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

ALGORITMOS I

INGENIERÍA DE SISTEMAS

Medellín, Colombia
Derechos Reservados ©2011
Primera edición
2020

Algoritmos I
José Antonio Polo
Facultad de Ingenierías

Comité académico
Jorge Mauricio Sepúlveda Castaño
Decano de la Facultad de Ingenierías
jsepulveda@uniremington.edu.co

David Ernesto González Parra
Director de Educación a Distancia y Virtual
dgonzalez@unireminton.edu.co

Francisco Javier Álvarez Gómez
Coordinador CUR-Virtual
falvarez@uniremington.edu.co

Edición y Montaje
Dirección de Educación a Distancia y Virtual
Equipo de diseño gráfico

www.uniremington.edu.co
virtual@uniremington.edu.co

TABLA DE CONTENIDO

Pág.

PROPÓSITO GENERAL

ALGORITMOS I

Proporcionar a los estudiantes las herramientas básicas para la solución de problemas mediante un proceso de secuencias lógicas.

ALGORITMOS I

a

OBJETIVO GENERAL

Experimentar los conceptos básicos para el desarrollo de algoritmos aplicables a cualquier disciplina, que permitan la generación de la capacidad analítica y creativa en la solución e implementación de problema propuestos.

OBJETIVOS ESPECÍFICOS

- Utilizar las herramientas básicas para la construcción de soluciones lógicas
- Aplicar las estructuras de decisión y repetitivas, fundamentales en la construcción de un buen algoritmo.
- Reconocer las estructuras de registros, archivos, análisis de algoritmos y manejo de hileras de caracteres y las diferentes operaciones que se pueden realizar con cada una de ellas.

UNIDAD 1	UNIDAD 2	UNIDA 3
Introducción a los Algoritmos	Conceptos y Estructuras Básicas de Algoritmos	Manejo de Registros, archivos y Análisis de algoritmo

1 UNIDAD 1 INTRODUCCIÓN A LOS ALGORITMOS

1.1.1 OBJETIVO GENERAL

- Utilizar las herramientas básicas para la construcción de soluciones lógicas.

1.1.2 OBJETIVOS ESPECÍFICOS

- Leer el problema el número de veces que sea necesario hasta entenderlo para representar de forma lógica el problema.
- Identificar las componentes básicas de un algoritmo y la forma de utilizarlos.
- Mostrar paso a paso que las instrucciones planteadas en el algoritmo arrojan el resultado esperado con datos de prueba.
- Asignar valores iniciales a las variables en memoria principal.

1.1.3 PRUEBA INICIAL

- ¿Qué es un algoritmo?
- ¿Qué es la CPU (unidad central de procesamiento) y que elementos la componen?
- ¿Qué función cumple la memoria RAM?
- ¿Qué diferencia hay entre memoria RAM y memoria externas?

1.2 TEMA 1 CLARIFICAR EL PROBLEMA MEDIANTE ACCIONES ALGORÍTMICAS

1.2.1 PROBLEMAS, SOLUCIONES Y PROGRAMAS

Los pasos que se siguen en la construcción de un algoritmo son:

1. Análisis del problema.
2. Diseño de la solución.
3. Construcción del algoritmo.
4. Prueba de escritorio.

El **análisis del problema** consiste en:

Determinar exactamente cuáles son los datos de entrada que se requieren, cuál es la información que se desea producir y cuál es el proceso que se debe efectuar sobre los datos de entrada para producir la información requerida.

Se debe indagar por todas **las situaciones especiales** que se puedan presentar para tenerlas en cuenta en el diseño.

Con base en el análisis se elabora el diseño del algoritmo:

- Se asignan nombres a las variables,
- Se define el tipo de cada una de ellas,
- Se definen las operaciones y subprocesos que hay que efectuar y el método para resolver cada uno de ellos.

Los elementos para la construcción de un algoritmo son:

- Datos,
- Estructuras, e
- Instrucciones.

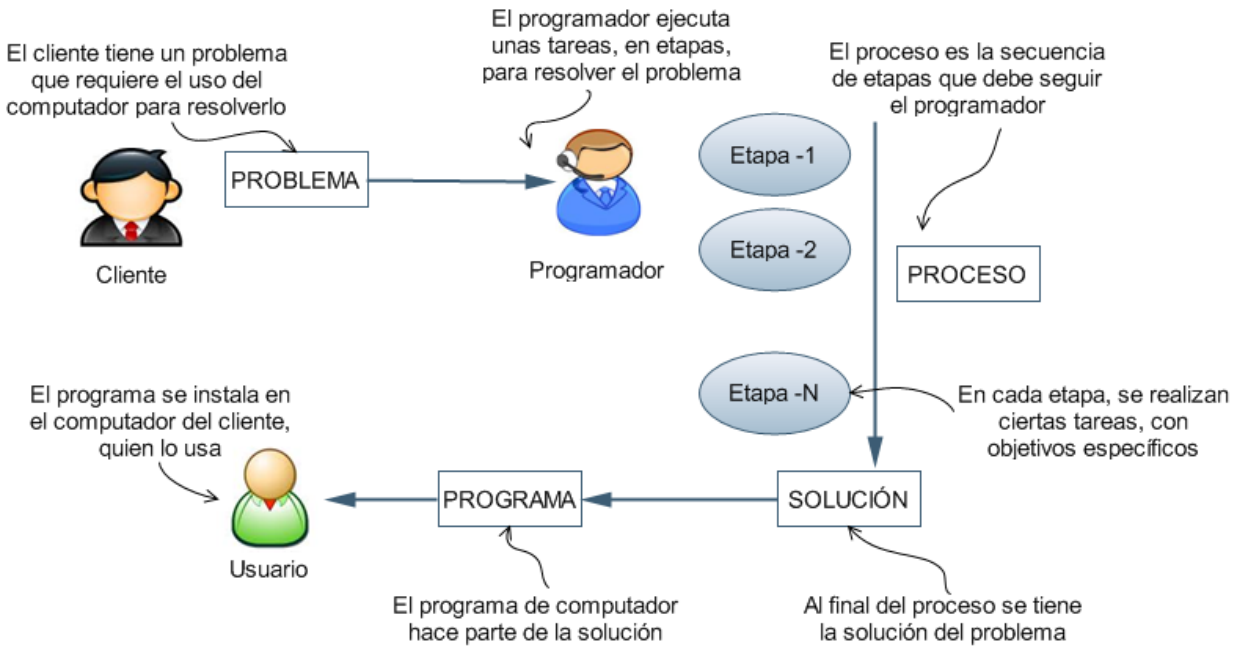
La **prueba de escritorio** consiste en asumir **la posición del computador** y **ejecutar el algoritmo** que se ha elaborado para ver cómo es **su funcionamiento**.
Esta parte es muy importante puesto que permite detectar errores de lógica sin haber hecho aún uso del computador. Aunque no garantiza que el algoritmo está bueno 100%, ayuda mucho en la elaboración de algoritmos correctos.

Habiendo superado los pasos anteriores:

- Se elige **un lenguaje de programación** (algunos de los más utilizados en la actualidad son: Java, Python, C++, PHP, entre otros),
- Se **codifica** el algoritmo en dicho lenguaje, y
- Se **pone en ejecución** en el computador disponible.

1.2.1.1 PASOS PARA LA SOLUCIÓN DE UN PROBLEMA

Veamos la forma gráfica para representar un problema en la siguiente figura:



En la primera sección nos concentramos en la definición del problema, en la segunda, en el proceso de construcción de la solución y, en la tercera, en el contenido y estructura de la solución misma.

Ahora veamos los pasos que debemos seguir para solucionar el problema en un computador:

- **Paso 1:**

Una persona u organización, denominada el **cliente**, tiene un problema y necesita la construcción de un programa para resolverlo. Para esto contacta una empresa de desarrollo de software que pone a su disposición un **programador**.

- **Paso 2:**

El programador sigue un conjunto de etapas, denominadas el **proceso**, para entender el problema del cliente y construir de manera organizada una **solución** de buena calidad, de la cual formará parte un **programa**.

- **Paso 3:**

El programador instala el programa que resuelve el problema en un computador y deja que el **usuario** lo utilice para resolver el problema. Fíjese que no es necesario que **el cliente** y **el usuario** sean **la misma persona**. Piense por ejemplo que el cliente puede ser el gerente de producción de una fábrica y, el usuario, un operario de la misma.

1.2.1.2 REPRESENTACIÓN DE ALGORITMOS

Para representar algoritmos se cuenta con tres formas distintas:

REPRESENTACIÓN DE ALGORITMOS
❖ **DFD (diagrama de flujo de datos):**
Representa las instrucciones del algoritmo utilizando figuras geométricas (el resultado de la representación algorítmica es un diagrama)
❖ **Diagramación rectangular:**
Utiliza rectángulos con algunas formas para representar las instrucciones del algoritmo.
❖ **Seudocódigo o lenguaje natural:**
Representa las instrucciones del algoritmo en el lenguaje nativo utilizando palabras claves para su construcción (es la más utilizada actualmente por su facilidad de uso)

➢ ***Video: " Solución de problemas empleando algoritmos"***

➢ ***" Ver Video":*** https://www.youtube.com/watch?v=KsaO6YMFhqY"

1.2.2 EJERCICIO DE APRENDIZAJE

Para explicar la solución algorítmica se usará el siguiente ejemplo:

Una persona se dirige a una institución bancaria a realizar una transacción. En este problema simple intentemos determinar cuáles son las partes de un algoritmo:

- **Entrada:** en el problema corresponde a la información que entrega el cliente para realizar la transacción (puede ser una consignación, un retiro o una transferencia de fondos)

- **Procesamiento:** de acuerdo con la entrada propuesta por el cliente corresponde al tipo de operación que debe realizar el cajero para efectuar la transacción. Si el cliente va a realizar una consignación debe incrementar el dinero que lleva a su cuenta, en caso de realizar un retiro debe decrementar el valor retirado de su cuenta y en caso de una transferencia debe decrementar su cuenta en el valor transferido y pasarlo a la otra cuenta.

- **Salida:** es el resultado de cualquiera de las operaciones que realiza el cliente con el cajero en el banco bien sea para incrementar o decrementas su cuenta.

1.2.3 TALLER DE ENTRENAMIENTO

Cuando el dueño de un automóvil va a una bomba de gasolina a tanquear el automóvil. Para este problema identificar claramente las partes de un algoritmo.

Para comprar los tiquetes de aviación en una empresa de vuelos se requiere la solicitud del cliente al punto de atención de la compañía. Identificar los pasos de un algoritmo para dar solución al requerimiento del cliente con respecto a la empresa de vuelo

TIPS

Debemos entender bien el problema para dar una solución algorítmica si no se entiende el problema la solución no será la mejor. Pertinente.

1.3 TEMA 2 COMPONENTES DE UN ALGORITMO

- ***Video: "Componentes de un Algoritmo"***
- ***" Ver Video":*** https://www.youtube.com/watch?v=bU7jqrhycoA"

1.3.1 VARIABLES Y CONSTANTES

Es la parte principal al realizar el análisis del algoritmo y consiste en definir cómo se va a realizar **el almacenamiento en la memoria**, la variable cambia su valor durante el procesamiento en el algoritmo y si se requieren valores que nunca cambian su valor se necesitarían **una constante asociada**.

Lo principal en la definición de variables es **el tipo asociado al almacenamiento** y este tiene que ver con los valores que maneja **la realidad asociada** a lo que almacena la variable por ejemplo no es lo mismo almacenar nombre, que edades o salarios, en cualquiera de los casos anteriores el almacenamiento es distinto por el tipo de variable.

1.3.2 ELEMENTOS PARA LA CONSTRUCCIÓN DE UN ALGORITMO

Los elementos con los cuales se construye un algoritmo son **las estructuras lógicas** y **los datos**.

Miremos los datos.

Para efectos de representación de datos en un computador, estos se clasifican en **numéricos** y **no numéricos**.

Los datos numéricos se clasifican en **enteros** y **reales**.

En términos de computación se denomina **tipo**, y se habla entonces de datos de:

- ✓ Tipo entero,
- ✓ Tipo real
- ✓ Tipo no numérico, entre otros.

Cuando se trabajan datos numéricos en un computador es muy importante considerar si el tipo es **entero** o **real**, puesto que, dependiendo de ello, los resultados que se obtienen al efectuar operaciones aritméticas pueden **variar sustancialmente**.

1.3.3 REPRESENTACIÓN DE DATOS EN UN COMPUTADOR

- La unidad básica de representación de datos es el **bit**.
- La siguiente unidad se denomina **byte**.
- La siguiente unidad de representación es el **campo**.
- La siguiente unidad de representación es el **registro**.
- La siguiente unidad de representación es el **archivo**.
- La siguiente unidad de representación es la **base de datos**.

1.3.4 OPERADORES ARITMÉTICOS, RELACIONALES Y LÓGICOS

En la lógica es preciso tener en claro el uso de los distintos operadores asociados a las instrucciones generadas por **la asignación** y **el procesamiento**. El uso de:

❖ **Los operadores aritméticos:**

En las expresiones son requeridos por el procesamiento de acuerdo con los cálculos asociados a las variables del problema,

❖ **Los operadores relacionales:**

Son utilizados cuando se necesita evaluar condiciones, y

❖ **Los operadores lógicos:**

Cuando se realizan condiciones compuestas asociadas a las preguntas en todos los casos se debe usar la precedencia de operadores como uno de los aspectos más importantes en la ejecución de instrucciones.

Nota: Para el manejo de precedencia de operadores se tiene en cuenta que la precedencia más alta se tiene con los paréntesis.

1.3.5 OPERADORES ARITMÉTICOS

Sirven para efectuar cálculos aritméticos. Ellos son:

SÍMBOLO	OPERACIÓN
$+$	Suma
$-$	Resta
$*$	Multiplicación
$/$ $(\boldsymbol{slash})$	División
\ $(\boldsymbol{backslash})$	División entera (Toma solo la parte entera de la división)
$\%$	Módulo (Toma el residuo de una división)
$**$, ^	Ambos potenciación y radicación (Se debe expresar la raíz como una potencia fraccionarla)

[1]Prioridades de los operadores:

Símbolo	Prioridad
$+, -$	Tienen la misma prioridad
$*, /, \backslash, \%$	Tienen la misma prioridad, pero mayor que la suma y la resta
$**$, ^	Tienen mayor prioridad que todos los anteriores

Si dos o más operadores consecutivos tienen **la misma prioridad**, las operaciones se ejecutarán en las instrucciones de **izquierda a derecha**.

Ejemplo: Si se tiene la expresión:
$A \ ** \ 2/5 \ * \ B \ - 5$ y los valores almacenados en A y B son 5 y 20 respectivamente, la evaluación de acuerdo con el orden de prioridad será:
$5 \ ** \ 2 \ = \ 25$
$25 \ / \ 5 \ * \ 20 \ = \ 100$
$100 - 5 \ = \ 95$

[1] Oviedo Efrain

Si se tiene una expresión con **dos o más potencias consecutivas** estas se realizan de **derecha a izquierda**. Por ejemplo:

$4 + 10 / 5 * 2 \wedge 2 \wedge 3$

De acuerdo con el orden de prioridad, el resultado sería:

$2 \wedge 3 = 8$

$2 \wedge 8 = 256$

$10 / 5 * 256 = 512$

$4 + 512 = 516$

Si se requiere que una o más operaciones se realicen primero que otras, entonces estas se encierran entre paréntesis y dentro de estos se conserva la jerarquía de los operadores.

Ejemplo:

La operación: $\frac{A+B}{C-A} + 20$ debe representarse como: $(A + B) / (C - A) + 20$

La operación: se representa como: $a/b - c/ (a\wedge (d + e) * 20)$

1.3.6 OPERADORES LÓGICOS RELACIONALES

Sirven para hallar el valor de verdad de una proposición Lógica simple. Se define una proposición lógica simple (PLS) como la comparación entre contenido de un campo variable y un valor constante o la comparación entre los contenidos de dos campos variables. Ellos son:

Símbolo	Función
==	Igual
<>, ! =	Diferente
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

Nota aclaratoria: Estos operadores pueden variar dependiendo el Lenguaje de Programación que se utilice. Para los procesos de comparación es bueno anotar que sólo son válidos si los campos variables a comparar han sido previamente asignados.

Ejemplos:

$$\frac{a}{b} - \frac{c}{d+e}$$

$$a \quad .20$$

CARLOS >= 6

Esto es una PLS, en la cual se compara el contenido del campo variable CARLOS con el valor numérico 6, esta puede ser verdadera o falsa dependiendo del contenido del campo CARLOS.

Nombre <> "*"

B == C

SALARIO <= 98700

EDAD > 100

1.3.7 OPERADORES LÓGICOS BOOLEANOS

Sirven para hallar el valor de verdad de una proposición lógica compuesta (PLC), entendiéndola como la conexión de dos o más PLS. En términos de Lógica matemática son los llamados conectivos lógicos. Ellos son:

Símbolos Matemáticos	Símbolos utilizados en programación	Nombre	Valor de Verdad o Definición
^	&&	Conjunción (AND), se lee Y	Se define como verdadera cuando las PLS que conectan son todas verdaderas
V	||	Disyunción (OR), se lee O	Se define como falsa cuando las PLS que conectan son todas falsas.
~	!	Negación (NOT), se lee NO	No es un conectivo lógico y su función es alterar el valor de verdad de las proposiciones lógicas.

1.3.8 PRIORIDAD DE LOS OPERADORES LÓGICOS BOOLEANOS

La prioridad de estos se clasifica de la siguiente manera:

1. Negación !
2. Conjunción **&&**
3. Disyunción **||**

Las variables lógicas son variables que sólo pueden tomar dos valores: **verdadero** o **falso**.

En general, una variable lógica, en el ámbito de los computadores, es una variable de un solo bit, el cual puede ser **0** o **1**. Por convención se ha adoptado que:

- El **0** representa falso, y
- El **1** verdadero.

Se establece por convención que, para formar una PLC, las **PLS** deben estar encerradas entre **paréntesis** y para hallar el **valor de verdad de una PLC** primero se evalúa **el valor de verdad de cada PLS por separado** y el valor de verdad de **la PLC** estará en función del **operador lógico booleano** usado para la conexión.

1.3.9 EJERCICIO DE APRENDIZAJE

Sean: A = 3; B = 5; C = 1. Hallar el valor de verdad de la siguiente PLC

(A < B) ^ (B < C)

V		F	Se halló el valor de verdad de cada PLS
V	^	F	Se aplicó la definición del operador lógico booleano conjunción
	⇩		
	F		La PLC es falsa

1.3.10 TALLER DE ENTRENAMIENTO

1. Dada la siguiente definición de variables con su respectivo tipo y contenido

Numéricos enteros	Numéricos reales
a = 4	y = 3.5
b = 7	x = 2.0
c = 3	z = 5.0
d = 2	w = 1.5

Determine el resultado de evaluar cada una de las siguientes expresiones:

a) a * b / 2 + 1
b) c / y ^ 2
c) a / ((b + c) / (d + 1) * (a + b) – b) ^ b ^ a + z
d) z / x + b * w * (c – b) / a

TIPS

Reconocer la prioridad de los operadores aritméticos y los operadores lógicos es fundamental en el desarrollo de las diferentes operaciones.

1.4 TEMA 3 PRUEBAS DE ESCRITORIO

Son **las simulaciones del comportamiento de un algoritmo** y esta permite determinar **la validez** de este. Para realizar una prueba de escritorio se debe generar una tabla con tantas columnas como variables tenga el algoritmo seguir las instrucciones y asignar los valores correspondientes en cada una de ellas, permitiendo así detectar errores, omisiones o darle una mejora al algoritmo. Los casos de prueba facilitan la comprobación de diferentes situaciones que puedan ocurrir en el algoritmo. Para esto se muestra un ejemplo de una prueba de escritorio a un algoritmo en el siguiente video:

- ***Video: "Ejemplo Prueba de Escritorio - Curso Fundamentos de Programación"***
- ***" Ver Video":*** https://www.youtube.com/watch?v=-5_1mMoIaxo

1.4.1 EJERCICIO DE APRENDIZAJE

Una prueba de escritorio va mostrando paso a paso el estado actual de una Variable, supongamos que se tiene asignado la siguiente información a las siguientes variables:

A=10

B=5

C=2

A= A+B

B= 20+A

C= A+B

Realizándole una prueba de escritorio a las anteriores instrucciones se obtiene:

A	B	C	Variables Involucradas en el proceso
10	5	2	Valores Iniciales para cada una de las variables
A= A+B A= 10+5 A=15			La variable A está sufriendo modificaciones al actualizarse arroja un nuevo resultado
	B= 20+A B= 20+15 B= 35		La variable B está sufriendo modificaciones al actualizarse arroja un nuevo resultado Esto significa que B toma el valor

			actualizado que tiene A, ya que el valor actualizado de A esta antes que el de la instrucción de B.
		C= A+B C= 15+35 C= 50	La variable C está sufriendo modificaciones al actualizarse arroja un nuevo resultado Esto significa que C toma el valor actualizado que tiene A y el que tiene B, ya que el valor actualizado de A y B están antes que el de la instrucción de C.

1.4.2 TALLER DE ENTRENAMIENTO

Se tiene un recipiente con 24 litros de leche el cual se desea repartir partes iguales entre tres personales, para ello se cuenta con tres recipientes auxiliares con las siguientes medidas: 1 recipiente de 13 litros, un recipiente de 11 litros y recipiente de 5 litros, ninguno de los recipientes cuenta con una medida donde se especifique1 litro, 3 Litros… se sabe que hay 13, 11, 5 litros cuando los recipientes están llenos.

