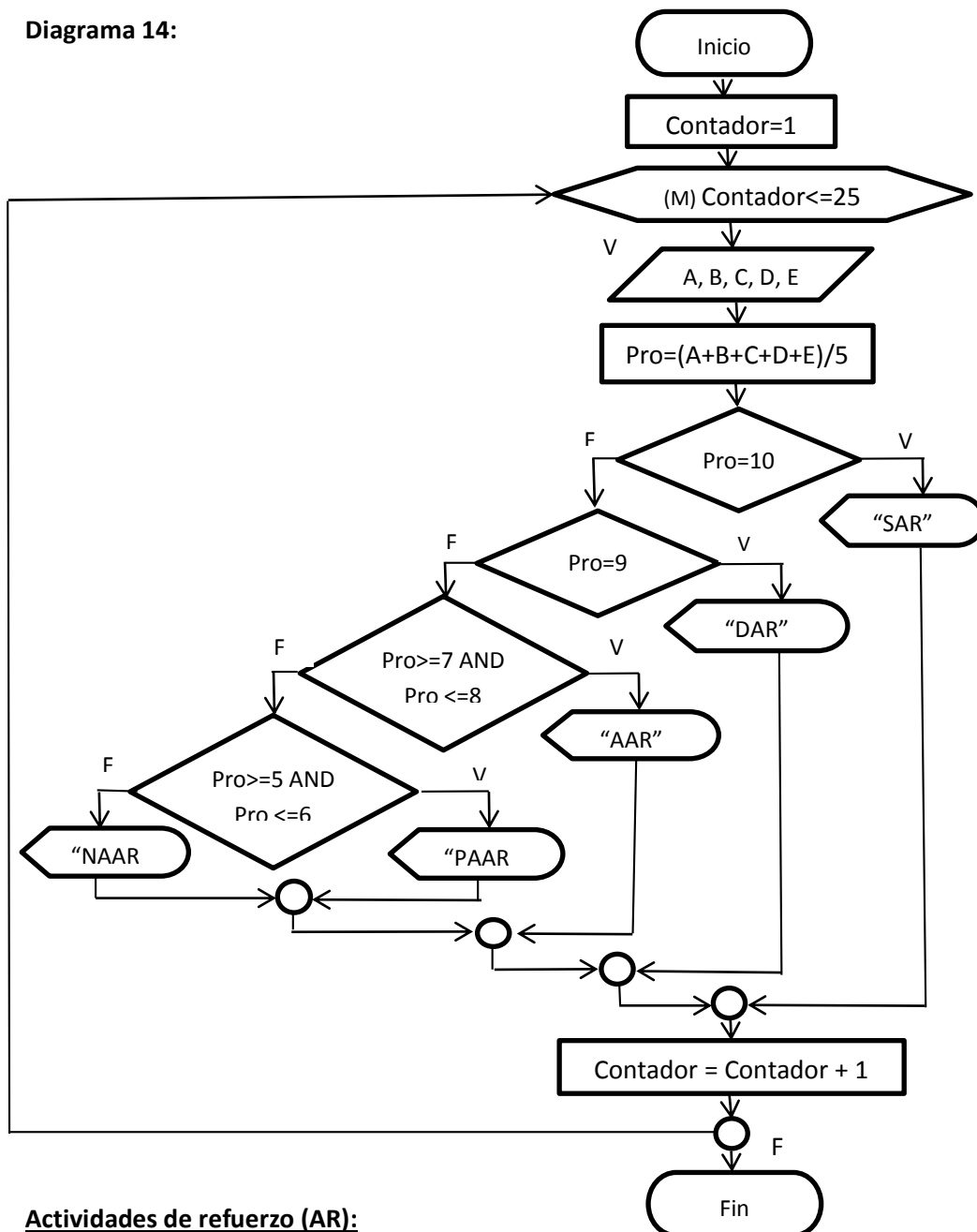


El ejercicio repetirá 10 veces mientras el *Contador* sea menor o igual a 10, por cada ciclo recibirá la venta del empleado (*VentaE*), y comparará si esa venta es mayor que la anterior, en caso de ser así pasará a ser la mayor (*MayorVenta = VentaE*).

Analice el siguiente problema: Una escuela de enseñanza básica necesita un programa que le permita calcular el promedio de 5 notas, el curso cuenta con 25 estudiantes, por cada estudiante se necesita mostrar en que categoría se encuentra considerando su promedio, en la siguiente tabla muestra cómo están categorizados:

ESCALA CUANTITATIVA	ESCALA CUALITATIVA
10	Supera los aprendizajes requeridos ( <b>SAR</b> ).
9	Domina los aprendizajes requeridos ( <b>DAR</b> )
7 - 8	Alcanza los aprendizajes requeridos ( <b>AAR</b> ).
5 - 6	Está próximo a alcanzar los aprendizajes requeridos ( <b>PAAR</b> )
≤ 4	No alcanza los aprendizajes requeridos ( <b>NAAR</b> ).

**Diagrama 14:**



**Actividades de refuerzo (AR):**

- AR38. Desarrolle un diagrama de flujo que encuentre la diferencia entre dos edades sin utilizar el operador aritmético de sustracción (-).
- AR39. Se necesita un diagrama de flujo que calcule el valor a pagar de una compra de 10 productos, la lógica debe solicitar el valor del producto y la cantidad comprada, antes de finalizar se debe mostrar el total de la compra más el incremento del IVA (12%).
- AR40. Desarrolle un diagrama de flujo que permita calcular el promedio general de un curso de 15 estudiantes, por cada estudiante se debe recibir tres notas y mostrar el promedio de cada uno.

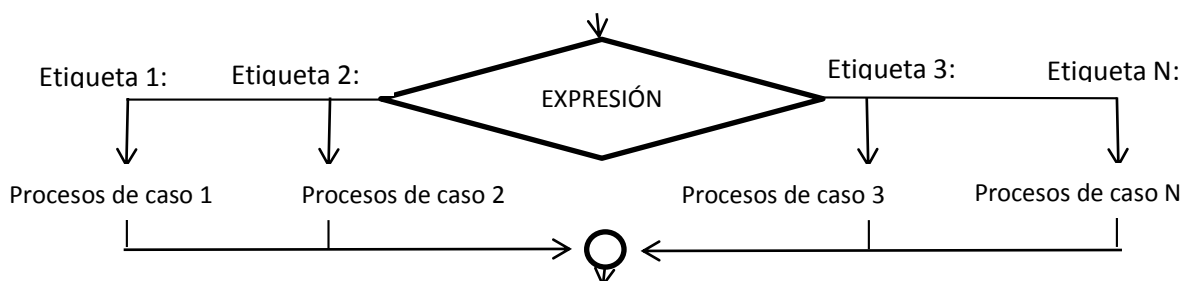
**ACTIVIDADES EXTRAS:**

- D1. Desarrollo un diagrama de flujo que solicite tres números, el programa debe mostrar el mayor y el menor de los tres números, considere que los números son diferentes.
- D2. La distribuidora de medicinas "NOVAMED" por aniversario desea premiar a los mejores empleados con el 25% de sus sueldo, existen visitadores médicos y entregadores de pedidos, para ser premiado deben tener un número de ventas superiores a 999 pedidos o los que tienen más de 40 horas ocupadas en la entrega de pedidos, considere el programa para un empleado cualquiera.
- D3. Modifique el ejercicio anterior de forma que permita aplicar el programa para 40 empleados

- D4. Aplicando el operador lógico NOT desarrolle un diagrama de flujo que permita ingresar un número, el programa deberá mostrar si el número es o no par.
- D5. La familia Pérez tiene registrado el año de nacimiento y el año de fallecimiento de 10 ancestros longevos de su árbol genealógico, el programa debe mostrar la edad máxima a la que han llegado en su familia.

### DIAGRAMAS DE FLUJO CON CONDICIONES DE CASO

El empleo de ésta técnica es poco utilizada en el desarrollo de soluciones algorítmicas, los formatos no están claros ya que no utiliza una figura geométrica única para las coincidencias de cada caso sino que hace uso de etiquetas para identificar la lista de casos, el formato que propone este texto es el siguiente:



Donde *Expresión* es el contenido que desea comparar con la lista de cada *Etiqueta*, considere que la comparación es únicamente de igualdad, *Etiqueta 1:*, al igual que todas las etiquetas poseen la palabra *Caso* seguido de una lista de posibles valores que coincidirán o no con el valor de expresión, observe que se utiliza los dos puntos (:) para definir el final de la etiqueta, los procesos de cada caso pueden ser cualesquiera que se considere necesario para resolver el problema, incluso otra condición de caso.

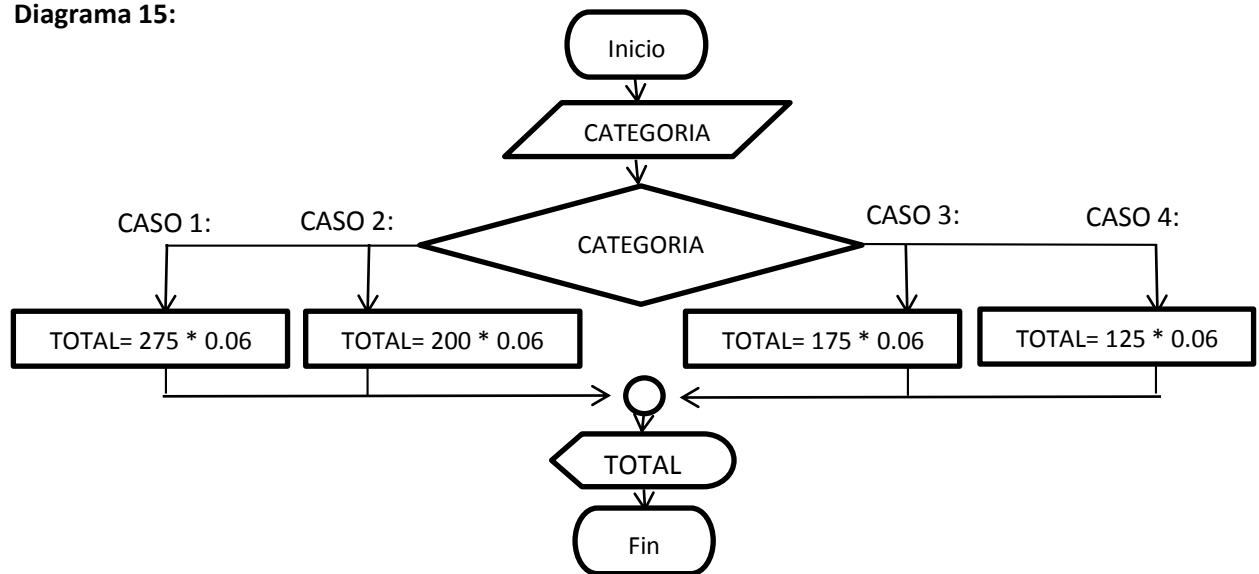
Para ilustrar el uso de ésta herramienta considere el siguiente ejemplo: El bono de desarrollo humano del gobierno nacional ha incluido dentro de sus beneficios subsidiar una cantidad de kilovatios hora dependiendo de la categoría en la que se encuentra el beneficiado, para esto ha creado una tabla que define la categoría y el beneficio:

CATEGORÍA	KILOVATIOS HORA	CATEGORÍA	KILOVATIOS HORA
1	275	3	175
2	200	4	125

Para mostrar los beneficios a la ciudadanía, se ha creado mesas de información para que los beneficiarios puedan consultar sus ventajas; Para dar cumplimiento se necesita un diagrama de flujo que permita solicitar únicamente la categoría del beneficiado, el programa deberá mostrar el número de kilovatios hora y la cantidad de dinero que el estado le subsidiará considerando que el kilovatio hora tiene un costo de 0.06 ctv.

Análisis: asumiendo que la categoría del beneficiario es dos, para calcular dinero subsidiado usted debe multiplicar  $200 \times 0.06$ , ya que cada kilovatio hora tiene un costo de 6 Ctv., considerando el ejemplo el programa debe mostrar como subsidio \$ 12.00.

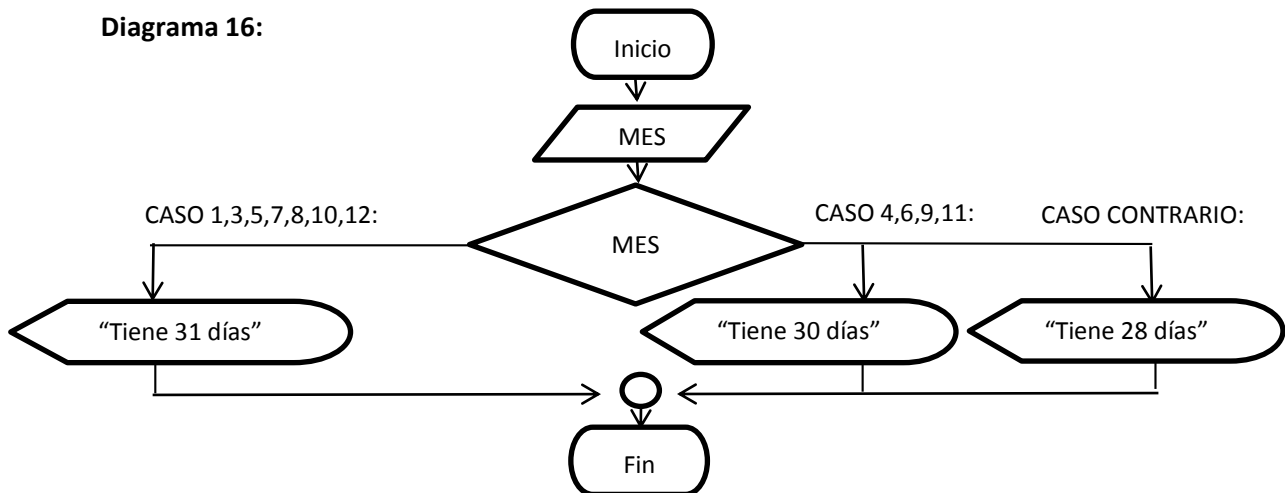
**Diagrama 15:**



La condición es el valor que tiene la variable *Categoría* y las comparaciones de igualdad son los valores que tienen cada caso, las acciones de respuesta a cada caso se producen después de los dos puntos (:) en cada secuencia de flecha, terminadas las acciones continúa al conector y de ahí al siguiente proceso.

Para considerar el uso de lista de valores a comparar y del caso contrario, el siguiente diagrama de flujo muestra en mensaje el número de días que posee un mes, para lograrlo se requiere el ingreso del número correspondiente del mes, como ejemplo del uso de “condiciones de caso”, el diagrama quedaría de la siguiente forma:

**Diagrama 16:**



#### **Actividades de refuerzo (AR):**

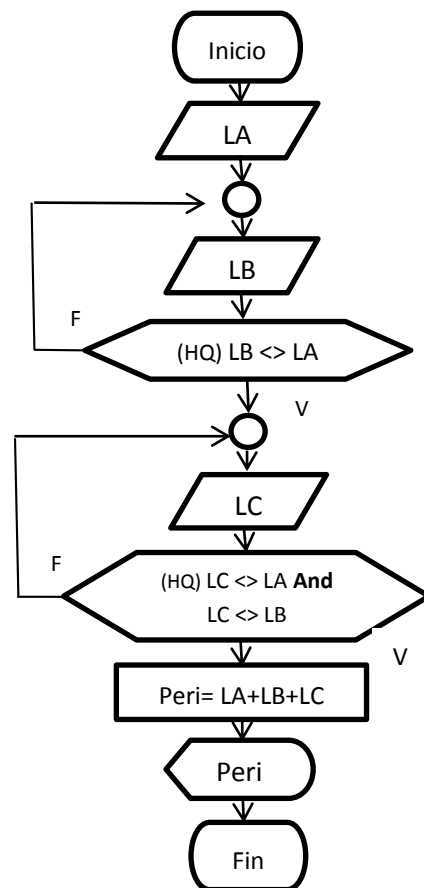
- AR41. La federación de fútbol barrial ha convocado a un grupo de 30 niños y jóvenes entre 5 y 18 años de edad, para agruparlos por categoría y formar las diferentes selecciones. Elabore un diagrama utilizando condiciones de casos que cuenten cuantos están entre 5 y 10, cuantos están entre 11 y 15, y cuantos están entre 16 y 18.
- AR42. Se necesita un diagrama de flujo que, utilizando condiciones de caso permita realizar una operación entre suma, resta, multiplicación, división o potencia, entre dos números.
- AR43. Desarrolle un algoritmo que permita calcular el promedio general de un curso de 20 estudiantes, por cada estudiante se debe recibir su promedio; antes de finalizar se debe mostrar cuantos regulares (1-5), buenos (6-8) y excelentes (9-10).

## PROCESOS REPETITIVOS CONTROLADOS CON “REPETIR ... HASTA QUE”

Esta técnica basada en el control condicional es utilizada por algunos lenguajes de programación, su principal uso se basa en la validación de datos y procesos, esta práctica de control al menos ejecuta una vez el proceso y para volver a repetir necesitará que no se cumpla la condición, es decir realiza el siguiente ciclo repetitivo cuando la respuesta a la condición es falsa y sale del ciclo cuando la respuesta a la condición es verdadera.

Para ilustrar esta definición considere el siguiente ejemplo: se requiere calcular el perímetro de un triángulo escaleno (tiene los tres lados desiguales), para esto el diagrama debe validar que el ingreso de los tres lados deben ser desiguales:

Diagrama 17:



*¿Qué datos entran a ser procesados?*

Tres números cualquiera validados para que obligatoriamente sean diferentes

*¿Qué resultados muestra?*

El perímetro del triángulo, aritméticamente la suma de sus lados

*¿Qué condiciones se presentan en el proceso?*

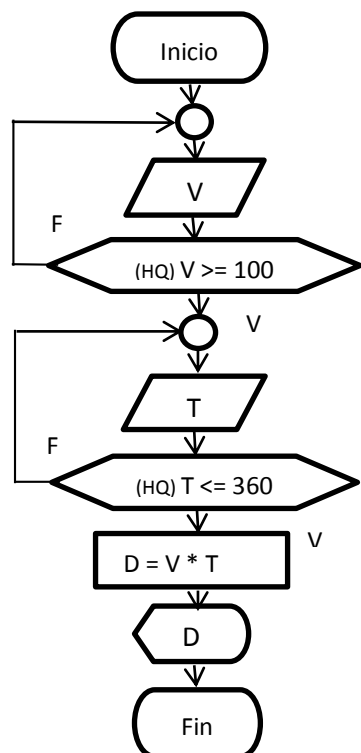
El primer lado se lo recibe sin ninguna restricción, el segundo lado está condicionado a ser diferente del primer lado y es aquí donde se presenta la primer condición, el tercer lado está condicionado a ser diferente al primer y segundo lado, por lo que la segunda condición tiene el operador lógico *And* que obliga a que ambas preguntas cumplan la condición.

Considere el siguiente problema a desarrollar: Proponga un diagrama de flujo que permita calcular la distancia recorrida en metros por un automóvil que tiene una velocidad constante (metros sobre segundos) durante un tiempo determinado, para resolver el diagrama considere que el movimiento es rectilíneo uniforme, obligatoriamente la velocidad no puede ser menor a 100 m/s y el tiempo no puede pasar los 360 segundos.

Análisis: el cálculo del problema planteado no es más que aplicar la siguiente fórmula ( $d = v * t$ ); es importante considerar que para resolver el problema, antes de realizar el cálculo se debe validar el ingreso de los valores que corresponden a velocidad y a tiempo.



**Diagrama 18:**



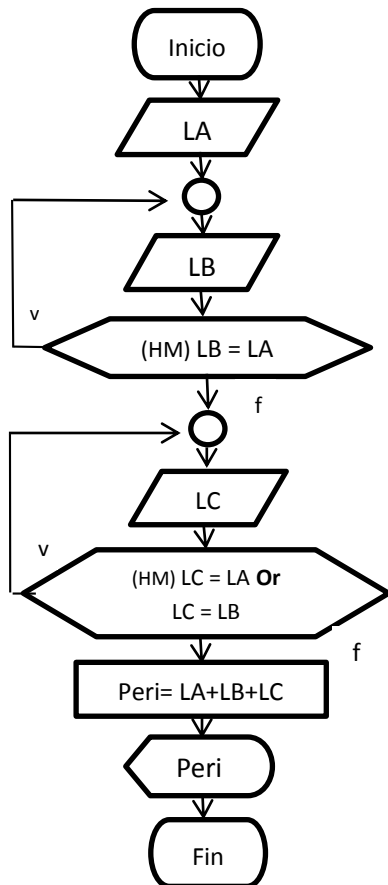
### **PROCESOS REPETITIVOS CONTROLADOS CON “HACER ... MIENTRAS”**

Esta técnica basada en el control condicional es ampliamente utilizada por casi todos los lenguajes de programación, su principal uso al igual que el control “Repetir... hasta que” se basa en aplicar la validación de datos y procesos, esta práctica de control al menos ejecuta una vez el proceso que se desea repetir, para que el proceso se vuelva realizar necesitara que la condición se cumpla, es decir repite mientras la condición de como respuesta verdadero y sale cuando la respuesta a la condición sea falsa.

Para ilustrar la definición del control “Hacer... Mientras” considere la utilización de los mismos ejemplos explicados en el control “Repetir... hasta que”, con la intención de diferenciar las dos estructuras:

En el ejemplo que requiere calcular el perímetro de un triángulo escaleno (tiene los tres lados desiguales), se utilizará el control “Hacer... mientras” para validar que el ingreso de los tres lados sean desiguales:

**Diagrama 19:**



*¿Qué datos entran a ser procesados?*

Tres números cualquiera validados para que obligatoriamente sean diferentes

*¿Qué resultados muestra?*

El perímetro del triángulo, aritméticamente la suma de sus lados

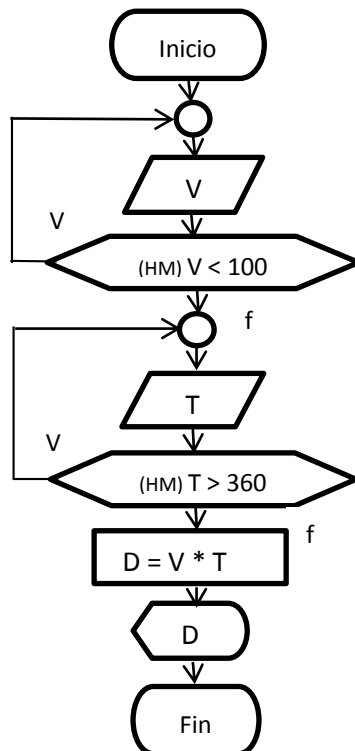
*¿Qué condiciones se presentan en el proceso?*

El primer lado se lo recibe sin ninguna restricción, el segundo lado está condicionado a ser diferente del primer lado y es aquí donde se presenta la primera condición, el tercer lado está condicionado a ser diferente al primer y segundo lado, por lo que la segunda condición tiene el operador lógico *And* que obliga a que ambas preguntas cumplan la condición.

Ahora en el siguiente problema a explicar: Se propone un diagrama de flujo que permite calcular la distancia recorrida en metros por un automóvil que tiene una velocidad constante (metros sobre segundos) durante un tiempo determinado, considere que el movimiento es rectilíneo y uniforme, para resolver el problema es obligatorio que la velocidad no pueda ser

menor a 100 m/s y el tiempo no puede pasar los 360 segundos.

**Diagrama 20:**



Análisis: el cálculo del problema planteado no es más que aplicar la siguiente fórmula ( $d = v * t$ ); es importante considerar que para resolver el problema, antes de realizar el cálculo se debe validar el ingreso de los valores que corresponden a velocidad y a tiempo.

### Actividades de refuerzo (AR):

AR44. Desarrolle un diagrama de flujo que permita calcular el área de una circunferencia cuyo radio debe estar entre 16 y 22.

AR45. Se necesita un diagrama de flujo que permita solicitar una cantidad múltiplo de 10, el diagrama debe mostrar la cantidad de números pares positivos menores a dicha cantidad.

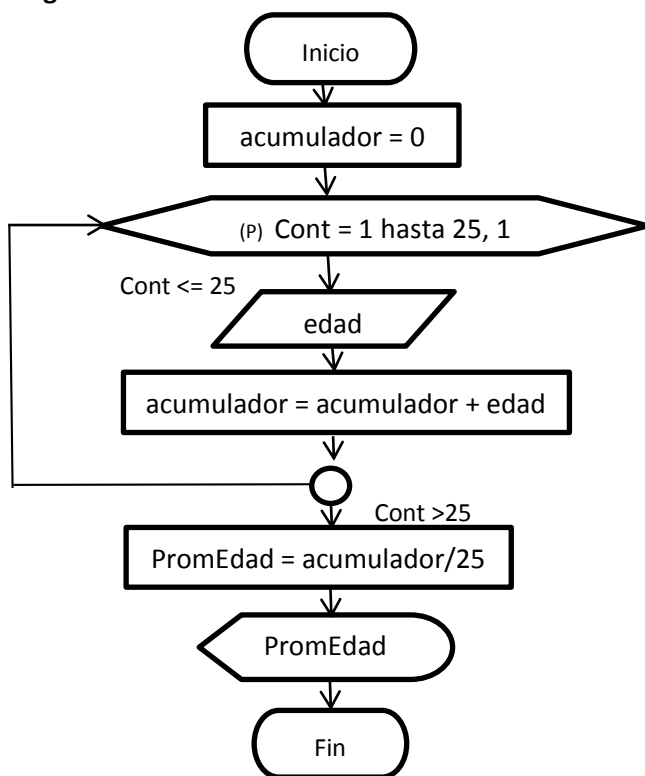
AR46. Desarrolle un algoritmo que permita recibir una nota entre 1 y 100 de un postulante que ha recibido un taller para un puesto de trabajo, se necesita mostrar si el postulante ha aprobado o reprobado la capacitación, se considera aprobado si obtiene una nota mayor a 70

### **PROCESOS REPETITIVOS CONTROLADOS CON CONTADORES AUTOMÁTICOS**

Al igual que los algoritmos tratados en apartados anteriores, que utilizan la técnica del “Para o Desde” para facilitar el control de procesos repetitivos, los diagramas de flujo también ofrecen la herramienta de control automático, definiendo un valor inicial y un valor final para establecer un número predeterminado de repeticiones.

Para ilustrar de mejor forma esta definición, considere el siguiente ejemplo: Se necesita un diagrama de flujo que permita calcular las edades promedio de un grupo de 25 jugadores juveniles, por cada jugador se debe recibir la edad.

**Diagrama 21:**



*¿Qué datos entran a ser procesados?*

Recibe 25 números cualesquiera que representan las edades de cada uno de los jugadores, el diagrama utiliza la variable *edad* para tomar una por una en cada ciclo de repetición, así la variable *acumulador* será utilizada para contener la suma de todas las edades.

*¿Qué resultados muestra?*

Antes de finalizar mostrará el promedio de las 25 edades.

*¿Qué condiciones se presentan en el proceso?*

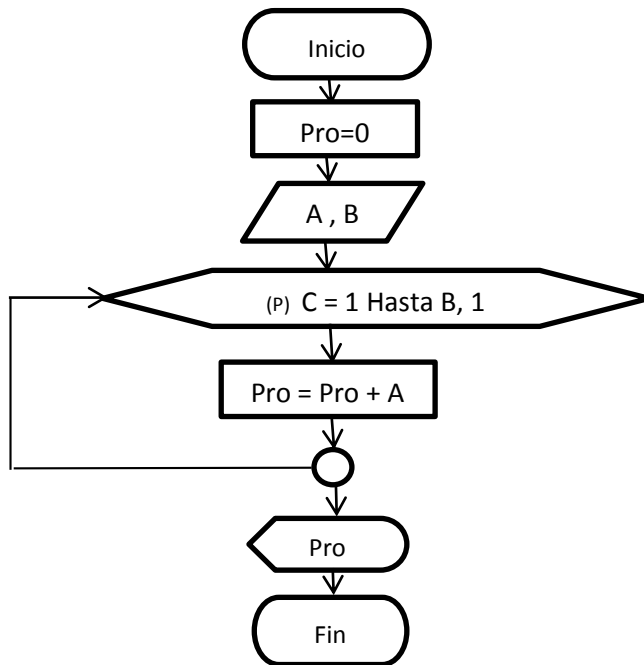
Solo una implícita, el proceso automático de repetición se cumple cuando *Cont* es menor o igual que 25

Considere el siguiente problema: Se necesita un diagrama de flujo que muestre

el producto de una multiplicación entre dos números cualquiera, la condición radica en NO utilizar el operador aritmético de multiplicación.

Análisis: Técnicamente el resultado de una multiplicación se la define como el número que especifica las veces que otro número debe sumarse, por ejemplo 5x4 el resultado es 20; podríamos decir que el 4 se suma 5 veces o lo contrario el 5 se suma 4 veces.

**Diagrama 22:**

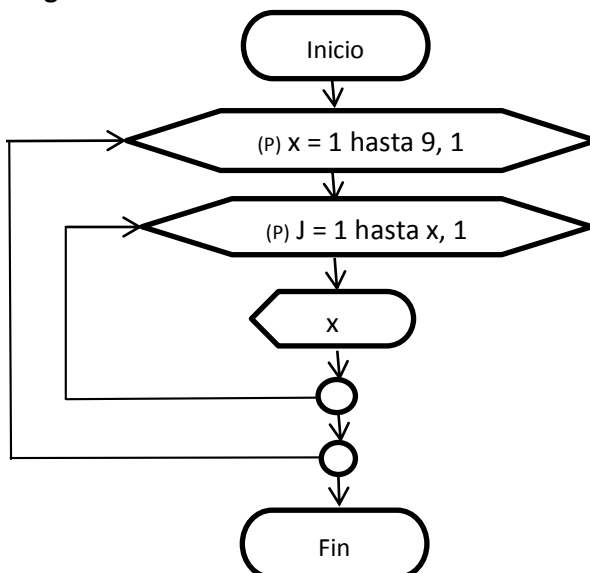


Suponga que se le plantea el siguiente problema: Desarrolle un diagrama de flujo que permita mostrar una vez 1, dos veces 2, tres veces 3, cuatro veces 4 y así sucesivamente hasta el nueve.

Análisis: La solución al problema planteado requiere de dos controles de repetición, así el control de repetición externo se encargará de recorrer los valores desde el 1 hasta el 9 (valores que se mostrarán uno a uno), y el segundo control de repetición se ejecutará dentro del primer control con la finalidad de repetir desde uno hasta el valor de la variable contador del primer control, por ejemplo si el contador del primer ciclo

de repetición vale 5, el segundo ciclo irá de 1 a 5, lo que significa que mostrará cinco veces 5.

**Diagrama 23:**



#### **Actividades de refuerzo (AR):**

AR47. Desarrolle un diagrama de flujo que calcule el pago de 50 oficiales de obra, para ello se necesita que por cada oficial se pida el número de horas trabajadas y el valor de la hora, por ley del estado las horas extras (Las que exceden de las 40 horas normales) de le debe incrementar el 50% del valor normal de la hora.

AR48. Modifique el diagrama 23 de forma que muestre la sumatoria de cada número mostrado en secuencia.

AR49. Desarrolle un algoritmo que permita del 1 al 100, mostrar cuantos son múltiplos de 3 y cuantos son múltiplos de 7.

## RESUMEN

El diagrama de flujo es una herramienta cuya utilización facilita el entendimiento de las instrucciones de forma gráfica, simplifica la comprensión de los algoritmos complejos y reduce la narración de las instrucciones, incluye figuras geométricas que reemplazan a las palabras pseudocodificadas, cada figura es única y tiene su propia definición de acción, los pasos a cumplir no están numerados ya que se utiliza flechas que indican el siguiente paso o acción a seguir, cada figura posee una entrada y una salida a excepción de las figuras condicionales e inicio y fin, cada estructura de control tiene sus propias normas y formas de uso que deben cumplirse para identificarlas y conocer su propósito, en la parte interna de cada figura geométrica puede incluir variables, constantes, operadores lógicos, operadores de comparación y operadores aritméticos, los diagramas de flujo empiezan con un óvalo que dice inicio y finaliza con otro óvalo que dice fin, los diagramas se leen de arriba hacia abajo o de izquierda a derecha, las figuras como el rombo (para incluir condiciones), la preparación (para controlar ciclos repetitivos) utilizan etiquetas en su parte externa, éstas permiten señalar las diferentes alternativas en respuesta a las condiciones, las etiquetas pueden incluir falso (f), verdadero (v), si (s), no (n) y en algunos casos comparaciones explicativas.

## IMPORTANTE

En sí los diagramas de flujo permiten simplificar el desarrollo y el análisis de las diferentes propuestas que proponen resolver un problema algorítmico, pero para desarrollar un programa que le dé aplicabilidad a la computadora se necesita obligatoriamente de un lenguaje de programación, ya que éstos poseen las instrucciones adecuadas que le permiten al computador entender y ejecutar las órdenes dadas por el programador, los diagramas son perfectos para analizar y proponer pequeñas soluciones complejas pero los lenguajes de programación ofrecen la alternativa de realizar programas más grandes e interactuar con los diferentes dispositivos físicos que se conectan a un computador.

## CAPITULO IV

### LENGUAJE DE PROGRAMACIÓN

En este capítulo se propone el desarrollo de programas mediante el uso de un lenguaje de programación, con la intención de utilizarlo como una herramienta de desarrollo en la aplicación de la lógica estudiada en los capítulos anteriores, todo programador debe tener claro las posibilidades que ofrece un lenguaje de programación, simbología utilizada, reglas de redacción, estructura y significado de control, instrucciones y expresiones que el lenguaje ofrece para que el programador realice las diferentes operaciones de control sobre el computador de forma física y lógica.

El estudio de este capítulo permitirá al lector introducirse en el significado e importancia de utilizar un lenguaje de programación, explica la importancia de conocer y aplicar las reglas de creación de un programa, define la forma de entender la base lógica de aplicar elementos nuevos e incorporarlos al programa, conocer la estructura y la organización interna de los diferentes elementos como las variables y los tipos de datos que aceptan, expone la codificación interna de almacenamiento que utilizan los sistemas operativos, incluye la explicación matemática de cómo se realizan los cálculos binarios, clasifica los diferentes caracteres y constantes que son utilizados como formatos para la entrada y salida de información para los diferentes tipos de datos, explica cómo se realizan la entrada de datos por teclado y la salidas de datos mediante la pantalla, ejemplifica la simbología utilizada en los operadores aritméticos, de comparación y lógicos, revela la forma como se definen y utilizan las constantes y las asignaciones condicionadas y describe la identificación de los diferentes errores que se pueden cometer al crear un programa.

**OBJETIVO DE APRENDIZAJE PARA EL SIGUIENTE APARTADO:** Aplicar la lógica en el desarrollo de programas interpretados por el ordenador, conoce las diferentes sintaxis, reconocer las posibilidades que ofrece el lenguaje y aplica las soluciones a los problemas planteados en el computador.

## PREÁMBULO A LOS LENGUAJES DE PROGRAMACIÓN

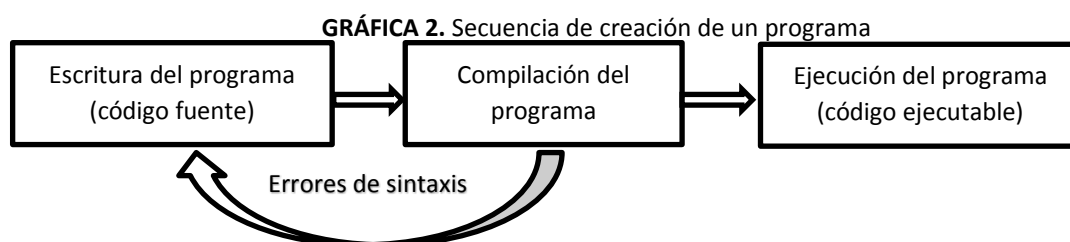
Los equipos digitales inteligentes como celulares, Tablet, computadores entre otros, utilizan software para maximizar sus posibilidades de gestión, los software o programas se constituyen en una secuencia de ordenes denominadas *instrucciones* que el equipo entiende y cumple una a una hasta lograr sus objetivos, estas instrucciones están escritas en un lenguaje de programación que el computador u otros dispositivos inteligentes lo entienden y obedecen.

Cuando se creó el computador inicialmente no existían lenguajes de programación, las instrucciones se limitaban a dar instrucciones mediante pulsos de encendido y apagado así como la provocación de interrupciones eléctricas, el lenguaje de programación nace con el desarrollo del lenguaje ensamblador (1950), pero el desarrollo de los programas más simples eran demasiados extensos, así la necesidad de desarrollar soluciones más inteligentes provocó la creación de los lenguajes de alto nivel, a lo largo de la historia se han creado varias propuestas de lenguajes, como ilustración técnica se muestran los nombres de los lenguajes de alto nivel más utilizados y por cronología de desarrollo: FORTAN (1957), LISP (1958), COBOL (1960), SNOBOL (1962), BASIC (1964), PASCAL (1970), para tener un mayor control de la máquina y desarrollar aplicaciones más complejas se crea un lenguaje de medio nivel llamado “*Lenguaje C*” (1972), se lo define como lenguaje de medio nivel por su cercanía al lenguaje ensamblador, posterior a éste se crean C++ (1983), PERL (1987), HTML (1991), Phython (1991), entre otros.