Realice una prueba de escritorio donde se muestre paso a paso como tendrían que hacer para repartirse los 24 litro de leche en partes iguales.

Nota: Se deben involucrar los cuatro recipientes para dicha repartición.

TIPS

Las pruebas de escritorio de escritorio son esenciales para ser un buen

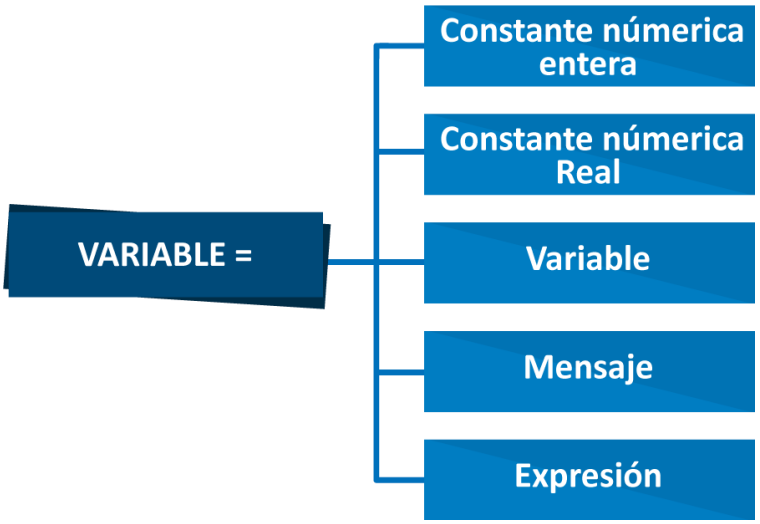
PROGRAMADOR

1.5 TEMA 4 VALORES, ASIGNACIONES A VARIABLES

- ***Video: "Asignación de variables"***
- ***" Ver Video":*** https://www.youtube.com/watch?v=ld5TzeMvm_E "

La instrucción de asignación consiste en llevar algún dato a una posición de memoria, la cual está identificada con el nombre de una variable.

La forma general de una instrucción de asignación es:



1.5.1 EJERCICIO DE APRENDIZAJE

1. a = 316
2. b = 3.14
3. c = "hola mundo"

4. d = a
5. e = a + b * a

En los ejemplos 1 y 2, a las variables a y b les estamos asignando una constante numérica:

- ✓ **Entera** en el primer ejemplo,
- ✓ **Real** en el segundo.

- En el ejemplo 3, a la variable **c** le estamos asignando **un mensaje**.
- En el ejemplo 4, a la variable **d** le estamos asignando el contenido de **otra variable**.
- En el ejemplo 5, a la variable **e** le estamos asignando el resultado de **evaluar una expresión**.

1.5.2 TALLER DE ENTRENAMIENTO

1. Si el valor de a = 4, b = 5, c = 1, l = Verdadero (TRUE); muestre cuales son los valores impresos en el siguiente algoritmo:

```
1.  CLASE OperadorYExpresion
2.      METODO PRINCIPAL ()
3.          VARIABLES: a, b, c, x, y, z (NUMÉRICAS)
4.                     l: (BOOLEANAS)
5.          a = 4
6.          b = 5
7.          c = 1
8.          l = TRUE
9.          x = b * a – b ^ 2 / 4 * c
10.         y = a * b / 3 ^ 2
11.         z = (((b + c) / 2 * a + 10) * 3 * b) – 6
12.         IMPRIMA (x, y, z)
13.     FIN(Método)
14. FIN(Clase)
```

2. Usando los valores de a, b, c y L del punto anterior, calcule el valor almacenado en las siguientes variables

   x = (b <= 100) && !(a>5)
   y = (b == 5) || (c == 30) && ! (a==10)
   z = ((b + 20) > (c – 1)) || ((a + 5) <= 50)

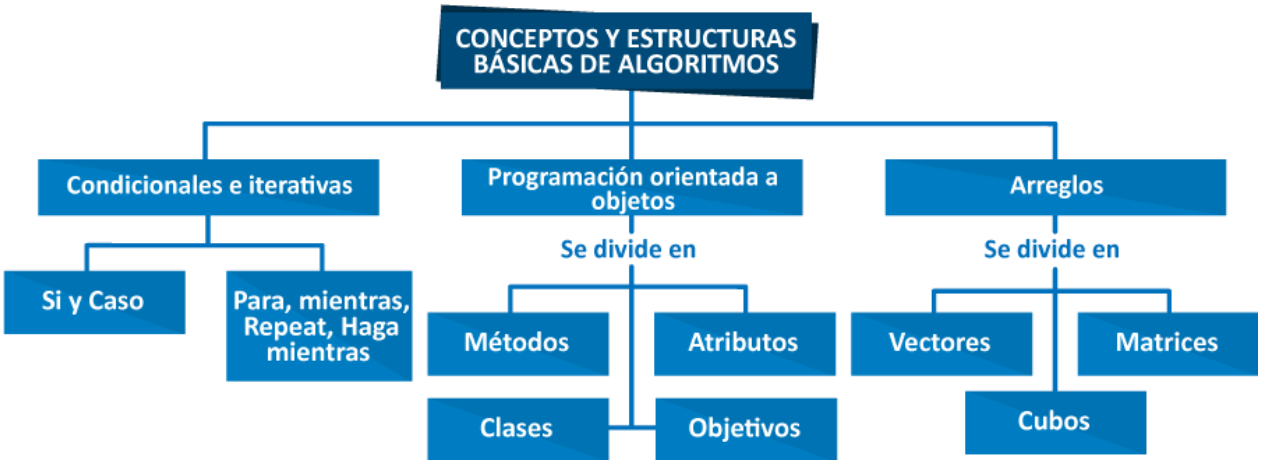
3. Elaborar un algoritmo que imprima el siguiente mensaje: “Que buenos son los algoritmos y la computación”.
4. ¿Cuáles son las instrucciones correspondientes a la estructura ciclo?
5. ¿Cómo se diferencia la escritura de mensajes de la escritura de datos de una instrucción de lectura?
6. Detecte y describa los errores que hay en el siguiente algoritmo:

```
1.    CLASE Error
2.        METODO PRINCIPAL ()
3.            VARIALBES: x, y (NUMÉRICAS)
4.            IMPRIMA (“Digite los datos para x y y:”)
5.            IMPRIMA (“dato x:”, x, “dato y:”, y)
6.            LEA (x, y)
7.            IMPRIMA (“Hasta pronto”) FIN(Método)
8.            FIN(Clase)
```

TIPS

Asignar información a las variables es importantísimo porque son los valores con los cuales se va a trabajar.

2 UNIDAD 2 CONCEPTOS Y ESTRUCTURAS BÁSICAS DE ALGORITMOS



2.1.1 OBJETIVO GENERAL

- Aplicar las estructuras de decisión y repetitivas, fundamentales en la construcción de un buen algoritmo.

2.1.2 OBJETIVOS ESPECIFICOS

- Reconocer cuando usar una estructura de condicionales o una estructura repetitiva dentro de un algoritmo.
- Reconocer de qué manera se puede reutilizar código dentro de un algoritmo
- Utilizar variables de tipo arreglo para la solución de un problema que requiera almacenar datos en memoria de forma consecutiva.

2.1.3 PRUEBA INICIAL

- ¿En qué consiste la estructura de decisión?
- ¿Cuál es la forma general de la instrucción SÍ?
- ¿En qué consiste la instrucción CASOS?

- ¿Qué es un ciclo?
- ¿Qué es un subprograma?
- ¿Qué es un vector?

2.2 TEMA 1 ESTRUCTURAS CONDICIONALES E ITERATIVAS

➢ ***Video: " Algoritmos estructuras condicionales - repetitivas"***

➢ ***" Ver Video":*** https://www.youtube.com/watch?v=iyIyeXNQyGk "

Es un mecanismo que permite **verificar el valor lógico de una pregunta**:

- Si es **verdadera** se ejecuta un bloque de instrucciones, y
- Si la pregunta es **falsa** se ejecuta otro bloque de instrucciones en caso de que exista.

Estas instrucciones se pueden componer de **una instrucción simple** o de **una instrucción compuesta.**

- ✓ En **la instrucción simple** solo se usa la opción verdadera de la condición, mientras que
- ✓ En **la instrucción compuesta** se utiliza su componente de lo contrario.

Estas estructuras se pueden implementar de forma anidada esto quiere decir que dentro de una condición se puede tener una o más condiciones

2.2.1 ESTRUCTURA DE DECISIONES SIMPLES Y COMPUESTAS

La estructura de decisión permite **instruir al computador** para que ejecute ciertas acciones (instrucciones) según **alguna condición**.

La forma general de la instrucción SI es:

SI (condición)
Instrucciones que se ejecutan cuando la condición sea verdadera.
SINO
Instrucciones que se ejecutan cuando la condición es fala.

```
FINSI
```

Ejemplo:

Elaborar un algoritmo que lea el salario actual de un empleado y que calcule e imprima el nuevo salario de acuerdo a la siguiente condición: si el salario es menor que 1000 pesos, aumentar el 10%; sino no hacer el aumento.

Análisis:

Datos de entrada: salario actual (SALACT).

Cálculos: determinar el aumento según la condición planteada.

Datos de salida: aumento (AU), nuevo salario (NUESAL).

Nuestro algoritmo es:

```
1.      CLASE AumentoCondicional
2.        METODO PRINCIPAL ()
3.              VARIABLES: SALACT, AU, NUESAL (NUMÉRICAS)
4.                      LEA (SALACT)
5.                      SI (SALACT < 1000)
6.                              AU = SALACT * 0.1
7.                      SINO
8.                              AU = 0
9.                      FINSI
10.                     NUESAL = SALACT + AU
11.                     IMPRIMA (“Nuevo salario:”, NUESAL)
12.       FIN(Método)
13.     FIN(Clase)
```

- En la instrucción 2, se definen las variables con las cuales vamos a trabajar.

- En la instrucción 4, se lee el salario actual.
- En la instrucción 5, se compara el salario leído con el dato de referencia planteado en el enunciado.
- Si la condición de la instrucción 5 es verdadera, se ejecuta la instrucción 6; sino se efectuará la instrucción 8.
- En la instrucción 6, se determina el aumento, el cual es el diez por ciento del salario actual, mientras que en la instrucción 8 se asigna cero al aumento.
- La instrucción 9, delimita el alcance de la instrucción SI.

- En la instrucción 10, se calcula el nuevo salario, y en la instrucción 11 se imprime el nuevo salario.

El anterior algoritmo se puede escribir sin utilizar el componente SINO, el cual, como habíamos dicho, es opcional.

Veamos nuestro nuevo algoritmo:

```
1.      CLASE AumentoCondicional (2)
2.        METODO PRINCIPAL ()
3.                VARIABLES: SALACT, AU, NUESAL (NUMÉRICAS)
4.                          LEA (SALACT)
5.                          AU = 0
6.                          SI (SALACT < 1000)
7.                                    AU = SALACT * 0.1
8.                          FINSI
9.                          NUESAL = SALACT + AU
10.                         IMPRIMA (“Nuevo salario: ”, NUESAL)
11.       FIN(Método)
12.     FIN(Clase)
```

La diferencia de este segundo algoritmo con el primero es que al aumento inicialmente se le asigna el valor de cero en la instrucción 5.

Cuando se compara el salario actual con el valor de referencia (1000), se modificará el aumento sólo si el salario actual es menor que el valor de referencia; en caso contrario el aumento permanecerá en cero.

En la práctica se presentan hechos en los cuales es necesario controlar situaciones dentro de situaciones ya controladas, es decir, comprobar condiciones dentro de condiciones, o comprobar varias condiciones a la vez. Estos acontecimientos implican el uso de la instrucción SI de una forma más compleja.

Ejemplo:

Elabore un algoritmo que lea tres datos numéricos y que los imprima ordenados ascendentemente.

Análisis:

Datos de entrada: tres datos numéricos (a, b, c).

Cálculos: determinar el menor de los tres datos para imprimirlo de primero, luego determinar el menor de los dos restantes para imprimirlo de segundo y luego imprimir el tercer dato.

Datos de salida: los mismos tres datos de entrada en orden ascendente.

Una primera forma de escribir este algoritmo es siendo exhaustivos en la comparación de los datos.

Realmente, las diferentes situaciones que se pueden presentar para escribir los tres datos son:

- La primera, cuando a es menor que b y b es menor que c.
- La segunda, cuando a es menor que c y c es menor que b.
- La tercera, cuando b es menor que a y a es menor que c,
- Y así sucesivamente.

A cada situación le corresponde una relación de orden diferente.

1. a, b, c
2. a, c, b

3. b, a, c
4. b, c, a
5. c, a, b
6. c, b, a

Un algoritmo para efectuar esta tarea es:

```
1.      CLASE Ordenar3Datos (1)
2.        METODO PRINCIPAL ()
3.              VARIABLES: a, b, c (NUMÉRICAS)
4.                      LEA (a, b, c)
5.                      SI (a < b) && (b < c)
6.                              IMPRIMA (a, b, c)
7.                      FINSI
8.                      SI (a < c) && (c < b)
9.                              IMPRIMA (a, c, b)
10.                     FINSI
11.                     SI (b < a) && (a < c)
12.                             IMPRIMA (b, a, c)
13.                     FINSI
14.                     SI (b < c) && (c < a)
15.                             IMPRIMA (b, c, a)
16.                     FINSI
17.                     SI (c < a) && (a < b)
18.                             IMPRIMA (c, a, b)
19.                     FINSI
20.                     SI (c < b) && (b < a)
21.                             IMPRIMA (c, b, a)
22.                     FINSI
```

```
23.        FIN(Método)
24.      FIN(Clase)
```

Las instrucciones 5 a 7 consideran la primera situación; las instrucciones 8 a 10 consideran la segunda situación; las instrucciones 11 a 13 consideran la tercera situación; las instrucciones 14 a 16 consideran la cuarta situación; las instrucciones 17 a 19 consideran la quinta situación; y las instrucciones 20 a 22 consideran la sexta situación.

Este algoritmo tiene el inconveniente de que cuando una situación sea verdadera, continúa preguntando por las demás, lo cual genera ineficiencia. Para evitar esta ineficiencia utilizamos la parte opcional SINO.

Veamos cómo queda nuestro algoritmo:

```
1.      CLASE Ordenar3Datos (2)
2.        METODO PRINCIPAL ()
3.              VARIABLES: a, b, c (NUMÉRICAS)
4.                      LEA (a, b, c)
5.                      SI (a < b) && (b < c)
6.                              IMPRIMA (a, b, c)
7.                      SINO
8.                              SI (a < c) && (c < b)
9.                                      IMPRIMA (a, c, b)
10.                             SINO
11.                                     SI (b < a) && (a < c)
12.                                             IMPRIMA (b, a, c)
13.                                     SINO
14.                                             SI (b < c) && (c < a)
15.                                                     IMPRIMA (b, c, a)
16.                                             SINO
17.                                                     SI (c < a) && (a < b)
18.                                                             IMPRIMA (c, a, b)
```

```
19.                                         SINO
20.                                             IMPRIMA (c, b, a)
21.                                         FINSI
22.                                     FINSI
23.                                 FINSI
24.                             FINSI
25.                         FINSI
26.                 FIN(Método)
27.         FIN(Clase)
```

De esta manera, cuando encuentre que una condición (situación) es verdadera, procede a imprimir los datos en forma ordenada y no sigue preguntando por las demás condiciones.

Una tercera forma en que podemos elaborar el algoritmo es la siguiente:

Comparamos $\boldsymbol{a}$ con $\boldsymbol{b}$.

Pueden suceder dos cosas:

- Una, que a sea menor que b, y
- Dos, que b sea menor que a.

Si $\boldsymbol{a}$ es menor que $\boldsymbol{b}$, implica que habrá que escribir el dato **a** antes que el dato **b**; por lo tanto, las posibles formas de escribir los tres datos son:

$$a, b, c$$

$$a, c, b$$

$$c, a, b$$

Si $\boldsymbol{b}$ es menor que $\boldsymbol{c}$, imprimimos la primera posibilidad: a, b, c; de lo contrario, debemos comparar $\boldsymbol{a}$ con $\boldsymbol{c}$.

Si $\boldsymbol{a}$ es menor que $\boldsymbol{c}$, imprimimos la segunda posibilidad: a, c, b; en caso contrario, imprimimos la tercera posibilidad: c, a, b.

Si $\boldsymbol{b}$ es menor que $\boldsymbol{a}$ implica que habrá que escribir el dato $\boldsymbol{b}$ antes que el dato $\boldsymbol{a}$; por lo tanto, las posibles formas de escribir los tres datos son:

$$b, a, c$$

$$b, c, a$$

$$c, b, a$$

Si $\boldsymbol{a}$ es menor que $\boldsymbol{c}$, imprimimos la primera posibilidad: b, a, c; de lo contrario, debemos comparar $\boldsymbol{b}$ con $\boldsymbol{c}$.

Si $\boldsymbol{b}$ es menor que $\boldsymbol{c}$, imprimimos la segunda probabilidad: b, c, a; si no, imprimimos la tercera probabilidad: c, b, a.

Con base en el anterior análisis nuestro algoritmo queda:

```
1.      CLASE Ordenar3Datos (3)
2.        METODO PRINCIPAL ()
3.              VARIABLES: a, b, c (NUMÉRICAS)
4.                      LEA (a, b, c)
5.                      SI (a < b)
6.                              SI (b < c)
7.                                      IMPRIMA (a, b, c)
8.                              SINO                            //c es menor que b
9.                                      SI (a < c)
10.                                             IMPRIMA (a, c, b)
11.                                     SINO                    //c es menor que a
12.                                             IMPRIMA (c, a, b)
13.                                     FINSI
14.                             FINSI
15.                     SINO                                    //b es menor que a
16.                             SI (a < c)
17.                                     IMPRIMA (b, a, c)
18.                             SINO                            //c es menor que a
19.                                     SI (b < c)
20.                                             IMPRIMA (b, c, a)
```

```
21.                                SINO                          //c es menor que b
22.                                      IMPRIMA (c, b, a)
23.                                FINSI
24.                          FINSI
25.                    FINSI
26.        FIN(Método)
27.     FIN(Clase)
```

2.2.2 ESTRUCTURA CASO O SELECTOR MÚLTIPLE

El selector múltiple sirve para **reemplazar una serie lógica**. Su estructura corresponde a la de un bloque de decisión múltiple, es decir, ofrece más de dos caminos a seguir simultáneamente. Para usar un selector múltiple se debe considerar:

- La presencia de una variable que contenga más de dos valores que sean enteros $(1, 2, 3, \ldots N)$ y dependiendo de ese valor se ejecute ciertas instrucciones según el camino lógico determinado.

No tiene sentido usar un selector para una variable como por ejemplo sexo (1; hombre, 2: Mujer), en este caso es más óptimo y eficiente **un bloque de decisión**, pero para una variable como programa (1: Sistemas, 2: Electrónica. 3: Secretariado, 4: Gestión Administrativa, 5: Contaduría) se debe usar **un selector**.

- EI selector múltiple se puede usar cuantas veces se requiera.
- En el selector se debe colocar todos los valores de la variable.
- El selector tiene la siguiente estructura:

```
CASOS
    CASO (VARIABLE==1)
            Grupo de instrucciones a ejecutar cuando la variable sea igual a 1
    SALTO
    CASO (VARIABLE==2)
            Grupo de instrucciones a ejecutar cuando la variable sea igual a 2
    SALTO
    CASO (VARIABLE==3)
            Grupo de instrucciones a ejecutar cuando la variable sea igual a 3
```

```
        SALTO
        ¦
        ¦
        CASO (VARIABLE==N)
                Grupo de instrucciones a ejecutar cuando la variable sea igual a N
        SALTO
        OTRO_CASO
Grupo de instrucciones a ejecutar cuando la variable sea diferente a las anteriores
        SALTO
FINCASOS
```

Después de ejecutar las instrucciones que se encuentran dentro de un caso específico la expresión SALTO nos remitirá hasta el fin de los casos.

Ejemplo:

Elaborar un algoritmo que lea el nombre de una persona y su estado civil.

El estado civil está codificado con un dígito con los siguientes significados:

1: Soltero

2: Casado

3: Separado

4: Viudo

5: Unión libre

El algoritmo debe imprimir el nombre leído y la descripción correspondiente al estado civil.

Análisis:

Datos de entrada: nombre (NOM), estado civil (EC).

Cálculos: Comparar el estado civil según el código establecido e imprimir la descripción correspondiente.

Datos de salida: nombre (NOM), estado civil (EC).

```
1.      CLASE EstadoCivil (1)
2.        METODO PRINCIPAL ()
3.              VARIABLES: NOM (CARACTER)
4.                         EC (NUMÉRICA)
5.                    LEA (NOM, EC)
6.                    CASOS
7.                          CASO (EC == 1)
8.                                IMPRIMA (NOM, “Soltero”)
9.                          SALTO
10.                         CASO (EC == 2)
11.                               IMPRIMA (NOM, “Casado”)
12.                         SALTO
13.                         CASO (EC == 3)
14.                               IMPRIMA (NOM, “Separado”)
15.                         SALTO
16.                         CASO (EC == 4)
17.                               IMPRIMA (NOM, “Viudo”)
18.                         SALTO
19.                         CASO (EC == 5)
20.                               IMPRIMA (NOM, “Unión libre)
21.                         SALTO
22.                         OTRO_CASO
23.                               IMPRIMA (EC, “Estado civil no válido”)
24.                         SALTO
25.                   FINCASOS
26.             FIN(Método)
27.       FIN(Clase)
```

Dentro de la instrucción CASOS cuando escribimos CASO (EC == 1): la máquina compara EC con 1, si EC es igual a 1 ejecuta las instrucciones correspondientes a ese caso.