Para desarrollar programas que puedan ser probados en la computadora, este libro propone utilizar lenguaje C como herramienta de programación, ya que es uno de los lenguajes más utilizados en el campo investigativo, es importante considerar que muchos de los lenguajes de programación actuales utiliza la estructura de desarrollo de lenguaje C, posee recursos de entradas y salidas de bajo y alto nivel, además fue utilizado para desarrollar varios sistemas operativos como las versiones iniciales de Windows y las múltiples variedades de Linux, entre otros.

Los lenguajes de programación necesitan de un software que les permita a los programadores *editar* la escritura y creación de los programas (a la escritura del programa se le denomina código fuente), otro software que se necesita es el compilador/interprete, éste se encarga de verificar la sintaxis (reglas de escritura) para que el lenguaje entienda y pueda obedecer las instrucciones dadas, una vez compilado este software permite traducir de lenguaje de programación a lenguaje de máquina, un programa compilado y convertido en lenguaje de máquina, puede ser ejecutado (el lenguaje de máquina solo es entendido por el computador) sin la necesidad de utilizar el compilador/interprete (al programa traducido se le denomina ejecutable) para que el computador mediante el intérprete del sistema operativo pueda entenderlo y aplicar instrucción por instrucción la lógica del programa.

Es importante entender que un programa no puede ejecutarse si presenta errores de sintaxis, esto quiere decir que el compilador/interprete no puede obedecer una instrucción que no entiende, técnicamente no puede existir errores de incumplimiento en las normas que exige el lenguaje de programación para escribir un programa, por lo que antes de ejecutar (el computador debe obedecer las instrucciones del programa) éste debe ser editado, después compilado (Detectar errores de sintaxis), por último deber ser probado y compilado a lenguaje de máquina para que el sistema operativo pueda interpretarlo y obedecer las instrucciones contenidas en el programa, como podrá notar, un programa en cada etapa sufre cambios importantes antes de ser considerado una aplicación (software que cumple un propósito):



**Fuente:** Autores del libro

Cuando se desarrolla un programa, normalmente la edición del código fuente siempre es almacenada como un archivo plano de texto y una vez compilado se convierte en código objeto que se representa en un archivo ejecutable.

## ESTRUCTURA DE UN PROGRAMA EN C.

Un programa escrito en lenguaje C puede ser probado en varios sistemas operativos, esto se debe a que las reglas impuestas por el lenguaje exigen el cumplimiento de sus estructuras, debido a las potencialidades que ofrece el lenguaje, en la actualidad existen varios editores y compilador/interprete, cada uno de ellos ofrece características propias con herramientas de desarrollo incorporadas que facilitan el desarrollo de los programas, sin importar el editor y compilador un programa debe cumplir la estructura que incluye, la cabecera, directivas del preprocesador y funciones.

Para explicar la estructura de un programa en lenguaje C considere el siguiente ejemplo, que muestra las diferentes áreas mencionadas y permite desarrollar el cálculo de una suma de dos números ingresados por teclado y a éste resultado se le resta 10:

```
C1: /* Cabecera del programa, normalmente nuestra información del programa y no es obligatorio
*/
#include <stdio.h>
#define CONSTANTE 10
main()
{
    int a, b, resultado;
    printf("Ingrese un número:");
    scanf("%i",&a);
    printf("Ingrese otro número:");
    scanf("%i",&b);
    resultado = a + b - CONSTANTE;
    printf("El resultado es:%i",resultado);
    return 0;
}
```

Sección de Directivas del preprocesador

Sección de funciones, aquí la función main() es la principal ya que es donde se ejecutan las instrucciones del programa

Todos los programas que se presentan en el libro se numeran utilizando la letra C seguido de una secuencia numérica, así el lector podrá utilizar la referencia numerada para probar el archivo fuente que se adjunta al libro, por ejemplo para probar C1 que explica la estructura de un programa en lenguaje C, se podrá abrir el archivo "C1.cpp" que se encuentra en el CD adjunto al libro.

En el ejemplo observe que un programa escrito en lenguaje C, las instrucciones se escribe en *minúsculas*, cada instrucción termina en punto y coma (;), en una línea puede incluir varias instrucciones, los nombres de variable, constantes, funciones o cualquier otro elemento cuya creación depende del programador puede utilizar las mayúsculas como parte del nombre, las instrucciones poseen elementos que se separan con comas (,) y sus elementos se encierran entre paréntesis ().

Para entender la importancia de cada sección se explicará uno a uno los elementos que se incluyen en un programa y sus reglas de uso para facilitar la aplicación de la lógica y desarrollar programas que cumplan con los criterios de sintaxis de un programa en lenguaje C.

## CABECERA.

Esta sección es descriptiva, es decir permite mostrar información referente al programa o a determinados procesos que el programador desea comentar, ésta información no es tomada como una instrucción del programa, su función es solamente informativa, para diferenciar los comentarios de las instrucciones reales, se hace uso de caracteres especiales que indican el inicio y final de un *bloque de comentarios*, por ejemplo los caracteres /\* (Barra inclinada seguido del asterisco) permite incluir información de una o varias líneas comentadas, el bloque de comentarios se cierra o finaliza

con los signos `*/` (Asterisco seguido de la barra inclinada), este tipo de comentario se lo pueden incluir en cualquier parte del programa, el único requisito es que empiece con `/*` y finalice con `*/`.

## DIRECTIVAS DEL PREPROCESADOR

En esta sección, el lenguaje C incluye instrucciones que incrementan las posibilidades de desarrollo de un programa, en ésta área se incluyen librerías o instrucciones realizadas por otros programadores, la única manera de incluir estas instrucciones y poder utilizarlas es mediante la directiva `#include`, las librerías incluidas mediante esta directiva son las herramientas que diferencian un compilador de otro, aunque existen librerías estándares, es decir, se encuentran en todos los compiladores, la directiva `#include` siempre se ubica en la parte superior del programa, las instrucciones adicionales se representan por medio de nombres denominados librerías, éstas librerías son archivos que contienen funciones, constantes y variables entre otros elementos, las librerías se identifican en los sistemas operativos por ser archivos de extensión `.h` y varían dependiendo del sistema operativo, por ejemplo el siguiente listado de librerías muestra las instrucciones o funciones que contienen, además se definen de forma general la naturaleza de su utilización:

`#include <stdio.h>`

Esta librería contiene las instrucciones que permiten controlar las entradas y salidas de datos, sus funciones son de carácter generalizado y son utilizadas en diferentes sistemas operativos sin variaciones en su uso, aquí el listado de las más importantes:

**TABLA 8.** Lista de funciones que posee la librería `stdio.h`

<code>clearerr()</code>	<code>fclose()</code>	<code>feof()</code>	<code>ferror()</code>	<code>fflush()</code>	<code>fgetc()</code>	<code>fgetpos()</code>
<code>fgets()</code>	<code>fopen()</code>	<code>formato()</code>	<code>fprintf()</code>	<code>fputc()</code>	<code>fputs()</code>	<code>fread()</code>
<code>freopen()</code>	<code>fscanf()</code>	<code>fseek()</code>	<code>fsetpos()</code>	<code>ftell()</code>	<code>fwrite()</code>	<code>getc()</code>
<code>getchar()</code>	<code>gets()</code>	<code>perror()</code>	<code>printf()</code>	<code>putc()</code>	<code>putchar()</code>	<code>puts()</code>
<code>remove()</code>	<code>rename()</code>	<code>rewind()</code>	<code>scanf()</code>	<code>setbuf()</code>	<code>setybuf()</code>	<code>sprintf()</code>
<code>sscanf()</code>	<code>tmpfile()</code>	<code>tmpnam()</code>	<code>ungetc()</code>	<code>vfprintf()</code>	<code>vprintf()</code>	<code>vsprintf()</code>

**Fuente:** Autores del libro

`#include <stdlib.h>`

Esta librería contiene las instrucciones o funciones que permiten la conversión de tipos de datos numéricos, generación de números randómicos o aleatorios, manipulación de memoria entre otras utilidades, aquí el listado de las más importantes:

**TABLA 9.** Lista de funciones que posee la librería `stdlib.h`

<code>abort()</code>	<code>abs()</code>	<code>atexit()</code>	<code>atof()</code>	<code>atoi()</code>	<code>atol()</code>	<code>bsearch()</code>
<code>calloc()</code>	<code>div()</code>	<code>exit()</code>	<code>free()</code>	<code>getenv()</code>	<code>labs()</code>	<code>ldiv()</code>
<code>malloc()</code>	<code>mblen()</code>	<code>mbstowcs()</code>	<code>mbtowc()</code>	<code>qsort()</code>	<code>rand()</code>	<code>Realloc()</code>
<code>srand()</code>	<code>strtod()</code>	<code>strtol()</code>	<code>strtoul()</code>	<code>system()</code>	<code>wctomb()</code>	

**Fuente:** Autores del libro

`#include <string.h>`

Esta librería contiene las instrucciones o funciones que permiten la manipulación de cadenas de caracteres, aquí el listado de las más importantes:

**TABLA 10.** Lista de funciones que posee la librería `string.h`

<code>memchr()</code>	<code>memcmp()</code>	<code>memcpy()</code>	<code>memmove()</code>	<code>memset()</code>	<code>strcat()</code>	<code>strchr()</code>
<code>strcmp()</code>	<code>strcoll()</code>	<code>strcpy()</code>	<code>strcspn()</code>	<code>strerror()</code>	<code>strlen()</code>	<code>strncat()</code>
<code>strncmp()</code>	<code>strncpy()</code>	<code>strpbrk()</code>	<code>strrchr()</code>	<code>strspn()</code>	<code>strstr()</code>	<code>strtok()</code>
<code>strxfrm()</code>						

**Fuente:** Autores del libro

`#include <math.h>`

Esta librería contiene las instrucciones que permiten el uso y manipulación de funciones matemáticas, aquí el listado de las más importantes:

**TABLA 11.** Lista de funciones que posee la librería `math.h`



Acos()	Asin()	atan()	atan2()	ceil()	cos()	cosh()
Exp()	Fabs()	floor()	fmod()	frexp()	ldexp()	log()
log10()	modf()	pow()	sin()	sinh()	sqrt()	tan()
tanh()						

**Fuente:** Autores del libro

#include <ctype.h>

Esta librería contiene funciones que permiten mostrar la naturaleza de un carácter, convertir de mayúsculas a minúsculas y viceversa; y valores enteros a códigos ASCII, aquí el listado de las más importantes:

**TABLA 12.** Lista de funciones que posee la librería ctype.h

tolower()	toupper()	isalnum()	isalpha()	isascii()	isdigit()	islower()
isctrl()	isgraph()	isprint()	ispunct()	isspace()	isxdigit()	isupper()

**Fuente:** Autores del libro

#include <time.h>

Esta librería proporciona las funciones, macros, y tipos para manipular la hora y la fecha del sistema, aquí el listado de las más importantes:

**TABLA 13.** Lista de funciones que posee la librería time.h

asctime()	clock()	ctime()	difftime()	Gmtime()	localtime()	mktime()
strftime()	time()					

**Fuente:** Autores del libro

#include <conio.h>

Esta librería contiene las funciones, macros, y constantes para preparar y manipular la pantalla en modo texto, es decir en el entorno de consola DOS, para sistema operativo Linux existe la librería <ncurses.h>, esta librería no viene por defecto instalada, su instalación se basa en distribuciones DEBIAN, para instalarla solo basta con ejecutar el comando: *apt-get update* y/o *apt-get install ncurses\**; aquí el listado de las más importantes funciones utilizadas en CONIO y en NCURSES:

**TABLA 14.** Lista de funciones que posee la librería conio.h

cgets()	clreol()	clrscr()	cprintf()	cputs()	cscanf()	delline()
getche()	getpass()	gettext()	gettextinfo()	gotoxy()	highvideo()	inport()
inline()	getch()	lowvideo()	movetext()	normvideo()	outport()	putch()
puttext()	setcursortype()	textattr()	textbackground()	textcolor()	textmode()	ungetch()

**Fuente:** Autores del libro

**TABLA 15.** Lista de funciones que posee la librería ncurses.h

initscr()	keypad()	endwin()	cbreak() / nocbreak()	printw() y putstr()
move()	nodelay()	refresh()	getstr() y getch()	echo() / noecho()

**Fuente:** Autores del libro

Existen varias directivas de preprocesadores que se tratarán poco a poco, pero la directiva obligatoria en todo programa es la que permite potenciar las posibilidades de acción del programa, entienda que las potencialidades radican en utilizar las funcionalidades que incluyen éstas librerías, para lograrlo se necesita de la instrucción *#include* que posee la siguiente sintaxis:

*#include <nombre de la librería>*, note que el nombre de la librería está encerrado entre los signos: menor que (<) y mayor que (>), lo que significa que el archivo se encuentra ubicado en una carpeta de librerías pre configurada, pero también se puede hacer uso de las comillas que cambia su ubicación.

`#include "nombre de la librería"`, la sintaxis que utiliza las comillas para encerrar el nombre de la librería, permite incluir como referencia una ruta de ubicación del archivo físico de la librería, por ejemplo `#include "c:\\milibrerías\\ejemplos\\nombre.h"`, el ejemplo indica que la ubicación del archivo está en el disco duro que contiene el sistema operativo(c:), dentro (\\) de la carpeta "milibrerías" y dentro (\\) de la carpeta "ejemplos"; tenga en cuenta que cuando no se incluye la ruta, el compilador asume que la ruta de ubicación es la misma donde se encuentra el compilador.

Como ya se ha indicado, en la actualidad existen muchos compiladores del lenguaje C, cada uno de ellos respeta el estándar de la sintaxis general, pero en la mayoría de los editores se diferencian en la ubicación física de las librerías y como se identifican, es decir en la explicación anterior se utiliza el nombre de la librería con la extensión (.h), pero otros compiladores no incluyen la extensión (.h) como el caso de Linux y otros compiladores.

## BLOQUES DE FUNCIONES

En esta área se puede crear pequeños programas dentro de un programas llamados funciones, por ahora se considerará solo el uso de la función `main()`, ya que es la primera que el intérprete del lenguaje C utilizará para ejecutar instrucción por instrucción hasta finalizar el programa, cada instrucción estará limitado por un punto y coma (;), es decir que pueden existir más de una instrucción en la misma línea de programación ya que el punto y coma (;) indica el final de una instrucción.

Las instrucciones del lenguaje C por regla siempre estarán en minúsculas y solo se permitirá instrucciones en mayúsculas cuando el programador cree sus propias instrucciones, es importante entender que cada letra del alfabeto incluido los diferentes signos y dígitos numéricos tiene sus propios códigos numerados de forma única, es decir asumamos que la letra (A) mayúscula tiene un código 65, ningún otro carácter tendrá el mismo código, por lo que la letra (a) minúscula tendrá un código diferente, aunque para nosotros son iguales, esto quiere decir que el lenguaje C si diferencia entre mayúsculas y minúsculas, en otras palabras para el lenguaje C la (a) minúscula es diferente de la (A) mayúscula y por supuesto diferente a la mismas tildadas (á,Á).

Por lo general la mayoría de los lenguajes de programación al igual que lenguaje C clasifica los tipos de datos en 2 grupos: *Alfanuméricos* (letras, símbolos y dígitos) que sirven para contener el código numérico de cada carácter que utiliza la computadora y los tipos de datos *numéricos* que son exclusivos para ser utilizados en los diferentes cálculos.

El compilador del lenguaje C obliga al programador a declarar las variables antes de poder utilizarla, el término declarar significa que el programador debe indicarle al compilador el nombre de la variable y el tipo de datos que almacenará, por lo que es importante que el programador conozca los diferentes tipos de datos que ofrece el lenguaje C, los límites, ventajas y desventajas de los diferentes tipos de datos y como se deben utilizar en un programa, el siguiente cuadro muestra los diferentes tipos de datos y los límites de almacenamiento:

**TABLA 16.** Tipos de datos predefinidos en lenguaje c

TIPO DE DATO:	Tamaño en memoria (byte)	Límites de almacenamiento
void	0	Normalmente utilizados solo para funciones, significa que "No devuelve nada" o sin valor
<b>VALORES ENTEROS:</b> números completos sin decimales o parte fraccionada y sus negativos		
int	2	-32768 a 32767
short int	1	-128 a 127
unsigned int	2	0 a 65535

long int	4	-2147483648 a 2147483647
unsigned long	4	0 a 4294967295
<b>VALORES REALES:</b> números que incluyen decimales o punto flotante		
float	4	3.4x10 <sup>-38</sup> a 3.4x10 <sup>38</sup>
double	8	1.7x10 <sup>-308</sup> a 1.7x10 <sup>308</sup>
long double	10	3.4x10 <sup>-4932</sup> a 1.1x10 <sup>4932</sup>
<b>VALORES ALFANUMÉRICOS:</b> Almacenan letras, dígitos, símbolos, signos de puntuación.		
char	1	0 a 255 (código único por carácter)

**Fuente:** Autores del libro

Para declarar una variable se debe cumplir con la siguiente sintaxis:

*TipoDeDatos ListaDeVariables;*

Observe que el tipo de datos se ubica primero de izquierda a derecha, se deja un espacio y a continuación el o los nombres de las variables que desee utilizar en el programa, es importante indicar que la lista de variables se separan mediante la coma (,) y se finaliza la declaración utilizando el punto y coma (;), por ejemplo:

*int a, val1, nota, area;*

En el ejemplo se crean cuatro variables de tipo int (Entero), esto quiere decir que las variables no podrán almacenar valores con partes fraccionadas, al igual que todos los lenguajes de programación, lenguaje C exige de ciertas reglas para crear nombres de variables, entre las más importantes tenemos:

- ✓ *No está permitido que dos o más variables compartan un mismo nombre;* una variable en la computadora se representa por un espacio de almacenamiento en la memoria RAM, por lo que es importante considerar que no se pueden identificar dos espacios de memoria con el mismo nombre, por ejemplo:

*int a,b,c;*

*float n1,a,d;*

En las dos declaraciones, se infringe esta regla al crear la variable (a) sin importar que sean de diferentes tipos de datos.

- ✓ *Una variable puede utilizar letras dígitos y el único símbolo permitido el guion bajo o sub guion (\_),* además la regla indica que los nombres de las variables *no pueden empezar* con dígitos ni símbolos diferentes del sub guion (\_), para mejorar la explicación de la regla considere el siguiente cuadro de ejemplos:

**TABLA 17.** Ejemplos de declaración de variables

EJEMPLOS INCORRECTOS	EJEMPLOS CORRECTOS	EXPLICACIÓN
1x	x1	No debe empezar con dígitos
NumDías	NumDias	Las vocales tildadas incluida la ñ se consideran símbolo y no se permiten
Edad Hombre	Edad_Hombre	No se acepta espacio como parte del nombre
-pago	_pago	Ningún signo aritmético debe ser parte del nombre de la variable

**Fuente:** Autores del libro

- ✓ *Una variable no puede utilizar como nombre, una palabra reservada del lenguaje C*

Las instrucciones del lenguaje C son utilizadas para realizar acciones sobre la lógica de un programa, por lo que no está permitido el uso de palabras reservadas como nombres de variables, de hacerlo el intérprete lo tomaría como una acción, mas no como una variable, aquí el listado de palabras reservadas del lenguaje C:

**TABLA 18.** Listado de palabras reservadas en lenguaje c

asm	auto	bool	break	case	catch
char	class	const	const_cast	continue	default
delete	do	double	dynamic_cast	else	enum
explicit	extern	false	float	for	friend
goto	if	inline	int	long	mutable
namespace	new	operator	private	protected	public
register	reinterpret_cast	return	short	signed	sizeof
static	static_cast	struct	switch	template	this
throw	true	try	typedef	typeid	typename
union	unsigned	using	while	void	volatile

**Fuente:** Autores del libro

La explicación de cada una de éstas instrucciones al igual que las ofrecidas en algunas librerías se describirán a medida que se desarrollan las actividades en el presente material bibliográfico.

## CONTENIDO DE MEMORIA Y LÍMITES DE ALMACENAMIENTO

Para explicar los límites de almacenamiento es importante conocer las unidades de medida que se aplica a la información:

**TABLA 19.** Unidades de medida de la información

UNIDADES DE MEDIDA	ABREVIATURA	EQUIVALENCIA	DESCRIPCIÓN
1 Bit	b	1(unos) o 0 (cero)	Sistema de numeración binaria
1 Byte	B	8 bits	Letra, símbolo, dígito (carácter)
1 Kilobyte	KB	1024 Byte	Mil caracteres
1 Megabyte	MB	1024 Kilobyte	Un millón de caracteres
1 Gigabyte	GB	1024 Megabyte	Mil millones de caracteres
1 Terabyte	TB	1024 Gigabyte	Un billón de caracteres

**Fuente:** Autores del libro

Las unidades de medida se aplican para definir la capacidad de almacenamiento que tiene un disco duro, un CD, un flash memory, la memoria RAM, entre otros dispositivos de almacenamiento.

### ¿Qué contiene la memoria principal de la computadora?

La memoria RAM es organizada y administrada por el sistema operativo, está compuesta por celdas que almacenan caracteres de 8 bit en algunos sistemas operativos y en otros 16 bit; un bit es la presencia de una variación de voltaje que se la conoce como bit (1) o la no existencia de dicha variación como bit (0), la combinación de bits ceros y unos generan un *código numérico único*, que identifica una letra (a..z, A..Z), un dígito (0..9) o un símbolo ( , . : ; - \_ + \* ? ¿ ! ¢ ñ á y otros más como los símbolos utilizados en otras escrituras).

Inicialmente se trabajó con una tabla de códigos llamadas ASCII de 127 caracteres del estándar ANSI C y C++, y que tanto Windows como Linux la utilizan para identificar las instrucciones y la sintaxis, lamentablemente 127 caracteres no cubren todos los caracteres como vocales tildadas, símbolos especiales o las eñes, así que la tabla se amplió a 256 caracteres, pero la necesidad de incluir símbolos de escrituras de otras lenguas generó que Windows creara su esquema de caracteres Win-1252 o el estándar ISO-8859-1, de forma paralela los otros sistemas operativo desarrollaron una tabla que

contienen todos los alfabetos y símbolos de todas las lenguas del planeta que tienen simbología o ideogramas en sus escrituras, a esta tabla se la llamó UNICODE cuyo estándar más utilizado es UTF-8 o ISO-8859, que por la gran cantidad de símbolos utiliza 16 bit de codificación única; La compatibilidad de éstos dos estándares se basan solo en los primeros 127 caracteres del ASCII, de los cuales no son imprimibles desde el código 0 hasta el código 32.

Debido a la compatibilidad entre estos dos estándares se utilizará ASCII para explicar la administración de variables y sus tipos de datos en la memoria y su funcionalidad, la tabla es la siguiente:

**TABLA 20.** Lista de caracteres del estándar ASCII

Código	Símbolo	Código	Símbolo	Código	Símbolo	Código	Símbolo	Código	Símbolo	Código	Símbolo
32	Esp	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	\$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	

**Fuente:** Autores del libro

Técnicamente cada vez que usted presiona una tecla, en realidad internamente se genera un código y se almacena dicho código en la memoria principal del computador (RAM), es decir asuma que ha escrito la palabra “Hola”, en la memoria se almacena (72 111 108 97) que corresponde al código de cada letra; de igual forma sucede con los números, si usted presiona 18 en realidad se almacena (49 y 56) que corresponden al código del uno y del ocho.

### ¿Cómo se realizan los cálculos en la computadora?

Si se necesita sumar 1+1 utilizando lo que está almacenado en memoria, no se podría realizar ningún cálculo, ya que los códigos serían 49 + 49, lo que significa que jamás daría un resultado real, ante estas dificultades se crearon los tipos de datos numéricos, que permiten transformar sus datos en una *versión calculable*, para lograrlo debe transformar los valores decimales a binario y realizar la operación, para simplificar la transformación se utilizará una tabla, que fija un valor dependiendo de la posición del bit, la tabla de transformación a utilizar será:

Posición →	8	7	6	5	4	3	2	1
Valor según la posición →	128	64	32	16	8	4	2	1

Observe que las posiciones se numeran de derecha a izquierda (8, 7, 6, 5, 4, 3, 2, 1) y corresponden a las 8 posiciones de una celda de memoria RAM, el *valor* es su equivalente conforme a la posición, empieza en 1 y se duplicará hasta la octava posición (128, 64, 32, 16, 8, 4, 2, 1); la regla de transformación es muy simple, solo hay que ubicar *unos* donde la sumatoria de sus valores den como resultado la cantidad a transformar, por ejemplo se transformará el 47 de decimal a binario:

128	64	32	16	8	4	2	1
		1		1	1	1	1

Observe que los *unos* se ubicaron en 32, 8, 4, 2 y 1, si usted suma estas cantidades le dará como resultado 47, las posiciones vacías se llenarán con ceros, quedando el 47 en binario de la siguiente forma:

128	64	32	16	8	4	2	1
0	0	1	0	1	1	1	1

Como ejemplo considere la siguiente operación aritmética que utiliza variables numéricas de tipo (int) cuyo tamaño por variable es de dos byte de memoria:

```
int a, b, c; //se crean las tres variables
a = 317;    //la variable a toma 317
b = 126;    //la variable b toma 126
c = a + b;  //la variable c tomará el resultado de sumar a y b
```

Cada variable declarada ocupa dos byte en la memoria, por lo que, la tabla se extiende a 16 posiciones:

signo	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	
0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	a= 137
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	b= 126

La última posición es para el signo, si tiene un valor de 0 significa que es positivo, y si es 1 significa que es negativo, para realizar la suma binaria de las dos variables se aplica las siguientes reglas:

- ✓ 0+1 o 1+0 es igual a 1
- ✓ 0+0 es igual a 0
- ✓ 1+1 es igual a 0 *llevando* 1
- ✓ La suma se realiza de derecha a izquierda

signo	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	
							1	1	1	1	1					
0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	a= 137
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	b= 126
0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1	c= 263

Las variables de tipo de dato entero (int) tienen límites de almacenamiento que va desde -32768 hasta 32767, esto significa que la variable (a) no podría almacenar 32800, esto se debe a que no existe más ubicaciones de posiciones para ubicar más dígitos, si se llena de unos todas las 16 posiciones se tiene:

signo	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	= 32767

La suma binaria se aplica incluso al cálculo de la resta, considere el mismo ejemplo de la suma:

```

int a, b, c; //se crean las tres variables
a = 317;    //la variable a toma 317
b = 126;    //la variable b toma 126
c = a - b;  //la variable c tomará el resultado de restar a y b

```

Observe que el valor de (b) es positivo:

signo	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	b= 126

Al aplicar la resta el valor de (b) debe ser transformado a un valor negativo, para realizarlo se aplica las siguientes reglas:

- ✓ De derecha a izquierda se escribe igual hasta encontrar el primer 1
- ✓ Pasando el primer 1 se transforma los ceros en unos y los unos en ceros

Sí aplicamos estas reglas, tendríamos:

signo	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	b= 126
1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	- b

Observe que la primera y segunda posición de derecha a izquierda se escribió igual ya que el primer uno se encontró en la segunda posición, pero de ahí en adelante se convierten los unos en ceros y los ceros en unos, al aplicar la suma entre el valor positivo de (a=137) y el valor negativo de (b=126) se tiene el siguiente resultado:

signo	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	
1	1	1	1	1	1	1	1									
0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	a= 137
1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	b= -126
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	c= 11

En caso de que el resultado sea negativo, se vuelve aplicar la transformación para ver el resultado en positivo.

### ¿Cómo está estructurada una función?

Las funciones son pequeños programas dentro de un programa principal, se identifican por medio de un nombre único y deben cumplir las mismas reglas que cumplen los nombres de las variables, éstos pequeños subprogramas puede o no intercambiar uno o varios datos, su propósito principal es cumplir con determinados procesos y/o devolver un resultado, por ahora solo se estudiará a la función *main()*, ya que se considera la más importante, al ser la primera función que el compilador y el intérprete buscan para ejecutar instrucción por instrucción la lógica propuesta por el programador.

Como regla principal en lenguaje C la función *main()* al igual que cualquier otra función empiezan las instrucciones a partir de la llave inicial de abrir "{" y finaliza con la llave de cierre "}", todas las funciones por su naturaleza se desarrollan para devolver un resultado a cualquier proceso, para lograrlo necesita de la instrucción *return* (Retornar) que sirve para finalizar la función y además permite devolver el resultado de un proceso escribiendo *return* seguido del valor o el dato a devolver, por ahora se utilizará la instrucción "*return 0;*" que significa que finalizo de forma correcta, para ilustrar estas definiciones considere el siguiente ejemplo que suma dos números y muestra el resultado:

```

C2:    # include <stdio.h>
        main()
        { int a,b,c;
          scanf("%i",&a);
          scanf("%i",&b);
          c = a + b;
          printf("El resultado es:%i",c);
          return 0;
        }

```

El ejemplo utiliza dos instrucciones nuevas `scanf()` y `printf()`, ambas son populares para recibir datos que se ingresaran desde el teclado (*scanf*) y para mostrar información por pantalla (*printf*), para explicar el uso de éstas instrucciones, primero hay que definir los caracteres especiales de formato, estos son recursos estandarizados por el lenguaje C para mejorar la funcionalidad y portabilidad entre las diferentes funciones y que se incluyen en las diferentes librerías, la siguiente tabla muestra los caracteres de mayor estandarización:

**TABLA 21.** Caracteres utilizados para especificar formatos de entrada y salida

CARÁCTER	DESCRIPCIÓN
d, i	entero decimal con signo
o	entero octal sin signo
u	entero decimal sin signo
x	entero hexadecimal sin signo (en minúsculas)
X	entero hexadecimal sin signo (en mayúsculas)
f	Coma flotante en la forma [-]dddd.dddd
e	Coma flotante en la forma [-]d.dddd e[+/-]ddd
g	Coma flotante según el valor
E	Como e pero en mayúsculas
G	Como g pero en mayúsculas
c	un carácter
s	cadena de caracteres terminada en '\0'
%	imprime el carácter %
p	puntero

**Fuente:** Autores del libro

**TABLA 22.** Constantes para realizar acciones en los formatos de entrada y salida

CONSTANTES	DESCRIPCIÓN
\a	Alerta, hace sonar la alarma del sistema
\b	Espacio atrás (backspace)
\f	Salto o avance de página



\n	Salto de línea o nueva línea
\r	Retorno de carro
\t	Tabulación horizontal, Mueve el cursor al próximo tabulador
\v	Tabulación vertical
\\	Barra invertida, Imprime un carácter de diagonal invertida
\'	Comilla simple, Imprime una comilla simple
\"	Comillas dobles, Imprime una doble comilla
\OOO	Visualiza un carácter cuyo código ASCII es OOO en octal
\xHHH	Visualiza un carácter cuyo código ASCII es HHH en hexadecimal
\?	Imprime el carácter del signo de interrogación

**Fuente:** Autores del libro

Las siguientes funciones son las más utilizadas en la entrada y salida de datos, éstas entre otras de similares características pertenecen a la librería *stdio.h*

#### **scanf("Formato",DireccionesDestinos);**

Esta función permite el ingreso de información que va desde el teclado hasta una posición en la memoria principal (variable), normalmente se debe indicar la dirección de destino que almacenará ésta información (dirección de la variable), esta instrucción en si permite escanear desde el teclado cualquier tipo de dato, como lenguaje C tiene varios tipos de datos, el programador debe obligadamente diferenciar entre los diferentes formatos al utilizar la instrucción, en la tabla 21 se muestra los caracteres especiales que ésta y otras funciones utilizan para diferenciar los formatos establecidos por el lenguaje, el formato empieza por el signo porcentaje (%) y seguido se ubica el carácter que corresponde al tipo de dato, para especificar la *dirección destino* se utiliza el signo i comercial (&) que en lenguaje C toma el significado "*La dirección de*", por ejemplo considere las siguientes declaraciones de variables y como utilizar la instrucción *scanf()* para escanear los diferentes tipos de datos desde el teclado y contenerlos en las diferentes variables; el ejemplo no tiene un significado relevante en cuanto al proceso propuesto, pero ejemplifica el uso de la función *scanf()* y los formatos utilizados para identificar el o los tipos de datos a manejar en el proceso de escaneo desde el teclado y llenado de las respectivas variables:

El programa incluye la función *system()* que pertenece a la librería *stdlib.h*, sirve para ejecutar comandos o instrucciones del sistema operativo, *system* ejecuta las ordenes "cls" que sirve para limpiar o borrar la pantalla de ejecución y "Pause" que sirve para pausa la ejecución del programa hasta que se presione cualquier tecla, la tecla que se presione no tendrá efecto sobre las variables o el programa.

```
C3:    #include<stdio.h>
        #include <stdlib.h>
        main()
        { int edad;
          float sueldo;
          long distancia;
          char sexo, nombre[30];
          double deudaExterna;
          system("cls");
          printf("Ingresa edad:"); scanf("%d",&edad);
```

```

printf("Ingrese sueldo:"); scanf("%f",&sueldo);
printf("Ingrese distancia:"); scanf("%ld",&distancia);
printf("Ingrese sexo (m/f):"); scanf("%s",sexo);
printf("Ingrese nombre:"); scanf("%s",nombre);
printf("Ingrese deuda externa:"); scanf("%lf",&deudaExterna);
printf("Edad:%d\n",edad);
printf("Sueldo: %f\n",sueldo);
printf("Distancia: %ld\n",distancia);
printf("Sexo: %s\n",sexo);
printf("Nombre: %s\n",nombre);
printf("Deuda externa: %lf",deudaExterna);
system("pause");
return 0;
}

```

Es importante tener en cuenta que la función `scanf()` permite escanear más de un tipo de dato en la misma instrucción, considere el mismo ejemplo pero esta vez modificado:

**C4:**

```

#include<stdio.h>
#include <stdlib.h>
main()
{ int edad;
  float sueldo;
  long distancia;
  char sexo, nombre[30];
  double deudaExterna;
  system("cls");
  printf("Ingrese edad, sueldo y distancia:"); scanf("%d%f%ld",&edad,&sueldo,&distancia);
  printf("Ingrese      sexo      (m/f),      nombre      y      deuda      externa:");
  scanf("%s%s%lf",sexo,nombre,&deudaExterna);
  printf("Edad:%d\n",edad);
  printf("Sueldo: %f\n",sueldo);
  printf("Distancia: %ld\n",distancia);
  printf("Sexo: %s\n",sexo);
  printf("Nombre: %s\n",nombre);
  printf("Deuda externa: %lf",deudaExterna);
  system("pause");
  return 0;
}

```

#### **printf("Formato",ContenidoVariable);**

Esta función es el complemento de `scanf()`, permite mostrar la información procesada utilizando principalmente la pantalla, la función es capaz de mostrar el contenido de cualquier tipo de dato, esto significa que el programador debe diferenciar entre los diferentes tipos de datos para identificarlos de forma similar a los formatos utilizados en `scanf()`, en la tabla 21 se muestran los caracteres especiales que ésta y otras funciones utilizan para diferenciar los formatos establecidos por el lenguaje C, el formato empieza por el signo porcentaje (%) y seguido se ubica el carácter que corresponde al tipo de dato, para ilustrar estas definiciones considere los siguientes ejemplos:

El siguiente formato permite la manipulación de textos:

```
const char* Fac = "FCI";
```

En este ejemplo declara una variable "Fac" de tipo *char* que contiene las iniciales "FCI", la instrucción **const** permite indicar al lenguaje que el contenido es una constante, es decir que su contenido no va a variar, al aplicar `printf()` con formatos de salida por pantalla se podría realizar lo siguiente:

```
printf("\t.%10s.\n \t.-%10s.\n \t.*s.\n", Fac, Fac, 10, Fac);
```

Esta instrucción utiliza los puntos para especificar límites iniciales y finales, al aplicarlo se tendría el siguiente resultado:

- FCI. -----> En éste ejemplo desde el punto inicial hasta el final hay 10 espacios 7 vacíos y los 3 caracteres
- FCI .
- FCI.

El formato es aplicado en un solo `printf()`, pero para detallar su explicación se dividirá en 3 partes:

- ✓ Este formato `"\t.%10s.\n"`, tiene tres acciones, primero avanza una tabulación de 5 espacios `"\t"` y muestra un punto `"."`, la segunda acción, en 10 espacios separados muestra un texto *alineado a la derecha* `"%10s"` y escribe otro punto `"."`, y por último la tercera acción se basa en saltar de línea `"\n"`.
- ✓ En la segunda parte del formato utiliza otra línea de la pantalla, vuelve a tabular los espacios, muestra un punto `"."` y en 10 espacios muestra un texto *alineado a la izquierda* `"%-10s"`, después muestra otro punto `"."` y salta de línea `"\n"`.
- ✓ En la última parte del formato destacado en negrilla `"\t.*s.\n"` produce como resultado lo mismo que la primera parte del formato, la diferencia radica en el uso del asterisco (\*) que permite definir los espacios a separar desde un argumento adicional.

```
printf("\t.%10s.\n \t.-%10s.\n \t.*s.\n", Fac, Fac, 10, Fac);
```

El siguiente ejemplo de uso de formatos, muestra como manipular caracteres especiales y su código numérico:

```
printf("Muestra el carácter de un código y el porcentaje:\tac %%\n", 66);
```

El resultado de aplicar el formato sería:

*Muestra el carácter de un código y el porcentaje:      B %*

El ejemplo utiliza el formato `"%c"` que sirve para mostrar un carácter, observe que muestra la B mayúscula cuyo código numérico correspondiente es 66 y además permite mostrar el carácter de formato porcentaje `"%%"` que no puede mostrarse de forma individual.