Un algoritmo equivalente al anterior, utilizando la instrucción SI, es el siguiente:

```
1.      CLASE EstadoCivil (2)
2.        METODO PRINCIPAL()
3.              VARIABLES: NOM (CARACTER)
4.                         EC (NUMÉRICA)
5.                    LEA (NOM, EC)
6.                    SI (EC == 1)
7.                          IMPRIMA (NOM, “Soltero”)
8.                    SINO
9.                          SI (EC == 2)
10.                               IMPRIMA (NOM, “Casado”)
11.                         SINO
12.                               SI (EC == 3)
13.                                     IMPRIMA (NOM, “Separado”)
14.                               SINO
15.                                     SI (EC == 4)
16.                                           IMPRIMA (NOM, “Viudo”)
17.                                     SINO
18.                                           SI (EC == 5)
19.                                                 IMPRIMA (NOM, “Unión libre)
20.                                           SINO
21.                                           IMPRIMA (EC, “Estado civil no válido”)
22.                                           FINSI
23.                                     FINSI
24.                               FINISI
25.                         FINSI
26.                   FINSI
27.             FIN(Método)
28.       FIN(Clase)
```

Un punto importante que se debe considerar en este sitio es: ¿cuándo utilizar la instrucción SI, y cuándo utilizar la instrucción CASOS?

La respuesta es sencilla:

- Cuando el resultado de una comparación sólo da dos alternativas, se utiliza la instrucción SI.
- Cuando el resultado de una comparación da más de dos alternativas, se utiliza la instrucción CASOS.

2.2.3 ESTRUCTURAS REPETITIVAS

Es un mecanismo que **permite repetir procesos lógicos** más de una vez, dando la posibilidad de solucionar problemas de más **alta complejidad**. Funciona con una pregunta asociada a una variable de control que:

- Si es **verdadera** ejecuta el bloque de instrucción, y
- En caso de ser **falsa** no se ejecuta el proceso repetitivo.

Las instrucciones repetitivas que se van a desarrollar son:

- El ciclo para…
- El mientras que…, y
- Haga mientras que…

Este tipo de mecanismos de repetición dan la posibilidad de usarse de manera anidada ósea que un ciclo puede estar dentro de otro ciclo, generando así soluciones aún más complejas dentro de la lógica de programación.

2.2.3.1 CICLO MIENTRAS QUE

Es una instrucción que permite que una acción se repita más de una vez, donde la condición que controla el ciclo puede ser una expresión relacional o una expresión lógica.

La forma general de la instrucción MIENTRAS QUE es:

```
MQ (condición)
        Instrucciones que se ejecutan mientras que la condición sea verdadera
FINMQ
```

La condición puede ser una expresión relacional o una expresión lógica. En la condición, por lo general, se evalúa el valor de una variable. Esta variable, que se debe modificar en cada iteración, es la variable controladora del ciclo.

2.2.3.1.1 Esquema Cualitativo

Son estructuras repetitivas que se pueden utilizar cuando no se conoce el número de iteraciones a ejecutar ejemplo cuando se hace una encuesta, el número de datos totales a precisar no es concreto, en este caso es más precedente usar un ciclo "mientras que", que un ciclo “para”, otros ejemplos serian calcular los salarios en una empresa que cambia constantemente el número de trabajadores, o las definitivas de una escuela en donde se pueden retirar alumnos o ingresar alumnos extemporáneamente; estos ciclos tienen las siguientes características:

CARACTERÍSTICAS
• El encabezamiento está formado por la frase MIENTRAS_QUE (o de forma abreviada: MQ), seguida de una proposición lógica simple o compuesta encerrada entre paréntesis. Esta proposición deberá estar integrada al menos por un campo variable que posibilite la alteración de su contenido dentro del ciclo, de manera que el proceso repetitivo se interrumpa en un momento predeterminado.
• El ciclo se ejecuta, si y Sólo si, la proposición lógica es verdadera, cuando se vuelva falsa, se sale automáticamente del ciclo.
• El control de las iteraciones lo tiene el usuario, es decir, el usuario determina cuando quiere interrumpir la ejecución del ciclo puesto que bastara dar un valor tal que la expresión lógica se vuelva falsa.

La proposición Lógica se puede formar de dos maneras:

PROPOSICIÓN LÓGICA

- Usando una variable adicional a las variables del diagrama la cual puede ser un switche o una variable llamada control.
- Usando una de las variables que contienen la información a procesar (nombre, edad, sexo, estado civil, entre otros), en este caso se dice que el ciclo está controlado por un campo centinela.

En ambos casos (usando un control, un switche o un centinela) se debe hacer lo siguiente:

- **Leer antes del encabezamiento del ciclo el campo control**, el switche o el campo centinela teniendo en cuenta que el valor asignado sea tal que vuelva verdadera la proposición Lógica si se quiere entrar al ciclo.

- **Hacer una segunda lectura del campo** (control, switche o centinela) dentro del ciclo y antes de llegar a la Línea de cierre del mismo teniendo en cuenta que como en ese punto se retorna al encabezamiento del ciclo, si se quiere iterar nuevamente el valor debe ser tal que la proposición Lógica se vuelva verdadera; si se quiere salir del ciclo se asignara un valor que la vuelva falsa, en este sentido es que se dice que el usuario tiene el control del ciclo y del número de iteraciones.

Ejemplo: Se está realizando una encuesta en la ciudad (no se conoce el número de encuestados) se va elaborando un archivo de datos y por cada registro tenemos: Edad, Estado civil (1: Soltero, 2: Casado), ingresos mensuales y conforme con el gobierno (1: si, 2: no).

Calcular e imprimir:

a. Qué porcentaje de personas están de acuerdo con el gobierno.

b. Promedio de edad de las personas inconformes con el gobierno.

c. Cuantas personas conformes con el gobierno, ganan menos de $150.00 I mensuales y son casadas.

d. Promedio de ingresos de todas las personas.

Solución: Necesitamos variables para almacenar los datos de entrada:

- ED: Para la edad.

- EC: Para el estado civil.

- ING: Para el ingreso mensual.
- CONFGOB: Conforme con el gobierno.
- PORCONF: Porcentaje de personas de acuerdo con el gobierno; para este cálculo se necesita contar a todas las personas encuestadas, porque no se conoce cuantas veces se va a procesar la información, si fuera un ciclo para, no habría la necesidad de este conteo porque el total de iteraciones está determinado por el registro identificador N, para este conteo se usará el contador llamado CONTOT (Contador total de personas encuestadas), adicionalmente se necesita contar a las personas que están de acuerdo con el gobierno y se hará con el campo CONCONF.
- PORMIN: Promedio de ingresos de todas las personas. Se necesita acumular los ingresos de todos y se hará en el campo acumulador llamado ACUING.
- PROMED: Promedio de edad de las personas en contra del gobierno. Se necesita acumular las edades de estas personas y para ello se usará acumulador ACUMEDAD y contar las personas inconformes y se hará con el contador CNCONF.
- CONTCAS: Contador de personas conformes con el gobierno que ganan menos de $150.000 y son casadas.

Solución con una variable adicional llamada control:

```
1.  CLASE MQCualitativo
2.     METODO PRINCIPAL ()
3.           VARIABLES: ED, EC, ING, CONFGOB, CONTOT, CONCONF,
4.                      CNOCONF, CONTCAS, ACUMING, ACUMEDAD, PORCONF,
5.                      PROMIN, PROMED (TIPO NUMÉRICO)
6.           CONTOT = 0
7.           CONCONF = 0
8.           CNOCONF = 0
9.           CONTCAS = 0
10.          ACUMING = 0
11.          ACUMEDAD = 0
12.          LEA (CONTROL)
13.          MQ (CONTROL > 0)
```

```
14.                 LEA (ED, EC, ING, CONFGOB)
15.                 CONTOT = CONTOT + 1
16.                 ACUMING = ACUMING + ING
17.                 SI (CONFGOB==1)
18.                         CONCONF = CONCONF + 1
19.                         SI (ING<150000) ^ (EC==2)
20.                                 CONTCAS = CONTCAS + 1
21.                         FINSI
22.                 SINO
23.                         CNOCONF = CNOCONF + 1
24.                         ACUMEDAD = ACUMEDAD + ED
25.                 FINSI
26.                 LEA (CONTROL)
27.         FINMQ
28.         PORCONF = CONCONF * 100 / CONTOT
29.         PROMIN = ACUMING / CONTOT
30.         PROMED = ACUMEDAD / CNOCONF
31.         IMPRIMA (PORCONF, PROMIN, PROMED, CONTCAS)
32.    FIN(Método)
33.  FIN(Clase)
```

En la solución anterior se usó una variable adicional que se llama control, su contenido se lee antes del ciclo y el encabezamiento se construyó con la PLS control > 0, aunque se debe aclarar que pudo haber sido cualquier otra, esto significa que para poder entrar al ciclo se le debe dar el control al valor positivo (usted pudo construir otra PLS como por ejemplo: Control < 0, control = 0, control <> 1, control < 10, etc., la que se quiera pero teniendo en cuenta que al leer control, el valor asignado debe ser tal que la PLS sea verdadera). Además, el control se vuelve a leer antes de retornar al encabezamiento del ciclo y si se quiere volver a iterar el valor leído debe ser tal que la PLS sea verdadera.

Por lo general la manera más usada de formar un ciclo mientras que, es usando variables centinela porque en un diagrama el uso de menos variables lo hace más óptimo, sin embargo, se presentan las otras dos opciones con el fin que se escoja la que más cómoda le parezca.

2.2.3.1.2 Esquema cuantitativo

El esquema cuantitativo es utilizado cuando se conoce **el número de veces** que debe **repetirse un ciclo determinado**, antes de activarse la estructura repetitiva.

El número de iteraciones o cantidad de registro puede ser un valor constante o, generalmente, una variable de entrada cuyo valor es proporcionado al algoritmo antes de activarse el ciclo. Para explicar este esquema se realizará el siguiente ejemplo:

Hacer un algoritmo que encuentre la suma de los primeros N números naturales.

Análisis:

Datos de entrada: La cantidad de números a tener en cuenta en la suma

Proceso: Primero se debe conocer la cantidad de números naturales a sumar y luego generar y sumar la cantidad de números comprendidos entre 1 y esa cantidad.

Datos de salida: La suma de los primeros N números naturales.

Definición de Variables:

- **N:** Cantidad de números naturales a sumar
- **NUM:** Contador que genera los números entre 1 y N y que a su vez controla el ciclo.
- **SUMA:** Suma de los números entre 1 y N

```
1.  CLASE MQCuantitativo
2.    METODO PRINCIPAL ()
3.      VARIABLES: N, SUMA, NUM (TIPO NUMÉRICO)
4.              SUMA = 0
5.              NUM = 1
6.              LEA (N)
7.              MQ (NUM <= N)
8.                      SUMA = SUMA + NUM
9.                      NUM = NUM + 1
10.             FINMQ
11.             IMPRIMA (SUMA)
12.   FIN(Método)
13. FIN(Clase)
```

Prueba de escritorio:

Si el valor de N es 7.

N	NUM	SUMA
7	~~+~~	~~0~~
	~~2~~	~~+~~
	~~3~~	~~3~~
	~~4~~	~~6~~
	~~5~~	~~10~~
	~~6~~	~~15~~
	~~7~~	~~21~~
	8	28

Salida: La suma es: 28

2.2.3.2 CICLO PARA

Es una instrucción que se utiliza cuando una acción se repite más de una vez, donde se conoce los valores inicial y final de las variables controladora del ciclo, y la variación de ella es constante.

Estos procesos cíclicos se caracterizan por el uso de una variable de iteración la cual tiene tres características: Esta inicializada, incrementada y controlada, es decir, tiene:

- **Límite inferior,**
- **Incremento, y**
- **Límite superior.**

Estas características se dan a manera de instrucciones ubicadas **espacialmente en el diagrama** en distintas posiciones, esto es lo que define un ciclo automático y una manera muy cómoda de diagramarlo es **agrupando las tres características** de la variable de iteración en una sola instrucción, seguida del proceso a iterar y terminando con la instrucción FINPARA.

La instrucción en donde están agrupadas las 3 características, se llama el **encabezamiento del ciclo**, esquemáticamente es:

```
PARA (I= LINF, LSUP, INC)
        Proceso
FINPARA
```

En donde:

- I es la variable de iteración o variable controladora del ciclo,
- $LINF$ es el límite inferior o valor inicial de la variable de iteración,
- $LSUP$ es el límite superior o control de la variable de iteración,
- INC es el valor del incremento.

NOTA: La instrucción PARA sólo se usa cuando se conocen el límite inferior (LINF) y límite superior (LSUP) de la variable controladora del ciclo.

Ejemplo: Elaborar un algoritmo que lea edad, estado civil (1: Soltero, 2: Casado), estatura y sexo (1: Hombre, 2: Mujer) de 500 personas. Calcular e imprimir: Cuantas personas cumplen simultáneamente ser mayor de edad, soltero, hombre y alto (estatura > 1.70), el porcentaje de esas personas con respecto a total de personas (500) y el promedio de edad y el promedio de estatura de esas personas.

Solución:

Datos de entrada: Se debe Leer edad, estatura, estado civil y sexo de 500 personas, notemos que al leer el campo sexo se debe almacenar en él un numero 1 o un numero 2, si el campo sexo contiene el número 1 se entiende que es un hombre; de lo contrario el único valor posible es un 2 y se entiende que es una mujer. Este análisis es válido para el campo estado civil.

Proceso: Se debe utilizar un ciclo, es decir, una variable de iteración; Se debe usar un bloque y una PLC para encontrar aquellas personas que cumplen simultáneamente la condición planteada. Se deben contar estas personas, lo cual sugiere el uso de un contador y como nos piden calcular

promedios debemos usar un acumulador de edad y un acumulador de estatura, con el contador es suficiente para calcular los porcentajes requeridos.

Resultados: Imprimir el contador, el promedio y el porcentaje. El diagrama quedaría así:

```
1.  CLASE CicloPara
2.      METODO PRINCIPAL ()
3.              VARIABLES: I, EDAD, EC, EST, SX, CONT, ACUMEDAD, ACUMEST,
4.                                  PROMEDAD, PROMEST, PORC (TIPO NUMÉRICO)
5.              CONT = 0
6.              ACUMEDAD = 0
7.              ACUMEST = 0
8.              PARA (I=1, 500, 1)
9.                      LEA (EDAD, EC, EST, SX)
10.                     SI (EDAD>=18) ^ (EC==1) ^ (EST>=1.70) ^ (SX==1)
11.                             CONT = CONT + 1
12.                             ACUMEDAD = ADUMEDAD + EDAD
13.                             ACUMEST = ECUMEST + EST
14.                     FINSI
15.             FINPARA
16.             PROMEDAD = ECUMEDAD / CONT
17.             PROMEST = ACUMEST / CONT
18.             PORC = CONT * 100 / 500
19.             IMPRIMA (PORC, PROMEDAD, PROMEST)
20.     FIN(Método)
21. FIN(Clase)
```

Note que la PLC del bloque de decisión es verdadera, si todas las PLS que la conforman son verdaderas.

- (EDAD >= 18) nos garantiza ser mayor de edad cuando es verdadera.
- (EC = 1), en donde la variable EC contiene el estado civil, si el contenido del campo EC es igual a 1 significa que es soltero.
- (SX = 1), en donde la variable SX contiene el Sexo. Si el contenido del campo SX es igual a 1 esta PLS es verdadera y significa ser hombre.

- (Est > 1.70), en donde la variable EST contiene la estatura. Según el problema si esta PLS es verdadera es una persona alta.

El campo CONT está contando las personas que cumplen la condición buscada, el campo ACUMEDAD está sumando las edades de esas personas, el campo ACUMEST está sumando las estaturas de esas personas; por Último, note que los promedios de edad (PROMEDAD), de estatura (PROMEST) y el porcentaje (PORC) se calculan por fuera del ciclo. Efectuemos una prueba de escritorio para los siguientes datos (Aunque pudieron ser otros).

EDAD	EC	SX	EST
18	1	1	1.80
20	1	1	1.60
22	1	1	1.75
22	2	2	1.70
26	1	1	1.80

EDAD	EC	SX	EST	CONT	ACUMEDAD	ACUMEST
				0	0	0
18	1	1	1.80	1	18	1.80
20	1	1	1.60	1	18	1.80
22	1	1	1.75	2	40	3.55
22	2	2	1.70	2	40	3.55
26	1	1	1.80	3	66	5.35

Con los valores de ACUMEDAD, ACUMEST y CONT, calculemos los promedios y el porcentaje:

PROMEDAD ← 66/3 => PROMEDAD = 22

PROMEST ← 5.35/3 => PROMEST = 1.78

PORC ← 3 * 100/ 5 => PORC = 60

Por último, se imprime: 3, 22, 1.78,60

Veamos otros ejemplos:

En una empresa se tiene 1000 empleados, por empleado lee la edad. Calcular e imprimir la cantidad de empleados que tienen edades mayores a 21 años y menores de 30 años.

```
1.  CLASE Empleados
2.      METODO PRINCIPAL ()
3.       VARIABLES: CONT, C, EDAD (TIPO NUMÉRICO)
4.                CONT = 0
5.                PARA (C = 1, 1000, 1)
6.                         LEA (EDAD)
7.                         SI (EDAD>21) ^ (EDAD<30)
8.                                  CONT = CONT + 1
9.                         FINSI
10.               FINPARA
11.               IMPRIMA (CONT)
12.     FIN(Método)
13. FIN(Clase)
```

En el ejemplo anterior se usó **la estructura cíclica automática** o **ciclo para** y se ha dicho que esta estructura se utiliza cuando se conoce **el número de iteraciones**, en este caso (1000), el número de iteraciones se ha controlado usando **un bloque de decisión** en el cual por medio de una PLS se compara el contenido actual de la variable de iteración con un valor constante, una manera de generalizar estos ciclos es comparar el contenido de la variable de iteración con el contenido de un campo el cual se ha leído previamente antes y por fuera del ciclo.

Este campo se conoce con el nombre de **REGISTRO IDENTIFICADOR.**

El problema anterior es para 1000 personas, pero lo podemos generalizar para un número N conocido de personas así:

```
1.  CLASE  EmpleadosN
2.     METODO PRINCIPAL ()
3.       VARIABLES: N, CONT, C, EDAD (TIPO NUMÉRICO)
4.                LEA (N)
5.                CONT = 0
6.                PARA (C = 1, N, 1)
7.                         LEA (EDAD)
8.                         SI (EDAD>21) ^ (EDAD<30)
9.                                  CONT = CONT + 1
10.                        FINSI
11.               FINPARA
12.               IMPRIMA (CONT)
13.     FIN(Método)
14. FIN(Clase)
```

Note que el ciclo va hasta N iteraciones, pero N ha sido leído previamente, es decir, el ciclo se puede hacer para 5, 100, 1000, 2000, Personas porque por medio de la lectura se le da el valor al campo N, se determina cuantas veces quiere repetir el proceso. La variable N se llama registro identificador.

2.2.3.3 CICLO QUE HAGA MIENTRAS QUE

Esta instrucción de ciclo tiene la característica que se ejecuta al menos una vez, ya que la condición que controla el ciclo se encuentra al final, después de haber ejecutado las instrucciones que se encuentran dentro de él.

Esta instrucción de ciclo tiene la característica de que se ejecuta mínimo una vez, ya que la condición se evalúa después de que se han ejecutado las instrucciones del ciclo.

La forma general de la instrucción HAGA MIENTRAS QUE es:

```
HAGA
        Instrucciones que se ejecutan mientras la condición sea verdadera
MQ (Condición)
```

Cuando se elaboran algoritmos es necesario, en algunas ocasiones, poder ejecutar primero las instrucciones correspondientes a un ciclo y luego evaluar las condiciones que determinan si se vuelven a ejecutar las instrucciones del ciclo. Una de las situaciones en las que es pertinente utilizar la instrucción HAGA MIENTRAS QUE es la elaboración de un menú, en el cual el usuario debe elegir una opción, ya que primero se deben presentar las posibles opciones para que el usuario escoja una, y luego evaluar la escogencia del usuario.

Ejemplo:

Elaborar un algoritmo que presente un menú en pantalla con las siguientes opciones:

1. Leer número.
2. Calcular factorial.
3. Determinar si es par.
4. Terminar.

El usuario elige una opción y el programa opera de acuerdo a la opción que él eligió.

Veamos cómo es el algoritmo:

```
1.  CLASE Menú (1)
2.      METODO PRINCIPAL ()
3.          VARIABLES: n, f, i, op (TIPO NUMÉRICAS)
4.                  HAGA
5.                      IMPRIMA (“Elija una opción del Menú:”)
6.                      IMPRIMA (“1. Leer número”)
7.                      IMPRIMA (“2. Calcular factorial”)
8.                      IMPRIMA (“3. Determinar si es par”)
9.                      IMPRIMA (“4. Terminar”)
10.                     LEA (op)
11.                     CASOS
```

```
12.                         CASO (op == 1)
13.                             LEA (n)                         SALTO
14.                         CASO (op == 2)
15.                             f = 1
16.                             PARA (i= 1, n, 1)
17.                                 f = f * i
18.                             FINPARA
19.                             IMPRIMA (f, “es el factorial de”, n)
20.                         SALTO
21.                         CASO (op == 3)
22.                             SI ((n % 2) == 0)
23.                                 IMPRIMA (n, “es par”)
24.                             SINO
25.                                 IMPRIMA (n, “es impar”)
26.                             FINSI
27.                         SALTO
28.                         CASO (op == 4)
29.                         SALTO
30.                         OTRO_CASO
31.                             IMPRIMA (“Elija una opción válida”)
32.                         SALTO
33.                     FINCASOS
34.                 MQ (op! = 4)
35.     FIN(Método)
36.  FIN(Clase)
```

En la instrucción 2, definimos las variables con las que vamos a trabajar: **n** el número leído, **f** para almacenar la factorial, **i** para efectuar las operaciones de cálculo del factorial y **op** para almacenar la opción elegida por el usuario.

En la instrucción 4, hemos colocado la instrucción HAGA, la cual significa que todas las instrucciones que hay desde la instrucción 5 hasta la 34 se van a repetir mientras la condición escrita en la instrucción 35 sea verdadera, es decir, mientras que la opción (**op**) sea diferente de 4.

En las instrucciones 5 a 9, escribimos los mensajes de las diferentes opciones que tiene el usuario.

En la instrucción 10, se lee la opción elegida por el usuario y se almacena en la variable **op**.

En la instrucción 11, planteamos la instrucción CASOS, ya que necesitamos comparar el contenido de la variable **op** con varios valores: si **op** vale 1, simplemente se leerá un valor para **n**; si **op** vale 2, se calcula el factorial de **n** y se imprime dicho resultado; si **op** vale 3, averiguamos si **n** es par e imprimimos el mensaje adecuado; si **op** vale 4, simplemente instruimos a la máquina para que no ejecute ninguna operación y vaya a la instrucción 35 para evaluar la condición de terminación del ciclo; y finalmente, si el valor de **op** no es ninguno de los anteriores, imprimimos el mensaje de que la opción elegida por el usuario no es válida. En las instrucciones correspondientes a cada una de las operaciones hemos colocado la instrucción SALTO, la cual hace que vaya a la instrucción 35 para evaluar la condición del ciclo.

En la instrucción 35, se evalúa la condición. Si **op** vale 4, la condición es falsa y la ejecución continúa en la instrucción 36; cualquier otro valor diferente de 4 significa que la condición es verdadera y por consiguiente regresa a la instrucción 4 para volver a ejecutar las instrucciones del ciclo.

2.2.3.4 CONVERSIÓN DE LA INSTRUCCIÓN HAGA MIENTRAS QUE EN INSTRUCCIÓN MIENTRAS QUE

En general, cualquier ciclo elaborado con la instrucción HAGA MIENTRAS QUE se puede construir usando la instrucción MIENTRAS QUE, aunque para ello se necesita una variable adicional. Llamaremos a esta variable adicional **sw**, con la cual controlaremos el ciclo MIENTRAS QUE. Inicialmente a esta variable le asignamos el valor de cero y planteamos el ciclo MIENTRAS QUE para que se ejecute mientras **sw** sea igual a cero. El valor del **sw** sólo lo pondremos en 1 cuando la opción elegida por el usuario sea 4.

Veamos cómo queda nuestro algoritmo utilizando la instrucción MIENTRAS QUE en lugar de utilizar la instrucción HAGA MIENTRAS QUE:

```
1.  CLASE Menú (2)
2.      METODO PRINCIPAL ()
3.        VARIABLES: n, f, i, op, sw (TIPO NUMÉRICAS)
4.                  sw = 0
5.                  MQ (sw == 0)
6.                          IMPRIMA (“Elija una opción del Menú:”)
7.                          IMPRIMA (“1. Leer número”)
8.                          IMPRIMA (“2. Calcular factorial”)
9.                          IMPRIMA (“3. Determinar si es par”)
10.                         IMPRIMA (“4. Terminar”)
11.                         LEA (op)
12.                         CASOS
13.                                 CASO (op == 1)
14.                                         LEA (n)
15.                                 SALTO
16.                                 CASO (op == 2)
17.                                         f = 1
18.                                         PARA (i= 1, n, 1)
19.                                                 f = f * i
20.                                         FINPARA
21.                                         IMPRIMA (f, “es el factorial de”, n)
22.                                 SALTO
23.                                 CASO (op == 3)
24.                                         SI ((n % 2) == 0)
25.                                                 IMPRIMA (n, “es par”)
26.                                         SINO
27.                                                 IMPRIMA (n, “es impar”)
```

```
28.                                 FINSI
29.                         SALTO
30.                         CASO (op == 4)
31.                                 sw = 1
32.                         SALTO
33.                         OTRO_CASO
34.                                 IMPRIMA (“Elija una opción válida”)
35.                         SALTO
36.                 FINCASOS
37.             FINMQ
38.     FIN(Método)
39.  FIN(Clase)
```

2.2.3.5 CICLO ANIDADOS

Es una estructura en la cual un ciclo se encuentra completamente dentro de otro ciclo.

Los ciclos anidados o nido de ciclos ocurren cuando dentro de un ciclo existe otro u otros ciclos. En estos casos **el ciclo más interno** se activa tantas veces como el ciclo externo permita entrar en el grupo de instrucciones.

Veamos que hace el siguiente algoritmo:

```
1.  CLASE CiclosAnidados
2.      METODO PRINCIPAL ()
3.          VARIABLES: I, J (TIPO NUMÉRICO)
4.              PARA (I=1, 3, 1)
5.                  PARA (J=5, 7, 1)
```

```
6.                              IMPRIMA (I, J)
7.                          FINPARA
8.                  FINPARA
9.      FIN(Método)
10. FIN(Clase)
```

Note que la variable I toma los valores 1, 2 y 3, por cada uno de estos valores la Variable J toma los valores 5, 6 y 7, es decir que el ciclo interno se activa 3 veces.

Veamos el seguimiento:

I	J	SE IMPRIME	
1	5	1	5
1	6	1	6
1	7	1	7

AI terminar las iteraciones del ciclo interno (**el de la J**), se retorna nuevamente al ciclo externo (**el de la I**) y la variable de iteración I toma el valor de dos (**I = 2**) lo cual obliga a entrar nuevamente al ciclo anidado (la I no ha terminado sus iteraciones) y esto implica que la variable **J** inicie nuevamente desde el comenzó, es decir, desde el valor de cinco.

El proceso se repite nuevamente, se sale del ciclo interno al ciclo externo y la variable **I** toma su último valor que es **tres** y se efectúa **la última iteración**, esto es:

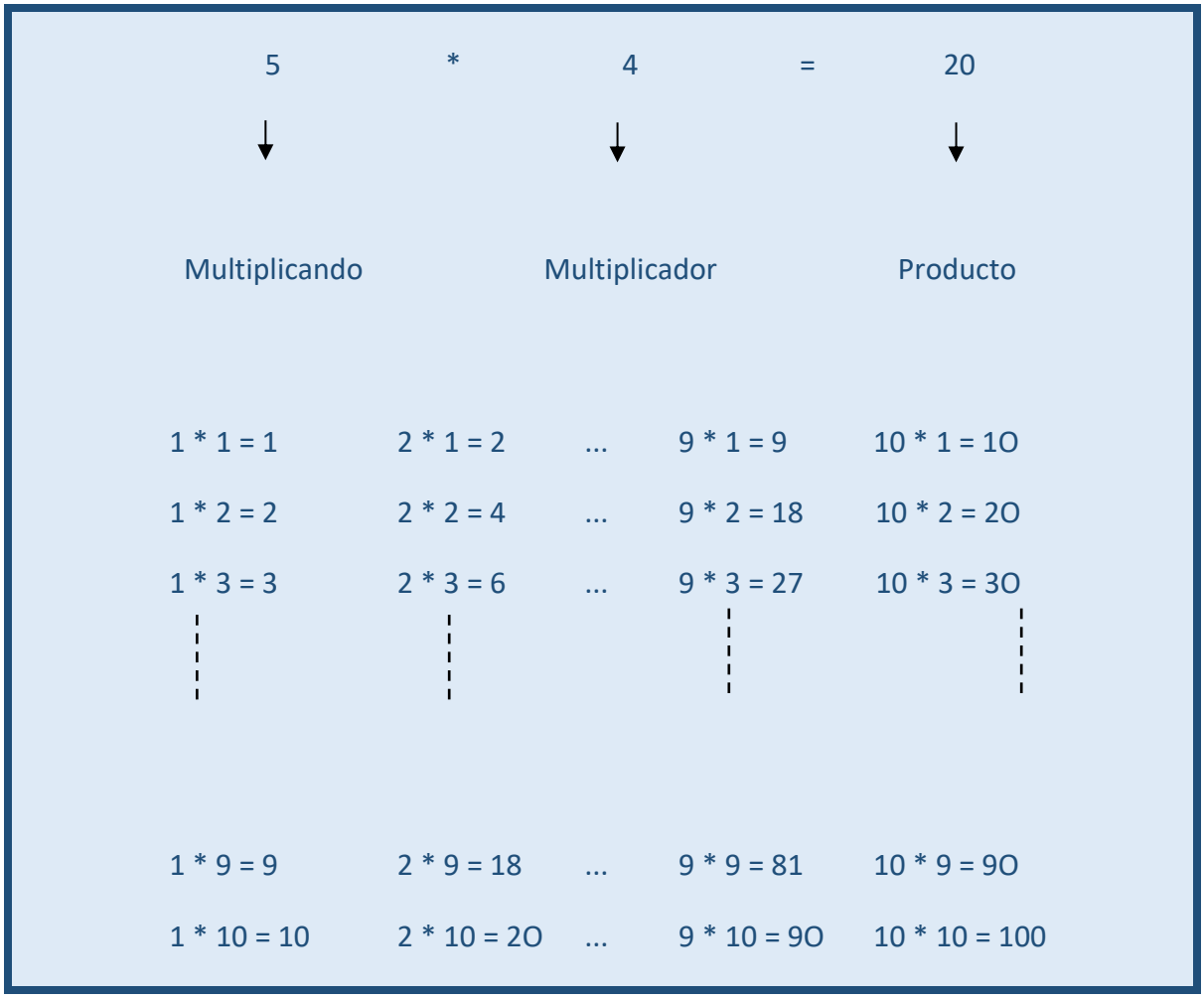
I	J	SE IMPRIME	
2	5	2	5
2	6	2	6
2	7	2	7
3	5	3	5
3	6	3	6
3	7	3	7

En este momento se sale del ciclo de la J y se procede a incrementar la I, pero cuando está ya tomó todos sus valores. Se sigue con la instrucción siguiente a este ciclo (el de Ia I) que es Terminar el proceso.

Nótese que, por cada valor del ciclo más externo, el ciclo interno toma todos sus valores.

Ejemplo 1: Elaborar un algoritmo que imprima las tablas de multiplicar del 1 al 10, se debe imprimir el multiplicando, el multiplicador y el producto.

Análisis:



Nótese que el multiplicando toma valores del 1 al 10, lo mismo que el multiplicador, pero el multiplicando permanece en un valor mientras que el multiplicador toma todos sus valores, lo que lleva a un nido de ciclos de los cuales el más externo generará el multiplicando y el más interno el multiplicador.

Definición de Variables:

- **I**: Índice del ciclo que generará el multiplicando

- **J**: Índice del ciclo que generará el multiplicador
- **Prod**: Producto del multiplicando y el multiplicador

```
1.  CLASE Multiplicación
2.    METODO PRINCIPAL ()
3.      VARIABLES: I, J, PROD
4.              PARA (I= 1, 10, 1)
5.                      PARA (J= 1, 10, 1)
6.                              PROD = I * J
7.                              IMPRIMA (I, "*", J, "=", PROD)
8.                      FINPARA
9.              FINPARA
10.   FIN(Método)
11. FIN(Clase)
```

Ejemplo 2: Elaborar un algoritmo que imprima la siguiente serie de números:

5 - 1 - 10

5 - 1 - 20

5 - 3 - 10

5 - 3 - 20

6 - 1 - 10

6 - 1 - 20

6 - 3 - 10

6 - 3 - 20

Análisis:

Aquí se encuentran tres variables distintas, la primera toma valores de 5 y 6, la segunda de 1 y 3, y la tercera de 10 y 20.

Por simple inspección el ciclo más externo será el que genere el 5 y el 6 (no cambia de valores mientras los otros sí); luego irá el ciclo que genere el 1 y el 3 (no cambia mientras el último sí) y por último, el más interno, será el que genere el 10 y el 20.

Definición de Variables:

K: Índice del ciclo que generará el 5 y el 6

M: Índice del ciclo que generará el 1 y el 3

N: Índice del ciclo que generará el 10 y el 20

```
1.  CLASE Serie
2.      METODO PRINCIPAL ()
3.       VARIABLES: K, M, N (TIPO NUMÉRICO)
4.                PARA (K= 5, 6, 1)
5.                        PARA (M= 1, 3, 2)
6.                                PARA (N= 10, 20, 10)
7.                                        IMPRIMA (K, M, N)
8.                                FINPARA
9.                        FINPARA
10.               FINPARA
11.   FIN(Método)
```

12. FIN(Clase)

Ejemplo 3: Una empresa produce canecas, para ello cuenta con una máquina que produce los discos de la parte inferior y otra que produce los cilindros de los lados. La máquina que produce los discos inferiores puede elaborarlos de 5 tamaños diferentes con radios de 30, 40, 50, 60 y 70 cms. La que produce los cilindros los puede hacer de alturas de 45, 60, 75, 90 cms.

Elabore un algoritmo que encuentre e imprima todos los posibles Volúmenes de las canecas que puede producir la empresa.

Análisis:

Es un nido de ciclos, ya que por cada tipo de discos de la base se tienen diferentes alturas de cilindros.

Volumen = Área de la Base * Altura

Área de la base = 3.1416 * R ** 2

Definición de Variables:

R = Índice del ciclo que generará los radios

A = Índice del ciclo que generará las alturas de los cilindros

ÁREA = Campo variable que contendré las áreas de los discos

VOL = Campo variable que contendrá los volúmenes

```
1.  CLASE Canecas
2.     METODO PRINCIPAL ()
3.      VARIABLES:
4.              PARA (R= 30, 70, 10)
5.                      ÁREA = 3.1416 * (R ^2)
6.                      PARA (A= 45, 90, 15)
7.                              VOL = ÁREA * A
8.                              IMPRIMA (VOL)
9.                      FINPARA
10.             FINPARA
11.     FIN(Método)
12.  FIN(Clase)
```

2.2.4 EJERCICIO DE APRENDIZAJE

Elaborar un algoritmo que lea edad, estado civil (1: Soltero, 2: Casado), estatura y sexo (1: Hombre, 2:

Mujer) de 500 personas. Calcular e imprimir: Cuantas personas cumplen simultáneamente ser mayor de edad, soltero, hombre y alto (estatura > 1.70), el porcentaje de esas personas con respecto a total de personas (500) y el promedio de edad y el promedio de estatura de esas personas.

Solución:

Datos de entrada: Se debe Leer edad, estatura, estado civil y sexo de 500 personas, notemos que al leer el campo sexo se debe almacenar en él un numero 1 o un numero 2, si el campo sexo contiene el número 1 se entiende que es un hombre; de lo contrario el único valor posible es un 2 y se entiende que es una mujer. Este análisis es válido para el campo estado civil.

Proceso: Se debe utilizar un ciclo, es decir, una variable de iteración; Se debe usar un bloque y una PLC (proposición lógica compuesta) para encontrar aquellas personas que cumplen simultáneamente la condición planteada. Se deben contar estas personas, lo cual sugiere el uso de un contador y como nos piden calcular promedios debemos usar un acumulador de edad y un acumulador de estatura, con el contador es suficiente para calcular los porcentajes requeridos.

Resultados: Imprimir el contador, el promedio y el porcentaje. El algoritmo quedaría así:

1. CLASE CicloPara

```
2.    METODO PRINCIPAL ()
3.        VARIABLES: i, edad, ec, est, sx, cont, acumedad, acumest,
4.                   promedad, promest, porc (TIPO NUMÉRICO)
5.        cont = 0
6.        acumedad = 0
7.        acumest = 0
8.        PARA (i=1, 500, 1)
9.            LEA (edad, ec, est, sx)
10.           SI (edad>=18) && (ec==1) && (est>=1.70) && (sx==1)
11.               cont = cont + 1
12.               acumedad = acumedad + edad
13.               acumest = ecumest + est
14.           FINSI
15.       FINPARA
16.       promedad = ecumedadaj / cont
17.       promest = acumest / cont
18.       porc = cont * 100 / 500
19.       IMPRIMA (porc, promedad, promest)
20.   FIN(Método)
21. FIN(Clase)
```

Note que la PLC del bloque de decisión es verdadera, si todas las PLS que la conforman son verdaderas.

- (edad >= 18) nos garantiza ser mayor de edad cuando es verdadera.
- (ec = 1), en donde la variable ec contiene el estado civil, si el contenido del campo ec es igual a 1 significa que es soltero.
- (sx = 1), en donde la variable sx contiene el Sexo. Si el contenido del campo sx es igual a 1 esta PLS es verdadera y significa ser hombre.
- (est > 1.70), en donde la variable est contiene la estatura. Según el problema si esta PLS es verdadera es una persona alta.

El campo cont está contando las personas que cumplen la condición buscada, el campo acumedad está sumando las edades de esas personas, el campo acumest está sumando las estaturas de esas personas; por Último, note que los promedios de edad (promedad), de estatura (promest) y el

porcentaje (porc) se calculan por fuera del ciclo. Efectuemos una prueba de escritorio para los siguientes datos (Aunque pudieron ser otros).

edad	ec	sx	est
18	1	1	1.80
20	1	1	1.60
22	1	1	1.75
22	2	2	1.70
26	1	1	1.80

edad	ec	sx	est	cont	acumedad	acumest
				0	0	0
18	1	1	1.80	1	18	1.80
20	1	1	1.60	1	18	1.80
22	1	1	1.75	2	40	3.55
22	2	2	1.70	2	40	3.55
26	1	1	1.80	3	66	5.35

Con los valores de acumedad, acumest y cont, calculemos los promedios y el porcentaje: promedad □ 66/3 => promedad = 22
promest □ 5.35/3 =>
promest = 1.78 porc □ 3 *
100/ 5 => porc = 60

Por último, se imprime: 3, 22, 1.78, 60

2.2.5 TALLER DE ENTRENAMIENTO

1. Elabore un algoritmo que lea un entero n y que genere e imprima todos los números primos desde 1 hasta n.

2. Elabore un algoritmo que lea dos enteros positivo m y n, y que calculen e impriman el resultado de elevar m a la potencia n utilizando únicamente la operación de suma.

3. Elabore un algoritmo que imprima los enteros desde 1 hasta n de la siguiente manera: 1 22 333 4444 55555 …

4. Una empresa utiliza la siguiente fórmula para calcular el sueldo de sus empleados: Sueldo = (100 + edad + (1+2+3+...+años en la compañía)) /años en la compañía.

Elabore un programa que permita imprimir el sueldo y el nombre de cada uno de los 40 empleados de la compañía, así como el total acumulado de sueldos y el nombre del empleado que gana más y el que gana menos.

TIPS

Asignar información a las variables es importantísimo porque son los valores con los cuales se va a trabajar.

2.3 TEMA 2 INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS

➢ ***Video: " Explicación de programación orientada a objetos (clases, métodos, atributos y objetos)"***

➢ ***" Ver Video":*** https://www.youtube.com/watch?v=EEyKr5-ui8Y&ab_channel=SOSeducaGu%C3%ADasyTutoriales ***"***

2.3.1 MÉTODOS

La solución de problemas complejos se facilita considerablemente si **se divide en problemas más pequeños**. La solución de estos problemas pequeños se realiza con **pequeños algoritmos** que se denominan **métodos** y cuya ejecución depende de **un método principal**.

Los métodos son unidades de programa que están diseñados para ejecutar una tarea específica.

Estos métodos pueden ser de **tipo función** o de **tipo void**, los cuales se escriben sólo una vez, pero pueden ser referenciados en diferentes partes del método principal evitando así **la duplicidad** de instrucciones en un mismo programa.

El método por ser un algoritmo debe cumplir con las mis las características de este y sus tareas deben ser similares como:

- Aceptar datos,
- Escribir datos, y
- Hacer cálculos.