El siguiente formato muestra las posibilidades de salida por pantalla al manipular valores enteros:

```
printf("Sistema de numeración en decimal:\t%i %d %.6i %i %.0i %+i %u\n", 1, 2, 13, 17, 22, 4, -1);
```

El ejemplo arrojaría el siguiente resultado:

*Sistema de numeración en decimal:    1 2 000013 17 22 +4 65526*

Observe que 1 y 2 se muestran de forma normal (sin formato especial), tanto `"%i"` como `"%d"` son formatos iguales al mostrar los valores enteros, el valor de 13 se lo mostrará en 6 espacios separados y llenados con ceros a la izquierda `"%.6i"`, 17 y 22 no tendrán formato ya que su valor así lo define `"%i%.0i"`, el 4 lo mostrará con el signo + ya que el formato así lo dispone al incluir el signo (+) `"%+i"`, el -1 lo mostrará conforme a la tabla de transformación binaria ya que `"%u"` es entero sin signo.

El computador trabaja con el sistema de numeración hexadecimal para referenciar direcciones internas, es importante que se utilicen formatos especiales para entender y relacionar sus valores, el siguiente ejemplo aplica formatos a los valores hexadecimales:

```
printf("Sistema de numeración en hexadecimal:\t%x %x %X %#x\n", 5, 10, 10, 6);
```

El ejemplo muestra el siguiente resultado:

*Sistema de numeración en hexadecimal:    5 a 0x6*

Estos valores decimales se mostrarán transformados a su correspondiente valor hexadecimal, cada uno en diferentes formatos de presentación hexadecimal, así el 5 seguirá siendo 5 en hexadecimal, 10 en decimal se mostrará como `"a"` en minúscula que corresponde a su valor en hexadecimal, el

También se puede utilizar formatos para el sistema de numeración octal, considere el siguiente ejemplo:

Al aplicar el formato dará el siguiente resultado:

Este ejemplo es similar al utilizado en el sistema hexadecimal con la diferencia que lo mostrará en el sistema octal, observe que el 10 en decimal es 12 en octal.

```
printf("Completando:\t%f %.0f %.32f\n", 1.5, 1.5, 1.3);
```

El ejemplo al ser aplicado daría el siguiente resultado:

El ejemplo propuesto manipula valores con decimales, así de izquierda a derecha el primer valor 1.5 se mostrará sin ningún formato `“%f”`, el segundo valor 1.5 será redondeado al entero y mostrará 2, observe que el formato define el no mostrar parte decimal `“%.0f”`, el último valor 1.3 se mostrará con 32 posiciones decimales más de precisión `“%.32f”`.

El siguiente ejemplo aplica otros formatos que pueden ser utilizados para mostrar valores con parte fraccionada:

```
printf("Ajustado:\t%05.2f %.2f %5.2f\n", 1.5, 1.5, 1.5);
```

Mostrará como resultado:

Ajustando: 01.50 1.50 1.50

De izquierda a derecha el primer valor 1.5 se mostrará anteponiéndole el cero 0 a la parte entera y complementa con 0 en caso de tener un solo decimal “%05.2f”, el segundo valor 1.5 se mostrará con 2 decimales, considerando que el formato lo propone “%.2f”, el último formato “%5.2f” establece que el valor a mostrar tendrá 5 enteros y 2 decimales.

También es posible presentar valores en notación científica, por ejemplo:

```
printf("Notación científica:\t%E %e\n", 1.5, 1.5);
```

Se obtendría el siguiente resultado:

Notación científica: 1.500000E+00

Observe que el formato utiliza las letras “e” en minúscula y la otra con la “E” en mayúscula, valores cuya diferencia cambia el formato pero se obtiene el mismo resultado.

Todos los ejemplos analizados, se incluyen en el siguiente ejercicio para verificar las diferentes salidas:

```
C5: #include <stdio.h>
#include <stdlib.h>
int main()
{ system("cls");
printf("Aquí un mensaje y un salto de línea:\n");
const char* Fac = "FCI";
printf("\t.%10s.\n\t.-%10s.\n\t.*s.\n", Fac, Fac, 10, Fac);
printf("Muestra el carácter de un código y el porcentaje:\tac %%%\n", 66);
printf("Manipulación de enteros:\n");
```

```

printf("Sistema de numeración en decimal:\t%i %d %.6i %i %.0i %+i %u\n", 1, 2, 13, 17, 22, 4, -1);
printf("Sistema de numeración en hexadecimal:\t%x %x %X %#x\n", 5, 10, 10, 6);
printf("Sistema de numeración en octal:\t%o %#o %o\n", 10, 10, 4);
printf("Manipulación de puntos flotantes:\n");
printf("Redondeando:\t%f %.0f %.32f\n", 1.5, 1.5, 1.3);
printf("Ajustado:\t%05.2f %.2f %5.2f\n", 1.5, 1.5, 1.5);
printf("Notación científica:\t%E %e\n", 1.5, 1.5);
system("pause");
return 0;
}

```

**OBJETIVO DE APRENDIZAJE PARA EL SIGUIENTE APARTADO:** Conocer y aplicar los diferentes operadores de control para desarrollar operaciones aritméticas y lógicas, entender y definir pseudónimos aplicados a los valores como constantes que facilitan el detalle de las operaciones y los procesos y aplicar asignaciones condicionadas.

## OPERADORES DE CONTROL UTILIZADOS EN LENGUAJE C

Los operadores de control son aquellas herramientas que permiten que el programa realice operaciones aritméticas, operaciones de comparación y operaciones lógicas de condición, con la finalidad de gestionar los diferentes tipos de variables y/o constantes según la necesidad, por ejemplo para realizar operaciones aritméticas se aplica la simbología aritmética que ofrece el lenguaje C, éstos evalúan y operan de izquierda a derecha según la jerarquía del operador.

## OPERADORES ARITMÉTICOS

Lenguaje C ofrece operadores similares como los que operan en otros lenguajes de programación y permiten realizar las operaciones aritméticas como la suma, resta, multiplicación y división, pero también ofrece operadores de incremento y decremento que no son comunes en otros lenguajes de programación:

**TABLA 23.** Lista de operadores aritméticos

OPERADOR	NOMBRE	DESCRIPCIÓN
+	Suma	Extrae la suma de dos datos numéricos
-	Resta	Extrae la resta de dos datos numéricos
/	División	Extrae el cociente de dividir dos números
*	Producto	Extrae el producto de multiplicar dos números
%	Resto	Extrae el residuo de una división de dos números enteros

**Fuente:** Autores del libro

El lenguaje C incluye entre sus operadores herramientas que facilitan operaciones comunes de incremento y decremento, considere la siguiente tabla:

**TABLA 24.** Lista de incrementadores y decrementadores de lenguaje C

OPERADOR	NOMBRE	DESCRIPCIÓN	EJEMPLOS
++	Incremento	Permite incrementar en uno al valor de una variable	b++; es igual a utilizar b=b+1;
--	Decremento	Permite reducir en uno el valor de una variable	b--; es igual a utilizar b=b-1;
=	Asignación	Permite asignar un valor dado a una variable, actúa pasando datos de derecha a izquierda	b=369; c=17; a=b+c;
+=	Incremento por adición	Permite incrementar en una variable un valor asignado, el valor de incremento se ubica a la derecha	b+=5; es igual a utilizar b=b+5;
-=	Decremento por sustracción	Permite decrementarse en una variable un valor asignado, el valor de sustracción se ubica a la derecha	c-=8; es igual a utilizar c=c-8;

<code>*=</code>	Incremento por producto	Permite multiplicar al valor de una variable con un valor asignado, el valor del multiplicador se ubica a la derecha	<code>a*=3;</code> es igual a utilizar <code>a=a*3;</code>
<code>/=</code>	Asignación de la división	Permite dividir el valor de una variable para un número dado, el valor del divisor se ubica a la derecha	<code>x/=10;</code> es igual a utilizar <code>x=x/10;</code>
<code>%=</code>	Asignación del resto	Permite dividir el valor de una variable para un número dado, su resultado es la asignación del residuo y no el cociente.	<code>x%=2;</code> es igual a utilizar <code>x=x%2;</code>

**Fuente:** Autores del libro

## OPERADORES DE COMPARACIÓN

Los operadores de comparación son símbolos que permiten evaluar (de izquierda a derecha) una expresión de comparación, internamente realiza un calculo lógico que da como resultado solo una de dos posibles respuestas: verdadero (es cualquier valor diferente de cero) o Falso (siempre será cero), el lenguaje C solo reconoce la respuesta por *falso* y sin importar el compilador falso es sinónimo de cero, y el verdadero siempre será un valor diferente de cero, lenguaje C utiliza los siguientes símbolos para realizar comparaciones:

**TABLA 25.** Lista de símbolos utilizados por lenguaje C para realizar comparaciones

OPERADOR	NOMBRE	DESCRIPCIÓN
<code>==</code>	Igual a	Permite comparar dos valores si son iguales.
<code>!=</code>	Diferente de	Permite comparar dos valores si son diferentes.
<code>&gt;</code>	Mayor que	Permite comparar si el valor izquierdo es mayor que el valor derecho.
<code>&gt;=</code>	Mayor o igual que	Permite comparar si el valor izquierdo es mayor o igual que el valor derecho.
<code>&lt;</code>	Menor que	Permite comparar si el valor izquierdo es menor que el valor derecho.
<code>&lt;=</code>	Menor o igual que	Permite comparar si el valor izquierdo es menor o igual que el valor derecho.

**Fuente:** Autores del libro

## OPERADORES LÓGICOS

Los operadores lógicos son símbolos que permiten evaluar el resultado de una o varias comparaciones y devolver solo una respuesta lógica de verdadero o falso, los símbolos utilizados son:

**TABLA 26.** Lista de operadores lógicos utilizados en lenguaje C

OPERADOR	NOMBRE	DESCRIPCIÓN
<code>!</code>	No ( NOT ) lógico	Invierte el resultado lógico de una condición
<code>&amp;&amp;</code>	Y ( AND ) lógico	Todas las comparaciones deben ser verdadera para que el resultado sea verdadero
<code>  </code>	O ( OR ) lógico	Al menos una de las comparaciones debe ser verdadera para que el resultado sea verdadero

**Fuente:** Autores del libro

El siguiente ejemplo detalla el funcionamiento de algunos operadores aritméticos y su forma de uso:

**C6:**

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int a,b,x,y;
  system("cls");
  a=10;
  b=a++; //A la variable b se le asigna el valor de a; luego el valor de la variable a se incrementa en 1
  printf("Valor de a=%d y el valor de b=%d\n",a,b);
  x=10;
  y=++x; //El valor de x es incrementado en 1 y luego este valor es asignado a la variable y
  printf("Valor de x=%d y el valor de y=%d\n",x,y);
  system("pause");
```

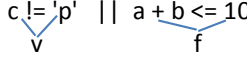

```

    return 0;
}

```

El siguiente cuadro muestra el funcionamiento de los operadores de comparación y lógicos, para esto asuma que la variable “a” es de tipo entero (int) con un valor de 8, la variable “b” es numérica con coma flotante (float) con un valor de 6.5 y la variable “c” de tipo carácter (char) que contiene la letra 'w' minúscula.

**TABLA 27.** Ejemplos de operadores de comparación y lógicos

EXPRESIÓN	RESULTADO DE LA COMPARACIÓN	VALOR
b > 6	Verdadero	1 u otro valor diferente de 0
(a+b) <= 10	Falso	0
(a>=6) && (c=='w')	Verdadero	1 u otro valor diferente de 0
c != 'p'    a + b <= 10 	Verdadero	1 u otro valor diferente de 0
!(b > 5)	Falso	0
a > (b+1)	Verdadero	1 u otro valor diferente de 0
a >=6 && c=='w'	Verdadero	1 u otro valor diferente de 0
!( a > (b+1) )	Falso	0
c != 'p'	Verdadero	1 u otro valor diferente de 0
(b < 11) && (a > 10) 	Falso	0

**Fuente:** Autores del libro

## USO DE LA DIRECTIVA #define

Esta instrucción permite asignar un nombre a un valor como si se tratara de una variable, se diferencia con la variable en que su comportamiento es una *constante*, esto significa que dicho valor no podrá ser cambiado y el nombre que utilizó como constante no puede ser utilizado como una variable, es decir las variables pueden cambiar de valor durante la ejecución de un programa, las constantes como su nombre lo indica, son valores que permanecen constantes durante toda la ejecución de un programa, los valores constantes pueden contener valores de diferentes sistemas de numeración, entre ellos valores decimales (enteros o punto flotante), octales y hexadecimales, la sintaxis es la siguiente:

```
# define NombreConstante valor
```

La directiva *#define* se la escribe después de incluir las librerías con *#include*; entre la directiva y el nombre de la constante se deja un espacio, a continuación otro espacio y por último su valor, para ilustrar estas definiciones considere el siguiente ejemplo:

<pre> #define PI 3.141592 int main() { float area, radio;   ...   area=PI*radio*radio;   ...   area=3.141592*radio*radio;   ... } </pre>	<pre> <b>CONST1:</b> #include &lt;stdio.h&gt;  #define b 2.4  #define h 9.2  #define peri b*2 + h*2  int main() { float area;   area=b*h;    printf("Area:%f,Perimetro:%f",area,peri);    return 0; } </pre>
--	--

En los ejemplos anteriores se utiliza la definición de constantes denominadas definiciones simbólicas, esta definición se debe porque no se especifican tipos de datos para cada constante, también existe la posibilidad de que el lenguaje C cree constantes considerando el tipo de dato, a este tipo de constante se la puede crear mediante la instrucción *const* que permite declarar las constantes definiéndolas con un tipo de dato específico, su sintaxis es la siguiente:

```
const TipoDato NombreConstante = valor;
```

Es importante conocer que al declarar una constante, si usted no incluye el "TipoDato" el lenguaje C asume que es de tipo entero (int), las políticas de uso son las mismas utilizadas en la gestión de variables, los nombres deben cumplir las mismas reglas de creación de nombres de variables, se diferencia en que su uso es constante y sin permiso de variación a lo largo de la ejecución del programa, considere los siguientes ejemplos de declaraciones de constantes:

```
const int DiasSemana=7;
const char Arroba ='@';
const float Valor_Hora=9.33;
const char Saludo[ ] ="Hola mundo..!" ;
```

Para ilustrar el uso de constantes considere el siguiente ejemplo, éste programa transforma una cantidad de segundos en horas, minutos y segundos:

```
CONST2:#include <stdio.h>
#include <stdlib.h>
int main()
{
    int sec, hr, min;
    const int SegHora=3600; //Una hora tiene 3600 segundos
    const int SegMin=60;    //Un minuto tiene 60 segundos
    system("cls");
    printf("Ingrese segundos: "); scanf("%d",&sec);
    hr = sec / SegHora;    //Extrae el número de horas
    sec %= SegHora;        //Quita los segundos utilizados en las horas
    min = sec / SegMin;    //Extrae el número de minutos
    sec %= 60;            //Quita los segundos utilizados en los minutos
    printf("Equivalencia:%d horas %d minutos %d segundos\n",hr,min,sec);
    system("pause");
    return 0;
}
```

## ASIGNACIONES CONDICIONADAS

Una asignación es simplemente el traspaso o envío de datos desde un lugar a otro, así una asignación puede ser la toma de valores como resultado de un proceso o cálculo, también puede ser el paso de valores constantes como los dados por el programador o también el paso de contenidos entre una variable y otra, por ejemplo:  $X=3.1416*5*5$ ;  $a=20$ ;  $pago=deuda$ ;

Las asignaciones condicionadas son aquellas que presentan condiciones y cuyo valor a ser asignado esta entre dos alternativas, la asignación dependerá de una condición, esta asignación estará controlada por los signos (?,:), su sintaxis está estructurada de la siguiente forma:

```
[destino]=<condición>?<expresión-verdadero>:<expresión-falso>
```

La asignación condicionada funciona de la siguiente forma: el resultado del proceso es opcionalmente contenida en la variable destino, Primero se evalúa la <condición>, si el resultado es *verdadero*, ejecutará <expresión-verdadero>, caso contrario, se ejecutará <expresión-falso>. Para ilustrar el uso de esta herramienta considere el siguiente ejemplo que muestra si un valor es par o impar:

```
C7: #include <stdio.h>
#include <stdlib.h>
int main()
{ int n;
```



```

system("cls");
printf("Ingrese un valor:");
scanf("%d",&n);
printf( (n%2 == 0) ? "Es par\n " : "Es impar\n ");
system("pause");
return 0;
}

```

Observe que en el ejemplo, el resultado de la condición es pasado a la función printf(), la condición evalúa el resultado del cálculo, al efectuar la operación para 2, si el residuo es cero, la asignación condicionada devolverá el mensaje "Es par\n" en caso contrario devolverá el mensaje "Es Impar\n", otra alternativa de uso, podría ser la verificación de mayor de edad de una persona:

```

printf("Ingrese su edad:");
scanf("%d",&edad);
r= (edad-18 < 0) ? -1 : 1;
if( r == -1 ) printf("Es menor de edad");
else printf("Es mayor de edad");

```

**OBJETIVO DE APRENDIZAJE PARA EL SIGUIENTE APARTADO:** Conocer los diferentes tipos de errores que normalmente se presentan al desarrollar un programa, identificar y aplicar posibles alternativas de solución.

## ERRORES EXISTENTES AL ESCRIBIR UN PROGRAMA

Cuando se desarrolla un programa en cualquier lenguaje de programación, el programador puede incurrir en cualquiera de los tres tipos de errores conocidos, errores de escritura (sintaxis), errores de ejecución y errores de lógica.

### ERRORES DE SINTAXIS

Todo lenguaje de programación posee reglas que permiten al compilador/intérprete entender la lógica y cumplir las instrucciones dadas por el programador, el no cumplirlas es un error de sintaxis, el compilador del lenguaje es el encargado de comprobar si se han cometido estos tipos de errores, los errores de sintaxis son los más fáciles de detectar y corregir en el código fuente.

Para ejemplificar la existencia de los errores de sintaxis considere el siguiente ejemplo:

**Error 1:** include <stdio.h>

```

imt main()
{
int base, altura;
base = 4;
altura =13;
area= base * altura;
printf( "%d", area )
retunr 0;
}

```

El ejemplo presenta 5 errores de sintaxis: *include* está escrito sin el #, lo correcto sería *#include*, se ha escrito *imt* antes de *main()* y lo correcto es *int*, falta un punto y coma (;) al momento de cerrar la instrucción *printf*, además se hace uso de la variable (*area*) que no ha sido declarada, y por último antes de finalizar se ha escrito *retunr* y lo correcto es *return*.

### ERRORES DE EJECUCIÓN

Una vez corregidos los errores de escritura o de sintaxis, se debe revisar que no se pueda producir un error en la ejecución del programa, considere que un error de ejecución se produce cuando el computador no puede ejecutar una instrucción de forma correcta. El siguiente ejemplo muestra un programa que tiene un error de ejecución:

**Error 2:** #include <stdio.h>

```
int main()
{
    int x;
    float y;
    x = 0;
    y = 6.4 / x;
    printf( "%f", y );
    return 0;
}
```

El programa no tiene errores de sintaxis pero si produce un error de ejecución, al ejecutar la instrucción

$y = 6.4 / x$ ; considere que x vale 0 ( $x=0$ ), y no se puede dividir para cero.

## ERRORES DE LÓGICA

Los errores de lógica son los producidos por el programador de forma inconsciente y son los más difíciles de detectar, estos se presentan después de superar errores de sintaxis y errores de ejecución, los errores de lógica se presentan cuando los resultados obtenidos no son los esperados.

El siguiente ejemplo ilustra una posibilidad de producirse errores de lógica:

**Error 3:** #include <stdio.h>

```
int main()
{
    float base, altura;
    base = 6.3;
    altura = 4.;
    printf( "El área es: %f", base * altura / 3 );
    return 0;
}
```

El programa muestra como resultado “El área es: 8.4”, el programa no tiene errores de sintaxis ni errores de ejecución, pero conociendo de antemano que el programa calcula el área de un triángulo, se esperaba que el resultado fuese: “El área es: 12.6”, al momento de ejecutar el programa y no evaluar los resultados esperados el error quedará involuntariamente, es recomendable desarrollar un ejemplo real para evaluar los resultados y de existir algún error lógico tendrá la posibilidad de corregirlo, considerando el ejemplo presentado el error está en la expresión “ $base * altura / 3$ ”, en su lugar se debe escribir la expresión “ $base * altura / 2$ ”.

El siguiente ejercicio propone identificar los errores en general y corregirlos, se trata de un programa que solicita el nombre y las notas de tres parciales, el programa mostrará como aprobado al estudiante si la sumatoria es mayor o igual a 70 puntos, caso contrario mostrará que reprobó:

**Error 4:** #include <stdio.h>

```
#include <stdlib.h>
int main()
{ char *nombre;
  double n1,n2,n3;
  double sum;
  system("cls");
  printf("Ingrese nombre del estudiante:");scanf("%s",&nombre);
```

```

printf("Ingrese nota 1:");scanf("%lf",&n1);
printf("Ingrese nota 2:");scanf("%ld",&n2);
printf("Ingrese nota 3:");scanf("%d",&n3);
sum=n1 + n2 + n3;
if (sum >= 70); printf("%f Aprobó\n",&sum);
else printf("%f Reprobó\n",&sum);
system("pause");
}

```

**OBJETIVO DE APRENDIZAJE PARA EL SIGUIENTE APARTADO:** Conocer las diferentes estructuras de control que posee el lenguaje y su similitud con otros lenguajes estructurados, aplicar la correcta sintaxis tanto para los controles condicionales como los controles de repetición, identificar y emplear las diferentes herramientas propias del lenguaje y aplicarlos en la resolución de problemas.

## ESTRUCTURAS DE CONTROL

Todos los lenguajes de programación poseen instrucciones que permiten controlar la secuencia lógica de los programas, estas instrucciones se clasifican en estructuras de control condicional y estructura de control de repetición, estas estructuras de control solo pueden ejecutar solo una instrucción, en caso de incluir dos o más instrucciones se debe crear un bloque de instrucciones, los bloques inician con la llave de apertura "{" y finaliza con la llave de cerrado "}", los bloques de instrucciones no necesitan finalizar con un punto y coma ";" y se pueden incluir en todas las estructuras de control.

### ESTRUCTURA DE CONTROLES CONDICIONALES

El Lenguaje C ofrece al programador varias estructuras de control condicional, las instrucciones se ejecutarán dependiendo de una condición, al aplicar las condiciones solo podrá realizar una de varias alternativas de acción, el lenguaje ofrece la estructura de control if() y switch(), note que todas las estructura de control utilizan paréntesis, se debe a que en su interior se debe incluir la expresión a validar.

#### ESTRUCTURA CONDICIONAL IF.

Esta estructura de control permite al programador aplicar condiciones basadas en comparaciones o en expresiones que pueden o no ser contenidas en variables o productos de resultados de cálculos, para utilizar la herramienta se aplica cualquiera de las siguientes sintaxis:

- ✓ En su forma más básica incluye solo la respuesta por verdadero, su formato es:

```

if(condición_expresión)
    instrucción_verdadero;

```

Al evaluar la *condición\_expresión* se ejecuta la *instrucción\_verdadero*, recuerde que *verdadero* es el resultado de cualquier valor diferente de cero (0).

- ✓ En su forma completa incluye también el *caso contrario*:

```

if (condición_expresión)
    instrucción_verdadero;
else
    instrucción_falso;

```

Al evaluar la *condición\_expresión* se ejecuta la *instrucción\_falso*, cuando el resultado es igual a cero (0).

**IMPORTANTE:** En algunos formatos que explican la sintaxis incluyen corchetes [...], esto significa que su uso es opcionalidad.

Considere los siguientes ejemplos que permiten definir el comportamiento de las evaluaciones condicionales:

**TABLA 28.** Ejemplos de casos de usos aplicado a la estructura condicional if()

float x=20; if (x<77) printf("%f",x);	<b>En éste ejemplo el valor de la variable “x” sí se muestra ya que su contenido es menor que 77</b>
int y=0; if (y) printf("%d",y);	En éste ejemplo el valor de la variable “y” no es mostrado ya que su contenido es cero (0), en lenguaje C cero representa falso
int y=5*3; if (y) printf("%d",y);	A diferencia del ejemplo anterior el valor de la variable “y” aquí sí es mostrado ya que su contenido es diferente que cero (0), en lenguaje C cualquier valor diferente de cero representa verdadero
if (sexo=='M') printf("Varón "); else printf("Mujer "); printf("Dios te ama.");	Sí la variable “sexo” contiene la letra M, se mostrará la palabra “Varón “, caso contrario se mostrará la palabra “Mujer “, y sin importar que acción se haya ejecutado siempre se mostrará “Dios te ama”, el resultado final podría ser “Varón Dios te ama” o “Mujer Dios te ama”

**Fuente:** Autores del libro

## ESTRUCTURA DE CONTROL IF CON CONDICIONES MÚLTIPLES O ANIDADAS

Se puede aplicar condiciones múltiples para resolver un problema, se aplica los mismos formatos estudiados, por ejemplo si se desea incluir dos o más sentencias como alternativa de acción ante una respuesta, se deben utilizar bloques de sentencias, es importante entender que las condiciones múltiples es el resultado de aplicar nuevas condiciones anidadas por verdadero o por los casos contrarios, es decir que como respuesta a una condición se pueden incluir otras condiciones, y así repetidamente por cada condición, la siguiente sintaxis expresa la forma o el formato de uso:

```
if (condición_expresión)
{
    < instrucción_verdadero >;
    [Más_instrucciones_verdadero]
}
else if (condición_expresión)
{
    < instrucción_verdadero >;
    [Más_instrucciones_verdadero]
}
.
.
else if(condición_expresión)
{
    < instrucción_verdadero >;
    [Más_instrucciones_verdadero]
}
else { < instrucción_falso >;
    [Más_instrucciones_falso]
}
```

Como una regla de oro es importante destacar que, para el lenguaje C, la parte que corresponde al caso contrario (*else*) al presentarse, se asociará al último if(), es decir, se considera que el *else* pertenece al if() más próximo considerando que se encuentra en el mismo bloque que el *else* y que no esté asociado con otro *if()*, para detallar estas definiciones considere la siguiente tabla que ejemplifica el uso de if anidados:

**TABLA 29.** Ejemplos de controles if

if (edad >=11 && edad <=15) { printf("Tiene %d años", edad); x= 18 - edad;	<b>En éste ejemplo el valor de la edad debe ser mayor o igual que 11 y menor o igual 15 para mostrar la edad, calcular cuántos</b>
--	--

<pre> printf("faltan %d años para ser mayor de edad", x); } if (Sueldo&gt;=1000) { CapPago=Sueldo*0.5;                     Monto=CapPago*12; } else { CapPago=Sueldo*0.3;         Monto=CapPago*12; } printf("Préstamo %f pago mensual: %f",Monto,CapPago); if (N mod 2 == 0)     if (N &gt;=10 &amp;&amp; N &lt; 100) printf("%d es par y de 2 dígitos",N);     else printf("%d es par de uno o más de 2 dígitos",N); if (N mod 2 == 0) {     if (N &gt;=10 &amp;&amp; N &lt; 100)         printf("%d es par y de 2 dígitos",N);     } else printf("%d es impar",N); printf ("Este mensaje se mostrará siempre"); if (HorasTrab) {     if (Categ==1) Sueldo=HorasTrab*3.50;     if (Categ==2) Sueldo=HorasTrab*6.80;     else Sueldo=HorasTrab*12.20; } else Sueldo=0; printf ("Usted recibe: %f", Sueldo); </pre>	<p><b>años le faltan para ser mayor de edad y mostrar el resultado del cálculo.</b></p> <p>Verifica si el sueldo es mayor o igual que 1000 para dar una capacidad de pago del 50 %, caso contrario la capacidad de pago será del 30%, sin importar el resultado de la condición se mostrará el monto del préstamo y su capacidad de pago</p> <p>En el ejemplo hay dos condiciones y un caso contrario, observe que el caso contrario se asocia al if() más cercano como una regla base</p> <p>A diferencia del ejemplo anterior el caso contrario se asociará al if() más cercano, pero considere que el if() interno se encuentra en el bloque de código de la respuesta por verdadero en el if() externo.</p> <p>La condición no posee una comparación pero aplica la regla que indica que cualquier valor diferente que cero es verdadero; utilizando bloques de llaves, los caso contrario (else) cambian y se ajustan a las condiciones incluidas dentro de bloque de llaves.</p>
---	--

**Fuente:** Autores del libro

Considere el siguiente ejercicio que permite el ingreso de tres números cualquiera para mostrar solo el mayor:

**C8:**

```

#include <stdio.h>
#include <stdlib.h>
main() {
    int a,b,c;
    system("cls");
    printf("Ingrese 3 números cualquiera:\n");
    scanf("%d %d %d",&a,&b,&c);
    if (a > b)
        if ( a > c)
            printf("Mayor: %d",a);
        else
            printf("Mayor: %d",c);
    else if ( b > c)
        printf("Mayor: %d",b);
    else
        printf("Mayor: %d",c);
    system("pause");
    return 0;
}

```

### **Actividades de refuerzo (AR):**

AR50. Modifique el programa C8 de forma que muestre los números ordenados descendientemente.

AR51. Se necesita de un programa que permita identificar la madurez de una persona, mediante el uso la edad, considere que es un "Infante" si la edad es menor que 14, es un "Joven" si la edad está entre 15 y 40, es un "Adulto" si la edad esta entre 41 y 60, y por último es un "Adulto mayor" si la edad es mayor que 60.

AR52. Desarrolle un programa que permita calcular el porcentaje de interés por mora en el pago de préstamos rápidos de máximo 30 días con un interés del 10%, el socio de la cooperativa conforme a la siguiente tabla se sujetará al monto de castigo:

PLAZO DE PAGO	INTERES APLICADO
Antes de los 30 días	-1%
A los 30 días	Normal del 10%
Desde los 31 días hasta 60 días	15%
Desde los 61 días en adelante	20%

### ESTRUCTURA DE CONTROL CONDICIONAL DE CASO SWITCH.

Esta estructura de control se la utiliza para agilizar la toma de decisiones múltiples para casos de comparaciones de igualdades, mejora y simplifica el uso de la estructuras `if()` anidadas, su sintaxis es la siguiente:

```
switch(expresión)
{
    case ListaValores_1: Instrucciones;
                        break;
    case ListaValores_2: Instrucciones;
                        break;
    .
    .
    .
    case ListaValores_n: Instrucciones; break;
    default: Instrucciones; break;
}
```

Cada `case` solo realiza comparaciones de igualdad entre cada elemento de la lista de valores con la expresión contenida en el `switch()`, si alguna de las comparaciones coincide con el valor de *expresión* se ejecutan las instrucciones o bloque de sentencias correspondientes hasta encontrar la instrucción *break*, es importante resaltar que si no encuentra la palabra reservada *break* continuará con la ejecución de instrucciones del siguiente bloque hasta encontrar la instrucción *break*.

El uso de la alternativa *default* es opcional, ésta se ejecuta en caso de que el valor de la *expresión* no coincida en ninguna de las lista de valores expresadas en cada caso. La estructura `switch` no permite que dos valores iguales estén en diferentes casos.

El siguiente ejemplo solicita el ingreso de dos números enteros cualesquiera, mediante el uso de opciones el usuario podrá seleccionar un cálculo entre: suma, resta, multiplicación o división

```
C9: #include <stdio.h>
#include <stdlib.h>
void main()
{
    int opcion, a, b;
    system("cls");
    printf("Ingrese un número: "); scanf("%d",&a);
    printf("Ingrese otro número: "); scanf("%d",&b);
    printf("Seleccione:\n1. Sumar\n2. Restar\n3. Dividir\n4. Multiplicar\n");
    printf("Escriba el número de la operación: ");
    scanf("%d",&opcion);
    switch(opcion)
    {
        case 1: printf("%d + %d = %d\n", a, b, a+b);
                break;
        case 2: printf("%d - %d = %d\n", a, b, a-b);
                break;
        case 3: printf("%d / %d = %d\n", a, b, a/b);
                break;
        case 4: printf("%d x %d = %d\n", a, b, a*b);
                break;
    }
```

```

        default: printf("Opción no valida\n");
                break;
    }
    system("pause");
}

```

El siguiente ejemplo muestra un programa que calcula el sueldo de un jornalero que gana por horas de trabajo, el programa debe cumplir con la disposición de la gerencia que consiste en incrementar la paga dependiendo del número de hijos que tenga el jornalero, para esto se utiliza la siguiente tabla:

**C10:**

```

#include <stdio.h>
#include <stdlib.h>
void main()
{ int hijos, horasT, valorH;
  system("cls");
  printf("Ingrese el número de hijos: ");
  scanf("%d", &hijos);
  printf("Ingrese el número de horas trabajadas: "); scanf("%d", &horasT);
  printf("Ingrese el valor de horas trabajadas: "); scanf("%d", &valorH);
  switch(hijos)
  { case 1:
    case 2: printf("Num. Horas: %d valor horas: %d Incremento 10%\n", horasT, valorH);
            printf("Sueldo: %6.2f \n", horasT * valorH * 1.1);
            break;
    case 3: printf("Num. Horas: %d valor horas: %d Incremento 12%\n", horasT, valorH);
            printf("Sueldo: %6.2f \n", horasT * valorH * 1.12);
            break;
    case 4: printf("Num. Horas: %d valor horas: %d Incremento 15%\n", horasT, valorH);
            printf("Sueldo: %6.2f \n", horasT * valorH * 1.15);
            break;
    case 5:
    case 6:
    case 7:
    case 8: printf("Num. Horas: %d valor horas: %d Incremento 20%\n", horasT, valorH);
            printf("Sueldo: %6.2f \n", horasT * valorH * 1.2);
            break;
    default: printf("Num. Horas: %d valor horas: %d Incremento 0%\n", horasT, valorH);
            printf("Sueldo: %6.2f \n", horasT * valorH);
            break;
  }
  system("pause");
}

```

NÚMERO DE HIJOS	PORCENTAJE DE INCREMENTO
1 o 2	10%
3	12%
4	15%
De 5 hasta 8	20%

Considere el siguiente problema: Un agricultor ha cosechado sandías y las ha puesto a la venta, por la calidad se han clasificado en tres tipos de sandías: tipo A, tipo B y Tipo C los precios son \$1.50, \$3.50 y \$5.00 respectivamente. Dependiendo del número de sandías se hace un descuento, para esto considere el siguiente cuadro:

CALIDAD	CANTIDAD VENDIDA	PORCENTAJE DE DESCUENTO
A	10 - 20	5%
	21 -	10%
B	10 - 30	7%
	31 -	12%

C	10 - 50	15%
	51 -	18%

Para resolver este ejercicio se utilizará la función *getchar()* que pertenece a la librería *stdio.h* (tabla 8), esta función permite devolver el código o carácter de una tecla presionada.