Se clasifica según la forma en la que se invoca el método:

CLASIFICACIÓN
• **Método de instancia:**
Se invoca desde objetos instanciados de una clase
• **Método de clase o método estático:**
Se invocan desde la clase, sin necesidad de instanciar objetos

Los métodos de instancia serán explicados en los subtemas siguientes, por lo que en este subtema no serán referencia de ellos. Los métodos de clase comúnmente llamado **métodos estáticos**,

están asociado a **una clase en particular**, por lo que su invocación se hace con **el nombre de la clase**. En este subtema los métodos se declararán:

Públicos para que se pueda acceder a ellos por **fuera de la clase** a la que pertenecen, y

Estático para invocarlo **desde la clase** y no como una instancia de esta. La forma general de un método estático es:

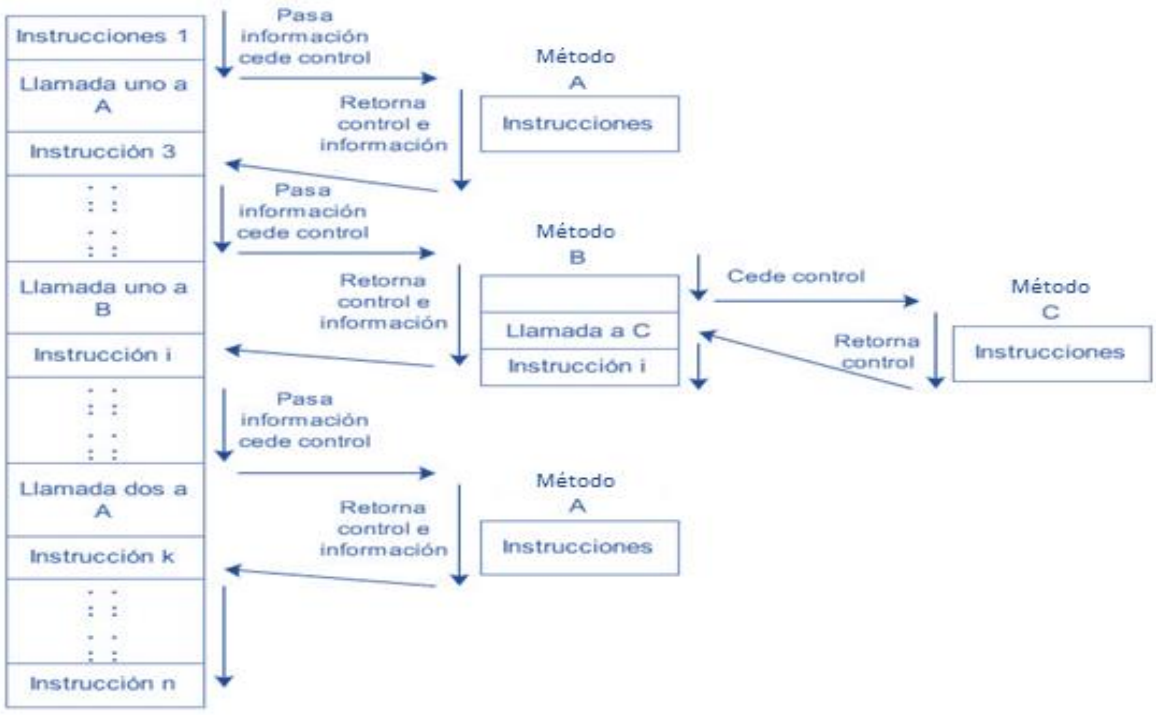
PUBLICO ESTÁTICO TIPO_DE_RETORNO Nombre (Parámetros si los tiene)

Instrucciones

FIN Nombre

Los conceptos de público y estático se ampliarán en los próximos subtemas.

Un método puede ser invocado tantas veces como se necesite y, a su vez, los métodos pueden invocar a otros métodos, como puede verse en la siguiente figura:



2.3.2 MÉTODOS TIPO FUNCIÓN

Los métodos son algoritmos que se **elaboran de forma independiente** y que tienen la característica de que se pueden invocar desde **cualquier programa** cuando se necesiten. El objetivo principal de utilizar métodos es elaborar **algoritmos:**

- ✓ Más cortos,
- ✓ Menos complejos,
- ✓ Más legibles, y
- ✓ eficientes.

La característica principal de un método tipo función es que retorna un valor a la variable desde la cual fue invocada en el método principal, por eso es necesario definir nuestro método del mismo tipo que la variable desde la cual se invocó. Su forma general es:

```
1. PUBLICO ESTATICO TIPO DE DATO Nombre (Parámetros, si los tiene)
2.      Definición Variables:
3.      Instrucciones o cuerpo del método
4.      RETORNE valor
5. FIN(Método)
```

Ejemplo:

En el tema de análisis combinatorio (conteo) es importante determinar el número de combinaciones que se pueden construir con **n** elementos, tomados en grupos de a **r** elementos.

Por ejemplo, si tenemos tres elementos **a**, **b**, y **c**, y queremos formar combinaciones de dos elementos, las posibles combinaciones son **ab**, **ac** y **bc**.

El total de combinaciones de tres elementos (n==3) tomando de a dos elementos (r==2) es tres.

Si el número de elementos es cuatro (n==4) **a**, **b**, **c** y **d**, las combinaciones posibles tomando de a dos elementos (r==2) son **ab**, **ac**, **ad**, **bc**, **bd** y **cd**; es decir, seis posibles combinaciones.

Para determinar el total de combinaciones de **n** elementos tomados en grupos de a **r** elementos se tiene una fórmula matemática que se escribe así:

$$ {}^{n}C_{r} = \frac{n!}{r! \cdot (n - r)!} $$

La cual se lee: el total de combinaciones de **n** elementos tomados en grupos de a **r** elementos es el factorial de **n** dividido por el producto del factorial de **r** con el factorial de **n – r**.

Si nuestro objetivo es elaborar un algoritmo en el cual se lean los datos de **n** y **r**, y calcular el número de combinaciones que se pueden construir, debemos definir una variable para el factorial de **n**, otra para el factorial de **r** y otra para el factorial de **n – r**. Llamaremos a estas variables **fn**, **fr** y **fnr**.

Hagamos un algoritmo para ejecutar esta tarea: leer dos datos enteros **n** y **r**, y calcular e imprimir el total de combinaciones que se pueden construir con **n** elementos tomados en grupos de a **r** elementos:

```
1.  CLASE Combinación (1)
2.    METODO PRINCIPAL ()
3.      VARIABLES: i, n, r, fn, fr, fnr, tc (ENTEROS)
4.              IMPRIMA ("Ingrese el número de elementos que desea combinar")
5.              LEA (n)
6.              IMPRIMA ("Ingrese de cuánto desea el grupo de combinaciones")
7.              LEA (r)
8.              fn = 1
9.              PARA (i = 1, n, 1)
10.                     fn = fn * i
11.             FINPARA
12.             fr = 1
13.             PARA (i = 1, r, 1)
14.                     fr = fr * i
15.             FINPARA
16.             fnr = 1
17.             PARA (i = 1, (n – r), 1)
18.                     fnr = fnr * i
```

```
19.             FINPARA
20.             tc = fn / (fr * fnr)
21.             IMPRIMA (“Total de combinaciones:”, tc)
22.     FIN(Método)
23.  FIN(Clase)
```

En este algoritmo las instrucciones 8 a 11, 12 a 15 y 16 a 19 son exactamente las mismas, la única diferencia es que actúan sobre diferentes datos.

Las instrucciones 8 a 11 trabajan con las variables **fn** y **n**, las instrucciones 12 a 15 con las variables **fr** y **r**, y las instrucciones 16 a 19 con las variables **fnr** y el resultado de **n – r**.

Esta situación, en la cual tenemos un grupo de instrucciones que se repiten en diferentes partes del programa con datos diferentes, amerita una herramienta que nos permita obviar estas repeticiones.

Veamos cómo pudimos haber escrito nuestro algoritmo si tuviéramos un método que determine el factorial de un número entero cualquiera:

```
1.  CLASE Combinación (2)
2.     METODO PRINCIPAL ()
3.       VARIABLES: n, r, fn, fr, fnr, tc (ENTEROS)
4.             IMPRIMA (“Ingrese el número de elementos que desea combinar”)
5.             LEA (n)
6.             IMPRIMA (“Ingrese de cuánto desea el grupo de combinaciones”)
7.             LEA (r)
8.             fn = Combinacion.Factorial (n)
9.             fr = Combinacion.Factorial (r)
10.            fnr = Combinacion.Factorial (n – r)
11.            tc = fn / (fr * fnr)
12.            IMPRIMA (“Total de combinaciones:”, tc)
13.    FIN(Método)
14. FIN(Clase)
```

En este nuevo algoritmo hemos reemplazado las instrucciones 8 a 11 por una sola instrucción:

```
fn = Combinacion.Factorial (n)
```

Las instrucciones 12 a 15 por una sola instrucción:

fr = Combinacion.Factorial (r)

Y las instrucciones 16 a 19 por una sola instrucción:

fnr = Combinacion.Factorial (n – r)

Como se puede ver, el algoritmo (2) de *combinaciones* es más compacto y más legible que el algoritmo (1) de *combinaciones*.

Hemos hecho uso de un método que llamamos *Factorial*, el cual calcula y retorna el factorial de un número entero que se envía a este método.

Como se observa se invocó al método Factorial y se hizo a través de una asignación el nombre de la clase. Y el nombre del método. Esto quiere decir que el formato general para invocar a un método que retorna un valor se realizar de las siguientes maneras:

Variable=NombreClase.NombreFunción (Argumentos – si tiene parámetros)
ESCRIBA: NombreClase.NombreFunción (Argumentos – si tiene parámetros)

Veamos cómo funciona el método Factorial:

```
1.  PUBLICO ESTATICO ENTERO Factorial (m)
2.      DV: i, f (ENTEROS)
3.          f = 1
4.          PARA (i = 1, m, 1)
5.              f = f * i
6.          FINPARA
7.          RETORNE (f)
8.  FIN(Método)
```

En la instrucción 1 definimos el nombre del método y lo precedemos con la palabra ENTERO, la cual indica que este método retorna un valor numérico de tipo entero. En esta misma instrucción 1 definimos una variable **m**, la cual será la variable que recibe el dato con el que trabajará el método.

La instrucción 3 asignamos a la variable f el valor de 1 por ser el módulo de la multiplicación

Las instrucciones 4 a la 6 son las instrucciones propias para calcular el factorial de un número **m**.

En la instrucción 7 se retorna el valor que fue calculado para el dato **m**.

2.3.3 EJERCICIO DE APRENDIZAJE

1. Elaborar un método tipo función que encuentre X^y por multiplicaciones sucesivas.

2. Elaborar un método que reciba un número y determine si dicho número es perfecto o no. Un número N es perfecto si la suma de todos sus divisores, excepto por él mismo, da N.

3. Elaborar un algoritmo que lea dos números N y M, donde N<M, e invoque dos métodos: uno que sume los números entre N y M, y otro calcule el promedio entre ellos.

2.3.4 MÉTODOS TIPO VOID

Hemos visto un tipo de métodos llamados funciones, que retornan un valor. Ahora veamos otro tipo de métodos tipo función que no retornan valores, sino que sólo ejecutan tareas. Este tipo de métodos, que sólo ejecutan una tarea, se denominan funciones tipo void, pueden tener parámetros o no: si no tienen parámetros los paréntesis son obligatorios y deben estar vacíos. la palabra void se usa para declarar funciones que no retornan valor. Ahora recordemos que la definición de público y estático será adoptada en este tema para desarrollar los métodos; sin embargo, estas expresiones y otras serán analizadas y ampliadas en el tema siguiente

La forma general de nuestros métodos tipo void será:

```
1. PUBLICO ESTATICO VOID Nombre (Parámetros, si los tiene)
2.       Definición Variables:
3.       Instrucciones o cuerpo del método
4. FIN(Método)
```

Uso de Void

La función podria no retornar valores en este caso es de tipo Void.

Cuando se escribe con (void), se especifica que cuando se llame esta no tomará ningún valor de parametro

```
// void function example
#include <iostream>
using namespace std;

void mensaje () //podría ser también
                void mensaje (void)
{
  cout << "Hola estudiantes Ingeniería
automatica!";
}

int main ()
{
  mensaje ();
  system ("pause");
}
```

Funciones tipo void

Se utiliza el tipo **void** para indicar que la función no regresa ningún valor.

Una función tipo **void** no debe aparecer en una instrucción de asignación en.

Ejemplo:

```
void despliegaNombre(){
  cout << "Programa hecho por Fulanito de
Tal.\n";
}
```

Llamada a la función:

```
despliegaNombre();
```

Ejemplos de Void

```
char car1='B', car2;
void *ptg;
ptg = &car1;
car2 = *((char *)ptg) + 3;
```

```
char cad[20], car3;
void *ptg;
ptg = cad;
strcpy(cad, "Ejemplo");
car3 = *((char *)(ptg) + 3) + 5;
```

Ejemplo:

Es muy común, en muchas actividades, colocar encabezados que siempre tendrán los mismos títulos, y cuyos datos son siempre los mismos o la variación es mínima.

Para no tener que escribir en cada algoritmo las mismas instrucciones de escritura podemos elaborar un método tipo void que efectúe esta tarea.

Por ejemplo, supongamos que los títulos son:

CORPORACIÓN UNIVERSITARIA REMINGTON

FACULTAD DE CIENCIAS BÁSICAS E INGENIERÍA

INGENIERÍA DE SISTEMAS

ALGORITMOS I

ALGORITMO: nombre del algoritmo

En donde la parte subrayada corresponde a un dato que se enviará como parámetro.

El método sería:

```
1. PUBLICO ESTATICO VOID Títulos (nombre)
2.         IMPRIMA ("CORPORACIÓN UNIVERSITARIA REMINGTON")
3.         IMPRIMA ("FACULTAD DE CIENCIAS BÁSICAS E INGENIERÍA")
4.         IMPRIMA ("INGENIERÍA DE SISTEMAS")
5.         IMPRIMA ("ALGORITMOS I")
6.         IMRPIMA ("ALGORITMO:", nombre)
7. FIN(Método)
```

Nótese que este método no utiliza variables propias, por ello no hay necesidad de hacer la definición de variables.

En la primara instrucción definimos el nombre del método, el cual llamamos *Títulos*, y le definimos un parámetro, llamado **nombre**, en el que se recibe el nombre del algoritmo cuyo título desea imprimir.

Ese dato que se recibe como parámetro lo imprimimos en la instrucción 6.

Las instrucciones 2, 3, 4 y 5 simplemente escriben los mensajes correspondientes a los títulos que se desean imprimir.

Teniendo estos métodos, *Factorial* (m) y *Títulos* (nombre), podremos escribir nuestro algoritmo combinaciones así:

```
1.  CLASE Combinación (3)
2.     METODO PRINCIPAL ()
3.          VARIABLES: n, r, fn, fr, fnr, tc (ENTEROS)
4.                IMPRIMA ("Ingrese el número de elementos que desea combinar")
5.                LEA (n)
6.                IMPRIMA ("Ingrese de cuánto desea el grupo de combinaciones")
7.                LEA (r)
8.                Combinacion.Titulos ("combinaciones")
9.                fn = Combinacion.Factorial (n)
10.               fr = Combinacion.Factorial (r)
11.               fnr = Combinacion.Factorial (n – r)
12.               tc = fn / (fr * fnr)
13.               IMPRIMA ("n=", n, "r=", r, "total de combinaciones=", tc)
14.    FIN(Método)
15. FIN(Clase)
```

Al ejecutar este algoritmo con los datos 5 y 3, para **n** y **r** respectivamente, el resultado de la ejecución sería:

CORPORACIÓN UNIVERSITARIA REMINGTON

FACULTAD DE CIENCIAS BÁSICAS E INGENIERÍA

INGENIERÍA DE SISTEMAS

ALGORITMOS I

ALGORITMO: combinaciones

n = 5 r = 3 total de combinaciones = 10

2.3.5 EJERCICIO DE APRENDIZAJE

1. Elaborar un método tipo void que reciba como parámetros de entrada 3 valores diferentes y los ordene en forma descendente.

2. Elaborar un método tipo void que encuentre los primeros N números perfectos.

3. Elabore un método tipo void que obtenga el interés generado por X pesos que se invierte a Y por ciento mensual en un período de N meses.

Observe que la instrucción numero 8 del algoritmo anterior se invocó al método Titulo de la siguiente manera:

Combinacion.Titulos (“combinaciones”)

Donde se tiene el nombre de la clase. Y el nombre del método y entre paréntesis y entre comillas dobles el mensaje que se le está enviando al método, esto quiere decir que cuando se invoca a un método su formato general es el siguiente:

NombreClase.Nombre (argumentos – si el método tiene parámetros)

Los argumentos son nombres de campos (variables o constantes) que usa el método llamante para enviar y recibir información del método invocado y que tiene una correspondencia biunívoca con los parámetros con los cuales se construyó; por lo tanto, estos deben ser iguales en número y tipo de datos que los parámetros, ya que a cada argumento le corresponde un parámetro y viceversa. Los argumentos también son de dos clases: unos que envían información al método y otros que reciben información (si hay parámetro de envió).

Nota: No todos los lenguajes de programación utilizan parámetros de envió.

2.3.6 PARÁMETROS Y VARIABLES

Cuando se construyen métodos, éstos requieren datos para poder ejecutarlos. Estos datos hay que suministrarlos al método usando otras variables. Estas variables reciben el nombre de

parámetros. Adicionalmente, el método requiere manejar otra serie de datos, bien sea definidos dentro del método o tomados del método principal. Todo lo anterior nos lleva a definir lo que son los conceptos de:

- Parámetros por valor,
- Parámetros por referencia,
- Variables locales, y
- Variables globales.

Como vimos anteriormente en un método tipo función que retorna valor, el tipo de dato que retorna puede ser entero, real, alfanumérico, etc.

En el ejemplo del Factorial, el tipo que definimos para el método tipo función fue **entero**, ya que el dato que retorna es **un número entero**.

El llamado de la función debe estar en una **instrucción de asignación**.

El llamado de un método tipo void es simplemente una en la cual se hace referencia al **nombre de la clase**. El nombre del método, con sus respectivos argumentos si hay parámetros

2.3.7 VARIABLES LOCALES Y VARIABLES GLOBALES

Dentro de un método se pueden definir nuevas variables.

- Las variables que se definen dentro del método se denominan ***variables locales*** y sólo existirán mientras se esté ejecutando el método; cuando termina la ejecución del método dichas variables desaparecen.
- Si dentro del método se utilizan variables definidas por fuera del método, éstas se llaman ***variables globales*.**

Los datos necesarios para la ejecución de un método se denominan ***parámetros formales***.

2.3.8 PARÁMETROS DE UN MÉTODO

Los parámetros, por lo general, son **datos de entrada de un método**, aunque también puede haber parámetro de entrada y salida, o sólo de salida.

- Los parámetros de entrada se denominan ***parámetros por valor***.
- Los parámetros de entrada y salida, o sólo de salida, se denominan ***parámetros por referencia***.

Cuando se define un método **hay que especificar** cuáles parámetros son por **valor** y cuáles parámetros son por **referencia**.

Elaboremos un ejemplo para mostrar la diferencia entre parámetros por valor y parámetros por referencia:

```
1.  CLASE Parámetro
2.     METODO PRINCIPAL ()
3.       VARIABLES: a, b, c (ENTEROS)
4.                IMPRIMA (“Ingrese tres números”)
5.                LEA (a, b, c)
6.                Parametro.Demo (a, b, c)
7.                IMPRIMA (a, b, c)
8.     FIN(Método)
9.  FIN(Clase)
```

```
1.  PUBLICO ESTATICO VOID Demo (por valor: x, y; por referencia: z)
2.                x = x + 3
3.                y = y * x
4.                z = x + y
5.                IMPRIMA (x, y, z)
6.    FIN(Método)
```

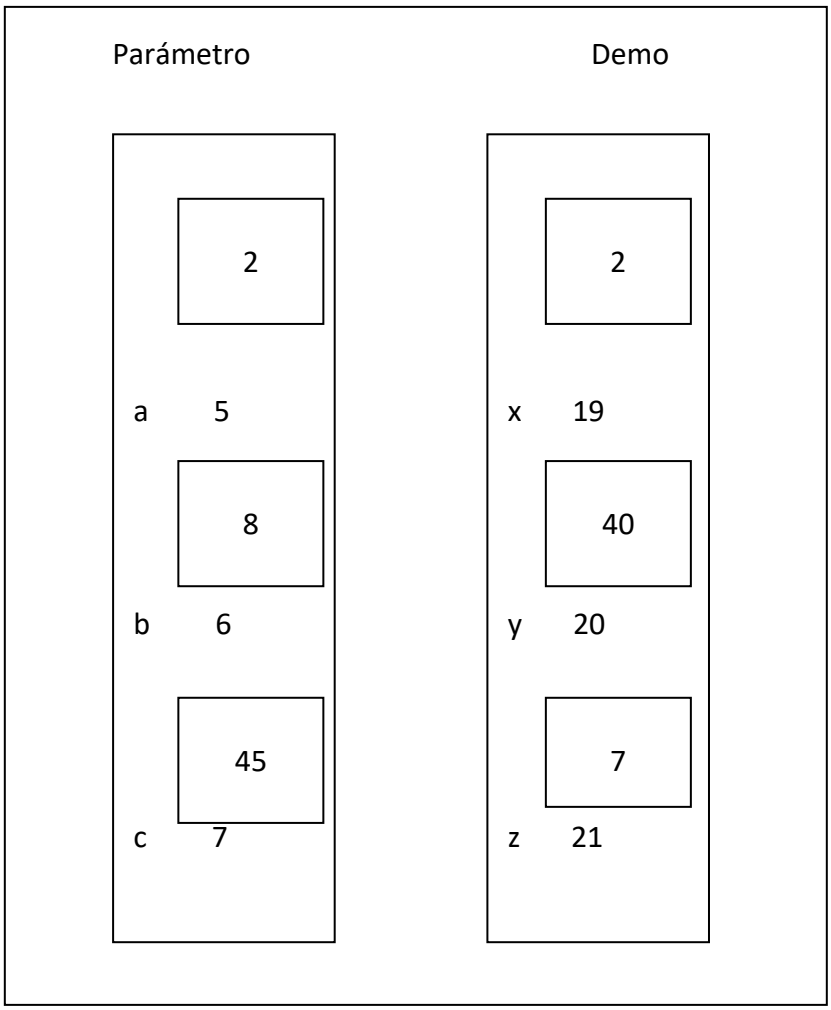
En la clase que hemos llamado ***Parámetro***, se definen tres variables enteras **a**, **b** y **c**, las cuales vamos a suponer que se hallan en las posiciones 5, 6 y 7 de memoria, respectivamente, como veremos más adelante.