```
C11:  #include <stdio.h>
        #include <stdlib.h>
        void main()
        { int cantidad;
          float total;
          char tipo;
          system("cls");
          printf("Ingrese la calidad (A,B,C): "); tipo=getchar();
          printf("Ingrese el número de sandías a comprar: "); scanf("%d",&cantidad);
          switch(tipo)
          { case 'A':
            case 'a': if(cantidad>=10 && cantidad<=20)
                      total=(cantidad*1.5)*0.95;
                      else if(cantidad>=21)
                      total=(cantidad*1.5)*0.90;
                      else
                      total=cantidad*1.5;
                      break;
            case 'B':
            case 'b': if(cantidad>=10 && cantidad<=30)
                      total=(cantidad*3.5)*0.93;
                      else if(cantidad>=31)
                      total=(cantidad*3.5)*0.88;
                      else
                      total=cantidad*3.5;
                      break;
            case 'C':
            case 'c': if(cantidad>=10 && cantidad<=50)
                      total=(cantidad*5)*0.85;
                      else if(cantidad>=51)
                      total=(cantidad*5)*0.82;
                      else
                      total=cantidad*5;
                      break;
            default: printf("Calidad incorrecta...\n");
                     break;
          }
          printf("Total a cobrar: %6.2f \n",total);
          system("pause");
        }
```

Observe en el ejemplo el uso de la instrucción "*tipo=getchar()*;" cuando el compilador/interprete encuentra esta instrucción, espera que el usuario presione cualquier tecla para devolver el carácter o el código del carácter presionado (recuerde que la memoria solo almacena códigos numéricos y no el dibujo del carácter); este programa se lo puede mejorar utilizando la función *toupper()* que sirve para devolver la mayúscula de uno o varios caracteres alfabéticos pertenece a la librería *ctype.h*, al aplicar la modificación de la instrucción quedaría de la siguiente forma: "*tipo=toupper(getchar())*;", así el programa sería:

```
C12:  #include <stdio.h>
        #include <stdlib.h>
        #include <ctype.h>
        void main()
```



```

{ int cantidad;
  float total;
  char tipo;
  system("cls");
  printf("Ingrese la calidad (A,B,C): "); tipo=toupper(getchar());
  printf("Ingrese el número de sandías a comprar: "); scanf("%d",&cantidad);
  switch(tipo)
  { case 'A': if(cantidad>=10 && cantidad<=20)
              total=(cantidad*1.5)*0.95;
            else if(cantidad>=21)
              total=(cantidad*1.5)*0.90;
            else
              total=cantidad*1.5;
            break;
    case 'B': if(cantidad>=10 && cantidad<=30)
              total=(cantidad*3.5)*0.93;
            else if(cantidad>=31)
              total=(cantidad*3.5)*0.88;
            else
              total=cantidad*3.5;
            break;
    case 'C': if(cantidad>=10 && cantidad<=50)
              total=(cantidad*5)*0.85;
            else if(cantidad>=51)
              total=(cantidad*5)*0.82;
            else
              total=cantidad*5;
            break;
    default: printf("Calidad incorrecta...\n");
            break;
  }
  printf("Total a cobrar: %6.2f\n",total);
  system("pause");
}

```

### **Actividades de refuerzo (AR):**

AR53. Desarrolle un programa que permita mostrar el número de días que tiene un mes considerando si es un año bisiesto, para esto se debe solicitar en valores el mes y el año.

AR54. Se necesita de un programa que permita a los administradores de un teatro verificar si la edad de un asistente es acta o no para ver un espectáculo en una determinada área, para conseguirlo se debe solicitar el año de nacimiento y el número de entradas, considere la siguiente tabla:

EDADES	CATEGORÍA DE PELÍCULA	PRECIOS
34 – 38	1.- Palco central	30
51 – 55	2.- Palco frontal	45
45 – 50	3.- Terraza izquierda	40
39 – 44	4.- Terraza derecha	40
Cualquier edad	5.- Suite presidencial	50

AR55. El supermercado de la ciudad por aniversario está realizando descuentos por las compras mayores a \$50.00, el cliente podrá escoger entre tres colores (Verde, Rojo o Azul) seleccionando la letra del color y número cualquiera (Par o Impar) se aplica el descuento de acuerdo a la siguiente tabla:

COLOR	NÚMERO	% DE DESC.
Verde (V)	Par	10 %

Diseñe un programa que a partir del total de la compra y el color seleccionado, muestre lo que debe pagar dicho cliente.

Rojo (R)	Impar	15 %
	Par	25 %
Azul (A)	Impar	20 %
	Par	30 %
	Impar	35 %

## ESTRUCTURAS DE CONTROL DE PROCESOS REPETITIVOS

Las estructuras de control repetitivas permiten cumplir o ejecutar instrucciones tantas veces como el programador lo determine, para controlarlos el lenguaje C ofrece los siguientes controles: *while()* “Mientras”, *do ... while()* “Hacer ... Mientras” y *for()* “Para”.

### ESTRUCTURA WHILE.

Esta estructura de control primero evalúa una expresión antes de ejecutar o cumplir con las instrucciones escritas en su estructura de repetición, la condición es cualquier expresión simple que al evaluarse devuelve el valor de verdadero o falso; el bucle se repite mientras la condición sea *verdadera*, al momento de ser falsa, el programa pasa a la instrucción siguiente después del cuerpo de la estructura, esta estructura tiene el siguiente formato:

```
while (condición_expresión)
    instrucción_verdadero;
```

En caso de tener más de una instrucción, aplicamos el bloque de sentencias usando las llaves “{ }”.

```
while (condición_expresión) {
    instrucción_verdadero;
    [Más_ instrucciones];
}
```

El siguiente ejercicio muestra los números desde el 1 hasta el 100 separados por tabulaciones, para complementarlo se utilizará la librería *conio.h* de la empresa Borland, este ejemplo no se podrá utilizar en otros compiladores que no pertenezcan a Borland C, considere la siguiente lista de funciones:

**clrscr.-** Esta función tiene la misma utilidad que la función “system(“cls”);”, ambas permiten limpiar la pantalla de cualquier contenido, su sintaxis es:

```
clrscr();
```

**getch.-** Esta función permite leer cualquier tecla presionada, al momento de presionarla, esta devuelve su código, es similar a la función “getchar()”, la diferencia radica en que la función getch() no tiene reflejo, es decir que al presionar la tecla no se muestra en la pantalla, también es utilizada como la función “system(“pause”);”, ya que el programa pausa hasta presionar cualquier tecla, la diferencia con ésta utilidad es que la función getch() no muestra el mensaje “Press any key to continue” su sintaxis es: *getch()*;

```
C13: #include <stdio.h>
#include <conio.h>
main()
{ clrscr();
  int digito=1;
  while (digito<=100) {
    printf("%d\t",digito);
    ++digito;
  }
  getch();
```

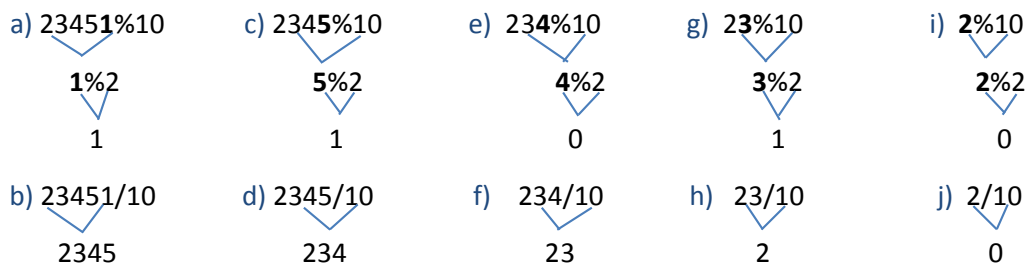
}

El siguiente ejercicio plantea mostrar el número de dígitos pares y el número de dígitos impares que tiene una cantidad entera positiva cualquiera.

Análisis: Para resolver esto se asume que se ha ingresado 23451, ésta cantidad posee dos dígitos pares (2 y 4) y tres dígitos impares (3, 5 y 1), considere los siguientes planteamientos matemáticos que permitirán resolver el ejercicio:

1. Toda cantidad dividida para 10, la coma recorre un puesto a la izquierda, es decir  $23451/10$  tendrá como resultado 2345,1
2. El dígito separado por la coma se considera el residuo, así que lo podemos obtener mediante la división por módulo  $23451\%10$  dará como resultado 1, que es el dígito después de la coma o el residuo
3. Si la variable que contiene 23451 es de tipo entero, al dividirlo para 10 la parte residual se pierde ya que los tipos de datos entero no contienen parte fraccionada o decimales
4. Un número es par si al dividirlo para 2 el residuo es 0, si se obtiene otro resultado entonces es impar

Considere el siguiente procedimiento matemático para ejemplificar el procedimiento técnico descrito en los puntos anteriores:



```
C14:  #include <stdio.h>
      #include <conio.h>
      main()
      { clrscr();
        int Num,ContDigP=0, ContDigI=0;
        printf("Ingrese una cantidad cualquiera:"); scanf("%d",&Num);
        while (Num>0) {
            if((Num%10)%2==0) // extrae cada dígito y comprueba si es par
                ContDigP++;
            else
                ContDigI++;
            Num/=10;
        }
        printf("Existen %d dígitos pares y %d dígitos impares:", ContDigP, ContDigI);
        getch();
      }
```

Considere como se resuelve el siguiente problema: La serie de Fibonacci es una sucesión de valores, generados por una norma infinita de sucesión, la serie comienza con los números 0 y 1, el siguiente es la suma de los dos últimos y así sucesivamente se genera la serie; la siguiente lista de valores corresponde a la aplicación de las reglas de Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610..., El siguiente programa genera la serie de los primeros 15 números de la serie de Fibonacci.

```
C15:  #include <stdio.h>
      #include <conio.h>
```

```

int main()
{ int anterior, actual, siguiente, cont=1;
  clrscr();
  anterior =0;
  actual =1;
  printf("0, 1, "); // printf("%d, %d",anterior, actual);
  while (cont<=15)
  { siguiente = anterior + actual;
    printf("%d, ", siguiente);
    anterior = actual;
    actual = siguiente;
    cont++;
  }
  getch();
  return 0;
}

```

Utilizando el mismo ejemplo, se modificará el programa de forma que muestre los números con diferentes colores; para lograrlo se utilizan las siguientes funciones:

**textcolor().**- esta función permite establecer un color de texto para el sistema, los colores parten desde 0 o BLACK hasta el 15 o WHITE, es decir se puede utilizar los valores de 0 a 15 o los nombres de los colores pero en mayúscula, por defecto se aplican 16 colores que son el formato estándar de los sistemas de video, esta función pertenece a la librería *conio.h*, se pueden aplicar más colores dependiendo de la configuración de las librerías gráficas que posea su compilador.

**TABLA 30.** Lista de colores básicos utilizados en lenguaje C

COLORES		VALOR
BLACK	Negro	0
BLUE	Azul	1
GREEN	Verde	2
CYAN	Cían	3
RED	Rojo	4
MAGENTA	Magenta	5
BROWN	Marrón	6
LIGHTGRAY	Gris Claro	7
DARKGRAY	Gris Oscuro	8
LIGHTBLUE	Azul Claro	9
LIGHTGREEN	Verde Claro	10
LIGHTCYAN	Cían Claro	11
LIGHTRED	Rojo Claro	12
LIGHTMAGENTA	Magenta Claro	13
YELLOW	Amarillo	14
WHITE	Blanco	15

**Fuente:** Autores del libro

**cprintf().**- Esta función pertenece a la librería *conio.h*, tiene el mismo formato y uso que la función *printf()*, la diferencia radica en que muestra con el color establecido por el sistema.

**C16:** #include <stdio.h>  
#include <conio.h>  
int main()  
{ int anterior,actual,siguiente,cont=1;  
 clrscr();  
 anterior =0;  
 actual =1;  
 printf("0, 1,");  
 while (cont<=15)  
 { **textcolor**(cont);  
 siguiente = anterior + actual;  
**cprintf**("%d, ", siguiente);  
 anterior = actual;

```

    actual = siguiente;
    cont++;
}
getch();
return 0;
}

```

### **Actividades de refuerzo (AR):**

AR56. Desarrolle un programa que permita mostrar si un número es o no primo, Un número primo es aquel que solo es divisible para la unidad o para sí mismo, por ejemplo: 1, 2, 3, 5, 7, 11, 13....

AR57. Se necesita de un programa que permita mostrar del 1 al 100 la lista de números impares, además se necesita la suma total de todos los números múltiplos de 3.

AR58. Desarrolle un programa que permita mostrar la siguiente tabla numérica:

```

1
22
333
4444
55555
NNNNN....

```

### **ESTRUCTURA DO-WHILE.**

Esta estructura *do while()* es similar a la aplicada en los apartados anteriores como el “Repetir... Hasta que o Hacer... mientras”, ese ciclo repetitivo primero realiza el proceso que se desea repetir y después evalúa la condición que permite o no repetir el ciclo repetitivo, en otras palabras ésta estructura al ofrecer éste mecanismo de control de repetición, permite o facilita los procesos de validación, su formato básico de sintaxis es:

```

do Instrucción;
while (condición_expresión);

```

Por lo general cuando existe una sola instrucción no necesita utilizar las llaves, pero cuando se tienen más de una instrucción su formato completo es el siguiente:

```

do{
    Instrucciones;
}while (condición_expresión);

```

Esta estructura de control funciona ejecutando un bloque de instrucciones mientras la condición propuesta en el *while()* sea verdadera, al momento que dicha condición se haga falsa, saldrá del ciclo repetitivo y continuará con la siguiente instrucción después del *do...while()*, la propuesta de esta estructura de control, al menos realiza el bloque de instrucciones una vez, ya que la condición de salida (*condición\_expresión*) se encuentra al final de la estructura propuesta.

Para explicar el uso de ésta estructura de control considere el siguiente ejemplo que permite recibir cualquier tecla que presione el usuario y mostrar su correspondiente código ASCII, el programa finalizará cuando el usuario presione la tecla escape.

**C17:**

```

#include <stdio.h>
#include <conio.h>
main()
{ int tecla;
  clrscr();
  do{ tecla=getch();
    printf("%d\n",tecla);
  } while (tecla!=27); //27 es el código de la tecla Esc.

```

```

return 0;
}

```

El siguiente ejemplo que se analizará, permite al usuario mostrar una tabla de multiplicar entre el 2 y el 12, ejemplifica el uso de la herramienta *do...while()* para validar que el número de la tabla sea el correcto y además se lo utiliza para controlar la escritura de la tabla en la pantalla:

**C18:**

```

#include <stdio.h>
#include <conio.h>
main()
{ int tabla, cont=1;
  clrscr();
  do{ printf("Ingrese un número entre 2 y 12:");
    scanf("%d",&tabla);
  } while (tabla<2 || tabla>12);
  do{
    printf("%d x %d = %d\n", tabla, cont, tabla*cont);
    cont++;
  } while (cont<=12);
  getch();
  return 0;
}

```

Considere el siguiente problema: El Instituto de Ciencias Básicas, necesita un programa que permita la simplificación de cualquier quebrado, cada simplificación debe mostrarse por pantalla, antes de finalizar el programa se debe mostrar por pantalla el Máximo Común Divisor (MCD) de todas las simplificaciones.

Análisis: Un quebrado está formado por un numerador y un denominador, la simplificación consiste en dividir tanto el numerador como el denominador para un número común divisible, suponga que se tiene 18/6, como ambos son pares se los divide para 2 quedando 9/3, como ambos son divisibles los volvemos a simplificar dividiéndolos para 3 que dando  $3/1 = 3$ ; el MCD sería  $2 \times 3 = 6$ , el 2 corresponde a la primera simplificación y el 3 corresponde a la segunda simplificación.

**C19:**

```

#include<conio.h>
#include<stdio.h>
void main()
{ int num, den,mcd=1,i=2;
  clrscr();
  printf("ingrese numerador:");scanf("%d",&num);
  printf("ingrese denominador:");scanf("%d",&den);
  printf("%d/%d\n",num,den);
  do{
    if (num%i==0 && den%i==0) {
      mcd=mcd*i;
      num=num/i;
      den=den/i;
      printf("%d/%d\n",num,den);
      i=2; }
    else i++;
  }while(i<=num);
  printf("MCD:%d",mcd);
  getch();
}

```

### **Actividades de refuerzo (AR):**

- AR59. Desarrolle un programa que permita leer varios números entre 1 y 100, el programa finaliza cuando se presiona 0, antes de finalizar el programa debe mostrar el mayor de todos los números ingresados.
- AR60. Desarrolle un programa aplicando “do ... while()” que permita generar los primeros 20 números de la siguiente serie 3,4,6,7,9,10,12,13....
- AR61. Desarrolle un programa que permita al consejo Municipal que tiene N integrantes votar a favor o en contra de una ordenanza del alcalde, el programa debe mostrar el porcentaje de integrantes a favor y el porcentaje que está en contra, para desarrollarlo aplique la estructura de control do ... while para validar el voto de cada integrante del consejo.

## ESTRUCTURA for().

Esta estructura de control permite generar una serie de repeticiones controladas con un formato que incluye en su misma estructura, un valor inicial, la condición de repetición y el incremento o decremento de la variable de control, su sintaxis básica es la siguiente:

```
for (inicialización; condición; incremento/decremento)
    instrucción;
```

En su sintaxis completa incluye el uso de llaves para el bloque de código:

```
for (inicialización; condición; incremento/decremento)
{
    instrucciones;
}
```

Considere que cada elemento que compone la estructura se separan con punto y coma (;), el formato incluye la “*inicialización*” que expresa una simple asignación de un valor inicial a la variable de control, la “*condición*” que especifica una expresión lógica o de comparación para determinar la continuidad o el fin de las repeticiones, y el “*incremento/decremento*” que define una expresión o cálculo simple que modifica mediante un incremento o decremento a la variable de control al final de cada ciclo de repetición; la estructura cumple el siguiente orden de ejecución: *Primero* asigna un valor inicial a la variable de control (*inicialización*), *Segundo* evalúa la condición, si el resultado es verdadero, ejecuta las instrucciones dentro de la estructura, si el resultado es falso, finaliza la estructura de repetición for; ahora cuando termina la ejecución de las instrucciones a repetir, realiza el incremento o decremento de la variable de control y se vuelve al segundo paso. Para ilustrar estos conceptos considere el siguiente ejemplo que muestra un programa que calcula el promedio general de un curso de 15 estudiantes, por cada estudiante el programa pedirá 3 notas:

### C20a:

```
#include <stdio.h>
#include <conio.h>
void main()
{ int x, n1,n2,n3;
  float p,acum=0.0;
  clrscr();
  for(x=1; x<=15; x=x+1) {
    printf("Ingresa notas del estudiante %d:\n",x);
    printf("Ingresa nota 1:"); scanf("%d",&n1);
    printf("Ingresa nota 2:"); scanf("%d",&n2);
    printf("Ingresa nota 3:"); scanf("%d",&n3);
    p=(n1+n2+n3)/3;
    printf("Promedio del estudiante: %2.2f\n",p);
    acum=acum+(n1+n2+n3)/3;
  }
  printf("El promedio general es: %2.2f ",acum/15);
  getch();
```

### C20b:

```
#include <stdio.h>
#include <conio.h>
void main()
{ int n1,n2,n3;
  float acum=0.0;
  clrscr();
  for(int x=1; x<=15; x++) {
    printf("\nIngresa notas del estudiante %d:\n",x);
    printf("Ingresa nota 1:"); scanf("%d",&n1);
    printf("Ingresa nota 2:"); scanf("%d",&n2);
    printf("Ingresa nota 3:"); scanf("%d",&n3);
    printf("Promedio del estud.:%2.2f\n",float((n1+n2+n3)/3));
    acum+=(n1+n2+n3)/3;
  }
  printf("El promedio general es: %2.2f ",acum/15);
  getch();
```

}

}

El control for() que ofrece el lenguaje, es uno de los más completos en comparación a otros lenguajes de programación, la estructura de control según su sintaxis llena o vacía es una estructura que cumplirá los ciclos de repetición, es decir este control permite añadir elementos en su estructura así como quitarlos pero sin eliminar el separador (;) ya que éste es obligatorio, los siguientes ejemplos permitirán identificar las diferentes posibilidades de quitar elementos en el control; el siguiente programa calcula el factorial de un número cualquiera; este cálculo matemático consiste en multiplicar desde el uno hasta el mismo número, por ejemplo  $5! = 1 \times 2 \times 3 \times 4 \times 5$

**C21a:** (Permite declarar la variable de control)

```
#include<stdio.h>

#include<stdlib.h>
void main()
{ int fac;

    long res=1;

    system("cls");
    printf("Ingresa número para calcular el factorial ");

    scanf("%d",&fac);
    for (int x=1; x<=fac; x++)
        res=res*x;
    printf("El factorial de %d es: %ld ", fac,res);

    system("pause");
}
```

**C21b:** (Se quita la inicialización)

```
#include<stdio.h>

#include<stdlib.h>
void main()
{ int fac,x=1;

    long res=1;

    system("cls");
    printf("Ingresa número para calcular el factorial ");

    scanf("%d",&fac);
    for (; x<=fac; x++)
        res=res*x;
    printf("El factorial de %d es: %ld ", fac,res);

    system("pause");
}
```

**C21c:** (Sin inicialización y sin incremento)

```
#include<stdio.h>

#include<stdlib.h>

int main()

{ int fac, x=1;

    long res=1;

    system("cls");

    printf("Ingresa número para calcular el factorial ");

    scanf("%d",&fac);

    for (; x<=fac; ){

        res=res*x;

        x++;

    }

}
```

**C21d:** (Totalmente vacía la estructura)

```
#include<stdio.h>

#include<stdlib.h>

int main()

{ int fac, x=1;

    long res=1;

    system("cls");

    printf("Ingresa número para calcular el factorial ");

    scanf("%d",&fac);

    for ( ; ; ){

        if(x>fac) break;

        res=res*x;

        x++;

    }

}
```



printf("El factorial de %d es: %ld ", fac,res);	}
system("pause");	printf("El factorial de %d es: %ld ", fac,res);
}	system("pause");
	}

Considere que también es posible agregar más expresiones al formato normal del for(), para esto se utiliza la coma (,) como separador de expresiones, por ejemplo el siguiente programa muestra la lista de números *pares* de forma descendente empezando en 100 además muestra los *impares* de forma incrementada empezando desde el uno; el programa termina al momento de cruzarse los pares e impares:

**C22:**

```
#include<stdio.h>
#include<stdlib.h>
void main()
{ system("cls");
  for (int i=1, p=100; i<=p && p>=i ; i+=2, p-=2)
    printf("%d:%d\t", i, p);
  system("pause");
}
```

El siguiente programa muestra una cuenta regresiva en segundos, para el cumplimiento de este propósito considere el uso de cualquiera de las siguientes funciones:

**sleep()** .- Esa función pertenece a la librería <dos.h>, *permite pausar el proceso que realiza el computador en segundos* su formato es el siguiente: sleep(int segundos). En otros compiladores de lenguaje C existe la función *Sleep*(int milisegundos), observe que la primera letra está en mayúscula, tiene la misma utilidad pero el tiempo de espera se expresa en milisegundos, esta función pertenece a la librería <windows.h>.

**delay()**.-Esa función pertenece a la librería <dos.h>, *permite pausar el proceso que realiza el computador en milisegundos* su formato es el siguiente: delay(int milisegundos).

**gotoxy()**.- Esta función permite ubicar el cursor en una posición determinada de la pantalla, el cursor es un indicador que muestra al usuario donde está ubicado, normalmente es una raya parpadeante que puede ser horizontal en modo texto o vertical en Windows, su formato es el siguiente: gotoxy(Pos\_Col, Pos\_Fil); *Pos\_Col* es un valor que representa la ubicación referente a la columna, la pantalla en modo texto posee 80 columnas, es decir que máximo entran 80 letras en una fila, y *Pos\_Fil* es un valor que representa la ubicación en fila, por lo general en su forma más básica la pantalla en modo texto tiene 25 filas de escritura, considere que la pantalla en su forma más básica es una matriz de 80 columnas por 25 filas.

**C23:**

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
void main()
{ clrscr();
  int nseg;
  printf("Ingrese el número de segundos de retardos:");
  do{ scanf("%d",&nseg);
  }while(nseg<1 || nseg>60);
  for (int i=nseg; i>0 ; i--){
    textcolor(i); gotoxy(20,10); cprintf("%d", i);
```

```

        sleep(1);
        textcolor(0); gotoxy(20,10); cprintf("%d", i);
    }
    textcolor(14); cprintf("\nL I S T O . . . !");
    getch();
}

```

### **Actividades de refuerzo (AR):**

- AR62. Desarrolle un programa que permita leer dos números diferentes entre 1 y 100, el programa debe restarlos sin utilizar el signo de sustracción.
- AR63. Desarrolle un programa aplicando “for()” que permita generar y calcular la suma de los primeros 15 quebrados de la siguiente serie:  
 $3/2 + 6/4 + 9/6 + 12/8 + 15/10 + 18/12 + 21/14 + 24/16...$
- AR64. Desarrolle un programa que permita mostrar el tiempo en 30 segundos incluidos los milisegundos, considere que 1 segundo es igual a 1.000 milisegundos.

## **RESUMEN**

Los Lenguajes de programación tienen el propósito de permitir que el programador pueda desarrollar aplicabilidades adicionales a la computadora, los lenguajes de programación incluyen programas que tienen varias utilidades, uno permite editar la escritura de las instrucciones, otro para realizar la compilación con la finalidad de detectar errores de incumplimiento de las regla de escritura o sintaxis, y otro para interpretar y ejecutar las instrucciones, las reglas de los lenguajes de programación deben cumplirse para el compilador/interprete lo entienda y ejecute las instrucciones dadas por el programador, un programa en lenguaje C está estructurado por la cabecera, directivas del preprocesador y la sección de las funciones, el encabezado define la naturaleza de la aplicación mediante la redacción comentada, la directivas del preprocesador incluye librerías con funcionalidades extras y la definición de constantes generales, la sección de funciones incluye la función principal llamada main(), esta función posee las instrucciones principales que el compilador/interprete obedecerá al momento de ejecutar el programa, el lenguaje C posee tipos de datos clasificado en numéricos enteros, numéricos con comas flotantes y el tipo de dato alfanumérico o de registro de caracteres, la información que se contiene en la memoria se encuentra codificada, el lenguaje C utiliza el sistema de codificación inicial básico llamado ASCII, utiliza sus propios operadores aritméticos y lógicos, los operadores de comparación son similares a todos los lenguajes de programación, al momento de desarrollar un programa se puede presentar tres tipos de errores, de sintaxis, de ejecución y de lógica, el lenguaje ofrece controles condicionales como la estructura if() y switch(), controles de ciclos repetitivos como goto, while(), do... while() y for().

## CAPITULO V

### ADMINISTRACIÓN Y GESTIÓN DE ARREGLOS

Las variables son recursos de memoria que permiten contener un y solo un valor o dato para gestionar los procesos requeridos, cuando se presenta la necesidad de utilizar un grupos de valores o un grupo de datos identificados de forma individual, el uso de estas variables independientes no es recomendable, ya que se tendría que declarar cuantas variables sean necesarias, al ser así, la gestión y la administración se torna complicada considerando que cada valor tiene su propio nombre y la redacción de las instrucciones se tornarían extensas al cumplir su propósito, este capítulo ofrece la posibilidad de utilizar variables organizadas en una sola estructura, que comparte un mismo nombre pero se individualizan mediante el uso de índices numéricos únicos, las variables organizadas en forma de arreglos permiten avanzar en el desarrollo de programas más complejos, permite organizar grandes cantidades de datos y ofrecen la posibilidad de aplicar métodos de ordenamiento, permite clasificarlos y gestionarlos de forma más simple en comparación a las variables tratadas en los capítulos anteriores.

El estudio de este capítulo permitirá al lector comprender la gestión de grandes cantidades de datos, explica y ejemplifica la gestión de cadenas de caracteres, arreglos unidimensionales como los vectores, arreglos multidimensionales como matrices y cubos de datos, fundamenta la organización interna de la memoria al administrar grandes cantidades de datos, el contenido tratado en estos apartados definen las bases aplicadas en contenidos de gestión de archivos y estructura de datos.

**OBJETIVO DE APRENDIZAJE DEL SIGUIENTE APARTADO:** Conocer y administrar variables que gestionan grupos de caracteres alfanuméricos, aplicar herramientas disponibles del lenguaje incluidas en las librerías estándares como `string.h`, ejemplificar propuestas de gestión con menús en la pantalla entre otros.

### USO DE CADENAS DE CARACTERES

La manipulación de cadenas de caracteres está íntimamente ligado al uso de múltiples variables de tipo *char*, las variables de tipo *char* solo pueden almacenar un byte de espacio de memoria, es decir solo un carácter, por ejemplo:

```
char letra='s';
```

Las dificultades de uso se presentan al momento de desear almacenar más de un carácter en una variable, para resolver esta necesidad se propone el uso de cadenas de caracteres, que consiste en utilizar una variable tipo *char* organizada en forma de vector, considere que el lenguaje C no contiene un tipo de dato que satisfaga ésta necesidad como los tipos *string* que poseen otros lenguajes de programación, sin embargo, el lenguaje C incluye algunas funcionalidades que se incluyen en la biblioteca `<string.h>`.

Al agrupamiento de varios caracteres se le denomina cadenas de caracteres, una cadena en C es un array de caracteres de una dimensión (vector de caracteres), esta estructura de variable define que el primer carácter está en la posición cero y el último carácter está marcado por el carácter especial `'\0'` (barra inclinada y el cero) que significa *fin de la cadena*, la sintaxis para declarar una cadena en C es:

```
char nombre_variable[n];
```

Considere que `[n]` representa un valor entero positivo mayor que 1, el primer carácter siempre se ubicara en la posición `[0]` cero y el final de la cadena en la posición `[n-1]`, por ejemplo considere la siguiente declaración de una variable:

```
char m[8]="Andinos";
```

La variable `m` reserva 8 espacios de memoria que se distribuyen de la siguiente forma:

A	n	d	i	n	o	s	\0
---	---	---	---	---	---	---	----

---

m[0] m[1] m[2] m[3] m[4] m[5] m[6] m[7]

Observe que la representación interna de una cadena de caracteres está terminada por el símbolo '\0' en la posición [n-1], por seguridad se recomienda reservar el espacio de memoria, adicionando un espacio más "n+1"; el carácter '\0' es un elemento implícito (lo pone de forma automática el lenguaje) o explícito (lo pone el programador a conveniencia) de una cadena de caracteres y solo sirve para demarcar el final de una cadena de caracteres, es importante destacar que el carácter no es mostrable al utilizar cualquier función como el printf().

Al momento de declarar una variable de tipo de cadena de caracteres, también es posible inicializarlas (dar un valor inicial), considere los siguientes ejercicios de ejemplo:

```
C24:  #include <stdio.h>
        #include <stdlib.h>
        void main ()
        { char s1[5] = "Hola";
          char s2[5] = {'H','o','l','a',0};
          char s3[5] = {'H','o','l','a','\0'};
          system("cls");
          printf ("\nLas cadena son:\n");
          printf ("%s\n%s\n%s\n",s1,s2,s3);
          system("pause");
        }
```

También es posible crear variables sin definir el tamaño de las cadenas de caracteres:

```
C25:  #include <stdio.h>
        #include <stdlib.h>
        void main ()
        { char s1[5] = "Hola";
          char s2[5] = {'H','o','l','a',0};
          char s3[5] = {'H','o','l','a','\0'};
          char c1[] = "Chao hasta pronto...!";
          char c2[] = {'H','a','s','t','a',' ','l','u','e','g','o',0};
          char c3[] = {'B','y','e',' ','b','y','e','\0'};
          system("cls");
          printf ("\nLas cadena son:\n");
          printf ("%s\n%s\n%s\n",s1,s2,s3);
          printf ("%s\n%s\n%s\n",c1,c2,c3);
          system("pause");
        }
```

El uso de los corchetes vacíos "[]" es para variables cuya limitación de caracteres no está definida, la inicialización puede contener cualquier número de caracteres, pero cuando necesitamos una variable de ilimitados números de caracteres que no necesita una inicialización, lo recomendable es utilizar el asterisco (\*) anteponiéndoselo al nombre de la variable, por ejemplo (char \*dirección;), el asterisco define a una variable que contiene la dirección inicial de un grupo de espacios cuyo final será dado por el carácter '\0', observe el siguiente ejemplo:

```
char *nombre;
printf( "ingrese nombres y apellidos:"); gets(nombre);
printf( "Usted ha ingresado:%s", nombre );
```

Observe que en el ejemplo se está utilizando la función gets() que pertenece a la librería stdio.h, esta función es exclusiva para leer cadenas de caracteres, tome en cuenta que para leer varias palabras desde el teclado utilizando la función scanf() presenta dificultades, ya que al utilizar el espacio para la separación de cada palabra, la función asume que el contenido escrito corresponde a otra variable y el buffer interno se distribuye o se divide, considere que solo el contenido de una distribución pasa a la variable y las otras se mantienen en el buffer esperando el acceso a otras variables, es decir la función scanf() solo podría ser utilizada para escribir una palabra más no para escribir frases; otra

complicación resulta al comparar dos cadenas de caracteres ya que al ser arreglos o vectores de tipo char (grupos de variables) no es posible utilizar los operadores de comparación (==, >, <, etc) así como el operador de asignación igual (=) ya que estos operadores están desarrollados para operar sobre un valor y no sobre varios al mismo tiempo.

El siguiente ejemplo aplica la lectura de una frase utilizando scanf() y gets(), al ejecutarlo notará la diferencia en el resultado obtenido:

**STRING1:** #include <stdio.h>

#include <stdlib.h>

main()

{ char frase[40];

system("cls");

printf("Ingrese una frase:");

scanf("%s",frase);

printf("Lo ingresado es:\n %s \n",frase);

system("pause");

return 0;

}

**STRING2:** #include <stdio.h>

#include <stdlib.h>

main()

{ char frase[30];

system("cls");

printf("Ingrese una frase:");

gets(frase);

printf("Lo ingresado es:\n %s \n",frase);

system("pause");

return 0;

}

Observará que el ejemplo de STRING1 al ingresar una frase solo se registrará la palabra antes del primer espacio, en el ejemplo STRING2 ese inconveniente es superado ya que se está utilizando la función gets() que permite leer cadenas de caracteres, su funcionamiento y explicación se ampliará con mayor detalle más adelante en este mismo capítulo.

Considerando las dificultades presentadas con los operadores de comparación así como el operador de asignación, se recomienda utilizar las siguientes funciones de la librería *string.h* que permiten ampliar las posibilidades de uso de las cadenas de caracteres:

**strlen(cadena\_caracteres).** - Esta función devuelve el número de caracteres o la longitud de la cadena, sin tomar en cuenta el carácter de final de cadena '\0'.

```
char c1[50] = "Dios";
```

```
char c2[] = "siempre perdona cuando te arrepientes";
```

```
printf("strlen(c1) = %d\nstrlen(c2) = %d\n", strlen(c1), strlen(c2) );
```

Devolverá como resultado 4 para C1 y 37 para C2

**STRING3:** #include <stdio.h>

#include <stdlib.h>

#include <string.h>

main()

{ char frase[100];

system("cls");

printf("Ingrese una frase:");

gets(frase);

printf("Lo ingresado es:\n %s \n",frase);

printf("La frase tiene %d caracteres\n",strlen(frase));

system("pause");

return 0;

```
}
```

**strcpy**(cadena\_destino, cadena\_origen).- Esta función reemplaza al operador de asignación (=) para las asignaciones de las cadenas de caracteres, es decir permite copiar el contenido de *cadena\_origen* a *cadena\_destino*.

```
char c2[20] = "bendiciones..!";
char c1[20];
strcpy( c1, c2 );
printf( "c2=%s\n", c2 );
printf( "c1=%s\n", c1 );
```

Ambas cadenas contienen el mismo contenido.

```
STRING4:#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main()
{ char frase[100],frase2[100];
  system("cls");
  printf("Ingrese una frase:");
  gets(frase);
  printf("Lo ingresado es:\n %s \n",frase);
  printf("La frase tiene %d caracteres\n",strlen(frase));
  strcpy(frase2,frase); //frase2=frase
  printf("Lo copiado es:\n %s \n",frase2);
  system("pause");
  return 0;
}
```

**strcat**(cadena\_destino, cadena\_origen).- Esta función permite unir dos cadenas de texto en una sola, funciona concatenando el contenido de *cadena\_origen* al final de *cadena\_destino*.

```
char c1[11] = "Estudiar ";
char c2[18] = "da frutos";
strcat( c1, c2 );
printf( "c1=%s", c1 );
```

Al utilizar strcat() el contenido de la variable c1 es "Estudiar da frutos"

```
STRING5:#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main()
{ char frase[100],frase2[100]="Usted piensa:";
  system("cls");
  printf("Ingrese una frase:");
  gets(frase);
  printf("Lo ingresado es:\n %s \n",frase);
  printf("La frase tiene %d caracteres\n",strlen(frase));
  strcat(frase2,frase);
  printf("Lo concatenado es:\n %s \n",frase2);
  system("pause");
  return 0;
}
```

**strcmp**(cadena1, cadena2) y **strcmpi**(cadena1, cadena2).- Estas funciones reemplazan los operadores de comparación (==,>,<) para las cadenas de caracteres, funciona comparando *cadena1* y *cadena2*, dependiendo del resultado que arroja se conoce que operador aplica:

**TABLA 31.** Cuadro de resultados utilizando strcmp() y strcmpi()

**Sí el resultado es:**      Significa:

<b>0</b>	Que ambas cadenas de caracteres son iguales
<b>Mayor que 0 o 1</b>	Que <i>cadena1</i> es mayor que <i>cadena2</i>
<b>Menor que 0 o -1</b>	Que <i>cadena1</i> es menor que <i>cadena2</i> o dicho de otra forma <i>cadena2</i> es mayor que <i>cadena1</i>

**Fuente:** Autores del libro

Es importante destacar que, para la función *strcmp()* la letra “a” minúscula es diferente a la letra “A” mayúscula, es decir si diferencia entre mayúsculas y minúsculas; pero para la función *strcmpi()*, la letra “a” minúscula es igual a la letra “A” mayúscula, es decir no diferencia entre mayúsculas y minúsculas.

char c1[5] = "Abeja";	char c1[5] = "Abeja";
char c2[5] = "abeja";	char c2[5] = "abeja";
if(strcmp( c1, c2 ) == 0 ) printf( "Son iguales" );	if(strcmpi( c1, c2 ) == 0 ) printf( "Son iguales" );
else printf( "Son diferentes" );	else printf( "Son diferentes" );
El resultado de este ejemplo es: <i>"Son diferentes"</i>	El resultado de este ejemplo es: <i>"Son iguales"</i>

El siguiente ejercicio ejemplifica el uso de las 2 funciones *strcmp* y *strcmpi* al comparar dos palabras:

```

STRING6: #include <stdio.h>
            #include <stdlib.h>
            #include <string.h>
            main()
            { char palabra[20],palabra2[20];
              system("cls");
              printf("Ingrese una palabra:"); gets(palabra);
              printf("Ingrese otra palabra:"); gets(palabra2);
              printf("\nU t i l i z a n d o  strcmp():\n");
              if(strcmp(palabra,palabra2)==0)
                  printf("La palabra:%s y la palabra:%s son iguales\n",palabra,palabra2);
              else if(strcmp(palabra,palabra2)>0)
                  printf("La palabra:%s ES MAYOR que la palabra:%s\n",palabra,palabra2);
              else
                  printf("La palabra:%s ES MENOR que la palabra:%s\n",palabra,palabra2);
              printf("\nU t i l i z a n d o  strcmpi():\n");
              if(strcmpi(palabra,palabra2)==0)
                  printf("La palabra:%s y la palabra:%s son iguales\n",palabra,palabra2);
              else if(strcmpi(palabra,palabra2)>0)
                  printf("La palabra:%s ES MAYOR que la palabra:%s\n",palabra,palabra2);
              else
                  printf("La palabra:%s ES MENOR que la palabra:%s\n",palabra,palabra2);
              system("pause");
              return 0;
            }

```

**puts (cadena\_caracteres).**- Esta función pertenece a la librería *stdio.h*, se la utiliza para mostrar cadenas de caracteres, esta función no necesita el uso del “\n”, ya que de forma automática produce un salto de línea.