En el método *Demo* se definen tres parámetros **x**, **y**, **z**. Los parámetros **x**, **y** se definen por valor, **z** por referencia. Supongamos que estos tres parámetros **x**, **y**, **z** se almacenan en las posiciones de memoria 19, 20 y 21, respectivamente.

Al ejecutar la instrucción 5 de la clase *Parámetro*, se leen tres datos para **a**, **b** y **c**, digamos 2, 8 y 9. Estos datos quedan almacenados en las posiciones de memoria 5, 6 y 7.

Al ejecutar la instrucción 6, la cual invocas al método *Demo*, sucede lo siguiente: **x** e **y** fueron definidos parámetros por valor, por lo tanto el contenido de las variables **a** y **b** se copia hacia las posiciones de memoria correspondientes a **x** e **y**; **z** fue definido por referencia, lo cual implica que lo que se copia en la posición de memoria **z** es la posición de memoria en la cual se halla almacenada la variable **c**.

MEMORIA

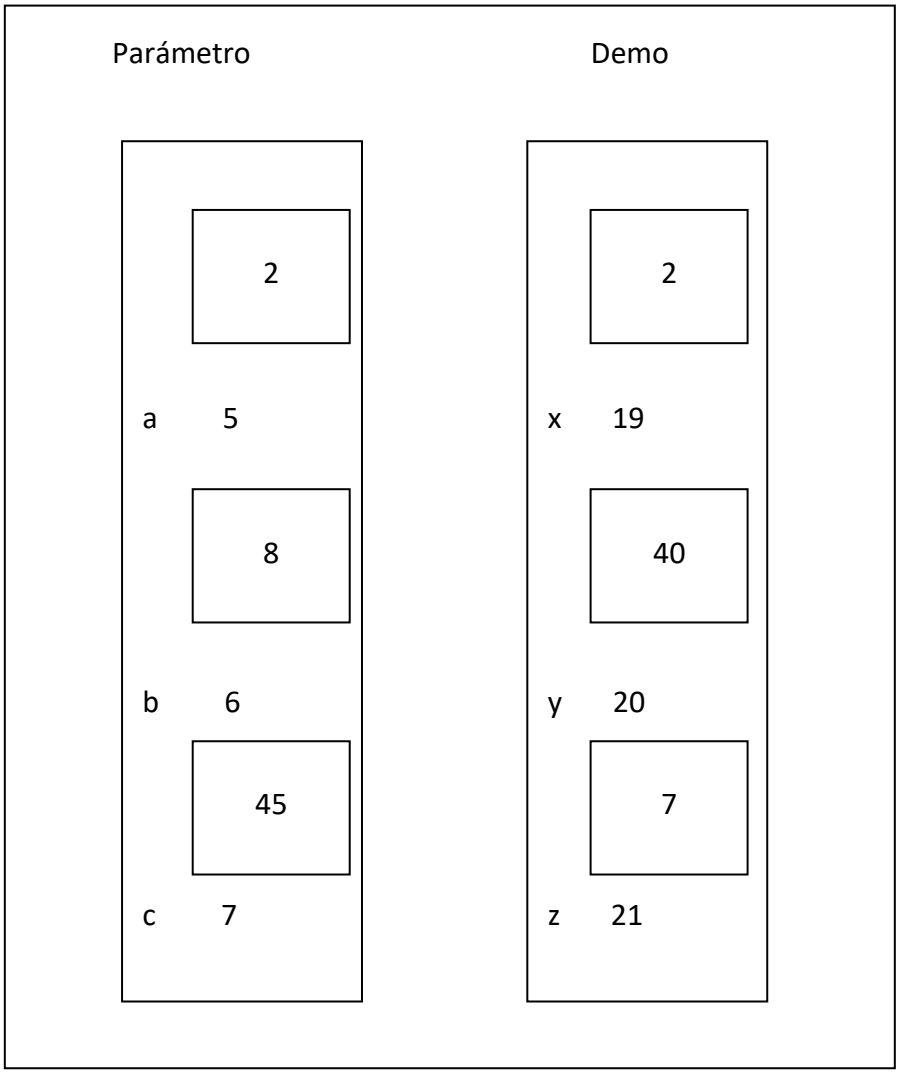


Estado de la memoria cuando se invoca el subprograma

Al ejecutarse el método *Demo*, sucede lo siguiente: en la instrucción 2 se le suma 3 a **x**, y el resultado se almacena en la misma posición de memoria **x**; en la instrucción 3 se multiplica **y** por **x** y el resultado se almacena en la misma posición de memoria **y**; y en la instrucción 4 sumamos **x** con **y**, y almacenamos el resultado en **z**. ¿Pero qué es **z**?: **z** es una posición de memoria: la dirección de memoria 7; por lo tanto el resultado de sumar el contenido de **x** con el contenido de **y** se almacena en la posición de memoria 7. Es como si desde dentro del método hubiéramos tenido acceso a la variable **c**, definida en el método principal.

El resultado de efectuar estas operaciones se refleja en la siguiente figura:

MEMORIA



Estado de la memoria después de ejecutar el método

Al ejecutar la instrucción 5 del método *Demo*, imprime 5, 40 y 45.

Al ejecutar la instrucción 7 del algoritmo *Parámetro*, imprime 2, 8 y 45.

2.3.9 TALLER DE ENTRENAMIENTO

1. Elabore las pruebas de escritorio respectivas a cada uno de los ejercicios de métodos tipo función e identifique las variables locales y las variables globales.

2. Elabore las pruebas de escritorio respectivas a cada uno de los ejercicios de métodos tipo void e identifique los parámetros por valor y los parámetros por referencia.

2.3.10 CLASE

2.3.10.1 LECTURA TODO ES UN OBJETO

Es una introducción teoría de los conceptos más importante de la programación orientada a objeto, donde lo más relevante es que el estudiante pueda abstraer el concepto de un objeto a partir de una realidad dada.

Para enriquecer todo el concepto de orientación a objeto es importante que visiten los siguientes links:

- https://www.dropbox.com/s/97jm0jnvrpgg7j4/Piensa%20en%20_Java_%20Bruce_Eckel_4ta%20Edici%C3%B3n_Espa%C3%B1ol_Manybadilla.pdf?dl=0 (página 23 - 43)

- http://sistemasremington.webnode.com/introduccion-al-dllo-de-software/documentos-complementarios/

2.3.10.2 DIAGRAMA DE CLASE Y OBJETO)

Antes de hablar de diagrama de clase y objeto es importante dar una definición de lo que es una clase y lo que es un objeto.

Cuando se quiere programar bajo el paradigma de **programación orientada a objetos** (**POO**) consiste en **realizar o modelar los objetos del mundo real**. En una perrera, por ejemplo, se puede identificar diferentes tipos de animales como entes con características y acciones propias. Como ejemplo práctico vamos a tomar un perro.

El perro que tenemos en mención (Pepe) es un objeto con características específicas. Claro está, en la perrera pueden existir otros perros con diferente edad, color, peso, medida y raza. De esta manera se puede definir la CLASE Perro como una abstracción a todos los perros existentes en la perrera;

- Las características son las VARIABLES MIEMBRO O ATRIBUTOS DE LA CLASE, y
- Las acciones son los MÉTODOS MIEMBRO DE LA CLASE o COMPORTAMIENTO DE LA CLASE.

Con el ejemplo anterior podemos deducir la definición de clase y objeto.

Una **clase** es una plantilla para crear objetos que constituye una abstracción del mundo real, como la clase PERRO, la clase persona, la clase cuenta, la clase carro, entre otros.

Las clases se nombran como **sustantivos en singular** y poseen variables que definen la información que se desea almacenar y métodos que definen las acciones que se desean realizar.

- Las **variables** se nombran como **sustantivo**s, y
- Los **métodos** como **verbos en infinitivo**.

Las clases se pueden representar gráficamente mediante un diagrama cuyas especificaciones son establecidas por el lenguaje unificado de modelo *(UML – Unified Modeling Lenguaje versión 2.4.4)*, el cual permite diseñar visualmente los sistemas de software. Uno de los diagramas de UML es denominado *diagrama de clases,* donde se utiliza la siguiente representación:

Nombre de la clase
variable1 variable2 …
método1() método2() …

Así, el diagrama de la clase Perro es:

Perro
edad color peso medidas raza
comer () dormir () correr () ladrar ()

Por otro lado:

Un **objeto** es una instanciación de una clase, es decir, la materialización de la clase. Cuando se instancia un objeto se asignan datos a las variables de la clase y se pueden ejecutar los métodos.

Fácilmente se pueden diferenciar las clases de los objetos con el siguiente paralelo:

Las **clases** son moldes de panes y **los objetos** son los panes creadas con los moldes.

Todos los panes creados con el mismo molde son iguales, así que todos los objetos instanciados de la misma clase tienen las mismas variables y los mismos métodos disponibles. Se diferencian en los valores que se asignan a las variables.

Gráficamente, los objetos se representan mediante un ***diagrama de objetos*** establecido por UML así:

Nombre objeto: Clase
variable1=valor variable2=valor …
método1() método2() …

El perro Pepe que mencionamos inicialmente es un objeto instanciado de la clase Perro, que podemos representar de la siguiente manera:

Pepe : Perro
Edad = 5 Color = blanco negro Peso = 35 Kg Medidas = 60 cm largo

Raza = dálmata
Comer () Dormir () Correr () Ladrar ()

2.3.10.3 DECLARACIÓN DE CLASE

Antes de hablar de la declaración de clase es importante que se miren algunas propiedades de la programación orientada a objeto.

La programación objetual define un conjunto de propiedades básicas que los lenguajes de programación orientados a objetos deben cumplir:

- Abstracción.
- Encapsulamiento.
- Ocultamiento de información.
- Sobrecarga.
- Polimorfismo.
- Herencia.
- Reutilización.

En el desarrollo de esta unidad abordaremos algunas de estas propiedades. Adicionalmente, se podrán introducir otros conceptos relacionados con el tema.

- **Abstracción**

La abstracción es la característica más básica de la POO y puede definirse como la acción de identificar las cualidades y acciones que un objeto puede realizar, lo que permite diferenciar los conceptos de clase objeto. Se puede crear la abstracción Perro, compuesta de las características: edad, color, peso, medidas y raza. También se puede crear la abstracción Bicicleta, compuesta de varias características y acciones:

CARACTERÍSTICAS DE LA BICICLETA

- Marca.
- Modelo.
- Tipo.
- Color.

Estas características son comunes a todas las bicicletas.

ACCIONES QUE PUEDE REALIZAR LA BICICLETA

- Calcular velocidad máxima.
- Calcular tiempo máximo de vida útil

Estos comportamientos son comunes a todas las bicicletas.

Existen muchos tipos bicicletas con diferentes marcas, modelo, tipo y color, así que la abstracción Bicicleta agrupa todos los tipos de bicicletas existentes que se pueden definir con esas características. De esta manera tenemos que la abstracción Bicicleta es una clase y todos los tipos de bicicletas que puedan existir con las características definidas en la clase son objetos. El diagrama de la clase Bicicleta es:

Bicicleta
Marca Modelo Tipo Color
calcularVelocidad () calcularVidaUtil ()

Ejercicio resuelto

Diagramar el objeto GW como una instancia de la clase Bicicleta presentada anteriormente.

Siguiendo la clase Bicicleta presentada, en el objeto GW se puede diagramar de la siguiente manera:

GW : Bicicleta
Marca : GW Modelo : 2016 Tipo : Mountain Color : Blanca
calcularVelocidad () calcularVidaUtil ()

Ejercicio resuelto

Diagramar una clase que permita representar un estudiante universitario y crear un objeto de dicha clase.

Un estudiante universitario tiene datos básicos que deben ser tenidos en cuenta como: nombre, fecha de nacimiento, número de identificación, dirección, teléfono, programa, semestre actual y promedio crédito. Adicionalmente, se pueden necesitar algunos cálculos básicos de cada estudiante como: calcular edad y calcular tiempo faltante para grado. Con estos datos es posible crear una clase Estudiante así:

Estudiante
Nombre fechaNacimiento NumId dirección Teléfono Programa Semestre Promedio
calcularEdad( ) calcularTiempoGrado( )

De la clase Estudiante es posible instanciar objetos como:

Estudiante
Nombre= Carolina Cruz fechaNacimiento= 09-05-81 NumId= 12345 Dirección= Calle 15 C # 23-78 Teléfono= 5134583 Programa= Ingeniería sitema Semestre= 5 Promedio= 4.5
calcularEdad( ) calcularTiempoGrado( )

- **Encapsulamiento y ocultamiento de la información**

El encapsulamiento es la propiedad que tienen las clases de agrupar las características y las acciones relacionadas con una abstracción bajo una misma unidad de programación. Con la encapsulación, las clases son vistas como cajas negras donde se conocen las características y las acciones (variables y métodos) pero no sus detalles internos, es decir, que se conoce lo que hace la clase, pero no la forma en que lo hace.

Por su parte, el ocultamiento de la información protege los objetos restringiendo y controlando el acceso en la clase que los define. Esto permite al programador definir exactamente cuales variables y métodos serán visibles para otras clases, asegurándose de que el estado interno de un objeto no pueda ser cambiado de forma inesperada por una entidad externa. Con esta propiedad se logra que los métodos de la clase Perro puedan modificar sin ningún problema las variables de la misma clase, pero no las variables de la clase Bicicleta, a menos que se asigne un permiso específico. De igual manera, los métodos de la clase Bicicleta pueden acceder y modificar las variables de su propia clase, pero no pueden modificar las variables de la clase Perro. La capacidad de restringir y controlar el acceso en las clases se logra por medios de unos especificadores o modificadores de acceso que se describen a continuación.

Los especificadores o modificadores condicionan el acceso de las variables, de los métodos de una clase, definiendo su nivel de ocultamiento. Los especificadores pueden ser:

ESPECIFICADORES
• $(+)$ ***Publico*:**
El elemento puede ser accedido desde cualquier clase. Si es un dato miembro, cualquier clase puede acceder al elemento. Si es un método, cualquier clase puede invocarlo.
• $(-)$ ***Privado*:**
Solo se puede acceder al elemento desde métodos miembros de la clase donde se encuentra definido, o solo puede invocarse desde otro método de la misma clase.
• $(\#)$ ***Protegido*:**
Proporciona acceso público para las clases derivadas (se revisará en la sección de herencia) y acceso privado para el resto de clases.
• $()$ ***Sin modificador*:**
Se puede acceder al elemento desde cualquier clase que este en la misma ubicación (carpeta) donde se define la clase.

Las clases no pueden declararse **ni protegidas ni privadas**, así que solo pueden declararse **públicas** o **sin modificador**. Por el contrario, **las variables** y **los métodos** pueden declararse con cualquiera de **los modificadores**. Por ejemplo, en la clase Estudiante, se van a incluir los modificadores de acceso de la siguiente manera:

Estudiante
+nombre
+fechaNacimiento
+numId
-dirección
-teléfono
+programa

+semestre -promedio
+calcularEdad( ) +calcularTiempoGrado()

Como la clase descrita no tiene modificador de acceso, significa que solo puede ser accedida por las clases que se encuentren en la misma ubicación.

Las variables nombre, fechaNacimiento, numId, programa y semestre pueden ser consultadas y modificadas desde cualquier clase, pero las variables dirección, teléfono y promedio solo pueden ser consultadas y modificadas desde métodos de la clase Estudiante. El criterio para definir cuando una variable es privada, publica, protegida o sin modificador depende del problema que se va a resolver. Por ejemplo, se podría definir que las variables nombre, fechaNacimiento, numId, programa y semestre no necesitan protección, mientras que las variables dirección, teléfono y promedio tienen información que debería ser accedida con precaución.

Finalmente, el método calcularEdad() puede ser invocado desde cualquier clase. Por el contrario, el método calcularTiempoGrado() solo puede ser invocado desde otro método de la clase Estudiante.

Ejercicio resuelto

Diagramar una clase que permita representar los datos básicos de una persona y calcular su índice de masa corporal y su edad.

Para cada persona se pueden identificar algunas características básicas como nombre, año de nacimiento, sexo, peso y estatura. Estas constituyen las variables de la clase. Teniendo en cuanta las variables almacenadas para la clase Persona, se pueden crear métodos que permitan calcular el índice de masa corporal y calcular la edad de la persona.

Persona
+nombre

+añoNacimiento +sexo -peso -estatura
+calcularIndiceMasaCorporal( ) +calcularEdad( )

La clase Persona puede ser accedida por cualquier clase que se encuentre en la misma ubicación, y las variables nombre, añoNacimiento y sexo pueden ser accedidas desde cualquier clase. Sin embargo, las variables peso y estatura solo pueden ser accedidas desde métodos de la clase Persona. Finalmente, ambos métodos pueden ser accedidos desde cualquier clase.

- **Formato para representar una clase**

Con los conceptos vistos anteriormente se nos permite representar una clase. Partiendo del diagrama de clase, por lo cual nuestro formato tiene la siguiente estructura:

```
MODIFICADOR_ACCESO CLASE Nombre Clase

    MODIFICADOR_ACCESO TIPO variable1
    MODIFICADOR_ACCESO TIPO variable2
    MODIFICADOR_ACCESO TIPO variable3
    …
    MODIFICADOR_ACCESO TIPO_RETORNO metodo1()

    …

    FIN_metodo1
    MODIFICADOR_ACCESO TIPO_RETORNO metodo2()
    …
    FIN_metodo2
    MODIFICADOR_ ACCESO TIPO_RETORNO metodo3()
    …
    FIN_metodo3

FIN_CLASE
```

A continuación, se va a crear la clase Persona. El siguiente pseudocódigo codifica la clase Persona:

```
CLASE Persona

        PUBLICO CADENA nombre
        PUBLICO ENTERO añoNacimiento
        PUBLICO CARÁCTER sexo
        PRIVADO REAL peso
        PRIVADO REAL estatura
        PUBLICO VOID calcularMasaCorporal()

                índice=peso/estatura^2
                ESCRIBA: índice

        FIN_calcularMasaCorporal
        PUBLICO VOID calcularEdad ()

                Edad=2013-añoNacimiento
                ESCRIBA: edad

        FIN_calcularEdad

FIN_CLASE
```

En los métodos de la clase Persona se puede observar que no necesita ingresar como parámetros las variables que son miembro de la clase, ya que todos los métodos de esta pueden acceder directamente a las variables de la clase. Si el método requiere **variables adicionales** a las que tiene la clase, entonces se pueden ingresar como **parámetro**s o **declararlas en su contenido**. Sin embargo, la anterior representación de la clase Persona tiene un problema:

> Las variables de la clase nunca son inicializadas, por lo que contiene valores "basura".

Solucionaremos este problema mediante un método llamado **constructor**.

2.3.11 OBJETOS

Define todos los aspectos de **la abstracción del problema** relacionando el nombre de la clase los atributos necesarios con sus respectivos tipos de datos asociados, según las necesidades de procesamientos para los objetos se definen los métodos en cada caso.