**gets().**-Esta función pertenece a la librería *stdio.h*, permite leer desde el teclado solo cadenas de caracteres, a diferencia de la función *scanf()*, la función *gets()* permite el uso del espacio y todos los símbolos de redacción como la coma (,) sin alterar el registro de ingreso de datos a la variable, para

finalizar el proceso solo es necesario presionar la tecla ENTER; para ilustrar el uso de las funciones *puts()* y *gets()* considere el siguiente ejemplo:

```
C26:    #include <stdio.h>
        #include <stdlib.h>
        void main()
        { char cadena[50];
          system("cls");
          puts("Escriba su nombre y apellido:"); gets(cadena);
          printf("Eres afortunado:");
          puts(cadena);
          system("pause");
        }
```

Observe que se está utilizando tanto la función *printf()* como la función *puts()* en el mismo programa ya que la combinación no se afecta el normal desenvolvimiento del programa; el problema surgiría al utilizar *gets()* y *scanf()*, el uso de la función *gets()* está limitada al ingreso de cadenas de caracteres, mientras que la función *scanf()* es una instrucción mucho más amplia en cuanto a sus posibilidades en la que puede ser utilizada para diferentes tipos de datos, esto genera distribuciones y seccionamientos en el uso del *buffer* (memoria temporal entre el teclado y la RAM), en el ejemplo C27A observará que no funciona *gets()* después de utilizar *scanf()*, ya que se produce un seccionamiento en el buffer, en el ejemplo C27B, *gets()* funciona ya que no utiliza memoria seccionada y *scanf()* la secciona después:

```
C27A:  #include <stdio.h>
        #include <stdlib.h>
        int main(){
        char cadena[50];
        int a;
        system("cls");
        printf("Ingrese un entero: "); scanf("%d",&a);
        printf("Ingrese otra cadena: "); gets(cadena);
        printf("El entero ingresado es:%d\n",a);
        printf("La cadena ingresada es:"); puts(cadena);
        system("pause");
        return 0;
        }
```

```
C27B:  #include <stdio.h>
        #include <stdlib.h>
        int main(){
        char cadena[50];
        int a;
        system("cls");
        printf("Ingrese otra cadena: "); gets(cadena);
        printf("Ingrese un entero: "); scanf("%d",&a);
        printf("El entero ingresado es:%d\n",a);
        printf("La cadena ingresada es:"); puts(cadena);
        system("pause");
        return 0;
        }
```

Existen programas en los que se necesitará utilizar el orden de ingreso utilizado en el ejemplo C27A, para resolver el problema se utiliza la función *fflush()*:

**fflush().**- Esta función sirve para limpiar los buffer, tanto en la entrada como en la salida, los buffer son necesarios ya que mantienen los datos de forma temporal cuando realizan procesos de entrada y de salida, por alguna razón los datos no son vaciados de forma adecuada y es cuando se necesita utilizar ésta función, puede utilizar como argumento "stdin" que significa 'Standard input', es para limpiar el ingreso por teclado; "stdout" que sirve para vaciar el buffer de escritura o de salida.

```
C28:  #include <stdio.h>
        #include <stdlib.h>
        int main(){
        char cadena[50];
        int a;
        system("cls");
        printf("Ingrese un entero: "); scanf("%d",&a);
        fflush(stdin);
        printf("Ingrese otra cadena: "); gets(cadena);
```



```

printf("El entero ingresado es:%d\n",a);
printf("La cadena ingresada es: "); puts(cadena);
system("pause");
return 0;
}

```

Considere el siguiente problema: Se necesita un programa que permita ingresar una frase cualquiera, el programa debe mostrar el número de vocales a, e, i, o y u, que existen en la frase ingresada.

**C29:**

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(){
char *frase;
int ca,ce,ci,co,cu;
ca=ce=ci=co=cu=0;
system("cls");
printf("Ingrese una frase preferida: "); gets(frase);
for (int i=0;i<strlen(frase);i++)
    switch(frase[i]){
        case 'A':
        case 'a': ca++; break;
        case 'E':
        case 'e': ce++; break;
        case 'I':
        case 'i': ci++; break;
        case 'O':
        case 'o': co++; break;
        case 'U':
        case 'u': cu++; break;
    }
printf("El total de vocal a=%d\t e=%d\t i=%d\t o=%d\t u=%d\n ",ca,ce,ci,co,cu);
system("pause");
return 0;
}

```

**tolower(int c).**-Esta función pertenece a la librería *ctype.h* sirve para transformar de mayúscula a minúscula, las variables del tipo char internamente contienen códigos ASCII de cada carácter en vez del dibujo del carácter, se trata de un valor que comprende entre 0 y 255, la función *tolower()* le suma 32 al número entero correspondiente al código ASCII del carácter, por ejemplo "A" tiene código ASCII 65, si le sumamos 32 dará 97 que es el código ASCII de "a".

Para ilustrar el uso de ésta funciones considere el siguiente problema: Una palabra palíndroma es una palabra que se lee igual de izquierda a derecha como de derecha a izquierda, por ejemplo: Aérea, Ama, Ana, Anilina, Nadan, Neuquén, Ojo, Orejero, Oro, Erigiré, Radar, Rajar, Rallar, Reconocer, Rodador, Rotomotor, Rotor, Salas, Sedes, entre otras. Se necesita un programa que permita el ingreso de una palabra y mostrar si es o no palíndroma.

**C30:**

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
int main(){
char palabra[50];
int iz, de;
system("cls");
printf("Ingrese una palabra: "); gets(palabra);
for (iz=0,de=strlen(palabra)-1; iz<de && de>iz ;iz++,de--)

```

```

        if (tolower(palabra[iz]) != tolower(palabra[de])) break;
        if(iz<de) printf("La palabra %s no es palíndroma\n ",palabra);
        else printf("La palabra %s si es palíndroma\n ",palabra);
        system("pause");
        return 0;
    }

```

### **Actividades de refuerzo (AR):**

AR65. Desarrolle un programa que permita contar cuantas palabras tiene una frase.

AR66. Una frase palíndroma es aquella que se puede leer de izquierda a derecha y de derecha a izquierda sin considerar los espacios, por ejemplo: “Amad a la dama”, “Aman a Panamá”, “Anita lava la tina”, “Anula la luz azul a la Luna”, entre otras. Desarrolle un programa que permita ingresar una frase y mostrar si la frase es o no palíndroma.

**OBJETIVO DE APRENDIZAJE DEL SIGUIENTE APARTADO:** Conocer y administrar estructuras de variables en arreglos (arrays) tales como vectores (arreglos unidimensionales), matrices (bidimensionales) y otros arreglos como los cubos (multidimensionales).

## **INTRODUCCIÓN A LOS ARRAYS**

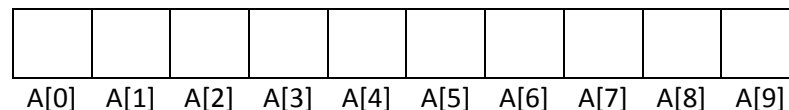
Un arreglo es un grupo de espacios de memoria (grupos de variables) que se organiza por ubicaciones con la finalidad de almacenar información del mismo tipo de dato y con el mismo propósito. Cada posición referencia a una celda de almacenamiento que a partir de ahora se denominará elemento del arreglo, por ejemplo el vector es un array unidimensional, esto significa que utilizará un índice o valor que referencia la posición de una celda de almacenamiento; una matriz es un array bidimensional, esto quiere decir que utilizará dos índices o valores para referenciar la posición de una celda de almacenamiento.

Para declarar la existencia de un arreglo se escribe: primero el tipo de dato, se deja un espacio y de forma seguida el nombre del vector y por último el número de elementos o celdas de almacenamiento encerrado entre corchetes, cada corchete encierra una dimensión del arreglo, considere las siguientes declaraciones y su respectiva representación interna que ejemplifican los arreglos y sus dimensiones:

El siguiente ejemplo declarar un vector de 10 elementos de tipo entero:

```
int A[10];
```

Observe que solo utiliza una dimensión de 10 elementos, su representación gráfica sería:

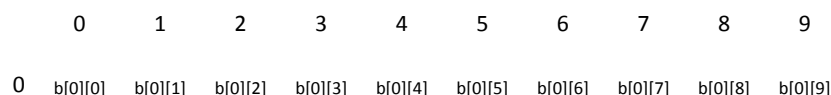


Cada elemento del arreglo unidimensional está identificado por un índice que referencia su posición referente al vector, al ser el vector de tipo entero, cada celda podrá almacenar entre -32768 y 32767.

El siguiente ejemplo declara una matriz de 8 filas y cada una de las filas con 10 columnas, ésta matriz podrá almacenar valores con decimales:

```
float b[8][10]; //el primer índice es para la fila y el segundo índice es para la columna
```

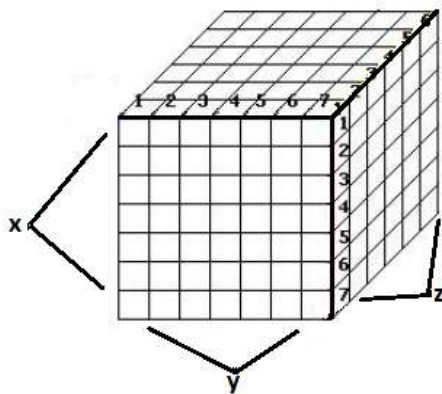
Observe que en la declaración se utilizan dos dimensiones, la primera dimensión de referencia 8 filas y la segunda dimensión 10 columnas por cada fila, su representación gráfica sería:



1	b[1][0]	b[1][1]	b[1][2]	b[1][3]	b[1][4]	b[1][5]	b[1][6]	b[1][7]	b[1][8]	b[1][9]
2	b[2][0]	b[2][1]	b[2][2]	b[2][3]	b[2][4]	b[2][5]	b[2][6]	b[2][7]	b[2][8]	b[2][9]
3	b[3][0]	b[3][1]	b[3][2]	b[3][3]	b[3][4]	b[3][5]	b[3][6]	b[3][7]	b[3][8]	b[3][9]
4	b[4][0]	b[4][1]	b[4][2]	b[4][3]	b[4][4]	b[4][5]	b[4][6]	b[4][7]	b[4][8]	b[4][9]
5	b[5][0]	b[5][1]	b[5][2]	b[5][3]	b[5][4]	b[5][5]	b[5][6]	b[5][7]	b[5][8]	b[5][9]
6	b[6][0]	b[6][1]	b[6][2]	b[6][3]	b[6][4]	b[6][5]	b[6][6]	b[6][7]	b[6][8]	b[6][9]
7	b[7][0]	b[7][1]	b[7][2]	b[7][3]	b[7][4]	b[7][5]	b[7][6]	b[7][7]	b[7][8]	b[7][9]

Observe que para referenciar a cada celda se utilizan dos índices, el primero para la fila y el segundo para las columnas, para el ejemplo se han creado 80 (8x10) celdas. Para declarar una matriz de tres dimensiones o también conocidos como cubos, se utilizan tres dimensiones:

```
long c[7][7][6]; //las dimensiones se representan en x (filas), y (Columnas), z (Nivel)
```



Es importante destacar que cada celda de cualquier arreglo es el equivalente al espacio utilizado por una determinada variable, así:

El vector “A” del ejemplo es de tipo entero “int”, cuando el compilador encuentra esta declaración reserva la memoria necesaria para los 10 elementos del vector, considerando que cada variable de tipo entero ocupa 2 byte de memoria, el compilador reservaría 20 byte en la memoria principal para el vector.

De forma similar funciona la reserva de memoria para la matriz “b”, considere que cada celda es de tipo float lo que significa que, separa 4 byte por cada celda, esto se traduce en reservar 8x10x4, ocho filas, diez columnas y cuatro byte por celda, dando como resultado 320 byte de memoria RAM para uso exclusivo de la matriz, esto ocurre para cada arrays sin importar sus dimensiones.

La manipulación de cada elemento del arreglo, se la aplica de la misma forma que fue utilizada las cadenas de caracteres, por ejemplo, el primer elemento del vector tiene la posición [0], el último elemento tiene la posición [n-1], es importante destacar que en arreglos de tipo numéricos no se utiliza ‘\0’ el carácter de fin de cadena, pero si utiliza índices para hacer referencia a cada elemento del arreglo, la inicialización de un arreglo se lo aplica de la misma forma utilizada hasta ahora:

```
int vector1[5] = { 10, 20, 30, 40, 50 };
int vector2[] = { 10, 20, 30, 40, 50 };
```

El siguiente programa almacena 10 números cualquiera, el programa deberá mostrar primero los números pares y después los números impares:

```
C31: #include <stdio.h>
#include <stdlib.h>
int main()
```

```

{ int i, vector[10];
  system("cls");
  for(i = 0; i < 10; i++){
    printf("Ingresa el número %d: ",i+1);
    scanf("%d",&vector[i]);
  }
  printf("\nlos números pares ingresados son:\n");
  for(i = 0; i < 10; i++)
    if(vector[i]%2 == 0) printf("%d\t",vector[i]);
  printf("\nlos números impares ingresados son:\n");
  for(i = 0; i < 10; i++)
    if(vector[i]%2 != 0) printf("%d\t",vector[i]);
  printf("\n");
  system("pause");
  return 0;
}

```

El siguiente ejemplo utiliza números *generados automáticamente* de forma aleatoria o randómica, para lograrlo utiliza las siguientes funciones:

**randomize()**.- Esta función pertenece a la librería *stdlib.h*, sirve para generar un valor inicial por el cual se podrá generar una serie de número al azar con valores diferentes, para lograrlo usa el reloj interno como semilla, ya que éste siempre será diferente al momento de generarlo, su sintaxis es:

```
randomize();
```

**rand()** .- Esta función también pertenece a la librería *stdlib.h*, sirve para tomar un número al azar de una serie de valores iniciados por *randomize()*, su sintaxis es variada según el propósito, por ejemplo:

```
rand();
```

Al utilizarla de ésta forma, retorna números entre 0 y 2.147.483.647.

Para limitar el número a retornar se necesita hacer lo siguiente:

```
rand()%101;
```

Al utilizar la función de esta forma, retorna números entre 0 y 100.

La siguiente sintaxis permite tomar números definiendo el límite inferior y superior:

```
10+(rand()%91);
```

Este ejemplo retorna números entre 10 y 100:

**C32:**

```

#include <stdio.h>
#include <stdlib.h>
int main()
{ int i, vector[10];
  system("cls");
  randomize();
  for(i = 0; i < 10; i++) vector[i]= rand()%101;
  printf("\nlos números pares tomados son:\n");
  for(i = 0; i < 10; i++)
    if(vector[i]%2 == 0) printf("%d\t",vector[i]);
  printf("\nlos números impares tomados son:\n");
  for(i = 0; i < 10; i++)
    if(vector[i]%2 != 0) printf("%d\t",vector[i]);
  printf("\n");
  system("pause");
  return 0;
}

```

Considere el siguiente problema: se necesita un programa que permita ingresar una lista de N elemento con una longitud máxima de 100 elementos, el programa no debe permitir ingresar números repetidos:

```
C33:  #include <stdio.h>
      #include <stdlib.h>
      int main()
      { int vector[100];
        int max, encontrado,i;
        system("cls");
        printf("Ingrese el número de elementos del vector:\n");
        scanf("%d",&max);
        for(i = 0; i < max; i++){
            printf("Ingresa el número %d: ",i+1);
            do {
                scanf("%d",& vector[i]);
                encontrado=0;
                for(int b=0; b < i && encontrado==0; b++)
                    if (vector[b]==vector[i]) encontrado=1;
                if(encontrado) printf("Número repetido...! \nVuelva a ingresarlo: ");
            } while(encontrado);
        }
        system("cls");
        for(i = 0; i < max; i++) printf("%d \t", vector[i]);
        system("pause");
        return 0;
      }
```

### **Actividades de refuerzo (AR):**

- AR67. Desarrolle un programa que permita ingresar un número binario de máximo 8 bit, el programa debe mostrar el número transformado en decimal.
- AR68. Desarrolle un programa que permita ingresar un número entre 1 y 255, el programa deberá transformarlo a binario y almacenarlo en un vector de no más de 8 elementos, antes de finalizar el programa mostrar el binario.

## **VECTORES DE CADENAS DE CARACTERES**

Cuando se desea utilizar una lista de palabras de forma similar al uso de un vector numérico, se encontrará que una cadena de caracteres como tal, es un vector o un arreglo unidimensional, recuerde que cada elemento del arreglo solo tiene el espacio para contener un carácter y no una palabra; para dar el mismo tratamiento de la lista de valores a una lista de palabras, se debe recurrir al uso de la matriz como si se tratase de un vector, es decir que para crear un vector de palabras se debe crear una matriz o un arreglo con dos índices, donde el primer índice corresponde a la ubicación de cada fila y se lo define como la ubicación de cada elemento del vector de palabras, y el segundo índice corresponde a la posición de cada letra de la palabra, por ejemplo el siguiente código muestra la inicialización de un vector que contiene los días de la semana y el otro vector contiene los diferentes estados civil:

```
char dia[8][10]={“Lunes”, “Martes”, “Miércoles”, “Jueves”, “Viernes”, “Sábado”, “Domingo”}
char EstCivil[6][13] = { {‘s’,‘o’,‘l’,‘t’,‘e’,‘r’,‘o’}, {‘c’,‘a’,‘s’,‘a’,‘d’,‘o’}, {‘d’,‘i’,‘v’,‘o’,‘r’,‘c’,‘i’,‘a’,‘d’,‘o’},
                        {‘v’,‘i’,‘u’,‘d’,‘o’}, {‘u’,‘n’,‘i’,‘ó’,‘n’}};
```

Observe que para declarar un vector de palabras necesita los dos índices, se puede inicializar en grupo utilizando las comillas ("" ) o de forma individual utilizando los apostrofes (‘’), también podemos asignar datos de forma individual por fila, observe en el siguiente ejemplo donde el

arreglo se comporta como un vector, ya que para referenciar un elemento (fila) solo se utiliza el primer índice:

```
char nombres[10][20];
strcpy(nombres[0], "Gabriel");
strcpy(nombres[1], "Paola");
strcpy(nombres[2], "Alejandro");
strcpy(nombres[3], "María");
gets(nombres[4]);
scanf("%s",nombres[5])
puts(nombres[0]);
puts(nombres[3]);
printf("%s",nombres[4]);
```

Los siguientes ejemplos describen las diferentes formas de gestionar vectores de cadenas de caracteres, el siguiente ejemplo crea un menú utilizando la librería *conio.h*, permite utilizar las teclas de dirección *izquierda* y *derecha* para permitir al usuario ubicarse en cada opción; para comprender mejor la propuesta del funcionamiento del menú, considere el uso de las siguientes funciones:

**\_setcursortype()**.- Esta función pertenece a la librería *conio.h*, permite cambiar la apariencia del cursor en cualquiera de las tres siguientes formas:

**TABLA 32.** Lista de formatos para cambiar la apariencia del cursor

<b>_setcursortype(_NOCURS);</b>	<b>Oculto o desactiva el cursor</b>
<b>_setcursortype(_NORMALCURSOR);</b>	Muestra el cursor normal
<b>_setcursortype(_SOLIDCURSOR);</b>	Muestra el cursor como un cuadrado relleno

**Fuente:** Autores del libro

**strlwr()**.-Esta función pertenece a la librería *string.h*, permite convertir el contenido de una cadena de caracteres de mayúsculas a minúsculas, el resultado de la transformación es almacenado en la misma cadena de caracteres.

**strupr()**.-Esta función también pertenece a la librería *string.h*, permite convertir el contenido de una cadena de caracteres de minúsculas a mayúsculas, el resultado de la transformación es almacenado en la misma cadena de caracteres.

```
C34: #include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
void main()
{ char menu[5][15]={"sumatoria","sustracción","multiplicación","división","salir"};
  int tecla, col=2, i, indice=0;
  int a,b;
  clrscr();
  //proceso de imprimir el menú en pantalla
  textcolor(YELLOW);
  for (i=0;i<5;i++){ gotoxy(col,2);cprintf("%s",menu[i]);
                      col+=15;}

  col=2; i=1;
  //Proceso de controlar el movimiento
  do{
    do{ _setcursortype(_NOCURS);
        textcolor(RED);
        gotoxy(col,2);cprintf("%s",strupr(menu[indice]));
        tecla=getch();
    }while(tecla!=77 && tecla != 75 && tecla!=13);
    textcolor(YELLOW);
    gotoxy(col,2);cprintf("%s",strlwr(menu[indice]));
```

```

//Proceso de chequear la tecla presionada
switch(tecla){
case 77: if (indice==4){col=2;indice=0;}
        else {col+=15;indice++;}
        break;
case 75: if (indice==0){col=62;indice=4;}
        else {col-=15;indice--;}
        break;
case 13: _setcursortype(_NORMALCURSOR);
        if (indice==0){gotoxy(10,10);printf("Ingrese un número:");scanf("%d",&a);
        gotoxy(10,11);printf("Ingrese otro número:");scanf("%d",&b);
        gotoxy(10,12);printf("La sumatoria da:%d\n",a+b);
        gotoxy(10,13);system("pause");
        gotoxy(10,10);printf("                ");
        gotoxy(10,11);printf("                ");
        gotoxy(10,12);printf("                ");
        gotoxy(10,13);printf("                ");
        }
        if (indice==1){}
        if (indice==2){}
        if (indice==3){}
        if (indice==4) i=0;
    }
}while(i);
}

```

**textbackground(int color).**- Esta función pertenece a la librería *conio.h*, permite establecer un color de fondo especificado por el argumento *color*, su uso debe ser complementado con las funciones que envían datos a la pantalla en modo texto. El argumento *color* debe ser un número entero entre 0 y 7 de la tabla 30; recuerde que también se pueden utilizar los nombres de los colores en inglés y en mayúsculas de forma similar a los utilizados en la función *textcolor()*.

**C35:** `#include <stdio.h>`  
`#include <conio.h>`  
`#include <string.h>`  
`#include <stdlib.h>`  
`void main()`  
`{ char menu[5][15]={"Sumatoria","Sustracción","Multiplicación","División","Salir"};`  
`int tecla, col=2, i, indice=0;`  
`int a,b;`  
`clrscr();`  
`//proceso de imprimir el menú en pantalla`  
`for (i=0;i<5;i++){ gotoxy(col,2);cprintf("%s",menu[i]);`  
`col+=15;}`  
`col=2;i=1;`  
`//Proceso de controlar el movimiento`  
`do{`  
`do{_setcursortype(_NOCURSOR);`  
`textbackground(6);`  
`gotoxy(col,2); cprintf("%s",menu[indice]);`  
`tecla=getch();`  
`}while(tecla!=77 && tecla != 75 && tecla!=13);`  
`textbackground(0);`  
`gotoxy(col,2);cprintf("%s",menu[indice]);`  
`//Proceso de chequear la tecla presionada`  
`switch(tecla){`  
`case 77: if (indice==4){col=2; indice=0;}`  
`else {col+=15; indice++;}`  
`break;`

```

case 75: if (indice==0){col=62; indice=4;}
        else {col-=15; indice--;}
        break;
case 13: _setcursortype(_NORMALCURSOR);
        if (indice==0){gotoxy(10,10);printf("Ingrese un número:");scanf("%d",&a);
        gotoxy(10,11);printf("Ingrese otro número:");scanf("%d",&b);
        gotoxy(10,12);printf("La sumatoria da:%d\n",a+b);
        gotoxy(10,13);system("pause");
        gotoxy(10,10);printf("                ");
        gotoxy(10,11);printf("                ");
        gotoxy(10,12);printf("                ");
        gotoxy(10,13);printf("                ");
        }
        if (indice==1){}
        if (indice==2){}
        if (indice==3){}
        if (indice==4) i=0;
    }
}while(i);
}

```

### **Actividades de refuerzo (AR):**

AR69. Desarrolle un programa que muestre el menú de forma vertical, aplique el mismo formato utilizado en la presentación del menú horizontal.

**OBJETIVO DE APRENDIZAJE DEL SIGUIENTE APARTADO:** Entender y aplicar las técnicas de ordenamiento de la información, conocer los fundamentos aplicados por las bases de datos, hojas de cálculo y otras aplicaciones que ordenan listas de datos y su importancia en el tratamiento de la información.

## **MÉTODOS DE ORDENAMIENTO APLICADOS A VECTORES**

Entiéndase por ordenar un vector, al proceso de reorganizar un conjunto de datos, de forma que se cree una secuencia de valores o datos con criterios ascendente o descendente, por lo general se necesita este proceso de ordenamiento para facilitar la búsqueda de datos con mayor eficiencia y rapidez, los desarrolladores han aplicado y mejorado diferentes algoritmos con el objeto de mejorar los tiempos de reorganización de datos evitando ocupar mayores recursos del computador, estos algoritmos han tenido la finalidad de aplicarlos entre otros ejemplos en la: búsqueda y localizaciones en la guía telefónica, índices de libros, roles de empleados, archivos de bibliotecas, diccionarios, fechas de eventos, entre otros.

En este apartado se explica algunos de los algoritmos de ordenamiento más conocidos:

### **ALGORITMO DE ORDENAMIENTO APLICANDO EL MÉTODO BURBUJA**

El método de ordenamiento aplica un recorrido principal y múltiples recorridos secundarios, el recorrido principal ubica el valor menor o mayor de todos los demás elementos del vector y los recorridos secundarios permiten la búsqueda y reorganización de los valores menores o mayores encontrados, el siguiente algoritmo describe el *ordenamiento burbuja*.

Antes de observar y analizar la lógica aplicada al método de ordenamiento, conozca el propósito de las variables utilizadas en el algoritmo:

**(n)** Esta variable contiene el número de elementos del vector

**(Pos)** Abreviación de posición, es la variable que apunta la posición de comparación fija y ubica uno a uno los valores menores o mayores encontrados en el recorrido interno.



**(Comp)** Abreviación de comparar, es la variable que recorre uno a uno los elementos del vector para compararlo con la posición fija del recorrido externo.

```
for (Pos=0 ; Pos < n-1 ; Pos++)
    for (Comp=Pos+1 ; Comp < n ; Comp++)
        if (Vector[Pos] > Vector[Comp]) {
            aux = Vector[Pos];
            Vector[Pos] = Vector[Comp];
            Vector[Comp] = aux;
        }
```

*Explicación:* El control for() del recorrido externo se encarga de ubicar de izquierda a derecha, posición a posición los menores encontrados, el for() del recorrido interno se encarga de apuntar posición a posición para permitir la comparación con la posición del recorrido externo buscando los menores o mayores e intercambiarlos cuando se cumpla la condición, para explicar la lógica con mayor detalle considere que se desea ordenar el siguiente vector:

7	8	13	16	9	15	10	4	5	11
Vector[0]	Vector[1]	Vector[2]	Vector[3]	Vector[4]	Vector[5]	Vector[6]	Vector[7]	Vector[8]	Vector[9]

Al aplicar la lógica del ordenamiento burbuja, se genera la siguiente prueba de funcionamiento:

**Pos, Comp ---> lista de elementos del vector, los valores subrayados son los elementos comparados**

0,1--->7 <u>8</u> 13 16 9 15 10 4 5 11	2,3--->4 5 <u>13</u> <u>16</u> 9 15 10 7 8 11	5,6--->4 5 7 8 9 <u>16</u> <u>15</u> 13 10 11
0,2--->7 8 <u>13</u> 16 9 15 10 4 5 11	2,4--->4 5 <u>13</u> 16 <u>9</u> 15 10 7 8 11	5,7--->4 5 7 8 9 <u>15</u> 16 <u>13</u> 10 11
0,3--->7 8 13 <u>16</u> 9 15 10 4 5 11	2,5--->4 5 <u>9</u> 16 13 <u>15</u> 10 7 8 11	5,8--->4 5 7 8 9 <u>13</u> 16 15 <u>10</u> 11
0,4--->7 8 13 16 <u>9</u> 15 10 4 5 11	2,6--->4 5 <u>9</u> 16 13 15 <u>10</u> 7 8 11	5,9--->4 5 7 8 9 <u>10</u> 16 15 13 <u>11</u>
0,5--->7 8 13 16 9 <u>15</u> 10 4 5 11	2,7--->4 5 <u>9</u> 16 13 15 10 <u>7</u> 8 11	6,7--->4 5 7 8 9 10 <u>16</u> <u>15</u> 13 11
0,6--->7 8 13 16 9 15 <u>10</u> 4 5 11	2,8--->4 5 <u>7</u> 16 13 15 10 9 <u>8</u> 11	6,8--->4 5 7 8 9 10 <u>15</u> 16 <u>13</u> 11
0,7--->7 8 13 16 9 15 10 <u>4</u> 5 11	2,9--->4 5 <u>7</u> 16 13 15 10 9 8 <u>11</u>	6,9--->4 5 7 8 9 10 <u>13</u> 16 15 <u>11</u>
0,8--->4 8 13 16 9 15 10 7 <u>5</u> 11	3,4--->4 5 7 <u>16</u> <u>13</u> 15 10 9 8 11	7,8--->4 5 7 8 9 10 11 <u>16</u> <u>15</u> 13
0,9--->4 8 13 16 9 15 10 7 5 <u>11</u>	3,5--->4 5 7 <u>13</u> 16 <u>15</u> 10 9 8 11	7,9--->4 5 7 8 9 10 11 <u>15</u> 16 <u>13</u>
1,2--->4 <u>8</u> <u>13</u> 16 9 15 10 7 5 11	3,6--->4 5 7 <u>13</u> 16 15 <u>10</u> 9 8 11	8,9--->4 5 7 8 9 10 11 13 <u>16</u> <u>15</u>
1,3--->4 <u>8</u> 13 <u>16</u> 9 15 10 7 5 11	3,7--->4 5 7 <u>10</u> 16 15 13 <u>9</u> 8 11	<b>Resultado:</b>
1,4--->4 <u>8</u> 13 16 <u>9</u> 15 10 7 5 11	3,8--->4 5 7 <u>9</u> 16 15 13 10 <u>8</u> 11	<b>4 5 7 8 9 10 11 13 15 16</b>
1,5--->4 <u>8</u> 13 16 9 <u>15</u> 10 7 5 11	3,9--->4 5 7 <u>8</u> 16 15 13 10 9 <u>11</u>	
1,6--->4 <u>8</u> 13 16 9 15 <u>10</u> 7 5 11	4,5--->4 5 7 8 <u>16</u> <u>15</u> 13 10 9	

```

1,7--->4 8 13 16 9 15 10 7 5 11
11
4,6--->4 5 7 8 15 16 13 10 9
1,8--->4 7 13 16 9 15 10 8 5 11
11
4,7--->4 5 7 8 13 16 15 10 9
1,9--->4 5 13 16 9 15 10 7 8 11
11
4,8--->4 5 7 8 10 16 15 13 9
11
4,9--->4 5 7 8 9 16 15 13 10
11

```

El siguiente programa muestra la aplicabilidad del algoritmo en ordenar una lista de valores:

```

C36:  #include <stdio.h>
        #include <stdlib.h>
        int main()
        { int i, vector[10];
          int Pos, Comp, aux;
          system("cls");
          for(i = 0; i < 10; i++){
            printf("Ingresa el número %d: ", i+1);
            scanf("%d", &vector[i]);
          }
          for (Pos=0 ; Pos < 9 ; Pos++)
            for (Comp=Pos+1 ; Comp < 10 ; Comp++)
              if (vector[Pos] > vector[Comp]) {
                aux = vector[Pos];
                vector[Pos] = vector[Comp];
                vector[Comp] = aux;
              }
          printf("\nlos números ordenados son:\n");
          for(i = 0; i < 10; i++)
            printf("%d\t", vector[i]);
          printf("\n");
          system("pause");
          return 0;
        }

```

## ALGORITMO DE ORDENAMIENTO APLICANDO EL MÉTODO DE SELECCIÓN

El siguiente algoritmo se lo conoce como el método de ordenamiento por selección de intercambio, este método se basa en buscar en cada repetición interna el valor menor de la lista y lo ubica en la posición de la variable “Fijo”, considere el significado de cada variable utilizada en el algoritmo de ordenamiento:

**(n)** Esta variable contiene el número de elementos del vector

**(Fijo)** Abreviación de posición fija, es la variable que apunta la posición que contendrá el valor menor encontrado en cada recorrido externo.

**(PosMenor)** Abreviación de posición del menor, es la variable que busca uno a uno los elementos menores o mayores del vector para intercambiarlo con la posición de la variable “Fijo” del recorrido externo.

**(Comp)** Abreviación de comparar, es la variable que recorre uno a uno los elementos del vector desde la posición “Fijo+1” hasta el límite del vector con la finalidad de buscar los menores para señalar la posición del elemento menor encontrado.

```

for (Fijo=0; Fijo<n; Fijo++) {
  PosMenor= Fijo;
  for (Comp= Fijo+1; Comp<n; Comp++) {
    if(Vector[Comp]<Vector[PosMenor]) PosMenor=Comp;

```

```

    }
    aux = Vector[Fijo];
    Vector[Fijo] = Vector[PosMenor];
    Vector[PosMenor] = aux;
}

```

*Explicación:* El algoritmo utiliza 2 ciclos repetitivos, el for() externo es utilizado para recorrer uno a uno y ubicar de izquierda a derecha los menores encontrados, utiliza la variable “PosMenor” para señalar el elemento menor encontrado, el for() del recorrido interno se encarga de, posición a posición encontrar el valor menor para que la variable “PosMenor” obtenga la posición, una vez cumplido el recorrido se produce el intercambio entre el valor señalado por el índice “Fijo” y el valor señalado por el índice “PosMenor”. Para explicar la lógica con mayor detalle considere que se desea ordenar el siguiente vector:

7	8	13	16	9	15	10	4	5	11
Vector[0]	Vector[1]	Vector[2]	Vector[3]	Vector[4]	Vector[5]	Vector[6]	Vector[7]	Vector[8]	Vector[9]

Al aplicar la lógica del ordenamiento por selección, se genera la siguiente prueba de funcionamiento:

Fijo				PosMenor					
7	8	13	16	9	15	10	4	5	11
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Fijo				PosMenor					
4	8	13	16	9	15	10	7	5	11
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Fijo				PosMenor					
4	5	13	16	9	15	10	7	8	11
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Fijo				PosMenor					
4	5	7	16	9	15	10	13	8	11
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Fijo				PosMenor					
4	5	7	8	9	15	10	13	16	11
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Fijo				PosMenor					
4	5	7	8	9	15	10	13	16	11

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Fijo						PosMenor			
4	5	7	8	9	10	15	13	16	11
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Fijo						PosMenor			
4	5	7	8	9	10	11	13	16	15
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Fijo						PosMenor			
4	5	7	8	9	10	11	13	16	15
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

El siguiente programa muestra la aplicabilidad del algoritmo de ordenamiento por selección al ordenar una lista de 10 valores:

```
C37:  #include <stdio.h>
        #include <stdlib.h>
        int main()
        { int i, vector[10];
          int Fijo, PosMenor, Comp, aux;
          system("cls");
          for(i = 0; i < 10; i++){
              printf("Ingresa el número %d: ", i+1);
              scanf("%d", &vector[i]);
          }
          for (Fijo=0; Fijo<10; Fijo++) {
              PosMenor= Fijo;
              for (Comp= Fijo+1; Comp<10 ; Comp++) {
                  if(vector[Comp]<vector[PosMenor]) PosMenor=Comp;
              }
              aux = vector[Fijo];
              vector[Fijo] = vector[PosMenor];
              vector[PosMenor] = aux;
          }
          printf("\nlos números ordenados son:\n");
          for(i = 0; i < 10; i++)
              printf("%d\t", vector[i]);
          printf("\n");
          system("pause");
          return 0; }
```

### ALGORITMO DE ORDENAMIENTO APLICANDO EL MÉTODO POR INSERCIÓN

En este algoritmo se aplica una búsqueda binaria en lugar de las búsquedas secuenciales para esto utiliza variable límites uno a la izquierda "**Izq**" y otro a la derecha "**Der**", y una variable intermedia "**Media**", que permite reubicar los límites izquierdo y derecho para lograr la inserción del valor menor; una vez localizado el valor menor y los límites se produce un desplazamiento hacia la derecha de los elementos entre la ubicación de la derecha y la ubicación izquierda, los reajustes de izquierda y derecha no varían si se cumple el requisito de menor a mayor (es decir que estén ordenados), el

algoritmo se repite desde el segundo elemento hasta el n-ésimo elemento, considere el significado de cada variable utilizada en el algoritmo de ordenamiento:

**(n)** Esta variable contiene el número de elementos del vector

**(cai)** Abreviación de contenido a insertar, es la variable que apunta a la posición que verificará si se inserta entre los límites izquierdo "Izq" y derecho "Der".

**(temp)** Abreviación de valor temporal, es la variable que contiene el valor apuntado por "cai" para rescatarlo en caso de existir desplazamiento de valores en la inserción, además es utilizado el contenido para ser insertado en forma ordenada.

**(Izq)** Abreviación de límite izquierdo, es la variable que siempre empezará a limitar desde la posición 0 y variará hasta la posición donde insertar el valor de forma ordenada.

**(der)** Abreviación de límite derecho, es la variable que siempre empezará a limitar desde la posición "cai-1", es decir desde una posición menor al valor que será analizado para ser o no insertado de forma ordenada.

**(cd)** Abreviación de contenido a desplazar, es la variable que permitirá el desplazamiento de valores desde la posición "cai" con el valor de posición anterior hasta llegar a la posición "Izq".

```
for (cai=1; cai<n; cai++) {
    temp = V[cai];
    Izq = 0;
    Der = cai-1;
    while (Izq <= Der){
        Medio = (Izq+Der)/2;
        if (temp < V[Medio]) Der = Medio - 1;
        else Izq = Medio + 1;    }
    for (cd=cai-1; cd>=Izq; cd--){
        V[cd+1]=V[cd];
    }
    V[Izq] = temp;
}
```

Para explicar con mayor detalle el efecto del algoritmo sobre un vector, considere el ordenamiento del siguiente vector:

7	8	13	16	9	15	10	4	5	11
V[0]	V[1]	V[2]	V[3]	V[4]	V[5]	V[6]	V[7]	V[8]	V[9]

Las siguientes graficas representan cada estado del vector al aplicar el algoritmo paso por paso al incrementarse la variable "cai", observe que, cada vez que se incrementa la variable "cai", el valor se insertará de forma ordenada en la posición que apunta a la variable "Izq", si se encuentra ordenada simplemente no se inserta.

		Der							
		No se produce inserción porque esta ordenado entre Izq y Der							
Izq	cai								
		Der							
7	8	13	16	9	15	10	4	5	11
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

		Der							
		No se produce inserción porque esta ordenado entre Izq y Der							
Izq	Der	cai							
7	8	13	16	9	15	10	4	5	11



---

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

El siguiente programa muestra la aplicabilidad del algoritmo en ordenar una lista de 10 valores:

```
C38:  #include <stdio.h>
        #include <stdlib.h>
        int main()
        { int i, cd, cai, vector[10];
          int temp, lzq, Der, Medio;
          system("cls");
          for(i = 0; i < 10; i++){
              printf("Ingresa el número %d: ",i+1);
              scanf("%d",&vector[i]);
          }
          for (cai=1 ; cai<10; cai++) {
              temp = vector[cai];
              lzq = 0;
              Der = cai-1;
              while (lzq <= Der){
                  Medio = (lzq+Der)/2;
                  if (temp < vector[Medio]) Der = Medio - 1;
                  else lzq = Medio + 1;
              }
              for (cd =cai-1; cd >=lzq; cd--){
                  vector[cd+1]= vector[cd];
              }
              vector[lzq] = temp;
          }
          printf("\nlos números ordenados son:\n");
          for(i = 0; i < 10; i++)
              printf("%d\t",vector[i]);
          printf("\n");
          system("pause");
          return 0;
        }
```

### **Actividades de refuerzo (AR):**

AR70. Desarrolle un programa que permita ordenar una lista de nombres, el programa puede utilizar cualquiera de los métodos descritos en éste apartado.

**OBJETIVO DE APRENDIZAJE DEL SIGUIENTE APARTADO:** Conocer y administrar estructuras de variables en arreglos bidimensionales o matrices, definir su importancia en el tratamiento de la información.

### **ARREGLOS BIDIMENSIONALES O MATRICES**

Las matrices son arreglos de dos dimensiones, cada dimensión está representada por un índice, en la matriz cada dimensión se representa por filas y columnas, otro tipo de matriz puede tener cualquier número de dimensiones, como los cubos que tienen tres dimensiones, pero las matrices más utilizadas son las que poseen dos dimensiones, para ejemplificar una matriz considere el tablero de un ajedrez, que posee ocho filas por ocho columnas. En las siguientes declaraciones se crean 2 arreglos bidimensionales, el primero es una matriz de 10 filas y 8 columnas, el segundo arreglo es una matriz de 10 filas y 10 columnas, cuando es igual el número de filas y el número columnas se dice que es una *matriz cuadrada*:

```
int x[10][8];
float m[10][10];
```

Al momento de crear una matriz también es posible inicializarla, el siguiente ejemplo inicializa dos matrices, la primera con los números pares y la segunda con los números primos, la siguiente ilustración indica que es válida la aplicación de cualquiera de los ejemplos:

```
int matriz1[5][3] = { 2,4,6,8,10,12,14,16,18,20,22,24,26,28,40 };
long matriz2[5][3] = { {1,2,3}, {5,7,11}, {13,17,23}, {29,31,37}, {41,43,51} };
```

A diferencia de los vectores que necesitan básicamente solo un ciclo repetitivo para recorrer la estructura, la matriz necesita de dos ciclos repetitivo para recorrerla ya que una matriz utiliza dos índices, el primer índice corresponde al número o posición de la fila y el segundo índice corresponde al número o posición de la columna, para ejemplificar el recorrido de una matriz se utilizará como herramienta el control `for()`:

```
for(int f = 0; f < NumFilas; f++) {
    for(int c = 0; c < Numcolumnas; c++) {
        Matriz[f][c] = f % c;
    }
}
```

Como ejemplo de utilización de matrices considere el siguiente problema matemático: se necesita un programa que permita el registro de valores solo la primera fila de la matriz, las siguientes filas se llenarán automáticamente con el cuadrado que corresponde a cada elemento de la fila anterior, considere que se tiene una matriz cuadrada de 5x5.

Para desarrollar la solución al problema presentado se necesita de una función que permita resolver el cuadrado de un valor, para este ejemplo se utilizará la función `pow()`:

**pow(base, exponente).**- Esta función pertenece a la librería `math.h`, permite calcular la potencia de cualquier número; la función devuelve un resultado de tipo `double`, se utiliza este tipo de dato, ya que es el que tiene mayor capacidad de cálculo entre todos los tipos de datos numéricos.

**sqrt(valor).**- Esta función también pertenece a la librería `math.h`, sirve para obtener la raíz cuadrada de un número cualquiera.

```
C39: #include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main()
{ int f, c;
  double matriz[5][5];
  system("cls");
  for(c = 0; c < 5; c++){
      printf("Ingresa el número fila 0 columna %d: ",c+1);
      scanf("%lf",&matriz[0][c]);
  }
  for(f = 1; f < 5; f++)
      for(c = 0; c < 5; c++)
          matriz[f][c] = pow (matriz[f-1][c],2);

  printf("\nLa matriz ha generado lo siguiente:\n");
  for(f = 1; f < 5; f++){
      for(c = 0; c < 5; c++)
          printf("%6.0lf\t",matriz[f][c]);
      printf("\n");
  }
  system("pause");
  return 0;
}
```

### **Actividades de refuerzo (AR):**



AR71. Modifique el programa anterior de forma que pida el número de filas y columnas, la matriz debe ser cuadrada y como máximo puede tener 10 fila y 10 columnas, si el resultado del cuadrado de la fila anterior da un valor mayor a 1000, se lo debe reducir automáticamente a la raíz cuadrada.

Considere el siguiente problema a resolver: Desarrolle un programa que permita llenar automáticamente una matriz de la siguiente forma:

1	0	0	0	0	0	0
2	2	0	0	0	0	0
3	3	3	0	0	0	0
4	4	4	4	0	0	0
5	5	5	5	5	0	0
6	6	6	6	6	6	0
7	7	7	7	7	7	7

**C40:**

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int f, c;
  int matriz[7][7];
  system("cls");
  for(f = 0; f < 7; f++)
    for(c = 0; c < 7; c++)
      if (c<=f) matriz[f][c]= f+1;
      else matriz[f][c]= 0;
  printf("\nLa matriz ha generada es la siguiente:\n");
  for(f = 0; f < 7; f++){
    for(c = 0; c < 7; c++)
      printf("%d\t",matriz[f][c]);
    printf("\n");
  }
  system("pause");
  return 0;
}
```

#### **Actividades de refuerzo (AR):**

AR72. Modifique el programa anterior de forma que genere la siguiente matriz:

0	0	0	0	0	0	1
0	0	0	0	0	2	2
0	0	0	0	3	3	3
0	0	0	4	4	4	4
0	0	5	5	5	5	5
0	6	6	6	6	6	6
7	7	7	7	7	7	7

Ahora considere el siguiente problema: Desarrolle un programa que permita llenar automáticamente una matriz de la siguiente forma:

0	0	0	1	0	0	0
0	0	2	2	2	0	0
0	3	3	3	3	3	0
4	4	4	4	4	4	4

**C41:**

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int f, c;
  int matriz[4][7];
  system("cls");
  for(f = 0; f < 4; f++){
    for(c = 0; c < 7; c++){
      if (c>=3-f && c<=3+f) matriz[f][c]= f+1;
      else matriz[f][c]= 0;
    }
  }
  printf("\nLa matriz ha generada es la siguiente:\n");
  for(f = 0; f < 4; f++){
    for(c = 0; c < 7; c++){
      printf("%d\t",matriz[f][c]);
    }
    printf("\n");
  }
  system("pause");
  return 0;
}
```

#### Actividades de refuerzo (AR):

AR73. Modifique el programa anterior de forma que genere la siguiente matriz:

4	4	4	4	4	4	4
0	3	3	3	3	3	0
0	0	2	2	2	0	0
0	0	0	1	0	0	0

Utilizando la matriz se puede desarrollar problema como el siguiente: En una matriz de 3 filas por 8 columnas desarrolle un programa que permita ingresar dos números binarios, cada binario ubicado en las dos primeras filas, el programa deberá almacenar el resultado en la tercera fila:

64	32	16	8	4	2	1	
0	1	1	1	0	1	1	59
0	0	1	1	1	0	1	29
1	0	1	1	0	0	0	88

**C42:**

```
#include <stdio.h>
#include <conio.h>
void main()
{ int f, c, tecla, Acarreo=0, r;
  int binario[3][8];
  clrscr();
  for(f = 0; f < 2; f++){
    printf("\nIngresa el %d número binario:\n",f+1);
    for(c = 0; c <= 7; c++){
```

```

do{ tecla=getch(); } while(tecla<48 || tecla>49);
binario[f][c]= tecla-48;
printf("%d", binario[f][c]);
}
}
for(c = 7; c >= 0; c--) {
    r = Acarreo+ binario[0][c]+ binario[1][c];
    switch(r){
        case 0: binario[2][c]=0; Acarreo=0; break;
        case 1: binario[2][c]=1; Acarreo=0; break;
        case 2: binario[2][c]=0; Acarreo=1; break;
        case 3: binario[2][c]=1; Acarreo=1; break;
    }
}
for(f = 0; f < 3; f++){
    printf("\n");
    for(c = 0; c <= 7; c++) printf("%d\t", binario[f][c]);
}
getch();
}

```

### **Actividades de refuerzo (AR):**

AR74. Modifique el programa anterior de forma que muestre cada número tanto en binario como en decimal

### **RESUMEN**

Las variables organizadas en estructuras de arreglos son capaces de contener grandes cantidades de datos, para referenciarse cada una utilizan el mismo nombre e índices únicos para identificarse de forma individual, las cadenas de caracteres son vectores o arreglos unidimensionales de caracteres alfanuméricos, utiliza al final de una cadena un carácter especial que define el límite de la estructura contenida utilizando '\0', con las cadenas de caracteres no se puede hacer uso de los operadores de comparación y aritméticos, ya que su estructura es de múltiples variables contenidas y los operadores solo trabajan con valores o variables de un único valor, para crear vectores de palabras se hace uso de la organización bidimensional o matrices pero su tratamiento es como un vector de valores, es decir utilizando solo un índice, los vectores se pueden inicializar, es decir asignarles un valor o dato inicial, para lograrlo utiliza llaves '{ }' y comas ',' para separar cada elemento de la inicialización, para crear un vector se escribe el tipo de dato que desea contener, el nombre de la estructura y entre corchetes el número de elementos a crear, es importante destacar que el primer elemento de un arreglo tiene la posición cero [0] y el último la posición [N-1], está permitido crear vectores sin límites en una inicialización, para lograrlo se utiliza corchetes vacíos '[ ]', toda estructura unidimensional tiene la posibilidad de aplicar métodos de ordenamiento, el capítulo explica y ejemplifica el método de ordenamiento burbuja, de selección y de inserción, considere que existen muchos más.

Las matrices utilizan dos índices para referenciar a cada elemento de su estructura, los cubos utilizan tres índices, al igual que los vectores las matrices se crean considerando el mismo orden, primero el tipo de dato seguido el nombre que desee para la estructura y por último dos valores cada uno encerrado entre corchetes, el primero define el número de filas y el segundo el número de columnas.

## CAPITULO VI

### USO DE LAS FUNCIONES

Este capítulo introduce al programador en la importancia, organización y empleo de las funciones, con la finalidad de reducir los diferentes grupos de instrucciones repetidos que realizan el mismo proceso en diferentes partes del programa, además en este capítulo se pretende promover en el programador, el desarrollo de un pensamiento planificador capaz de reutilizar soluciones desarrolladas por el programador u otros programadores en futuras soluciones de desarrollo.

El capítulo empieza dando a conocer la necesidad de utilizar funciones en los nuevos programas, expone las reglas de creación de funciones, así como los formatos que se utilizan para utilizar una función y mejorar su desempeño, el apartado propone la creación de librerías propias que permiten al programador reutilizar las instrucciones en nuevas aplicaciones, además define y ejemplifica el intercambio de datos entre las diferentes funciones mediante la aplicación de paso de parámetros por valor y paso de parámetro por referencia, ejemplifica el uso de argumentos y retorno de resultados considerando los diferentes tipos de datos, aplica mediante la resolución de problemas el paso de parámetro por valor y su diferencia con el paso de parámetro por referencia, además expone el paso de variables estructuradas como los arreglos a las diferentes funciones, define la importancia de los ámbitos implícitos y explícitos de las diferentes declaraciones de las variables.

El estudio de éste capítulo sirve de base para el conocimiento de las nuevas metodologías de desarrollo que ofrecen otros lenguajes de programación, en sí la manipulación de las funciones permiten desarrollar y controlar proyectos más grandes y completos al simplificar, reciclar y reducir la programación.

**OBJETIVO DE APRENDIZAJE DEL SIGUIENTE APARTADO:** Desarrollar en el lector la capacidad de desarrollar programar mediante el uso de funciones (sub programas) dentro de los parámetros de la programación estructurada, permitiendo enviar datos y retornar resultados, utilizando pasos de parámetros por valor y por referencia y tendrá la posibilidad de crear sus propias librerías de funciones.

#### ¿Por qué simplificar la programación?

Es importante entender que cuando se desarrollan programas con múltiples utilidades para el usuario, se presentan diferentes adversidades que complican el desarrollo y el mantenimiento de los mismos, uno de los factores preponderantes es la extensión en contenido de sus instrucciones, dando como resultado programas lentos y difíciles de dar mantenimiento, es decir la documentación de las instrucciones programadas crecería conforme a la complejidad de lo que se desea realizar, esto implica el hecho de que se presentan procesos que se repiten en varias partes del programa; al desarrollar aplicaciones sin la debida planificación se hace uso de un incremento innecesario de variables que se utilizan en procesos pequeños y que ocupan demasiado recursos de memoria a lo largo la ejecución del programa.

Ante estas circunstancias se desarrolló la programación estructurada, que motiva la planificación estructural y modular de pequeños subprogramas llamados funciones, estos permiten realizar procesos de forma separada e individual, su propósito radica en desarrollar y/o devolver un determinado resultado, las funciones al ser individuales poseen sus propias variables sin afectar a las utilizadas por el programa principal, su utilidad puede ser aplicada en otros programas, la programación de las diferentes funciones pueden ser contenidas en archivos planos llamados librerías.

Las funciones permiten disminuir la redundancia de programación al concentrarla en un solo sitio es decir en un subprograma y mediante la utilización de su nombre, aplicar las instrucciones como si estuviesen las mismas líneas de programación, las funciones permiten el reciclaje de procesos, ya que sus instrucciones pueden ser aplicadas en otros programas, es decir se puede tener una función que devuelva la potencia de un número cualquiera, el resultado es lo esperado en cualquier programa que necesite realizar dicho calculo lo que significa que solo se debe incluir la función y no

volver a escribir las instrucciones que resuelvas dicho proceso. Todos los lenguajes de programación utilizan funciones, por lo tanto todo programador debe conocer su funcionamiento, los requisitos lógicos de trabajo y los recursos que estos proveen.

## ¿Qué es una función?

Una función es como un programa pequeño que tiene un nombre para ser identificado y un propósito funcional que cumple cuando se lo utiliza, éste pequeño programa (función) puede formar parte de un programa mayor o de una librería, por ejemplo `printf()`, `scanf()`, `gets()`, `pow()`, `system()`, entre muchas otras, son algunas de las funciones que más se han utilizado en el desarrollo del presente material bibliográfico, pero ahora usted aprenderá a crear sus propias funciones.

Es importante entender que las funciones al igual que las variables cumplen con la regla de identificación única, es decir que en un programa o en una librería no está permitido que dos funciones utilicen el mismo nombre, esto ya que al momento de ser utilizadas el intérprete no sabría a quién utilizar en el proceso, el nombre de la función cumple las mismas reglas de creación de nombres de variables, además obligatoriamente el nombre debe utilizar paréntesis que permiten recibir datos del programa principal que la utiliza; una función utiliza la siguiente sintaxis como regla para la creación de la misma:

### Formato de la sintaxis 1:

```
Tipo_de_dato Nombre_función([lista_de_parametros])
{
    [Cuerpo_de_la_Función]
    return expresión;
}
```

### Formato de la sintaxis 2:

```
void Nombre_función([lista_de_parametros])
{
    [Cuerpo_de_la_Función]
}
```

Observe que la sintaxis del formato 1 se utiliza “*tipo\_de\_dato*” de forma similar a la declaración de una variable (`int`, `float`, `long`, `double`, `char`); al aplicar este formato es obligatorio el uso de la instrucción “*return expresión;*” donde *expresión* significa que es el resultado que devuelve la función al programa mayor o principal, y puede ser el contenido de una variable, el resultado de un cálculo o el resultado de aplicar una condición; la sintaxis del formato 2 utiliza la instrucción `void` en vez de “*tipo\_de\_dato*” cuyo significado indica que la función *no devolverá nada*, por lo tanto no es necesario utilizar la instrucción “*return expresión;*”, en caso de pretender utilizarlo no se debe incluir la expresión solo “*return;*”.

En los dos formatos se utiliza “[*lista\_de\_parametros*]”, esto quiere decir que una función puede o no tener en su estructura la lista de parámetros, que básicamente permiten definir uno o varios valores de entrada y/o de salida, de querer hacer uso de la *lista de parámetros*, cada parámetro (variable) debe ser declarado de forma independiente y de existir más de uno, se separan mediante una coma (,); considere que una función puede recibir un *valor* o una *dirección de memoria* (dirección de una variable) como dato de entrada; observe el siguiente ejemplo de cómo declarar una función:

```
int cuadrado(int num);
```

En éste ejemplo se declara una función que recibe un valor de tipo entero y devuelve como resultado un valor del mismo tipo, esto no significa que los parámetros siempre deben ser del mismo tipo que devuelve la función, por ejemplo:

```
float interes(float capital, int PorInt, int tiempo);
```

En éste ejemplo se declara una función que recibe una lista de valores, uno de los parámetros es de tipo `float` y otros dos parámetros de tipo entero, observe que tanto la variable `PorInt` y `tiempo` cada una tiene *su propia declaración* a pesar de que son del mismo tipo, la función devolverá como resultado un valor de tipo `float`, observe la siguiente declaración:

```
double areas(float *a, int tipo, float b, float h);
```

En éste ejemplo se declara una función que recibe una lista de valores, la primera variable de la lista (\*a) recibe una dirección de memoria, observe que la variable al momento de su declaración se le antepone el asterisco, esto significa que recibirá la dirección de memoria en vez de valores del mismo tipo, este tipo de declaración se ampliará con mayor detalle más adelante.

Retomando la explicación de ambos formatos, considere que ambas sintaxis utilizan “[Cuerpo\_de\_la\_Función]”, esto significa que el programador puede incluir: declaraciones de variables, asignaciones, cálculos, estructuras condicionales, de repetición, en fin se incluye lo mismo que se incluye en un programa natural; para salir de una función se utiliza la sentencia *return*, esta instrucción puede devolver un valor conforme con el “*tipo\_de\_dato*” indicado en la declaración de la función.

Una función, al igual que una variable, antes de ser utilizada debe ser declarada o reconocida, esto significa que su declaración o desarrollo debe ubicarse de forma estratégica que evite errores de desconocimiento, para ilustrar esta regla considere el siguiente ejemplo:

#### Ejemplo 1: C43

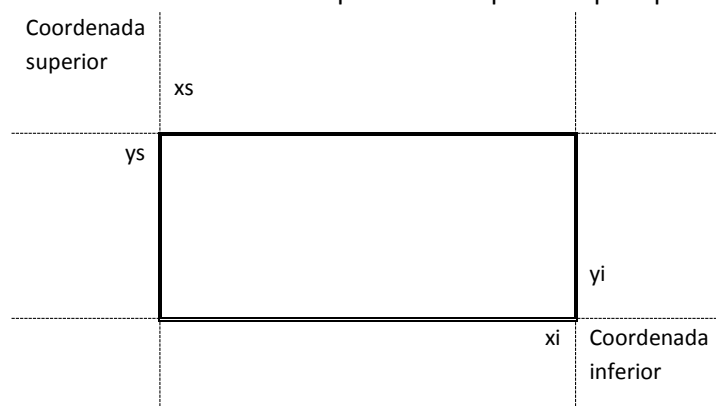
```
#include<stdio.h>
#include<stdlib.h>
int cuadrado(int num)
{ return num*num;
}
main()
{ int a, b;
  system("cls");
  printf("Ingrese un número: ");scanf("%d",&a);
  printf("Ingrese otro número: ");scanf("%d",&b);
  printf("Cuadrado 1er número es: %d\n",cuadrado(a));
  printf("Cuadrado 2do número es: %d\n",cuadrado(b));
  system("pause");
  return 0;
}
```

#### Ejemplo 2: C44

```
#include<stdio.h>
#include<stdlib.h>
int cuadrado(int num);
main()
{ int a, b;
  system("cls");
  printf("Ingrese un número: ");scanf("%d",&a);
  printf("Ingrese otro número: ");scanf("%d",&b);
  printf("Cuadrado 1er número es: %d\n",cuadrado(a));
  printf("Cuadrado 2do número es: %d\n",cuadrado(b));
  system("pause");
  return 0;
}
int cuadrado(int num)
{ return num*num;
}
```

En los dos ejemplos se anticipa el reconocimiento de la función *cuadrado()*, en el ejemplo uno, la declaración y desarrollo se encuentra antes de la función *main()*, observe que *main()* es la función que llama a la función *cuadrado()* dos veces, en el segundo ejemplo se declara la existencia de la función pero su desarrollo se encuentra después de la función *main()*, este método de desarrollo es similar al primer ejemplo, al no afectar la lógica, ya que se cumple la regla de reconocimiento antes de ser utilizada, observe que una función que tiene parámetros obligatoriamente debe pasarle valores conforme al número de parámetros, a estos valores se les llama *argumentos*.

La siguiente función permite dibujar un cuadro en la pantalla, para esto se debe pasar como argumento las coordenadas de 2 ubicaciones o puntos en la pantalla para poder mostrar el cuadro:



Para desarrollar el programa se utilizará la librería *conio.h*:

```

C45:  #include <stdio.h>
        #include <conio.h>
        /* Caracteres para doble línea
        Alt 205="=" 201="É" 200="È" 187="»" 188="¼" 186="||" */
        void cuadro(int, int, int, int);
        void main()
        { int x1,y1,x2,y2;
          clrscr();
          printf("\nIngresa coordenada superior x y:"); scanf("%d%d",&x1,&y1);
          printf("\nIngresa coordenada inferior x y:"); scanf("%d%d",&x2,&y2);
          clrscr();
          cuadro(x1,y1,x2,y2);
          getch();
        }
        void cuadro(int xs, int ys, int xi, int yi)
        { int i;
          //Dibujar líneas horizontales superior e inferior
          for(i=xs ; i<xi ; i++){
            gotoxy(i,ys);cprintf("-"); // Alt 196="-"
            gotoxy(i,yi);cprintf("-");
          }
          //Dibujar líneas verticales izquierda y derecha
          for(i=ys ; i<yi ; i++){
            gotoxy(xs,i);cprintf("|"); //Alt 179="|"
            gotoxy(xi,i);cprintf("|");
          }
          //Dibujar las esquinas del cuadro
          gotoxy(xs,ys);cprintf("┐"); // Alt 218="┐"
          gotoxy(xs,yi);cprintf("└"); // Alt 192="└"
          gotoxy(xi,ys);cprintf("┌"); // Alt 191="┌"
          gotoxy(xi,yi);cprintf("┘"); // Alt 217="┘"
        }

```

Observe que en el ejemplo la declaración de la función que se encuentra antes del main() solo tiene como argumento cuatro tipos de datos y no tienen nombres de variables, esto está permitido ya que el propósito es solo de dar a conocer la existencia de la función, pero no está permitido para el desarrollo o codificación de la misma.

Es importante conocer que existen solo dos formas de pasar valores o datos a una función, mediante el paso de una copia del contenido de una variable (conocido como paso de parámetros por valor) y mediante el paso de la dirección de la variable (conocido como paso de parámetro por referencia).

**OBJETIVO DE APRENDIZAJE DEL SIGUIENTE APARTADO:** Conocer y aplicar las reglas correspondientes para aplicar las diversas formas de compartir datos e información entre las diferentes funciones y el programa principal.

## PASO DE PARÁMETROS POR VALOR

Los ejemplos desarrollados hasta ahora utilizan pasos de parámetros por valor, se constituye en una simple asignación implícita, esto quiere decir que, a pesar de no utilizar el signo igual (=) se realiza la misma acción de transferir datos de una expresión a una variable, en este caso específico se produce la asignación implícita cuando el parámetro toma el dato del argumento, observe el siguiente ejemplo, la función “esprimo” recibe un valor cualquiera, ésta verificará si es o no un valor primo:

```

int esprimo(int num)
{ If (num==1 || num==2 || num==3) return 1;
  for (int i=2;i<=(num/2)+1 ; i++)
    if (num%i==0) return 0;
}

```

```

    return 1;
}

```

La función devolverá como resultado cero o uno dependiendo del proceso, devolverá el valor de uno “return 1;” cuando el proceso interno de comparación encuentre que el valor de la variable “num” es primo; o devolverá como resultado el valor de cero “return 0;” cuando el número *no es primo*; la función plantea como lógica de funcionamiento, representar al valor de 1 como verdadero es decir “es primo”, y el valor de 0 como falso; Observe que la función, inicialmente verifica si el número es 1 o 2 o 3 para devolver el valor de 1 (Verdadero) que a su vez “es primo”, en caso de encontrar una división exacta entre dos y la mitad más uno del número devolverá 0 que a su vez “no es primo”.

El siguiente ejemplo utiliza la función “*esprimo()*”, para mostrar la lista de números primos que se encuentran desde el 1 hasta el 100:

**C46:**

```

#include <stdio.h>
#include <stdlib.h>
int esprimo(int num)
{ if (num==1 || num==2 || num==3) return 1;
  for (int i=2;i<=(num/2)+1 ; i++)
    if (num%i==0) return 0;
  return 1;
}
void main()
{
  system("cls");
  printf("lista de números primos entre 1 y 100: \n ");
  for (int i=1 ; i<=100 ; i++)
    if ( esprimo( i ) ) printf("%d\t",i);
  system("pause");
}

```

Observe que existe una asignación implícita cuando se llama a la función “*esprimo(i)*”, ya que ocurre esto: “num=i”, “num” es la variable parámetro declarada en la función *esprimo()* y la variable “i” es el argumento que le pasa a la función, por lo tanto el contenido de la variable “i” es pasada una copia a la variable “num”, considere importante el hecho de que el contenido de la variable “i” no sufrirá ningún cambio al ser alterada la copia asignada a la variable “num”.

### **Actividades de refuerzo (AR):**

- AR75. Desarrolle un programa que permita mediante una función calcular el factorial de un número, la función deberá devolver el resultado como un entero largo.
- AR76. Desarrolle un programa que permita calcular la potencia de un número, para lograrlo, utilice una función que tome como argumento la base y el exponente.
- AR77. Se necesita un programa que permita mediante una función devolver el número de dígitos que tiene una cantidad, el programa debe utilizar como argumento un valor de tipo entero largo.

### **PASO DE PARÁMETROS POR REFERENCIA**

Este proceso es totalmente diferente al paso de parámetro por valor ya que exige lo siguiente:

- ✓ El parámetro debe ser declarado para aceptar direcciones de memoria y no valores en general, una variable para recibir direcciones debe ser declarada anteponiéndole el asterisco (\*), es importante indicar que las direcciones están en el sistema de numeración hexadecimal (éste sistema de numeración utiliza 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F) y no decimal (este sistema de numeración utiliza 0,1,2,3,4,5,6,7,8,9) que es el sistema de numeración que se utiliza normalmente, por ejemplo al declarar las siguientes variables “int \*a,b;”, la variable (a) podrá recibir valores hexadecimales como “a=0xacd;” que podría representar una dirección de



memoria, si asignamos este valor a la variable (b) provocará de seguro un error de sintaxis, ya que la variable (b) fue declarada para aceptar valores decimales como "b=1567;"

- ✓ El argumento debe ser la dirección de la variable y no su valor, para enviarlo como argumento se le antepone el (&) al nombre de la variable, el símbolo & significa "*la dirección de...*", para complementar el uso de variables con direcciones como valor de almacenamiento se utiliza el asterisco (\*) fuera de la declaración, este símbolo significa "*el contenido de la dirección...*", se lo utiliza anteponiéndole al nombre de la variable para referenciar al contenido de esa dirección, es decir al valor almacenado.

Para ejemplificar el uso de estos símbolos considere el siguiente ejemplo:

```
C47:  #include <stdio.h>
        #include <stdlib.h>
        void siguientepar(int *num)
        { *num = (*num%2==0)? *num+2 : *num+1;
        }

        void main()
        { int x;
          system("cls");
          printf("Ingrese un número y mostrará el siguiente par: "); scanf("%d",&x);
          siguientepar( &x );
          printf("%d\n",x);
          system("pause");
        }
```

Al ejecutar el programa notará que el valor de la variable "x" es modificado en el desarrollo de la función "*siguientepar()*", esto ocurre al pasar la dirección y no una copia de su contenido, note que la asignación implícita queda así "*\*num = &x*", dentro de la función aplica una asignación condicionada, al dividir *el contenido de la dirección de "x"* para dos chequea si el residuo es 0, de ser así *el contenido de la dirección de "x"* se le suma 2, caso contrario se le suma 1, el resultado de cualquiera de las dos operaciones es asignado como *el contenido de la dirección de "x"*; la función no devuelve nada como resultado ya que el contenido de la variable "x" es modificado; si la función tiene declarada una variable que recibe una dirección en vez de un valor, es obligatorio utilizar como argumento el & anteponiéndosele al nombre de la variable como el ejemplo utilizado en main() del programa principal "*siguientepar( &x );*".

Ahora considere el siguiente problema: se necesita un programa que permita calcular el área de un rectángulo, el programa debe aplicar la misma utilidad que ofrece la función *scanf()*, es decir debe crear una función para almacenar en cualquier variable de tipo *int* un valor de tipo entero, considere que la función solo debe aceptar las teclas numéricas y la tecla ENTER, la función debe devolver el contenido ingresado mediante el paso de parámetro por referencia. Para cumplir con el propósito solicitado se podrá utilizar cualquiera de las siguientes funciones:

**wherex()** y **wherey()**.- Estas funciones pertenecen a la librería *conio.h*, sirven para devolver la ubicación del cursor; *wherex()* devuelve la posición de la columna en donde se encuentra el cursor y *wherey()* devuelve la posición de la fila en donde se encuentra el cursor, ambas funciones no necesitan argumentos ya que se encargan de devolver solo la posición respectiva.

```
C48:  #include <stdio.h>
        #include <conio.h>
        void soloint(int *num);
        void main(){
          int a, b, h;
          clrscr();
          printf("Ingrese la base:");soloint(&b);
```

```

printf("Ingrese la altura:");soloint(&h);
a=b*h;
printf("El área es: %d",a);
getch();
}
void soloint(int *num)
{ int tecla, limi=whrex();
  *num=0;
  do{ tecla=getch();
    if (tecla>=48 && tecla<=57){
      printf("%d",tecla-48); //calcula y muestra el dígito presionado
      *num = *num * 10 + tecla-48; //almacena el carácter presionado
    }
    if (tecla==8 && limi<whrex()){
      *num = *num / 10;
      printf("\b \b");
    }
  }while(tecla!=13); //repite mientras tecla sea diferente de ENTER
  printf("\n");
}

```

### **Actividades de refuerzo (AR):**

AR78. Desarrolle un programa que permita mediante una función intercambiar dos números, es decir si se tienen las variables num1 y num2, el valor de num1 será el que tenga num2 e inversamente con el otro número, el programa pedirá 3 números cualquiera y utilizará la función de intercambio para mostrarlos de forma ordenada.

AR79. Desarrolle un programa que permita simplificar un quebrado mediante el uso de una función, se debe utilizar como parámetros el numerador y el denominador.

### **ARREGLOS COMO PASO DE PARÁMETROS**

Es importante destacar que, en lenguaje C los arreglos deben ser pasados por *referencia* y no por valor, como explicación recuerde que los pasos de parámetros funcionan como una asignación parámetro=argumento de forma implícita, donde el parámetro recibe un valor de entrada que se denomina argumento, este procedimiento de asignación funciona cuando se le pasa un único valor a otra variable; el inconveniente radica en que el arreglo es una estructura que contiene muchos valores y no se puede aplicar el principio de asignación a un grupo de datos en la misma asignación; para solventar la necesidad de pasar arreglos a las funciones se utiliza el paso de parámetros por referencia, ya que solo se debe pasar la dirección del arreglo (es decir solo un dato) y no todos los valores, esto implica que, todas las modificaciones que se realicen dentro de la función al arreglo, modificarán al arreglo original que fue utilizado como argumento, considerando que se le ha pasado la dirección física del arreglo y no una copia de sus valores.

Recuerde que para el lenguaje C, declarar una variable que almacena una dirección, se debe especificar el tipo de dato y el nombre anteponiéndole el asterisco o simplemente declarando un arreglo, ya que en ambos casos son iguales en su contenido, por ejemplo:

```

int *a, b[10], c[15][10];
char nombre[20], *apellido;

```

Las variables del ejemplo “a”, “b” y “c” son declaradas para contener direcciones de memoria, la variable “a” accede directamente a la variable y accede al contenido utilizando el asterisco “\*” como ya se detalló en apartados anteriores, mientras que las variables “b” y “c” necesitan de los índices para referenciar a cada elemento del arreglo, las variables *nombre* y *apellido* también son variables

que internamente contienen direcciones como contenido, ya que son arreglos, por ejemplo considere el uso de la función “*scanf()*”, el programador debe pasar como argumento la dirección de la variable, asumiendo que tiene una variable de tipo entero llamada “*edad*”, se tendría que pasar la dirección de la variable para que los datos digitados desde el teclado se almacenen en la misma: *scanf(“%d”, &edad)*; observe que se utiliza el & para especificar “la dirección de”, para almacenar desde el teclado una cadena de caracteres utilizando la variable “*nombre[20]*”, se aplica lo siguiente: *scanf(“%s”, nombre)*; observe que, a la variable “*nombre*” no se le antepone el &, ya que la variable como tal contiene la dirección de la cadena de caracteres y por lo tanto no necesita especificar “la dirección de” con el signo &.