2.3.12 CONSTRUCTOR

El constructor o **función constructora** es **un método público** que tiene el mismo nombre de la clase y se ejecuta automáticamente al crear un objeto de la clase. Como característica especial, los métodos constructores **no tienen valor de retorno ni parámetros de envío.** Este método se emplea para **inicializar las variables** de la clase o parte de ellas. Los valores utilizados para ello pueden ingresar como parámetros al método constructor o pueden leerse dentro del método. Por ejemplo, para inicializar la clase Persona podemos crear un método constructor que tenga todos los valores para inicializar las variables de la clase como parámetros que ingresan al método:

```
PUBLICO Persona (CADENA n, ENTERO an, CARÁCTER s, REAL p, REAL e)

        Nombre=n
        añoNacimiento=an
        sexo=s
        peso=p
        estatura=e

FIN_persona
```

También podemos crear el método constructor sin parámetros de entrada, y en el contenido del mismo método se leen las variables de la clase que se quieren inicializar:

```
PUBLICO Persona()

        ESCRIBA: “DIGITE EL NOMBRE”
        LEA: nombre
        ESCRIBA: “DIGITE AÑO DE NACIMIENTO”
        LEA: añoNacimiento
        ESCRIBA: “DIGITE EL SEXO”
        LEA: sexo
        ESCRIBA: “DIGITE EL PESO”
        LEA: peso
        ESCRIBA: “DIGITE LA ESTATURA”
        LEA: estatura

FIN_Persona
```

También pueden existir constructores que inicializan solo algunas de las variables, o incluso que no inicializan ninguna de las variables. En estos casos las variables pueden ser inicializadas en

otros métodos diferentes al constructor. Ahora, por ejemplo, presentamos un constructor para la clase Persona que inicializa solo algunas variables de la clase:

```
PUBLICO persona (CADENA n, CARÁCTER s)

    Nombre=n
    DIGITE: "AÑO NACIMIENTO"
    LEA: añoNacimiento
    sexo=s

FIN_Persona
```

Este es un constructor que inicializa con valores por defecto:

```
PUBLICO Persona()

    Nombre= "     "
    añoNacimiento=0
    sexo= "     "
    estatura=0

FIN_Persona
```

Aquí tenemos un constructor vacío, es decir que no inicializa ninguna de las variables de la clase persona:

```
PUBLICO Persona ()

    ESCRIBA: "CREANDO UN OBJETO DE LA CLASE PERSONA"

FIN_Persona
```

Mientras que el constructor se ejecuta automáticamente al instanciar una clase, existe otro método, llamado destructor, que se ejecuta automáticamente al destruirse la clase. Este método comúnmente es opcional y se utiliza para dar instrucciones finales como liberar memoria, cerrar archivos, desconectarse de base de datos o simplemente despedirse del usuario. El nombre de este método se forma anteponiendo una virgulilla "~" al nombre de la clase.

```
PUBLICO ~Persona ()

    ESCRIBA: "DESTRUYENDO EL OBJETO"
```

FIN_~Persona

2.3.13 EJERCICIO DE APRENDIZAJE

Crear un algoritmo que implemente la clase Persona correspondiente al siguiente diagrama:

Persona
+nombre +añoNacimiento +sexo -peso -estatura
+Persona(CADENA nom, ENTERO an, CARÁCTER sex, REAL pe, REAL est) +calcularIndiceMasaCorporal () +calcularEdad () +~Persona

Utilizando la codificación de la clase Persona presentada anteriormente, la clase completa con el método constructor y destructor es:

```
CLASE Persona

    PUBLICO CADENA nombre
    PUBLICO ENTERO añoNacimiento
    PUBLICO CARÁCTER sexo
    PRIVADO REAL peso
    PRIVADO REAL estatura
    PUBLICO Persona (CADENA nom, ENTERO an, CARÁCTER sex, REAL pe, REAL est)

        Nombre=nom
        añoNacimiento=an
        sexo=sex
        peso=pe
        estatura=est
```

```
        FIN_Persona

        PUBLICO VOID calcularMasaCorporal()

                REAL índice

                        Índice=peso/estatura^2
                        ESCRIBA: “EL INDICE ES;”, índice

        FIN_CalcularMasaCorporal
        PUBLICO VOID calcularEdad()

                ENTERO edad
                edad=2015-añoNacimiento
                ESCRIBA:”LA EDAD ES;”, edad

        FIN_calcularEdad

        PUBLICO ~Persona()
        FIN_~Persona()

FIN_CLASE
```

2.3.14 INSTANCIAR OBJETOS

De la clase Persona se pueden crear todos los objetos que sean necesarios, es decir, con la plantilla de la clase se pueden crear muchas personas. Este proceso es llamado **instanciación de la clase**. Para realizarlo se debe tener en cuenta que el constructor se **ejecuta automáticamente**; entonces se le deben enviar los argumentos que necesite este método, si los tiene. Para instanciar objetos de la clase Persona tenemos:

```
Persona objeto1, objeto2
objeto1=Persona (“SAMUEL”, 2010, “H”, 13.7, 90.5)
objeto2=Persona (“MARIA ISABEL”, 2002, “M”, 45, 140)
```

En la primera línea se están definiendo las dos variables que **almacenaran los objetos**, y en las dos siguientes se está invocando al **constructor de la clase** para que **cree los objetos**. Al instanciar estos dos objetos, en la memoria RAM se separa espacio para los dos objetos compuestos por todas las variables y los métodos de la clase Persona:

MEMORIA RAM

Objeto 1: Persona	Objeto 1: Persona
Nombre= SAMUEL añoNacimiento= 2010 sexo= H peso= 13.7 estatura= 90.5	Nombre= MARIA ISABEL añoNacimiento= 2002 sexo= M peso= 45 estatura= 140
+Persona(CADENA nom, ENTERO an, CARÁCTER sex, REAL pe, REAL est) +calcularIndiceMasaCorporal () +calcularEdad () +~Persona	+Persona(CADENA nom, ENTERO an, CARÁCTER sex, REAL pe, REAL est) +calcularIndiceMasaCorporal () +calcularEdad () +~Persona

2.3.15 DECLARACIÓN DE ATRIBUTOS

En esta sección se trabajará un ejercicio completo donde se tendrá en cuenta la clase, los objetos, la declaración de los atributos de la clase y todos los métodos que sean necesario para solucionar el problema.

Dentro del método principal es posible cambiar o imprimir el valor de las variables de los objetos creados. Esta manipulación se realiza por medio del operador punto (.) siempre que las variables hayan sido declaradas como públicas:

objeto. Variable

Por ejemplo, si es necesario cambiar el nombre del objeto1, entonces se realiza la siguiente operación:

objeto1.nombre=”DAVID”

Si se requiere mostrar por pantalla el sexo del objeto2, entonces:

ESCRIBA: objeto2.sexo

Por otro lado, para utilizar los métodos de las clases, la invocación se realiza desde el objeto por medio del operador punto (.) de la forma:

- Si el método es una función que retorna valor:
  retorno=objeto.Metodo()
- Si no retorna valor:
  objeto.Metodo()

Por ejemplo, siguiendo con la clase Persona y los objetos instanciados de esta clase, se puede invocar el método calcularEdad() del objeto2 con la siguiente instrucción:

objeto2.CalcularEdad()

Es de anotar que este método no tiene valor de retorno y produce una impresión por pantalla:

Finalmente, se obtiene un método principal con las siguientes instrucciones:

```
METODO PRINCIPAL
    Persona objeto1, objeto2
    objeto1=Persona (“SAMUEL”, 2010, “H”, 13.7, 90.5)
    objeto2=Persona (“MARIA ISABEL”, 2002, “M”, 45, 140)
    objeto1.nombre=”DAVID”
    ESCRIBA: ”EL SEXO ES:”, objeto2.sexo
    objeto2.calcularEdad()
FIN_PRINCIPAL
```

En la primera instrucción del método principal se declaran los dos objetos. En las siguientes dos instrucciones se le envían argumentos a la función constructora para inicializar los objetos (darles valores a cada una de sus variables). Estas dos instrucciones se pueden simplificar en una sola así:

```
Persona objeto1=Persona (“SAMUEL”, 2010, “H”, 13.7, 90.5)
Persona objeto2=Persona (“MARIA ISABEL”, 2002, “M”, 45, 140)
```

La ejecución de este método principal produce la siguiente impresión por pantalla:

2.3.16 EJERCICIO DE APRENDIZAJE

Ejercicio

Construir un programa que permita realizar las operaciones de suma, resta, multiplicación y división de dos números ingresados por el usuario.

1. **Identificación de datos, acciones y limitaciones**

- Datos de entrada: numero1 y numero2.
- Datos de salida: resultado de las operaciones.
- Acciones: suma, resta, multiplicación y división.
- Limitaciones: todas las operaciones se realizan con los mismos números.

2. **Definición de clases**

- Identificación de posibles clases (sustantivos relevantes): operacion números, usuarios.
- Relación de los sustantivos con los datos y las acciones:
- Operacion: número 1, número 2, sumar, restar, multiplicación y división.
- Números: N/A.
- Usuario: N/A.
- Clase seleccionada: Operación.
- Definición de constructores y destructores para cada clase:

- Operación(): este constructor lee los datos miembro de la clase.
- Diagramación de clases con sus variables y métodos:

Operación
-numero1 -numero2
+Operación() +sumar(): REAL +restar(): REAL +multiplicar(): REAL +dividir()

Explicación de los métodos:

- **Operación:** Constructor que lee las variables de la clase.
- **Sumar:** Calcula la suma de las variables de la clase y retorna el resultado tipo REAL.
- **Restar:** Calcula la resta de las variables y retorna el resultado tipo REAL
- **Multiplicar:** Calcula la multiplicación de las variables y retorna el resultado de tipo REAL.
- **Dividir:** Calcula la división de las variables y muestra el resultado por pantalla

3. **Definición del método principal**

Se debe crear un objeto de la clase Operación que permita invocar los diferentes métodos de la clase.

Definición de variables:

- **Objeto:** Objeto de la clase Operación para invocar los métodos de la clase.
- **Opción:** Variable para controlar la selección menú.
- **Respuesta:** Variable para almacenar los resultados de las operaciones.

Algoritmo

```
CLASE Operación
    PRIVADO REAL numero1
    PRIVADO REAL numero2
    PUBLICO Operacion()
        ESCRIBA: “DIGITE EL PRIMER NUMERO”
        LEA: numero1
        ESCRIBA:” DIGITE EL SEGUNDO NUMERO”
        LEA: numero2
        FIN Operación
        PUBLICO REAL sumar()
            REAL resultado
            resultado=numero1 + numero2
            RETORNE resultado
        FIN_sumar
        PUBLICO REAL restar()
            REAL resultado=numero1 – numero2
            RETORNE resultado
        FIN_restar
        PUBLICO REAL multiplicar()
            REAL resultado=numero1*numero2
            RETORNE resultado
        FIN_multiplicar
        PUBLICO VOID dividir()
            SI (numero2==0) ENTONCES
                ESCRIBA: “ERROR EN LA OPERACIÓN”
            SI_NO
                REAL resultado=numero1/numero2
                ESCRIBA: “DIVISION=”, resultado
            FIN_SI
        FIN_dividir
```

```
METODO PRINCIPAL ()
    ENTERO opción
    REAL respuesta
        Operación objeto=Operación()
    HAGA
        ESCRIBA: “MENU”
        ESCRIBA: “1: SUMAR”
        ESCRIBA: “2: RESTAR”
        ESCRIBA: “3: MULTIPLICAR”
        ESCRIBA: “4: DIVIDIR”
        ESCRIBA: “5: SALIR DEL MENU”
        ESCRIBA:” DIGITE OPCION”
        LEA: opción
        CASO DE (opción)
            CASO 1: respuesta=objeto.Sumar()
                ESCRIBA: “LA SUMA ES:”, respuesta
            CASO 2: ESCRIBA: “LA RESTA ES:”, objeto.restar()
            CASO 3: ESCRIBA: “LA MULTIPLICACION ES:”,
                    objeto.multiplicar()
            CASO 4: objeto.dividir()
        FIN_CASOS
    MIENTRAS (opción<>5)
FIN_PRINCIPAL
FIN_CLASE
```

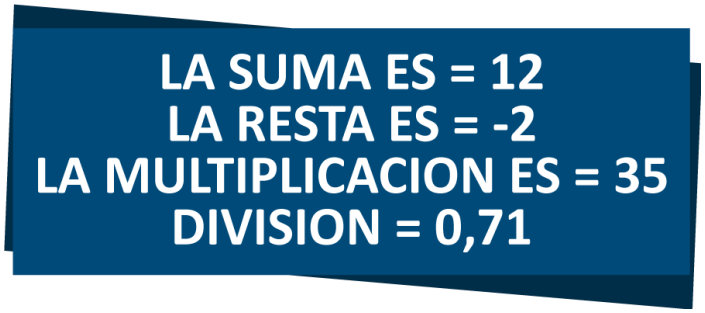
Prueba de escritorio

Si los valores ingresados en el constructor son 5 y 7, entonces el objeto tendrá los siguientes valores en sus variables de clase:

Objeto: Operación

-numero1= 5 -numero2= 7
+Operación() +sumar(): REAL +restar(): REAL +multiplicar(): REAL +dividir()

Salida



Elementos que se deben de tener en cuenta para solucionar problemas mediante la programación orientada a objeto (POO)

1. Para solucionar un problema mediante POO se debe tener en cuenta los siguientes elementos:

- Identificación de datos, acciones y limitaciones: se debe revisar cuidadosamente la descripción del problema para identificar los datos de entrada, los datos de salida, los procesos o acciones requeridas y las restricciones existentes.

2. Definición de clase: para definir las clases que se deben implementar es necesario realizar las siguientes actividades:

- Identificar las posibles clases buscando los sustantivos relevantes dentro del enunciado que no sean datos de entrada o de salida.

- Relacionar los sustantivos con los datos de entrada, los datos de salida y las acciones identificadas.

- Seleccionar como clases aquellos sustantivos que tienen asignado al menos un dato o al menos una acción. Los demás sustantivos son eliminados. Los datos asignados se convierten en las variables miembro y las acciones asignadas se convierten en los métodos miembro de las clases.

- Cada clase debe poseer al menos un método constructor. Se sugiere utilizar siempre este método para inicializar todos o algunos de los datos miembro de la clase. El método destructor es opcional.

- Nombrar cada clase con un sustantivo en singular y diagramarla con las variables, los métodos y sus respectivos modificadores de acceso

3. Definición del método principal: se debe analizar cuantas instancias de las clases son necesarias, es decir, cuantos objetos deben ser creados de cada clase. Igualmente, se debe identificar cuales métodos deben ser invocados para cada objeto.

4. Creación de pseudocódigo: este es el paso final para solucionar un problema mediante POO. Se debe partir del diagrama de clases.

5. Prueba de escritorio: este paso es muy importante para verificar que los datos de salida sean correctos.

2.3.17 TALLER DE ENTRENAMIENTO

Una entidad bancaria quiere disponer de una aplicación que cree una cuenta de un cliente y luego permita realizar movimientos sobre ella hasta que el usuario responda que no quiere realizar más. Cuando esto ocurra debe mostrar en pantalla el nombre de usuario, su número de identificación, el saldo de la cuenta y la cantidad de movimientos de depósito y de retiro realizados por él.

TIPS

La programación orientada a objetos la reutilización de código es un elemento fundamental para realizar algoritmos eficientes.

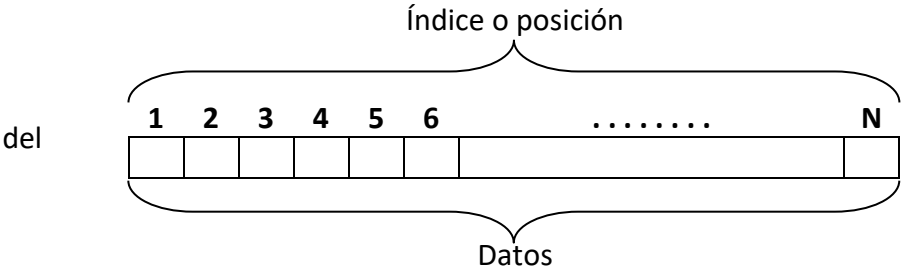
2.4 TEMA 3 ARREGLOS

- ***Video: " ¿Qué son los ARREGLOS en PROGRAMACIÓN?"***
- ***" Ver Video":*** *https://www.youtube.com/watch?v=4s2D-pr8Klo"*

2.4.1 VECTORES

Un vector es una colección finita, homogénea y ordenada de datos del mismo tipo que se almacenan en posiciones consecutivas de memoria y que reciben un nombre en común. Para referirse a un determinado elemento de un vector se debe utilizar un subíndice que especifique su posición en el arreglo.

Veamos la representación de un vector:



Para situarnos en una posición específica dentro de un vector; primero nombramos a nuestro vector, seguido del número de la posición en la cual nos ubicaremos, así: **vec[k]**; siendo **vec** el nombre de nuestro vector, y **k** la posición en él.

Por ejemplo, un vector con nombres:

	1	2	3	4	5	6
vnom	Ana	Pedro	Luis	Raúl	María	José

vnom[1] = “Ana”
vnom[2] = “Pedro”
vnom[3] = “Luis”
vnom[4] = “Raúl”
vnom[5] = “María”
vnom[6] = “José”

Existen 3 casos para almacenar información en un vector:

1. Cuando el tamaño del vector es constante
2. Cuando el tamaño del vector está dado por N
3. Cuando se desconoce el tamaño del vector

- **Tamaño constante**

```
1.  CLASE VectorConstante
2.      METODO PRINCIPAL ()
3.          VARIABLES: vnom[] (ALFANUMÉRICO)
4.                     i (ENTERO)
5.              PARA (i= 1, 10, 1)
6.                     IMPRIMA (“Ingrese nombre”)
7.                     LEA (vnom[i])
8.              FINPARA
9.              PARA (i= 1, 10, 1)
10.                    IMPRIMA (vnom[i])
11.             FINPARA
12.     FIN(Método)
13. FIN(Clase)
```

- **Tamaño N**

```
1.  CLASE VectorN
2.  METODO PRINCIPAL ()
3.      VARIABLES: vnom[] (ALFANUMÉRICO)
4.             i, n (ENTERO)
5.             IMPRIMA (“Tamaño del vector”)
6.             LEA (n)
7.             PARA (i= 1, n, 1)
```

```
8.                  IMPRIMA (“Ingrese nombre”)
9.                  LEA (vnom[i])
10.           FINPARA
11.           PARA (i= 1, n, 1)
12.                 IMPRIMA (vnom[i])
13.           FINPARA
14.     FIN(Método)
15.  FIN(Clase)
```

- **Tamaño desconocido**

```
1.  CLASE VectorDesconocido
2.        METODO PRINCIPAL ()
3.           VARIABLES: vnom[], nom (ALFANUMÉRICO)
4.                       n (ENTERO)
5.                 n = 0
6.                 IMPRIMA (“Ingrese nombre”)
7.                 LEA (nom)
8.                 MQ (nom != “xx”)
9.                       n = n + 1
10.                      vnom[n] = nom
11.                      IMPRIMA (“Ingrese nombre”)
12.                      LEA (nom)
13.                FINMQ
14.                PARA (i= 1, n, 1)
15.                      IMPRIMA (vnom[i])
16.                FINPARA
17.       FIN(Método)
18.  FIN(Clase)
```

- **Ejemplo:**

Consideramos la situación de que se hizo un censo en la ciudad de Medellín y que se grabo en archivo en disco, llamado “censo”, el cual contiene la siguiente información en cada registro: municipio, dirección y número de personas. Cada registro contiene los datos correspondientes a cada vivienda visitada.

Interesa procesar el archivo y calcular el total de personas que viven en Medellín

Definimos las siguientes variables:

- **mpio:** variable en la cual se almacena el código del municipio.
- **dir:** variable en la cual se almacena la dirección de la vivienda visitada.
- **np:** variable en la cual se almacena el número de personas de la vivienda visitada.
- **acmed:** variable en la cual llevaremos el acumulado de las personas que viven en Medellín.

Un algoritmo para procesar el archivo "censo" es el siguiente:

```
1.  CLASE CensoMedellin
2.    METODO PERINCIPAL ()
3.      VARIABLES: mpio, acmed, np (ENTEROS)
4.                 dir: alfanumérica
5.                    acmed = 0
6.                    ABRA (censo)
7.                    MQ (NOT EOF (censo))
8.                          LEA (censo: mpio, dir, np)
9.                          acmed = acmed + np
10.                   FINMQ
11.                   CIERRE (censo)
12.                   IMPRIMA ("Personas que viven en Medellín", acmed)
13.    FIN(Método)
14.  FIN(Clase)
```

Si el censo se hace en dos municipios (Medellín y bello), y los códigos asignados a los municipios son:

mpio: { 1. Medellín (**acmed**)
2. Bello (**acbel**)

Un algoritmo para procesar el archivo "censo" e imprimir el total de habitantes de cada municipio es:

```
1.  CLASE CensoMedellinBello
```

```
2.    METODO PRINCIPAL ()
3.      VARIABLES: mpio, acmed, acbel, np: (ENTEROS)
4.                      dir: (ALFANUMÉRICAS)
5.              ABRA(censo)
6.              acmed = 0
7.              acbel = 0
8.              MQ (NOT EOF (censo))
9.                      LEA(censo: mpio, dir, np)
10.                     SI (mpio = 1)
11.                             acmed = acmed + np
12.                     SINO
13.                             acbel = acbel + np
14.                     FINSI
15.             FINMQ
16.             IMPRIMA("habitantes Medellín:", acmed, "habitantes Bello:", acbel)
17.             CIERRE(censo)
18.     FIN(Método)
19.  FIN(Clase)
```

Observe que en este algoritmo ya necesitamos dos acumuladores: uno para el total de habitantes en Medellín (**cmed)** y otro para el total de habitantes de bello (**cbel)**.

Si el censo hubiera sido para Medellín, Bello, Itagüí y Envigado; se le asigna un código a cada municipio:

mpio:
1. Medellín: (**acmed**)
2. Bello: (**acbel**)
3. Itagüí: (**acita**)
4. Envigado: (**acenv**)

Nuestro algoritmo es:

```
1.  CLASE CensoAntioquia
2.      MÉTODO PRINCIPAL ()
3.          VARIABLES: mpio, acmed, acbel, acita, acenv, np (ENTEROS)
4.                          dir (ALFANUMÉRICO)
```

```
5.              ABRA(censo)
6.              acmed = 0
7.              acbel = 0
8.              acita = 0
9.              acenv = 0
10.             MQ (NOT EOF (censo))
11.                     LEA(censo: mpio, dir, np)
12.                     CASOS
13.                             CASO (mpio == 1)
14.                                     acmed = acmed + np
15.                             SALTE
16.                             CASO (mpio == 2)
17.                                     acbel = acbel + np
18.                             SALTE
19.                             CASO (mpio == 3)
20.                                     acita = acita + np
21.                             SALTE
22.                             CASO (mpio == 4)
23.                                     acenv = acenv + np
24.                             SALTE
25.                     FINCASOS
26.             FINMQ
27.             CIERRE(censo)
28.             IMPRIMA ("habitantes Medellín:", acmed, "habitantes bello:", acbel)
29.             IMPRIMA ("habitantes Itagüí:", acita, "habitantes envigado:", acenv)
30.     FIN(Método)
31. FIN(Clase)
```

Observe que en este nuevo algoritmo ya se necesita cuatro acumuladores: uno por cada municipio que se procese.

Si el censo hubiera sido para los 125 municipios del departamento, se requerirían 125 acumuladores: uno para cada municipio. Tendríamos que manejar 125 variables, nuestra instrucción CASOS sería muy extensa y el algoritmo sería completamente impráctico, ya que cada vez que varíe el número de municipios debemos modificar nuestro algoritmo.