En una función, se puede declarar un parámetro tipo arreglo, especificando primero el tipo de dato y después el nombre del arreglo seguido de corchetes que se abren y cierran, por ejemplo considere una función que necesita recibir un arreglo unidimensional (vector) que contiene un número binario de 8 bit, la función deberá devolver el decimal de dicho binario:

```
int binario_decimal(int binario[])
{ int v=128, acum=0;
  for(int i = 0; i < 8; i++){
    if(binario[i]==1) acum+=v;
    v/=2;
  }
  return acum;
}
```

En el ejemplo anterior, observe que la función “*binario\_decimal()*”, contiene como único parámetro un vector llamado “*int binario[]*”, que no tiene indicado el número de elementos, ya que el propósito es recibir una dirección de un arreglo y no la copia de sus elementos, para pasarle el argumento se podría utilizar la siguiente instrucción: “*resultado=binario\_decimal(vector)*”; note que no es necesario utilizar el & ya que la variable “*vector*” ya contiene como contenido la dirección del mismo; para ilustrar de forma más detallada, considere el ejemplo completo:

**C49:**

```
#include <stdio.h>
#include <conio.h>
int binario_decimal(int binario[])
{ int v=128, acum=0;
  for(int i = 0; i < 8; i++){
    if(binario[i]==1) acum+=v;
    v/=2;
  }
  return acum;
}
void main()
{ int i, tecla, res;
  int vector[8];
  clrscr();
  printf("\nIngresa un número binario:\n");
  for(i = 0; i < 8; i++){
    do{ tecla=getch(); } while(tecla<48 || tecla>49);
    vector[i]= tecla-48;
    printf("%d", vector[i]);
  }
  res=binario_decimal(vector);
  printf("\nEl decimal es:%d", res);
  getch();
}
```

En el ejercicio anterior se utiliza la función para recibir un arreglo, el contenido del arreglo no es afectado ya que el proceso solicitado solo realiza el recorrido del arreglo; observe que el proceso de ingreso se lo realiza dentro de la función main(), el siguiente ejercicio lo modifica y realiza el proceso de ingreso en otra función:

```
C50:  #include <stdio.h>
        #include <conio.h>
        int binario_decimal(int binario[])
        { int v=128, acum=0;
          for(int i = 0; i < 8; i++){
              if(binario[i]==1) acum+=v;
              v/=2;
          }
          return acum;
        }
        void ingreso_binario(int binario[])
        { int tecla,i;
          for(i = 0; i < 8; i++){
              do{ tecla=getch(); } while(tecla<48 || tecla>49);
              binario[i]= tecla-48;
              printf("%d", binario[i]);
          }
        }
        void main()
        { int i, tecla, res;
          int vector[8];
          clrscr();
          printf("\nIngrese un número binario:\n");
          ingreso_binario(vector);
          res=binario_decimal(vector);
          printf("\nEl decimal es:%d", res);
          getch();
        }
```

### Actividades de refuerzo (AR):

AR80. Desarrolle un programa que permita mediante una función llenar un vector de N elementos con números al azar entre 1 y 300, el programa debe mostrar el número de elementos pares y el número de elementos impares.

AR81. Desarrolle un programa que permita mediante una función llenar un vector de N elementos con números de la serie de Fibonacci, el programa debe mostrar entre paréntesis los números primos de la serie.

Considere el siguiente problema planteado y analice las alternativas presentadas como respuesta: Desarrolle un programa que permita mediante una función contar las palabras que tiene una frase, el programa mediante una función debe devolver la cantidad de palabras encontradas.

**C51:**

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int total_palabras(char texto[])

{ int cp=0;
```

**C52:**

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int total_palabras(char *texto)

{ int cp=0;
```

<pre> for (int i=0;i&lt;strlen(<b>texto</b>);i++) {      while (<b>texto</b>[i]!=' ') i++;      if (i != 0) cp++;      while (<b>texto</b>[i]==' ') i++;  }  return cp;  }  void main()  { char *frase;      system("cls");      printf("Ingrese una frase: "); gets(frase);      printf("Tiene: %d palabras\n", <b>total_palabras(frase)</b>);      system("pause");  } </pre>	<pre> for (int i=0;i&lt;strlen(<b>texto</b>);i++) {      while (<b>texto</b>[i]!=' ') i++;      if (i != 0) cp++;      while (<b>texto</b>[i]==' ') i++;  }  return cp;  }  void main()  { char *frase;      system("cls");      printf("Ingrese una frase: "); gets(frase);      printf("Tiene: %d palabras\n", <b>total_palabras(frase)</b>);      system("pause");  } </pre>
---	---

Analice el siguiente problema: Se necesita una función que permita devolver el día de la semana pasándole como argumento un valor entre 0 y 6, considerando que 0 es el primer día “Domingo”

**C53:**

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char * dia_semana(int Num_dia)
{ switch(Num_dia ) {
    case 0: return "Domingo";
    case 1: return "Lunes";
    case 2: return "Martes";
    case 3: return "Miércoles";
    case 4: return "Jueves";
    case 5: return "Viernes";
    case 6: return "Sábado";
}
return "Error, día incorrecto...!";
}

void main()
{ int dia;
  char *ds;
  system("cls");
  printf("\nIngrese un número de día (0=domingo ... 6=sábado):");
  scanf("%d",&dia);
  ds= dia_semana (dia);
  printf("\nEl día es:%s\n", ds);
  system("pause");
}

```

Ahora considere la modificación del programa anterior de forma que pida la fecha de nacimiento y devuelva el día de la semana en que nació, por ejemplo suponiendo que una persona nació el

25/08/2004, el programa mostrará “Miércoles”, es importante recalcar que el siguiente código o es autoría de quienes desarrollaron el presente material bibliográfico.

```
C54:  #include <stdio.h>
      #include <stdlib.h>
      #include <string.h>
      char * dia_semana(int Num_dia)
      { switch(Num_dia ) {
        case 0: return "Domingo";
        case 1: return "Lunes";
        case 2: return "Martes";
        case 3: return "Miércoles";
        case 4: return "Jueves";
        case 5: return "Viernes";
        case 6: return "Sábado";
        }
      return "Error, día incorrecto...!";
      }
      int extrae_dia( int Mes, int Dia, int anio)
      { if(Mes >= 3) Mes -= 2;
        else Mes += 10;
        if( (Mes == 11) || (Mes == 12) ) anio--;
        int NumCenten = (anio / 100);
        int NumAnios = anio % 100;
        int ds = (2.6 * Mes - 0.2);
        ds += (Dia + NumAnios);
        ds += NumAnios / 4;
        ds += (NumCenten / 4);
        ds -= (2 * NumCenten);
        ds %= 7;
        if(anio >= 1700 && anio <= 1751) ds -= 3;
        else {
            if(anio <= 1699) ds -= 4;
        }
        if(ds < 0) ds += 7;
        return ds;
      }
      void main()
      { int d, m, a, dia;
        char *ds;
        system("cls");
        printf("\nIngresa su fecha de nacimiento:\n");
        printf("\nIngresa el día:"); scanf("%d",&d);
        printf("\nIngresa el mes:"); scanf("%d",&m);
        printf("\nIngresa el año:"); scanf("%d",&a);
        dia=extrae_dia(m, d, a);
        ds= dia_semana(dia);
        printf("\nEl día de nacimiento es:%s\n", ds);
        system("pause");
      }
```

**Fuente:** Publicado en la dirección: <http://programador-apli.blogspot.com/2012/04/calcular-el-dia-de-la-semana-partir-de.html>

### **Actividades de refuerzo (AR):**

- AR82. Desarrolle un programa que permita mediante una función verificar si una palabra o una frase es o no palíndroma.
- AR83. Desarrolle un programa que permita mediante una función mostrar una frase o una palabra al revés.

**OBJETIVO DE APRENDIZAJE DEL SIGUIENTE APARTADO:** Comprender y administrar la existencia y reconocimiento de las diferentes variables necesarias en un programa, con la finalidad de optimizar y agilizar la funcionalidad de los programas.

## AMBITO DE UNA VARIABLE

El concepto de ámbito de una variable define dos aspectos importantes, el primero permite el reconocimiento de su existencia y el segundo especifica su tiempo de vida durante la ejecución del programa; es importante recordar que en lenguaje C todas las variables deben ser declaradas antes de ser utilizada en cualquier proceso, esto más la ubicación donde se produce la declaración definen su ámbito, así las variables pueden ser declaradas en tres sitios diferentes:

- ✓ Dentro de las funciones.- La existencia de la variable solo es reconocida dentro de la función donde fue declarada, se las conoce como *variables locales* ya que estas variables solo son reconocidas de forma local (dentro de la función) y no es reconocida en otras funciones o áreas del programa, estas variables se crean al momento de ejecutar la función y son borradas al momento de finalizar la función.
- ✓ Fuera de todas las funciones.- La existencia de estas variables es reconocida dentro todas las funciones, se las conoce como *variables globales* ya que todas las áreas del programa las pueden utilizar, estas variables se crean al momento de ejecutar el programa y son borradas al momento de finalizar el mismo.
- ✓ Como parámetro de la función.- Su existencia es solo reconocida dentro de la función donde es declarada, también se las conoce como *variables locales*, ya que solo es reconocida de forma local y no es reconocida en otras funciones o áreas del programa.

Para ilustrar estas definiciones conceptuales analice el siguiente programa: Se necesita un programa que permita mediante un arreglo, registrar la lista de N números enteros, como requisito se necesita encontrar el número mayor, el número menor, el total de números pares, el total de números impares y el total de números primos que existan en la lista:

```
C55:  #include <stdio.h>
        #include <stdlib.h>

        //variables globales
        int lista[100], n;

        void mayor_menor()
        { int mayor=0, menor=32767;      //mayor y menor son locales de la función
          for(int i = 0; i < n; i++){    // la variable i es local del bucle for()
            if (lista[i]>mayor) mayor= lista[i];
            if (lista[i]<menor) menor= lista[i];
          }
          printf("\nmayor: %d menor: %d", mayor, menor);
        }
        void pares_impares()
        { int np=0, ni=0;                //np y ni son locales de la función
          for(int i = 0; i < n; i++){    // la variable i es local del bucle for()
            if (lista[i]%2 == 0) np++;
            if (lista[i]%2 != 0) ni++;
          }
          printf("\npares: %d impares: %d", np, ni);
        }
        int esprimo(int num)
        { if (num==1 || num==2 || num==3) return 1;
          for (int i=2; i<=(num/2)+1; i++)
            if (num%i==0) return 0;
          return 1;
        }
```

```

void main()
{ int i, np=0;
  system("cls");
  printf("Ingrese el número de elementos ene 1 y 100:");
  do{ scanf("%d",&n);
  }while(n<1 || n>100);
  printf("Ingrese la lista de números:\n");
  for(i = 0; i < n; i++) scanf("%d",&lista[i]);
  pares_impares();
  mayor_menor();
  for(i = 0; i < n; i++)
    if ( esprimo(lista[i]) ) np++;
  printf("\nEl número de primos: %d\n", np);
  system("pause");
}

```

Observe que tanto la variable “n” como el arreglo “lista”, son variables de ámbito global y por lo tanto son reconocidos en todas las funciones sin necesidad de ser declaradas dentro de ellas o de realizar paso de parámetros por referencia para utilizar su contenido, las variables “np” y “i” son *locales* y únicamente reconocidas por la función “*pares\_impares()*”, al igual que las variables “mayor” y “menor” de la función “*mayor\_menor()*”, observe que existen otras variables locales como “i” que son reconocidas solo dentro del bucle de repetición *for()*, o la variable de parámetro “num” que solo es reconocida dentro de la función “*esprimo(int num)*”; considerando el ejemplo, podríamos concluir que el ámbito de la variable de forma implícita dependerá de la ubicación donde se realiza la declaración de la variable; lenguaje C permite definir de forma explícita el ámbito de una variable utilizando ésta sintaxis:

[clase] *Tipo\_de\_dato* Nombre\_variable [=expresión][,Nombre\_variable[=expresión][...]];

La definición [clase] determina el ámbito de la variable, es utilizado para expresar de forma explícita su reconocimiento y su existencia, la definición de [clase] puede ser reemplazada por cualquiera de las siguientes expresiones:

- ✓ **auto**: Esta expresión explícita indica que la variable es local, es decir, que será reconocida en esa función o bloque de instrucciones, toda declaración por omisión es local, es decir, cuando no se especifica una [clase] en la declaración de una variable, ésta siempre es automática.

```

void mayor_menor()
{ auto int mayor=0, menor=32767; //mayor y menor son locales de la función
  for(int i = 0; i < n; i++){ // la variable i es local del bucle for()
    if (lista[i]>mayor) mayor= lista[i];
    if (lista[i]<menor) menor= lista[i];
  }
  printf("\nmayor: %d menor: %d", mayor, menor);
}

```

- ✓ **extern**: Esta expresión explícita permite definir a las variables externas o globales de ámbito, el uso de la variable es reconocido en cualquier función, es importante destacar que la declaración de una variable externa no puede incluir una inicialización.

```

//variables globales
extern int lista[100],n;

```

- ✓ **static**: Esta expresión explícita permite definir a las variables cuyo contenido durará desde el comienzo hasta el final de la ejecución del programa, éstas variables se las considera locales en su declaración y a la vez global por su contenido, todas las variables globales, por defecto, son de clase estática, es decir cuando una variable local en una función es declarada de clase estática, su valor no es eliminado al terminar la función sino que conserva su valor de una



llamada a otra, por ejemplo el siguiente programa utiliza una función para generar la serie de Fibonacci, observe que el valor de las variables no se pierde entre llamadas a la función:

```
C56:    #include <stdio.h>
        #include <stdlib.h>
        void fibonacci_1a1()
        { static int anterior=0, actual=1, siguiente;
          siguiente = anterior + actual;
          printf("%d\t",siguiente);
          anterior = actual;
          actual = siguiente;
        }
        void main()
        { system("cls");
          for (int i=1; i<=10; i++) fibonacci_1a1();
          system("pause");
        }
```

- ✓ **register:** Esta declaración es opcional para el computador (es decir que si existe la posibilidad la puede cumplirla), register es la posibilidad de que el programador utilice un espacio en los registros de la CPU, la ventaja de utilizar ésta área radica en que la velocidad de acceso es más rápida que las variables normales declaradas en la memoria principal, las variables de tipo registro siempre son automáticas, lo que solo son reconocidas de forma local a una función, además solo acepta variables de tipo entero o carácter. Recordemos que éste lenguaje fue creado con el propósito de desarrollar sistemas operativos y esta posibilidad es apreciada al desarrollo y cumplimiento del mismo.

```
int potencia(int base, register int expo)
{ register int resul;
  resul=1;
  for ( ; expo; expo--)
    resul *= base;
  return resul;
}
```

**OBJETIVO DE APRENDIZAJE DEL SIGUIENTE APARTADO:** Afianzar en los programadores el conocimiento de aplicar el reciclaje de código mediante la creación de archivos contenedores de funciones denominadas librerías.

## CREACION DE LIBRERÍAS PROPIAS

La creación de librerías consiste en crear un archivo plano de texto, este archivo no debe incluir la función main y debe tener como extensión (.h), para crear la librería se puede utilizar el editor de texto del compilador o un editor de texto como el bloc de notas, como ejemplo considere el siguiente grupo de funciones en la librería propia.h:

### Propia.h

```
int cuadrado(int num)
{ return num*num;
}
void cuadro(int xs, int ys, int xi, int yi)
{ int i;
  //Dibujar líneas horizontales superior e inferior
  for(i=xs ; i<xi ; i++){
    gotoxy(i,ys);cprintf("-"); // Alt 196="-"
    gotoxy(i,yi);cprintf("-");
  }
  //Dibujar líneas verticales izquierda y derecha
  for(i=ys ; i<yi ; i++){
    gotoxy(xs,i);cprintf("|"); //Alt 179="|"
  }
}
```

```

        gotoxy(xi,i);cprintf("|");
    }
    //Dibujar las esquinas del cuadro
    gotoxy(xs,ys);cprintf("┌"); // Alt 218="┌"
    gotoxy(xs,yi);cprintf("└"); // Alt 192="└"
    gotoxy(xi,ys);cprintf("┐"); // Alt 191="┐"
    gotoxy(xi,yi);cprintf("┘"); // Alt 217="┘"
}

Int esprimo(int num)
{ int i;
  if (num==1 || num==2 || num==3) return 1;
  for (i=2;i<=(num/2)+1 ; i++)
    if (num%i==0) return 0;
  return 1;
}

void soloint(int *num)
{ int tecla, limi=wherex();
  *num=0;
  do{ tecla=getch();
    if (tecla>=48 && tecla<=57){
      printf("%d",tecla-48); //calcula y muestra el digito presionado
      *num = *num * 10 + tecla-48; //almacena el carácter presionado
    }
    if (tecla==8 && limi<wherex()){
      *num = *num / 10;
      printf("\b \b");
    }
  }while(tecla!=13); //repite mientras tecla sea diferente de ENTER
  printf("\n");
}

```

Se recomienda que lo guardes dentro de la carpeta *include* del compilador que esté utilizando, normalmente esta carpeta se encuentra en la misma carpeta del compilador, otra opción es guardarlo en la misma carpeta donde se encuentra el código fuente, en cualquiera de los casos se puede aplicar los siguientes formatos:

**#include <propia.h>** Cuando el archivo plano *propia.h* se encuentre en la carpeta *include* del compilador que esté utilizando.

**#include "milibreria.h"** Observe que se utilizan comillas en vez de los signos < > para encerrar el nombre del archivo plano *propia.h*, al hacerlo el compilador asume que se encuentra en la misma carpeta que el archivo que tiene el código fuente.

El siguiente ejercicio muestra el uso de la librería:

#### Librería1:

```

#include <stdio.h>
#include <conio.h>
#include "propia.h"
void main()
{ int x1,y1,x2,y2;
  clrscr();
  printf("\nIngrese coordenada superior x y:"); scanf("%d%d",&x1,&y1);
  printf("\nIngrese coordenada inferior x y:"); scanf("%d%d",&x2,&y2);
  clrscr();
  cuadro(x1,y1,x2,y2);
  gotoxy(1,1);printf("Ingrese su edad:");soloint(&y1);
  gotoxy(1,2);printf("El cuadrado de la edad es: %d",cuadrado(y1));
  getch();
}

```

}

Observe que las instrucciones de las funciones *soloint()*, *cuadrado()* y *cuadro()* no se encuentra en la programación de *libreria1.cpp*, ya que su código fuente se encuentran en la librería *propia.h*, el realizar este esquema de organización mediante el uso de librerías, evita tener que copiar el código fuente de cada función en los programas que desee su utilización, es importante describir la finalidad de cada función que incluya en la librería mediante el uso de cabeceras de comentarios ya que su descripción ayudará en conocer las herramientas disponibles que posee la librería.

## RESUMEN

Las funciones son utilizadas para realizar procesos de forma organizada, cumplen con recibir datos y/o realizar acciones que genere solución a un requerimiento solicitado, el uso de las funciones reducen las líneas de instrucciones así como la redundancia de procesos, aplicando la estructuración y organización de librerías, permiten reutilizar sus instrucciones en otros programas, las funciones deben tener nombres propios que no permitan su duplicidad, los nombres de funciones deben cumplir las mismas reglas de creación de nombres de variables, Las funciones permiten en intercambio de datos, se prepara para recibir información mediante la declaración de parámetros y se pasan los mismos mediante el uso de argumentos, una función puede recibir una copia del contenido de una variable lo que se conoce como paso de parámetros por valor o la dirección de la variable lo que se conoce como paso de parámetro por referencia, cuando se realiza el paso de datos mediante la copia el contenido original de la variable no sufre cambios, mientras que, cuando se pasa la dirección de la variable el contenido original podrá ser cambiado, cuando se declara un parámetro, éste solo podrá recibir un solo valor y nada más que uno, así el paso de variables estructuradas como los arreglos se realizan únicamente mediante el paso de parámetro por referencia, es decir se pasa la dirección y no su contenido, cada función posee sus propias variables que se crean al momento de ser llamadas y son eliminadas al momento de terminar la función, una función puede hacer uso de sus propias variables así como las variables globales o generales que se declaran fuera de toda función.

## CAPITULO VII

### ESTRUCTURAS DE DATOS PROPIOS

Este capítulo muestra las metodologías y herramientas utilizadas para diseñar y crear variables estructuradas, el estudio de éste material, le proporcionará al programador las técnicas necesarias para crear variables que permiten agrupar diferentes espacios de almacenamientos incluidos los diferentes tipos de datos en una sola estructura de almacenamiento, este capítulo tiene la finalidad de aprovechar la organización estructurada, para aplicarlos en la optimización de los recursos de almacenamiento tanto para la memoria principal de computador como para las unidades de almacenamiento secundarias.

El estudio de esta disciplina de organización de datos, explica el funcionamiento y la forma en que los diferentes software implementan para gestionar y almacenar grupos de datos que se procesan como si se tratara de una sola estructura, en este capítulo se proporcionan los fundamentos generales que se aplicaron para la administración de las bases de datos y otros programas de similares características, estas estructuras de almacenamiento permite al capítulo de gestión de archivos, definir las estructuras base de almacenamiento para aplicarlos a los formatos binarios de registros físicos, así como fundamento para gestionar estructuras dinámicas e inteligencia artificial.

**OBJETIVO DE APRENDIZAJE DEL SIGUIENTE APARTADO:** Conocer y crear tipos de datos propios que le permitan al lector crear variables estructuradas con capacidad de formar combinaciones de registros en una sola variable o en arreglos estructurados, los conocimientos planteados permitirán al lector conocer las bases fundamentales de las estructuras de datos así como la estructura de almacenamiento de los archivos binarios.

### VARIABLES ESTRUCTURADAS.

Una variable estructurada es un tipo de dato definido por el programador que permite contener un grupo de diferentes tipos de datos, es decir compuesta por diferentes variables que se agrupan en una estructura, considere que las estructuras se las define como *simple* cuando contienen variables de tipo básico como caracteres, enteros o de coma flotante entre otros; así como de tipo *compuestas* que contienen datos más estructurados como arreglos, otras estructuras, listas, entre otras. Sin importar el tipo de variable que incluya la estructura, a cada elemento o variable contenida dentro de la estructura se les denomina *miembro* de la estructura.

Es importante entender que para utilizar una estructura en lenguaje C primero se la debe *definir* y después se la puede *crear mediante el uso de las declaraciones*, para definir una estructura se utiliza la siguiente sintaxis:

```
struct [nombre_estructura]{  
    Tipo_de_dato miembro_estructura_1;  
    Tipo_de_dato miembro_estructura_2;  
    .....  
    Tipo_de_dato miembro_estructura_N;  
};
```

*struct.*- Esta instrucción es una palabra reservada del lenguaje C, permite definir la existencia de un tipo de dato estructurado, inicia la estructura con la llave "{" y termina con la llave "}" y el punto y coma.

*[nombre\_estructura].*- Este formato especifica que el nombre de la estructura es opcional, en caso de escribirlo, el nombre debe cumplir con las reglas de creación de nombres de variables, el nombre de la estructura será utilizado para identificar el tipo de dato que se utilizará para declarar las variables estructuradas.

*miembro\_estructura\_1, miembro\_estructura\_2,...:* Son las variables internas o elementos miembros que componen la estructura, al igual que las variables, éstas deben cumplir con el orden de declaración normal de una variable, es decir: primero el *tipo\_de\_dato* después la lista de nombres de variables separadas por comas y finalizar la declaración con el punto y coma.

Una vez definida la estructura, el programador puede declarar variables de tipo estructurada, esta declaración pueden utilizar cualquiera de las siguientes formas:

#### PRIMERA FORMA:

```
struct {
    long num_lote;
    char propietario[80];
    float largo, ancho;
    double valor;
} terreno1, terreno2;
```

#### SEGUNDA FORMA:

```
struct terreno {
    long num_lote;
    char propietario[80];
    float largo, ancho;
    double valor;
};

struct terreno ter1, ter2;
```

#### TERCERA FORMA:

```
struct terreno {
    long num_lote;
    char propietario[80];
    float largo, ancho;
    double valor;
} ter1, ter2;
```

En la primera forma, aprovecha la definición de la estructura para realizar la declaración de las variables “terreno1” y “terreno2” que se utilizarán en el programa, es importante destacar que no se podrá declarar más variables en otras partes del programa ya que la definición no incluye el *nombre de la estructura*; el no incluir un nombre a la estructura es permitido ya que el nombre es opcional, por ello en la definición de la sintaxis de la estructura, el nombre se encuentra entre corchetes.

Para la segunda forma, La definición de la estructura incluye un nombre “terreno”, este nombre será utilizado para declarar variables estructuradas en cualquier otra parte del programa.

En la tercera forma, se define la estructura y se crean 2 variables, en sí es una combinación de las dos formas anteriores, observe que a diferencia de la primera forma, ésta sí incluye el nombre “terreno” y puede ser utilizado para crear más variables en cualquier parte del programa.

### INICIALIZACIÓN DE UNA VARIABLE ESTRUCTURADA:

Las variables estructuradas tienen las mismas posibilidades de inicialización que una variable normal, la inicialización debe respetar el orden y tipos de datos utilizados en la definición de la estructura, utiliza el modelo de inicialización de grupos de variables utilizados en las inicializaciones de los arreglos, el siguiente ejemplo explica de forma más clara la inicialización:

```
C57: #include <stdio.h>
#include <stdlib.h>
struct ejemplo {
    int DIA;
    char *MES;
    int ANIO;
};
void main()
{ struct ejemplo fecha = {18,"Marzo",1973}, cumple;
  cumple = fecha;
  system("cls");
  printf("Fecha: %d de %s de %d\n",cumple.DIA,cumple.MES,cumple.ANIO);
  system("pause");
}
```

Observe que el ejemplo aplica la inicialización utilizando las llaves “{}” para agrupar la asignación, los valores en el mismo orden de definición y separados por comas “,” y los valores a ser asignados respetando los formatos normales correspondiente a cada tipo de dato.

Es importante destacar que para hacer uso de las variables estructuradas y las variables internas definidas, se debe especificar el nombre de la *variable estructurada*, el *punto (.)* y el *nombre de la variable interna* de la estructura:

*Nombre\_variable\_estructurada.nombre\_variable\_miembro\_estructura*

Considere el siguiente problema: Se necesita un programa que permita crear una estructura para un estudiante, la estructura debe contener el nombre del estudiante y tres notas, como funcionamiento del programa, la estructura solo debe ser conocida de forma local para la función `main()`, el programa antes de finalizar debe mostrar si la sumatoria de las tres notas es mayor o igual a 70 para considerarlo como aprobado o caso contrario se lo considera reprobado:

```
C58: #include <stdio.h>
#include <stdlib.h>
#include <string.h>
void main()
{ struct {
    char *nombre;
    int n1,n2,n3;
} estudiante;
system("cls");
printf("Ingrese nombre del estudiante:");gets(estudiante.nombre);
printf("Ingrese nota 1:");scanf("%d",&estudiante.n1);
printf("Ingrese nota 2:");scanf("%d",&estudiante.n2);
printf("Ingrese nota 3:");scanf("%d",&estudiante.n3);
if (estudiante.n1 + estudiante.n2 + estudiante.n3 >= 70) printf("Aprobó\n");
else printf("Reprobó\n");
system("pause");
}
```

Observe que la estructura fue definida dentro de la función `main()`, el formato es similar al utilizado en la declaración de una variable estándar, esta definición no incluye un nombre ya que solo se necesita ser reconocida internamente en la función, además el nombre de la variable estructurada sirve como elemento agrupador de las variables miembro de la estructura, así que siempre será necesario utilizarla antes de referencias a las variables internas.

## VARIABLES ESTRUCTURAS ANIDADAS

Una estructura se la considera anidada, cuando uno o varios de los elementos miembros de la estructura son el resultado de otra estructura, por ejemplo: considere que se necesita un programa que permita registrar los datos a descontar y pagar de un jornalero que gana por horas de trabajo, mediante una variable estructurada se almacenará los datos de ingresos y egresos económicos; como política de la empresa considere que un jornalero solo se puede realizar un anticipo, el programa antes de finalizar debe mostrar los datos registrados y el total a recibir:

```
C59: #include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct ingreso{
    int HT;
    float VH;
    float comi;
};
struct egreso{
    int dia,mes,anio;
    float valor;
};
struct jornalero{
    char nombre[30];
    struct ingreso sueldo;
```

```

        egreso anticipo;
    };
    void main()
    { jornalero jose;
      system("cls");
      printf("Ingrese Nombre: "); gets(jose.nombre);
      printf("Ingrese horas trabajadas: "); scanf("%d",&jose.sueldo.HT);
      printf("Ingrese valor de hora: "); scanf("%f",&jose.sueldo.VH);
      printf("Ingrese comisión: "); scanf("%f",&jose.sueldo.comi);
      printf("DATOS DE ANTICIPO:\n");
      printf("Ingrese dia: "); scanf("%d",&jose.anticipo.dia);
      printf("Ingrese mes: "); scanf("%d",&jose.anticipo.mes);
      printf("Ingrese año: "); scanf("%d",&jose.anticipo.anio);
      printf("Ingrese valor del anticipo: "); scanf("%f",&jose.anticipo.valor);
      system("cls");
      printf("Nombre: %s\n",jose.nombre);
      printf("Horas trabajadas: %d\n",jose.sueldo.HT);
      printf("Valor de hora: %6.2f\n",jose.sueldo.VH);
      printf("Comisión: %6.2f\n",jose.sueldo.comi);
      printf("TOTAL DE INGRESOS:%6.2f\n", (jose.sueldo.HT* jose.sueldo.VH)+ jose.sueldo.comi);
      printf("Fecha de anticipo: %d/%d/%d\n", jose.anticipo.dia, jose.anticipo.mes, jose.anticipo.anio);
      printf("Valor del anticipo: %6.2f",jose.anticipo.valor);
      printf("TOTAL          RECIBIR:%6.2f\n",          ((jose.sueldo.HT*jose.sueldo.VH)+jose.sueldo.comi)-
jose.anticipo.valor);
      system("pause");
    }

```

Analice el siguiente problema: La alcaldía de la ciudad ha dispuesto una ordenanza que prohíbe la libre circulación de vehículos de carga pesada dentro de la ciudad, por tal razón la empresa de entrega de encomiendas “ABC” necesita un programa que permita mostrar el número de vehículos de carga liviana por cada vehículo de carga pesada, para esto se necesita una estructura que contenga los datos comunes para ambos tipos de vehículos y dos estructuras adicionales para cada tipo de vehículos de carga.

```

C60: #include <stdio.h>
        #include <stdlib.h>
        #include <string.h>

        struct vehiculo{
            int num_llantas;
            float cap_carga;
            int num_pasajeros;
        };
        struct cargaliviana{
            int velocidad;
            vehiculo liviano;
        };
        struct cargapesada{
            char *origen;
            vehiculo pesado;
        };
        void main()
        { cargaliviana furgon;
          cargapesada trailer;
          system("cls");
          printf("VEHICULO DE CARGA PESADA:\n");
          printf("Ingrese origen del tráiler: "); gets(trailer.origen);

```

```

printf("Ingrese número de llantas: "); scanf("%d",&trailer.pesado.num_llantas);
printf("Ingrese la capacidad de carga: "); scanf("%f",&trailer.pesado.cap_carga);
printf("Ingrese número de pasajeros: "); scanf("%d",& trailer.pesado.num_pasajeros);
printf("VEHICULO DE CARGA LIVINA:\n");
printf("Ingrese velocidad: "); scanf("%d",&furgon.velocidad);
printf("Ingrese número de llantas: "); scanf("%d",&furgon.liviano.num_llantas);
printf("Ingrese capacidad de carga: "); scanf("%f",&furgon.liviano.cap_carga);
printf("Ingrese número de pasajeros: "); scanf("%d",&furgon.liviano.num_pasajeros);
printf("Se necesitan: ");
if(int(trailer.pesado.cap_carga) % int(furgon.liviano.cap_carga)!=0)
    printf("%d vehículos livianos \n", int(trailer.pesado.cap_carga / furgon.liviano.cap_carga)+1);
else
    printf("%d vehículos livianos \n", int(trailer.pesado.cap_carga / furgon.liviano.cap_carga));
system("pause");
}

```

### **Actividades de refuerzo (AR):**

AR84. Desarrolle un programa que permita registrar la atención médica de un paciente que tiene un seguro de salud, se pide crear una estructura para la clínica que permita registrar el nombre y la dirección, otra estructura que sirva para registrar tanto al médico como para al paciente (nombre y especialidad\_tratamiento), y una tercera estructura que registre la unión de las dos estructuras anteriores, que incluya información de la clínica, médico y paciente, además se debe registrar el costo de atención.

### **CREACIÓN DE ARREGLOS UTILIZANDO VARIABLES ESTRUCTURADAS.**

Las variables estructuradas y definidas en cualquier programa, también pueden ser utilizadas para la creación de arreglos como vectores, matrices y cubos, su declaración y comportamiento es similar a los tipos de datos normales, la única diferencia notoria es la utilización de los miembros internos de la estructura, por ejemplo:

*Al tener la siguiente estructura:*

*Se puede realizar:*

<b>struct</b> terreno {	struct terreno bienes[10], lote={123,"Paola Montero",20,40,15000};
long num_lote;	bienes[0]=lote;
char propietario[80];	bienes[2].num_lote=456;
float largo, ancho;	strcpy(bienes[2].propietario, "Alejandro Demera");
double valor;	bienes[2].largo=30;
};	bienes[2].ancho=50;

El ejemplo muestra una estructura que posee datos de tipo entero largo, cadena de caracteres y valores de precisión decimal como el float y el double; el ejemplo crea un vector llamado "*bienes*" de 10 elementos y una variable estructurada llamada *lote*, esta variable esta inicializada con el lote número 123 que pertenece a la propietaria "Paola Montero", entre otros datos relacionados con el terreno; observe que el contenido de la variable *lote* es completamente pasado al primer elemento del vector "*bienes[0]=lote;*" mediante una simple asignación, el tercer elemento del vector es llenado de forma individual elemento por elemento, observe que el índice se aplica a la variable estructurada y no a los elementos internos de la estructura.

El ejemplo descrito detalla que la gestión de un arreglo no sufre ningún cambio al compararlo entre arreglo que está definido y un arreglo normal de cualquier tipo de dato, es decir ambos arreglos utilizan índices para referenciar cada elemento, considere que, para identificar cada elemento, se



utiliza el nombre del arreglo y entre corchete la posición referenciada por el índice, cuando se utiliza arreglos de variables estructuradas, los miembros de la estructura se separan del nombre del arreglo utilizando el punto y pueden tomar asignaciones completas o de forma específica por cada miembro del arreglo.

Para ejemplificar de forma adicional el uso de arreglos con estructuras de datos se modificará el ejercicio del jornalero de forma que permita registrar los datos para N jornaleros:

```
C61:  #include <stdio.h>
        #include <stdlib.h>
        #include <string.h>
        extern unsigned _floatconvert;
        #pragma extref _floatconvert
        struct ingreso{
            int HT;
            double VH;
            double comi;
        };
        struct egreso{
            int dia,mes,anio;
            double valor;
        };
        struct jornalero{
            char nombre[30];
            struct ingreso sueldo;
            egreso anticipo;
        };
        void main()
        { jornalero Jorna[50];
          int i,N;
          system("cls");
          printf("Ingrese el número de jornaleros: ");
          do{ scanf("%d",&N); } while(N<1 || N>50);
          for( i=0; i<N; i++){
              printf("Datos del %d jornalero:\n ",i+1); fflush(stdin);
              printf("Ingrese Nombre: "); gets(Jorna[i].nombre);
              printf("Ingrese horas trabajadas: "); scanf("%d",&Jorna[i].sueldo.HT);
              printf("Ingrese valor de hora: "); scanf("%lf",&Jorna[i].sueldo.VH);
              printf("Ingrese comisión: "); scanf("%lf",&Jorna[i].sueldo.comi);
              printf("DATOS DE ANTICIPO:\n");
              printf("Ingrese dia: "); scanf("%d",&Jorna[i].anticipo.dia);
              printf("Ingrese mes: "); scanf("%d",&Jorna[i].anticipo.mes);
              printf("Ingrese año: "); scanf("%d",&Jorna[i].anticipo.anio);
              printf("Ingrese valor del anticipo: "); scanf("%lf",&Jorna[i].anticipo.valor);
          }
          system("cls");
          for(i=0; i<N;i++){
              printf("\nNombre: %s\n",Jorna[i].nombre);
              printf("Horas trabajadas: %d\n",Jorna[i].sueldo.HT);
              printf("Valor de hora: %6.2lf\n",Jorna[i].sueldo.VH);
              printf("Comisión: %6.2lf\n",Jorna[i].sueldo.comi);
              printf("TOTAL DE INGRESOS:%d\n", (Jorna[i].sueldo.HT* Jorna[i].sueldo.VH)+ Jorna[i].sueldo.comi);
              printf("Fecha de anticipo: %d/%d/%d\n", Jorna[i].anticipo.dia, Jorna[i].anticipo.mes,
Jorna[i].anticipo.anio);
              printf("Valor del anticipo: %6.2lf\n",Jorna[i].anticipo.valor);
              printf("A RECIBIR:%6.2lf\n", ((Jorna[i].sueldo.HT*Jorna[i].sueldo.VH)+Jorna[i].sueldo.comi)-
Jorna[i].anticipo.valor);
          }
          system("pause");
        }
```

Debido a las dificultades que presentan algunos compiladores de lenguaje C al utilizar variables con coma flotante como miembros de una estructura, esta limitación se presenta al momento de ejecutarse el programa, al encontrar la instrucción que haga referencia al miembro de la coma

flotante, el programa muestra el error *"floating point formats not linked"*, para evitarlo se recomienda utilizar las siguientes líneas después de las definiciones de las librerías:

```
extern unsigned _floatconvert;
#pragma extref _floatconvert
```

El error se muestra al tratar de ser eficiente el compilador, por defecto activa y desactiva el uso de la librería de gestión de puntos flotantes según sea necesario, pero en algunas ocasiones falla y produce el error, usualmente al utilizar las funciones que usan %f en llamadas a *printf()* o *scanf()*, las dos líneas solucionan el problema al indicar al compilador que se usará el tipo de dato *float* o *double* sin esperar que lo active o desactive de forma automática.

Considere el siguiente problema: La Policía Nacional está en una campaña de reclutamiento de jóvenes para formarlos como miembros de la institución, para ser aceptado el joven debe cumplir con la edad, talla y/o examen de conocimiento según la siguiente tabla:

Se considera aprobado:

- ✓ Sí obtiene 10 en el examen de conocimiento sin importar la edad ni la talla.
- ✓ Sí obtiene 8 o 9 en el examen de conocimiento debe tener una edad menor a 23 sin importar la talla.
- ✓ Sí obtiene 7 en el examen de conocimiento debe tener una talla mayor a 180 sin importar la edad.

Desarrolle un programa que permita registrar los datos de N postulantes, por cada postulante se debe recibir la edad, talla y nota del examen, el programa antes de salir debe mostrar la lista de aprobados y el promedio de las edades.

```
C62: #include <stdio.h>
#include <stdlib.h>
extern unsigned _floatconvert;
#pragma extref _floatconvert
void main()
{ struct {
    char nombre[20];
    int edad, nota;
    float talla;
} poli[100];
int i, num, sumedad=0, c=0;
system("cls");
printf("Ingrese el número de postulantes entre 1 y 100:");
do{ scanf("%d",&num); }while(num<1 || num>100);
for( i=0 ; i < num ; i++){
    fflush(stdin);
    printf("\nIngrese nombre del postulante %d:",i+1); gets(poli[i].nombre);
    printf("Ingrese la edad :");scanf("%d",&poli[i].edad);
    printf("Ingrese la talla :");scanf("%f",&poli[i].talla);
    printf("Ingrese la nota :");scanf("%d",&poli[i].nota);
}
for( i=0 ; i < num ; i++){
    if (poli[i].nota == 10) {
        printf("Nombre: %s Talla:%6.2f Edad:%d Nota:%d\n", poli[i].nombre, poli[i].talla, poli[i].edad, poli[i].nota);
        sumedad+= poli[i].edad; c++;
    }
    else if (poli[i].nota >= 8 && poli[i].nota <= 9 && poli[i].edad < 23 ) {
        printf("Nombre: %s Talla:%6.2f Edad:%d Nota:%d\n", poli[i].nombre, poli[i].talla, poli[i].edad, poli[i].nota);
        sumedad+= poli[i].edad; c++;
    }
    else if (poli[i].nota == 7 && poli[i].talla > 180 ) {
```

```

        printf("Nombre: %s Talla:%6.2f Edad:%d Nota:%d\n", poli[i].nombre, poli[i].talla, poli[i].edad,
        poli[i].nota);
        sumedad+= poli[i].edad; c++;
    }
}
if(c) printf("Promedio de edades: %6.2f \n",float(sumedad/c));
system("pause");
}

```

Observe que la estructura fue definida dentro de la función *main()*, sirvió para declarar el vector *poli* de 100 elementos, el programa activar los recursos de cálculo para valores con coma flotante y doubles, el programa valida el ingreso de número de elementos a utilizar en el vector entre 1 y 100.

### **Actividades de refuerzo (AR):**

AR85. Desarrolle un programa que permita registrar la atención médica de N pacientes que tiene un seguro de salud, se pide crear una estructura para la clínica que permita registrar el nombre y la dirección, otra estructura que sirva para registrar tanto a los médico como para los pacientes (nombre y especialidad\_tratamiento), y una tercera estructura que registre la unión de las dos estructuras anteriores, que incluya información de la clínica, médico y paciente, además se debe registrar el costo de atención, el programa antes de finalizar debe mostrar el total a pagar por los costos de atención a los clientes.

### **PASO DE ESTRUCTURAS COMO PARÁMETROS EN LAS FUNCIONES**

El uso de variables estructuradas como parámetros y argumentos es similar a las variables normales, el siguiente ejemplo utiliza una función que transforma de decimal a binario, para esto se utilizará una estructura que conserve tanto el valor decimal como el binario:

**C63:**

```

#include <stdio.h>
#include <stdlib.h>
struct bindec{
    int binario[8];
    int dec;
};
void convierte(int n, struct bindec x[])
{ int i, y, pos, aux;
  for( y=0; y<n; y++)
    for ( i=0; i<8; i++)
      x[y].binario[i]=0;
  for (y=0;y<n;y++){
    pos=0; aux=x[y].dec;
    while (aux>=2){
      x[y].binario[pos] = aux % 2;
      aux/=2;
      pos++;
    }
    x[y].binario[pos]=aux;
  }
}
void muestra (int n, struct bindec x[])
{ int y,i;
  for ( y=0; y<n; y++ ){
    printf("%d\t", x[y].dec);
    for (i=7;i>=0;i--) printf("%d\t", x[y].binario[i]);
    printf("\n");
  }
}

```

```

void main()
{
    bindec lista[10];
    int i, num;
    system("cls");
    do{ printf("Ingrese número de elementos a transformer (max 10): ");
        scanf("%d",&num);
    }while (num<1 || num>10);
    for( i = 0 ; i < num; i++){
        printf("Ingrese una cantidad entre 1 y 255: ");
        do{
            scanf("%d",&lista[i].dec);
        }while (lista[i].dec <1 || lista[i].dec >255);
    }
    convierte(num, lista);
    muestra(num, lista);
    system("pause");
}

```

El ejercicio muestra la forma de utilizar arreglos de estructura de datos como argumentos, observe que no existe diferencia entre los argumentos comunes, es decir utilizando los tipos de datos convencionales y los argumentos especiales con tipos de datos estructurados.

### **Actividades de refuerzo (AR):**

AR86. Desarrolle un programa que permita registrar la atención médica de N pacientes que tiene un seguro de salud, se pide crear una estructura para la clínica que permita registrar el nombre y la dirección, otra estructura que sirva para registrar tanto a los médico como para los pacientes (nombre y especialidad\_tratamiento), y una tercera estructura que registre la unión de las dos estructuras anteriores, que incluya información de la clínica, médico y paciente, además se debe registrar el costo de atención, el programa debe utilizar una función que muestre por cliente el costo real a cubrir por el seguro, considere que el porcentaje puede estar entre el 80 y el 100%, la función debe mostrar el total a pagar por los costos de atención a los clientes.

## **RESUMEN**

Las variables estructuradas son aquellas que permiten incorporar en una sola estructura varias variables internas que admiten ser presentadas como si se tratara de una sola variable, para crear una variable estructurada se utiliza la palabra reservada struct, internamente se puede crear como miembro de la estructura otra variable estructurada lo que se conoce como estructuras anidadas, en algunos materiales bibliográficos a esta estructura se la conoce como variables de registro o estructura de registro, las variables estructuradas se comportan como variables normales tanto así que los diferentes operadores no presentan errores al utilizarlos con las variables normales, la única diferencia notable radica en especificar a la variable estructurada más la variable interna unidas por el punto, se puede inicializar las variables al momento de ser declaradas, los arreglos utilizan el mismo formato o sintaxis que un arreglo estándar, la definición de parámetros utiliza el mismo formato de corchetes vacíos para recibir la dirección de un arreglo de variables estructuradas, aplica un índice para definir vectores, dos índices para definir matrices y así sucesivamente con la estructura deseada.

## **CAPITULO VIII**

### **GESTIÓN DE ARCHIVOS DIGITALES**

Es importante destacar que todo proceso realizado por la computadora necesita ser alimentado por datos o información pertinente que el usuario aporta al momento de ejecutar la aplicación, considere que hasta ahora se ha tratado con información contenida en las variables que ocupan espacio de

almacenamiento en la memoria RAM, dicha información está activa al momento que se ejecuta el programa pero ésta desaparece al momento de finalizar el mismo, considere que para cualquier usuario le resulta incómodo tener que registrar los mismos datos cada vez que se ejecuta el programa, este capítulo introduce al programador en la creación y administración de archivos digitales, explica los procedimientos necesarios para enviar la información contenida en la memoria a los diferentes dispositivos de almacenamiento, así también ejemplifica las diferentes técnicas de almacenamiento y las instrucciones necesarias para cumplir éste propósito.

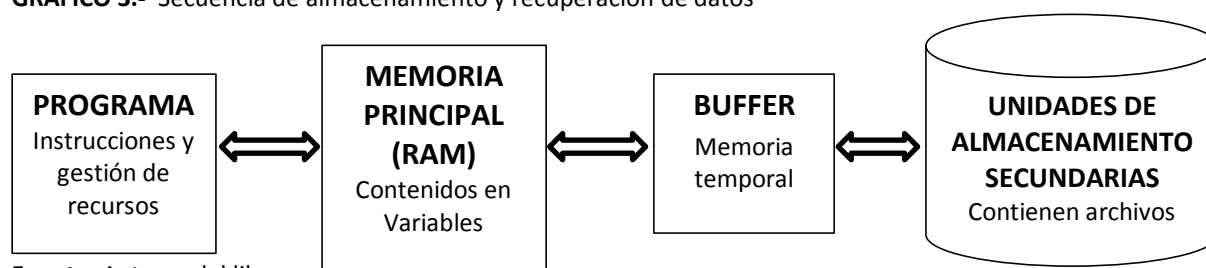
Es importante entender que los computadores funcionan mediante el uso del sistema operativo, esta plataforma controla al computador en su totalidad y por lo tanto posee sus propias reglas de gestión en general, ante ésta realidad, cada sistema operativo posee normas de organización y gestión de la información que incluye a los diferentes medios de almacenamiento, este capítulo toma como modelo de aprendizaje el sistema de almacenamiento utilizado por los sistemas operativos Windows, cabe indicar que, por la estructura dada que ofrece el lenguaje de programación, se utilizará el sistema de organización FAT ya que corresponde a la estructura propia del lenguaje en su contexto.

**OBJETIVO DE APRENDIZAJE DEL SIGUIENTE APARTADO:** Conocer y gestionar la creación y apertura de archivos digitales, haciendo uso de las herramientas que ofrece el lenguaje de programación, así como aplicar las estructuras de datos como método organizativo de los registros digitales en la creación y gestión de archivos de texto y binarios.

## ARCHIVOS

Todo trabajo realizado y almacenado en un computador es considerado un archivo, éstos se almacenan en unidades de almacenamiento como discos duros, flash memory, CD, DVD, entre otros, es decir las unidades de almacenamiento solo contienen archivos sin importar si son videos, documentos, imágenes, programas, animaciones, publicaciones, presentaciones o cualquier otro producto producido por el usuario mediante el uso del computador; técnicamente un archivo es el resultado de registrar o almacenar la información organizada internamente por el computador en variables y mediante la aplicación de instrucciones especializadas son enviadas y contenidas en cualquier unidad de almacenamiento secundaria, considere la siguiente gráfica que muestra los elementos que intervienen en el proceso:

**GRÁFICO 3.-** Secuencia de almacenamiento y recuperación de datos



**Fuente:** Autores del libro

Un archivo está compuesto por dos bloques de datos, el primer bloque y por lo general el más grande, es la información procesada proveniente de la memoria principal del computador y el segundo bloque y más pequeño de datos son los atributos que identifican al archivo, en general los archivos dependiendo del sistema operativo poseen entre otros los siguientes datos: Nombre, Identificador único, Tipo de archivo, Ubicación, Tamaño, Protección, Fechas, Identificación de propietario, Información de control, entre otros, considere que estos datos variarán dependiendo del sistema operativo, básicamente por la forma en cómo se almacenan, los archivos se pueden agrupar en dos tipos: archivos ASCII y archivos binarios.

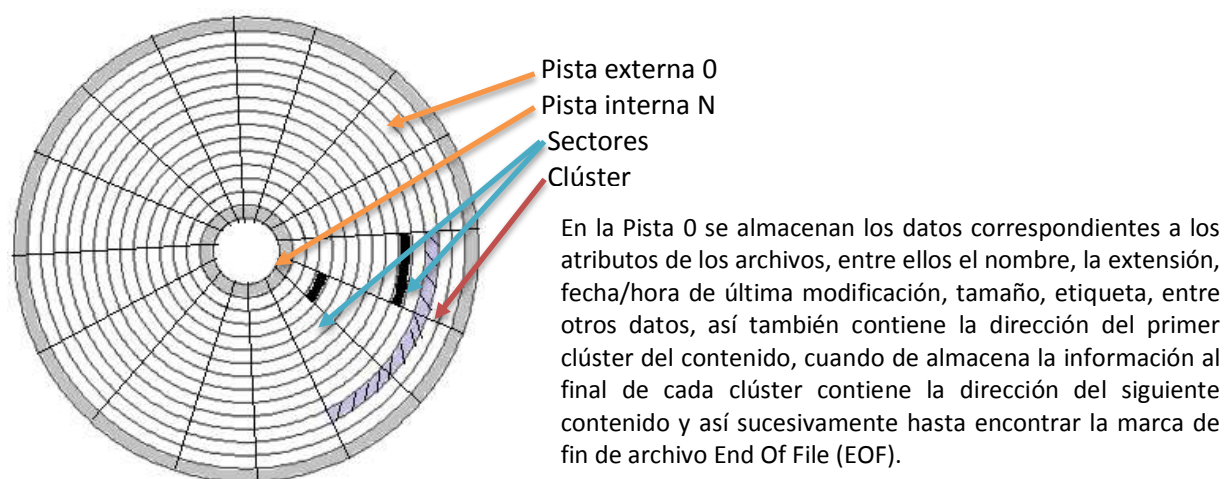
Los archivos ASCII o también conocidos como archivos de texto o archivos planos, son archivos cuyo contenido almacenado son caracteres ASCII, tal y cual como fueron ingresados a la memoria principal del computador, su estructura está organizada en forma de documento y no posee ningún formato

especial, estos archivos al momento de ser leídos, su contenido se reciben en secuencia y no de forma estructurada, es decir no respetan formatos ya que su naturaleza es plana; los archivos binarios se diferencian de los archivos planos al momento de ser leídos, ya que su estructura funcional y organizacional es respetada y por lo tanto se organizan estructuralmente en memoria, es decir considera la estructura original utilizada antes de ser almacenados sus datos, es por ello que son ampliamente utilizados en formatos de música, videos, imágenes, animaciones, documentos, presentaciones, entre otros.

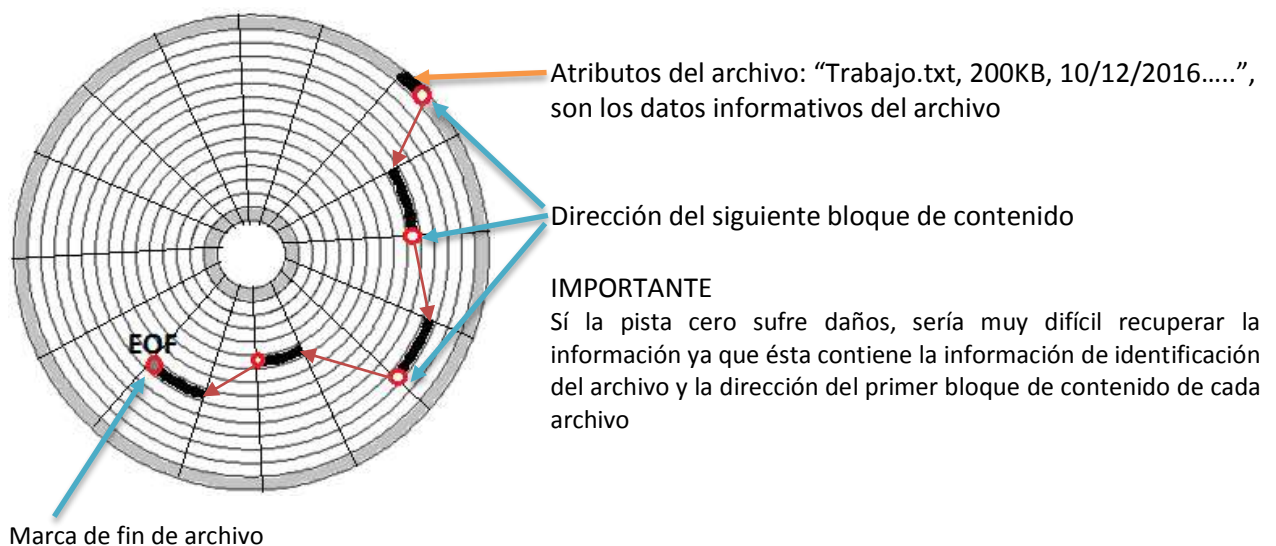
Los programadores necesitan conocer las actividades que se pueden desarrollar tanto con los archivos planos como con los archivos binarios, en ambos se puede realizar las siguientes actividades: Solicitar operaciones con el archivo (crear y/o abrir), liberar la operación solicitada (cerrado), leer el contenido de un archivo (Lectura), escribir el contenido a un archivo (Agregar al final o sobre escritura total), cambiar el puntero en cualquier operación, consultar los atributos de un archivo, manipular los atributos de cualquier archivo, entre otras operaciones.

En el gráfico 3 se muestra los elementos lógicos y físicos que intervienen en el proceso de lectura y almacenamiento de datos en los archivos, así considerando estos elementos en capítulos anteriores ya se ha detallado el funcionamiento y organización de la memoria RAM; considere los dos últimos elementos a la derecha del gráfico 3, la memoria temporal llamada BUFFER es un recurso que el computador utiliza para contener la información de paso desde la memoria principal hasta las unidades de entrada y/o salida, para ejemplificar el funcionamiento de las unidades de almacenamiento, se explicará la estructura de un disco para entender su organización y funcionamiento.

Un disco está compuesto por pistas, sectores y clúster, cada uno de ellos cumplen un propósito específico; así las pistas son divisiones físicas en forma de filas concéntricas circulares, su función permite la orientación de los mecanismos físicos y lógicos utilizados por la cabeza lectora/escritora que permiten leer y guardar la información en el disco. Los sectores son las divisiones que tiene una pista para almacenar la dirección y el contenido fragmentado de la información, su organización es parecida a una celda que almacena aproximadamente 512 Byte. Los clúster son organizaciones lógicas de sectores, es decir un clúster es la unión de varios sectores y su propósito radica en organizar la información en grupos de sectores y así disminuir la administración de grandes cantidades de direcciones por el uso de sectores independientes.



Para ejemplificar el proceso, considere que se ha almacenado un documento plano que contiene información cualquiera, asuma que el nombre es “trabajo” al ser plano su extensión es “.txt”, su estructura almacenada quedaría de la siguiente forma:



Los programas o aplicaciones en general cumplen con el propósito de enviar los datos a la unidad de almacenamiento y también se encarga de recibirlos, el programa en realidad no sabe dónde se almacenan dicha información, para establecer el envío y la recepción de datos almacenados, el programa debe apoyarse en el sistema operativo, que es quién establece las reglas de gestión como donde almacenarlo, tipo de acceso (solo lectura o lectura y escritura, etc), cantidad de espacio a utilizar, entre otros; técnicamente el programa utiliza un nombre lógico y un puntero que contiene la dirección física del archivo, estos datos son proporcionados por el sistema operativo.

Antes de que el programa pueda gestionar un archivo físico, el sistema operativo debe recibir las instrucciones adecuadas para que el programa tenga acceso a la dirección física del archivo, lenguaje C ofrece las siguientes funciones para crear y administrar los diferentes archivos:

**fopen().**- Esta función permite crear y/o abrir un archivo, si el archivo no existe procede a crearlo pertenece a la librería *stdio.h*, esta función devuelve la dirección física del archivo, por lo que es obligatorio crear una variable que almacene la dirección del archivo, para lograrlo se utilizar la estructura **FILE**, esta estructura proporcionada por la librería contiene la información necesaria sobre el nombre del archivo, su modo de apertura (solo lectura, lectura/escritura, etc.), estado, entre otros datos, esta variable se la considera un "puntero de archivo", por tanto, esta variable podrá ser utilizada por todas las funciones que quieran administrar el archivo; la sintaxis de la función `fopen()` es la siguiente:

```
FILE *fopen(const char *nombre_archivo, char *modo_acceso);
```

Observe que el formato utiliza como primer argumento una cadena de caracteres que representará el nombre válido de un archivo, esta cadena también puede especificar la ruta o ubicación lógica del archivo que desea abrir o crear; el argumento `modo_acceso` es un texto específico que indica al sistema operativo el modo de acceder al archivo, considere la siguiente tabla que muestra las posibilidades de apertura del archivo:

**Tabla 33.-** Modos de apertura de un archivo utilizando la función `fopen()`.

<b>MODO_ACCESO</b>	<b>PROPÓSITO</b>
<b>r</b>	Permite abrir un archivo para lectura.
<b>w</b>	Permite crear un archivo para escritura.
<b>a</b>	Permite abrir un archivo para añadir al final.
<b>rb</b>	Permite abrir un archivo binario para lectura.
<b>wb</b>	Permite crear un archivo binario para escritura.
<b>ab</b>	Permite abrir un archivo binario para añadir al final.
<b>rt</b>	Permite abrir un archivo de texto para lectura.

<b>wt</b>	Permite crear un archivo de texto para escritura.
<b>at</b>	Permite abrir un archivo de texto para añadir al final.
<b>r+</b>	Permite abrir un archivo para lectura/escritura.
<b>w+</b>	Permite crear un archivo para lectura/escritura.
<b>a+</b>	Permite abrir un archivo para leer/añadir.
<b>r+b</b>	Permite abrir un archivo binario para lectura/escritura.
<b>w+b</b>	Permite crear un archivo binario para lectura/escritura.
<b>a+b</b>	Permite abrir un archivo binario para leer/añadir.
<b>r+t</b>	Permite abrir un archivo de texto para lectura/escritura.
<b>w+t</b>	Permite crear un archivo de texto para lectura/escritura.
<b>a+t</b>	Permite abrir un archivo de texto para leer/añadir.

**Fuente:** Autores del libro.

**fclose().** Esta función pertenece a la librería `stdio.h`, sirve para cerrar la gestión con el archivo, su principal uso radica en vaciar el contenido del buffer a la dirección del archivo, esto es necesario ya que en algunas ocasiones la memoria temporal buffer no envía la información al archivo destino, `fclose` se asegura de que ésta información sea salvada, su sintaxis es la siguiente:

```
int fclose(FILE * dirección_archivo);
```

Para ejemplificar la creación de archivos y aplicar la gestión de almacenamiento de información en un formato plano o de texto, se utilizarán 3 ejercicios, el primer ejemplo muestra el almacenamiento, el segundo ejemplo muestra la extracción de datos almacenados y el tercer ejercicio es un compendio de los dos primeros, para el primer ejemplo se utilizará la siguiente función:

**fprintf().** Esta función sirve para enviar datos a un archivo, pertenece a la librería `stdio.h`, funcionan de forma similar a la función `printf()`, la única diferencia radica en especificar la dirección del archivo destino, la sintaxis es la siguiente:

```
int fprintf(FILE *dirección_archivo, const char *formato, variables...);
```

Esta función devuelve el número de caracteres enviados, o un valor negativo para mostrar la existencia de un error al momento de guardar información.

```
ARCHI1: #include <stdio.h>
        #include <stdlib.h>
        int main()
        { FILE *pa;
          char linea[70];
          system("cls");
          pa = fopen("primero.txt", "wt");
          printf("Ingrese una línea de texto para almacenarla en el disco:\n");
          gets(linea);
          fprintf(pa, "%s\n", linea);
          fclose(pa);
          return 0;
        }
```

Este ejercicio permite contener cualquier mensaje en la variable *línea*, mediante el uso de la función `fprintf()` dicho contenido es pasado al archivo *primero.txt*, observe que el archivo es abierto o creado para sobre escribir cualquier contenido, es decir el contenido antiguo es reemplazado por lo que se escriba al momento de ejecutar el programa.

El siguiente ejercicio ejemplifica el proceso de recuperar la información contenida en un archivo, este proceso es complementario al primer ejercicio, para lograrlo utiliza las siguientes funciones:



**fgets().**- Esta función pertenece a la librería `stdio.h`, permite leer información contenida en archivos planos, funciona de forma similar a la función `gets()`, la diferencia radica en que los datos son tomados desde un archivo y se debe indicar el número de caracteres que se desea recuperar, su sintaxis es la siguiente:

```
char *fgets(char *cadena_destino, int num_caracteres, FILE *dirección_archivo);
```

Es importante destacar que la función lee como máximo uno menos que el número de caracteres indicado por **num\_caracteres** desde la `dirección_archivo`.

**feof().**-La función `feof` pertenece a la librería `stdio.h` retorna un valor distinto a cero si y sólo si el indicador ha llegado al final del archivo, su sintaxis es la siguiente:

```
int feof(FILE *dirección_archivo);
```

```
ARCHI2 #include <stdio.h>
        #include <stdlib.h>
        int main()
        { FILE *pa;
          char linea[60];
          system("cls");
          pa = fopen("primero.txt", "rt");
          do{ fgets(linea, 60, pa);
              printf("%s\n\n", linea);
          }while (!feof(pa));
          fclose(pa);
          system("pause");
          return 0;
        }
```

Observe que el archivo *primero.txt* en el ejemplo es abierto para solo lectura, además es necesario realizar un ciclo repetitivo para leer desde el principio hasta el final el archivo, considere que se estará leyendo el archivo mientras no sea fin de archivo "`!feof(pa)`"; el siguiente ejemplo realiza un compendio de los dos ejercicios anteriores:

```
ARCHI3 #include <stdio.h>
        #include <string.h>
        #include <conio.h>
        int main()
        { FILE *pa;
          int opc;
          char linea[60];
          do{
            clrscr();
            printf("<1> Agregar nueva línea\n<2> Mostrar líneas grabadas:\n");
            printf("<3> Salir\nSelecione su opción:\n");
            do{ opc=getch(); }while(opc!=49 && opc!=50 && opc!=51);
            switch(opc){
              case 49: pa = fopen("primero.txt", "at");
                       printf("Ingrese una línea de texto para almacenarla en el disco:\n\n");
                       gets(linea);
                       fprintf(pa, "%s\n", linea);
                       fclose(pa);
                       break;
              case 50: pa = fopen("primero.txt", "rt");
```

```

        do{ fgets(linea, 60, pa);
            printf("%s\n",linea);
        }while (!feof(pa));
        fclose(pa);
        getch();
    }
    }while(opc!=51);
    return 0;
}

```

Observe que al momento de escoger la opción de agregar una nueva línea, el archivo se abre o se crea para añadir al final del mismo, esto permitirá conservar la información registrada con anterioridad. También es posible leer un archivo carácter a carácter, para esto se utiliza la función `getc()` que permite extraer carácter a carácter el contenido de un archivo:

**ARCHI4**

```

#include <stdio.h>
#include <stdlib.h>
main()
{ FILE *pa;
  char caracter;
  system("cls");
  pa = fopen("primero.txt","r");
  printf("Contenido del Archivo:\n");
  caracter = getc(pa);
  while (feof(pa) == 0) {
      printf("%c",caracter);
      caracter = getc(pa);
  }
  fclose(pa);
  printf("\n");
  system("Pause");
  return 0;
}

```

Existen muchas funciones que permiten gestionar la información almacenada en archivos planos como `getc`, `putc`, `fgets`, `fputs`, entre otras, cada una de ellas posee características que pueden ser aprovechadas según la necesidad de almacenamiento; ahora se propone la ejemplificación en la gestión de archivos binarios, éstos a diferencia de los archivos de texto, permiten almacenar información utilizando los diferentes tipos de datos, mientras que los archivos planos solo almacenan caracteres, los archivos binarios permiten trabajar con variables estructuradas o variables de tipo registro, el siguiente ejercicio registra en un archivo binario una lista de ciudades con los totales de hombres y mujeres como si se tratara de un censo, para conseguir este propósito se utilizará las funciones `fwrite` y `fread`.

`fwrite()`.- Esta función pertenece a la librería `stdio.h`, permite guardar información contenida en una variable a un archivo destino, la sintaxis aplicada cumple el siguiente formato:

tamaño\_guardado `fwrite(const void *variable_origen, size_t tamaño, size_t número, FILE *dirección_archivo);`

Este formato contienen los siguientes elementos: *\*variable\_origen* corresponde a la dirección de la variable que contiene la información que se desea almacenar, *tamaño* corresponde al tamaño en byte de toda la información que se desea almacenar (se recomienda utilizar la función `sizeof(tipo_dato)` que devuelve el tamaño de byte que son separados para cada variable utilizadas para contener la información), *número* corresponde al número de datos que desea almacenar, por lo general siempre es uno, *\*dirección\_archivo* corresponde a la variable que contiene la dirección

física del archivo utilizada en la apertura del archivo con *fopen*; es importante destacar que la función *fwrite* devuelve el número de elementos (no bytes) escritos físicamente en el archivo.

*fread()*.-Esta función pertenece a la librería *stdio.h*, permite leer información almacenada en un archivo a una variable destino con la misma estructura utilizada en el almacenamiento, la sintaxis aplicada cumple el siguiente formato:

```
size_t fread(const void *variable_destino, size_t tamaño, size_t número, FILE *dirección_archivo);
```

Considere que su formato es igual al utilizado por la función *fwrite* la diferencia radica en que la dirección de la variable ahora funciona para contener lo leído en el archivo, es decir la *variable\_destino* es la dirección de la variable donde se almacenará la información contenida en el archivo.

```
ARCHI5 #include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
struct censo
{ char ciudad[30];
  int TotalH;
  int TotalM;
};
main()
{ FILE *pa;
  char bc[30];
  int opc;
  censo poblacion;
  do{ system("cls");
    printf("<1> Agregar nueva línea\n<2> Mostrar líneas grabadas:\n");
    printf("<3> Buscar ciudad\n<4> Salir\nSelecione su opción:\n");
    do{ opc=getch(); }while(opc!=49 && opc!=50 && opc!=51 && opc!=52);
    switch(opc){
      case 49: pa = fopen("Conteo.txt","a+b");
        fflush(stdin);
        printf("Nombre de la ciudad: ");gets(poblacion.ciudad);
        printf("Total de hombres : ");scanf("%d",&poblacion.TotalH);
        printf("Total de mujeres : ");scanf("%d",&poblacion.TotalM);
        fwrite(&poblacion,sizeof(censo),1,pa);
        fclose(pa);
        break;
      case 50: pa = fopen("Conteo.txt","rb");
        printf("\tCIUDAD\tHOMBRES\tMUJERES\n");
        do{ fread(&poblacion,sizeof(censo),1,pa);
          printf("\t%s\t%d\t%d\n",poblacion.ciudad,poblacion.TotalH,poblacion.TotalM);
        }while (!feof(pa));
        fclose(pa);
        printf("\n");
        system("Pause");
        break;
      case 51: pa = fopen("Conteo.txt","rb");
        printf("Ciudad a buscar: ");gets(bc);
        printf("\tCIUDAD\tHOMBRES\tMUJERES\n");
        while (!feof(pa)){
```

```

        fread(&poblacion,sizeof(censo),1,pa);
        if(strncmp(poblacion.ciudad,bc)==0)
            printf("\t%s\t%d\t%d\n",poblacion.ciudad,poblacion.TotalH,poblacion.TotalM);

    }
    fclose(pa);
    printf("\n");
    system("Pause");
    break;
}
}while(opc!=52);
return 0;
}

```

El ejercicio de ejemplo muestra tres opciones para trabajar con los archivos binarios, la opción número uno cuyo código ASCII de la tecla es el 49, corresponde al ingreso de nuevos registro:

```

case 49: pa = fopen("Conteo.txt","a+b");
        fflush(stdin);
        printf("Nombre de la ciudad: ");gets(poblacion.ciudad);
        printf("Total de hombres : ");scanf("%d",&poblacion.TotalH);
        printf("Total de mujeres : ");scanf("%d",&poblacion.TotalM);
        fwrite(&poblacion,sizeof(censo),1,pa);
        fclose(pa);
        break;

```

El proceso empieza obteniendo la dirección del archivo “Conteo.txt” con fopen, cuyo modo de apertura es para crear o abrir un archivo binario con el propósito de agregar o leer información “a+b”, para registrar la información se utiliza la variable estructurada llamada “población”, observe que se utiliza de forma combinada gets y scanf, esto significa que se podrían presentar inconvenientes de organización en la memoria temporal buffer, para solucionarlo se utiliza la función fflush que permite limpiar el buffer y utilizar la función gets sin problemas, una vez llenos los datos en la variable estructura se guardan utilizando la función fwrite que envía al archivo lo que contiene la variable “poblacion”, y por último cierra el archivo con fclose.

La opción número dos que corresponde al código ASCII número 50 permite mostrar la lista de todos los datos almacenados en el archivo:

```

case 50: pa = fopen("Conteo.txt","rb");
        printf("\tCIUDAD\tHOMBRES\tMUJERES\n");
        do{ fread(&poblacion,sizeof(censo),1,pa);
            printf("\t%s\t%d\t%d\n",poblacion.ciudad,poblacion.TotalH,poblacion.TotalM);
        }while (!feof(pa));
        fclose(pa);
        printf("\n");
        system("Pause");
        break;

```

El proceso empieza obteniendo la dirección del archivo “Conteo.txt” con fopen, cuyo modo de apertura es para lectura de un archivo binario con el propósito de recorrer toda la información “rb”, muestra los encabezados de los datos y en un ciclo repetitivo lee el archivo registro a registro, por cada lectura del archivo el contenido es depositado en la variable estructurada “poblacion”, una vez mostrado dicho contenido verifica si no ha llegado al final del archivo para volver a leer el siguiente registro, recuerde que la función feof() devuelve verdadero cuando ya alcanzo el final de archivo. Al

momento de ejecutar el programa y escoger ésta opción notará que devuelve el último registro dos veces, esto se debe a que feof() informa cuando ya se ha pasado el final y encuentra vacío el registro por lo que lo imprime dos veces, para resolver éste inconveniente se considera el número de elementos leídos por fread, es importante destacar que la función devuelve el número de elementos leídos en el archivo, como ya se había indicado el fin de archivo se encuentra vacío por lo tanto devolverá un valor de cero, la propuesta de solución quedaría de la siguiente forma:

```
case 50: pa = fopen("Conteo.txt", "rb");
        printf("\tCIUDAD\tHOMBRES\tMUJERES\n");
        do{ if(fread(&poblacion,sizeof(censo),1,pa))
            printf("\t%s\t%d\t%d\n",poblacion.ciudad,poblacion.TotalH,poblacion.TotalM);
        }while (!feof(pa));
        fclose(pa);
        printf("\n");
        system("Pause");
        break;
```

La tercera opción permite al usuario realizar una búsqueda y mostrar un registro determinado, para el ejemplo se realiza una búsqueda por ciudad de toda la lista almacenada en el archivo, para lograrlo se aplica un recorrido desde el principio hasta el final, por cada registro leído del archivo se comparará el nombre de la ciudad a buscar con el dato correspondiente del registro leído en el archivo:

```
case 51: pa = fopen("Conteo.txt", "rb");
        printf("Ciudad a buscar: ");gets(bc);
        printf("\tCIUDAD\tHOMBRES\tMUJERES\n");
        while (!feof(pa)){
            fread(&poblacion,sizeof(censo),1,pa);
            if(strncmp(poblacion.ciudad,bc)==0)
                printf("\t%s\t%d\t%d\n",poblacion.ciudad,poblacion.TotalH,poblacion.TotalM);
        }
        fclose(pa);
        printf("\n");
        system("Pause");
        break;
```

En el ejemplo la variable "bc" es utilizada por el programa para almacenar el nombre de la ciudad a buscar, considere que éste dato es solicitado al momento de escoger la opción tres que tiene un código ASCII 51 de la respectiva tecla, observe que la búsqueda utiliza un ciclo repetitivo que se realiza mientras no se llegue al final del archivo, por cada ciclo repetitivo se lee el registro y luego lo compara con la función strcmp() que compara dos cadenas de texto, si son iguales se mostraran los datos del registro leído, caso contrario no se mostrará nada.

### **Actividades de refuerzo (AR):**

AR87. Desarrolle un programa que permita registrar la atención médica de varios pacientes, considere que por cada paciente se debe registrar: nombre del doctor(a), nombre del paciente, diagnóstico y tratamiento; es importante considerar que se puede actualizar cualquiera de los registros, los mismos que deben quedar plasmado en el archivo.

## RESUMEN

Se considera a un archivo como la estructura de información almacenada y contenida en cualquier dispositivo de almacenamiento físico que puede ser recuperada y actualizada cuando se lo desee, en sí, los archivos son datos que pueden mantenerse almacenado durante mucho tiempo, estos datos provienen de variables y estructuras organizativas de la memoria RAM, crear o actualizar un archivo no es más que vaciar el contenido de la memoria del computador a un dispositivo de almacenamiento, con la finalidad de hacer uso de ella en cualquier tiempo, existen 2 estructuras de almacenamiento, archivos con estructura binaria y archivos con estructura de texto o plana, los archivos binarios utilizan la estructura de datos para organizar los contenidos o registros, esta organización permite agrupar la información de diferentes tipos de datos en una sola organización física, esta organización se origina en los tipos de datos estructurados estudiados en el capítulo anterior, los archivos de texto son el resultado de vaciar la información sin considerar su estructura organizativa, es por ello que son conocidos como archivos planos, para acceder a un determinado archivo se utiliza función `fopen()` que devuelve la dirección del mismo, la gestión de dicha información puede ser realizada por múltiples funciones que utilizan la dirección provista por `fopen`, los archivos tienen varios modos de apertura que definen las posibilidades de gestionar la información que contienen, así un archivo puede ser abierto para solo lectura, lectura y escritura pero agregando datos al final del archivo y agregado para reemplazo de contenido.