2.4.2 SOLUCIÓN AL PROBLEMA. CONCEPTO DE VECTOR

Presentaremos una solución alterna utilizando una estructura arreglo en la cual definimos un área de memoria con cierto número de posiciones, e identificamos cada posición con un número entero.

Veamos dicha estructura:

acmpio

1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	1	6	28	9	4	5	7	6	3	2	7	5	8

Nuestra estructura la hemos llamado **acmpio** y tiene una capacidad de catorce elementos.

Para referirnos a algún elemento lo hacemos con el nombre de la estructura y un subíndice que indique la posición de dicho elemento.

Si nuestra estructura tiene los datos que se muestran y queremos imprimir el dato que se halla la posición 4 escribimos:

```
IMPRIMA (acmpio [4])
```

Y el resultado de ejecutar esta instrucción será mostrar el número 28.

Dicha estructura se conoce como *un* ***arreglo de una dimensión*** y en el contexto de algoritmos se denomina ***vector.***

En general, para referirnos al contenido de una posición lo hacemos con el nombre del vector y un entero que se refiere a la posición. Dicho entero lo escribimos entre corchetes y lo seguimos llamado ***subíndice***.

Para trabajar con un vector, lo primero que se debe hacer es construir el vector. Para ilustrar dicha construcción consideremos el siguiente algoritmo, que llamaremos ***censo***, el cual es una variación del algoritmo desarrollado anteriormente:

```
1.  CLASE Censo
2.     METODO PRINCIPAL ()
3.           VARIABLES: mpio, np, sum, acmpio[125] (ENTEROS)
4.                            dir (ALFANUMÉRICA)
5.                     ABRA(censo)
6.                     PARA (i= 1, 125, 1)
```

```
7.                      acmpio[i] = 0
8.              FINPARA
9.              MQ (NOT EOF(censo))
10.                     LEA(censo:mpio, dir, np)
11.                     acmpio[mpio] = acmpio[mpio] + np
12.             FINMQ
13.             CIERRE (censo)
14.             Censo.ImprimeVector (acmpio, 125)
15.             sum = Censo.SumarVector (acmpio, 125)
16.             IMPRIMA (sum)
17.    FIN(Método)
18.  FIN(Clase)
```

En este nuevo algoritmo ya no utilizamos una variable para cada acumulador que se necesite manejar, sino que definimos una sola variable, la cual llamamos **acmpio,** que tiene características de vector, y tendrá 125 posiciones. Este tamaño (125 posiciones) se define con base en el conocimiento que se tiene acerca del problema que se va a resolver. En nuestro ejemplo debemos manejar el total de habitantes de cada uno de los 125 municipios que tiene el departamento y por tanto el vector de acumuladores deberá tener este tamaño. Esta definición aparece en la instrucción 3 de la clase Censo en el método principal.

Definimos la variable **acmpio** con una capacidad de 125 elementos.

En las instrucciones 6 a 8 inicializamos cada una de esas posiciones en cero.

En la instrucción 11 de dicho algoritmo se configura cada una de las posiciones del vector **acmpio,** sumándole a la posición **mpio** el número de personas que se lee en cada registro.

En la instrucción 14 invocamos a un método llamado *ImprimeVector,* con el cual recorremos e imprimimos los datos del vector que se ha construido.

En la instrucción 15 invocamos un método llamado *SumarVector,* con el cual se suman todos los datos que se hallan en las diferentes posiciones del vector.

Ya hemos visto en nuestro algoritmo anterior que cuando se termina la construcción de un vector interesa conocer cuál es el valor almacenado en cada una de las posiciones del vector. Para efectuar esta tarea invocamos un método denominado *ImprimeVector*; el cual veremos a continuación:

```
1.  PUBLICO ESTATICO VOID ImprimeVector (V, n)
2.      VARIABLES: i (ENTERO)
3.              PARA (i= 1, n, 1)
4.                      IMPRIMA (i, V[i])
5.              FINPARA
6.  FIN(Método)
```

Este programa es bastante simple: los parámetros son el nombre del vector y el tamaño del vector.

Solo se requiere una variable local para recorrer las diferentes posiciones del vector. A dicha variable la llamamos **i**.

En instrucciones 3 a 5 escribimos un ciclo PARA, con el cual nos movemos en el vector desde la posición 1 hasta la posición **n**, imprimiendo el subíndice y el contenido de dicha posición.

2.4.3 SUMA DE LOS DATOS DE UN VECTOR

Es también muy frecuente determinar el total de los datos almacenados en un vector. Para ello definimos un método *función,* al cual llamamos *SumarVector*, que efectué esta suma y retorne el resultado de ella al método principal: :

```
1.  PUBLICO ESTATICO ENTERO SumarVector (V,n)
2.      VARIABLES: i, s( ENTEROS)
3.              s = 0
4.              PARA (i= 1, n, 1)
5.                      s = s + V[i]
6.              FINPARA
7.              RETORNE (s)
8.  FIN(Método)
```

Este programa es también bastante simple. Tiene como parámetros de entrada el nombre del vector y el número de elementos en él.

Se requieren **dos variables locales**:

- Una para recorrer el vector, y

- Otra en el cual se maneje el acumulado.

A dichas variables las llamamos **i** y **s**, respectivamente.

Es la instrucción 3 se inicializa el acumulador **s** en cero.

En instrucciones 4 a 6 escribimos nuestro ciclo de recorrido, y en vez de imprimir el contenido de cada posición, lo acumulamos en la variable llamada **s**.

En la instrucción 7 se retorna el contenido de **s** al método llamante.

2.4.4 OPERACIONES BÁSICAS: MAYOR DATO Y MENOR DATO EN EL VECTOR

Ya hemos visto que en un vector se almacenan los datos correspondientes a un conjunto de **situaciones** o **eventos**. Es importante conocer la situación o evento que más veces se presenta. Para lograr esto hay que elaborar un algoritmo que determine o identifique la posición en la cual se halla la situación que más veces ocurre.

A continuación, presentamos un método tipo función, que retorna en una variable la posición en donde se encuentra el mayor de los datos almacenados en el vector:

```
1.  PUBLICO ESTATICO ENTERO MayorDato (V, m)
2.      VARIABLES: mayor, i (ENTEROS)
3.              mayor = 1
4.              PARA (i= 2, m, 1)
5.                      SI (V[i] > V[mayor])
6.                              mayor = i
7.                      FINSI
8.              FINPARA
9.              RETORNE (mayor)
10. FIN(Método)
```

Para ilustrar como funciona este método consideremos el siguiente gráfico:

m = 7

	1	**2**	**3**	**4**	**5**	**6**	**7**
V	3	1	6	2	8	9	4

En la instrucción 2 se define dos variables: **mayor**, para almacenar la posición en la cual se halla el dato mayor, e **i**, para recorrer el vector e ir comparando el dato de la posición **i** con el dato de la posición **mayor.**

En la instrucción 3 se inicializa la variable **mayor** en 1, definiendo que el dato mayor se halla en dicha posición.

En la instrucción 4 se plantea el ciclo con el cual se recorre el vector desde la posición 2 hasta la posición m.

En la instrucción 5 se compara el dato de la posición **i** con el dato de la posición **mayor**. Si el dato que está en la posición **i** es mayor que el dato que está en la posición **mayor** se actualiza el contenido de la variable mayor en la instrucción 6.

Al terminar el ciclo, cuando la **i** es mayor que la **m**, en la variable **mayor** estará almacenada la posición en la cual se halla el mayor dato del vector.

Para nuestro ejemplo, **mayor** se inicializará en 1, lo cual significa que el mayor dato está en la posición 1 del vector **V**, es decir, el mayor dato es **V[mayor],** o sea 3.

Cuando se ejecuta el ciclo, **i** comienza valiendo 2, o sea que **V[i]** es 1. Al comparar **v[i]** con **V[mayor]** en la instrucción 5, el resultado es falso, por consiguiente, no ejecuta la instrucción 6 y continúa con la instrucción 8, la cual incrementa el valor de **i** en 1 y regresa a la instrucción 4 a comparar **i** con **m.**

Como **i** es menor o igual que **m,** vuelve a ejecutar las instrucciones del ciclo: compara **V[i]** con **V[mayor]**, es decir, compara **V[3]** con **V[1].** El dato de la posición **i** es mayor que el dato de la posición **mayor,** por tanto, ejecuta la instrucción 6, o sea que modifica el contenido de la variable **mayor**, el cual será 3, indicando que el mayor dato se halla en la posición 3 del vector.

Se llega de nuevo a la instrucción 8 e incrementa la **i** en 1 (**i** queda valiendo 4) y regresa de nuevo a la instrucción 4 a continuación ejecutando las instrucciones del ciclo hasta que la **i** sea mayor que la **m.**

Cuando la **i** sea mayor que la **m** termina la ejecución del ciclo y en la variable **mayor** quedara la posición en la cual se halla el mayor dato del vector.

Observe que hemos trabajado con las posiciones del vector, ya que con ellas tenemos acceso directo a los datos que se hallan en dichas posiciones.

En el ejemplo anterior desarrollamos un método en el cual se determina la posición en la cual se halla el mayor dato de un vector. Es también necesario, en muchas situaciones, determinar la posición en la cual se halla el menor dato. A continuación, se presenta un subprograma que efectúa esta tarea:

```
1.  PUBLICO ESTATICO ENTERO MenorDato (V, m)
2.      VARIABLES: menor, i (ENTEROS)
3.              menor = 1
4.              PARA (i= 2, m, 1)
5.                      SI (V[i] < V[menor])
6.                              menor = i
7.                      FINSI
8.              FINPARA
9.              RETORNE (menor)
10. FIN(Método)
```

Este algoritmo es similar al algoritmo de determina la posición en la cual se halla el mayor dato. La única diferencia entre este algoritmo y el anterior es que la variable en la cual se retorna el resultado de la búsqueda se denomina **menor,** en vez de **mayor,** y la comparación de **V[i]** con **V[menor]** se hace con el símbolo menor que (<), en vez del símbolo mayor que (>).

2.4.5 INTERCAMBIAR DOS DATOS EN UN VECTOR

Intercambiar dos datos en un vector es una operación frecuente en manipulación de vectores. Por ejemplo, cuando se desean ordenar los datos de un vector es necesario cambiar los datos de posición, y este cambio se hace por parejas. Un algoritmo que efectúa dicha tarea es el siguiente:

```
1.  PUBLICO ESTATICO VOID Intercambiar (V, i, k)
2.      VARIABLES: aux (ENTERO)
3.              aux = V[i]
4.              V[i] = V[k]
5.              V[k] = aux
6.  FIN(Método)
```

En la instrucción 1 se define el método con sus parámetros: **V**, el nombre del vector en el cual se desea hacer el intercambio, e **i** y **k,** las posiciones cuyos datos se desean intercambiar.

En la instrustración 2 se define una variable, llamada **aux,** que se utilizara para efectuar el intercambio.

En la instrucción 3 se guarda el contenido de la posición **i** del vector en la variable auxiliar **aux** con el fin de que dicho dato no se pierda.

En la instrucción 4 se lleva a la posición **i** del vector lo que hay en la posición **k** del vector.

En la instrucción 5 se lleva lo que hay en la variable auxiliar, es decir, lo que había en la posición **i** del vector, a la posición **k** del vector.

Como ejemplo, consideremos el vector que veremos a continuación, y que se desea intercambiar el dato que se halla en la posición 2 **(i == 2)** con el dato que se halla en la posición 5 **(k == 5).**

	1	**2**	**3**	**4**	**5**	**6**	**7**
V	3	1	6	2	8	9	4

Al ejecutar nuestro método *Intercambiar*, el vector quedara como se ve en la siguiente gráfica:

	1	**2**	**3**	**4**	**5**	**6**	**7**
V	3	8	6	2	1	9	4

2.4.6 OPERACIONES CON VECTORES

Consiste en la definición de los métodos asociados a la clase arreglo y que permite el **procesamiento de la información** de **los datos almacenados en la estructura**. Entre los métodos más importantes asociado a la clase se pueden destacar:

- ✓ Crear,
- ✓ insertar,
- ✓ borrar,
- ✓ ordenar, y
- ✓ Mostrar entre otros.

2.4.6.1 PROCESO DE BORRADO EN UN VECTOR

Para borrar un dato primero debemos determinar en qué posición se encuentra dicho dato. El proceso para determinar en cual posición del vector se halla un dato es bastante similar al proceso de buscar en donde insertar. En el siguiente método presentamos dicho proceso:

```
1. PUBLICO ESTATICO ENTERO BuscarPosicionDato (V, m, d)
2.       VARIABLES: i (ENTERO)
3.                 i = 1
4.                 MQ (i <= m) ^ (V[i] != d)
5.                           i = i + 1
6.                 FINMQ
7.                 RETORNE (i)
8. FIN(Método)
```

La única diferencia de este algoritmo con el algoritmo para buscar donde insertar es que la instrucción MIENTRAS la comparación de **V[i]** con **d** se efectúa con el operador diferente (!=) y no con el operador menor (<).

Nuestro algoritmo de buscar dato funciona independiente de que los datos se encuentren ordenados o no. En caso de que el dato a buscar no se halle en el vector, nuestro algoritmo retorna en la variable **i** el valor de **m** + **1,** lo cual usaremos como condición para detectar si el dato que se está buscando se halla en el vector o no. En otras palabras, si **i** es igual a **m** + **1** el dato no está en el vector **V.**

Como ejemplo, consideremos el siguiente vector, y que deseamos borrar el dato 2:

							m			n
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
V	3	1	6	2	8	9	4			

Al ejecutar nuestro método *BuscarPosicionDato,* el contenido de la variable **i** será 4; es decir, en la posición 4 del vector se halla el dato 2:

				i			m			n
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
V	3	1	6	2	8	9	4			

Conocimos la posición **i** en la cual se halla el dato a borrar, el proceso de borrado implica que hay que mover todos los datos que se encuentran desde la posición **i + 1** hasta la posición **m** una posición hacia la izquierda y restarle 1 a **m.**

En el siguiente método se efectúa dicho proceso:

```
1.  PUBLICO ESTATICO VOID Borrar (V, m, i)
2.      VARIABLES: j (ENTERO)
3.          SI (i > m)
4.              IMPRIMA ("El dato no existe")
5.          SINO
6.              PARA (j= i, m, 1)
7.                  V[j] = V[j+1]
8.              FINPARA
9.              m = m – 1
10.         FINSI
11. FIN(Método)
```

En la instrucción 1 se define el método con sus parámetros: **V**, variable que contiene el nombre del vector en el cual hay que efectuar el proceso de borrar; **m,** variable que contiene el número de posiciones utilizadas en el vector; e **i**, variable que contiene la posición del dato a borrar.

En la instrucción 2 se define la variable **j,** la cual se utilizará para mover los datos del vector en caso de que sea necesario.

En la instrucción 3 se controla que el dato a borrar exista en el vector, en caso de que el dato no exista (**i > m**) se ejecuta la instrucción 4, la cual produce el mensaje de que el dato no está en el vector y finaliza sin ejecutar más acciones.

En caso de que el dato este en el vector se ejecuta la instrucción 6.

En la instrucción 6 se plantea la instrucción PARA, la cual inicializa el valor de **j** con el contenido de **i**, compara el valor de **j** con el valor de **m** y define la forma como variara el valor de **j.** La variación de la variable **j** será sumándole 1 cada vez que llegue a la instrucción FINPARA. Cuando el valor de **j** sea mayor que **m** se sale del ciclo. En caso de que el valor inicial de **j** sea mayor que **m** no ejecutara la instrucción 7 y continuará con la instrucción 9.

En la instrucción 7 se mueve el dato que está en la posición **j+1** hacia la posición **j.**

En la instrucción 8, que es el fin de la instrucción PARA, incrementa el valor de **j** en 1 y regresa a la instrucción 6 a comparar el valor de **j** con **m.** sí **j** es menor que **m**, ejecutará nuevamente las instrucciones 7 y 8, hasta que **j** sea mayor que **m**.

Cuando se sale del ciclo, ejecuta la instrucción 9, en la cual se le resta 1 a **m**, indicando que el vector ha quedado con un elemento menos.

Para nuestro ejemplo, al terminar de ejecutar el método *Borrar* el vector queda así:

						m				n
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
V	3	1	6	8	9	4				

NOTA: Es conveniente, en este punto, hacer notar algo que es muy importante: en la figura anterior, y solo con fines explicativos, hemos dejado el conjunto de la posición 7 en blanco, indicando que se ha borrado un dato. En realidad, en la posición 7 del vector seguirá estando en 4 hasta que se reemplace por otro valor, pero el contenido de m se ha disimulado en 1, lo cual significa que el dato de la posición 7 ya no pertenece al vector.

2.4.7 MATRICES

Se llama matriz de orden **m*n** a todo conjunto rectangular de elementos a_{ij} dispuestos en **m** líneas horizontales (filas) y **n** verticales (columnas) de la forma:

$$A = \begin{matrix} a_{11} & a_{12} & \dots & a_{1j} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2j} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{ij} & \dots & a_{in} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{mj} & a_{m2} & \dots & a_{mj} & \dots & a_{mn} \end{matrix}$$

Abreviadamente suele expresarse de la forma **A = (a_{ij})** con i **= 1, 2, 3, …, m** y **j = 1, 2, 3, …, n**.; donde los subíndices indican la posición del elemento dentro de la matriz: el primero **(i)** denota la fila y el segundo **(j)** la columna. Por ejemplo, elemento **a_{46}** será el elemento de la fila 4, columna 6.

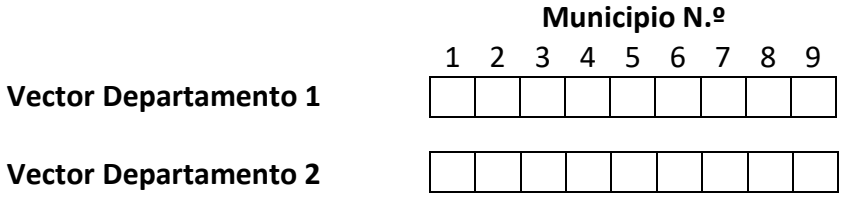
Para situarnos en una posición específica dentro de una matriz; primero nombramos a nuestra matriz, seguido del número de la fila y la columna en la cual nos ubicaremos, así: **mat[i][j]**, siendo **mat** el nombre de nuestra matriz, e **i** y **j** el número de la fila y la columna, respectivamente, en la cual nos ubicamos.

2.4.7.1 IDENTIFICACIÓN Y CONSTRUCCIÓN DE UNA MATRIZ

Con el fin de mostrar la necesidad de utilizar matrices vamos a considerar, de nuevo, un archivo con los datos correspondientes a un censo. En este archivo cada registro contiene la información correspondiente al departamento, municipio, dirección de residencia y número de personas en dicha residencia. Nuestro archivo es el que vemos a continuación:

Departamento	*Municipio*	*Dirección*	*N° personas*
1	3	-	3
1	2	-	1
1	1	-	6
2	1	-	2
2	4	-	8
2	3	-	9
1	3	-	4
EOF			

Si deseamos procesar este archivo y determinar el número de habitantes en cada departamento, debemos definir un vector para cada uno de los **m** departamentos que tenga el país. Veamos cómo sería:



Vector Departamento m

Manejar **m** vectores tiene los mismos inconvenientes que cuando se manejaban **n** variables para representar el número de habitantes de cada municipio de un departamento. Vamos a adoptar ahora una solución análoga: en vez de definir **m** vectores vamos a considerar una nueva estructura, la cual está conformada con filas y columnas. Cada fila es como si fuera un vector y la identificaremos con un subíndice.

En nuestro ejemplo cada fila representa un departamento, y las columnas, los municipios correspondientes a cada departamento. Al procesar completamente el archivo "*censo*", la matriz que nos proporciona la información acerca de los habitantes de cada municipio, de cada departamento, queda de la siguiente forma.

mat						**n**
		1		2	3	4
	1	6		0	7	0
	2	2		0	9	8
m	3	0		1	0	0

Ahora veamos el programa con el cual hemos construido esta matriz:

```
1.  PUBLICO ESTATICO VOID ConstruirMatriz (mat, m, n)
2.      VARIABLES: depto, mpio, np (ENTEROS)
                   direc (ALFANUMÉRICO)
3.          ConstruirMatriz.Inicialice (mat, m, n)
4.          ABRA (censo)
5.          MQ (NOT EOF(censo))
6.                  LEA (depto, mpio, direc, np)
7.                  mat[depto][mpio]= mat[depto][mpio] + np
8.          FINMQ
9.          CIERRE (censo)
10.         ConstruirMatriz.ImprimaPorFilas (mat, m, n)
11. FIN(Método)
```

En la instrucción 1 se define el nombre del método con sus parámetros: la matriz **mat** y sus dimensiones **m** y **n**, siendo **m** el número de filas y **n** el número de columnas.

En la instrucción 2 se definen las variables con las cuales trabajará nuestro algoritmo: **depto**, para el código del departamento; **mpio**, para el código del municipio; **np**, para el número de personas en cada vivienda; y **direc**, para la dirección de cada residencia.

En la instrucción 3 se invoca al método Inicialice el cual nos inicializa la matriz, como veremos en un momento.

En la instrucción 4 abrimos nuestro archivo, al que llamamos "***censo***".

En la instrucción 5 planteamos el ciclo, que se ejecutará mientras que no se llegue a la marca de *fin de archivo* (EOF) de "***censo***".

En la instrucción 6 leemos los datos correspondientes a cada registro.

En la instrucción 7 actualizamos el acumulado correspondiente al departamento y municipio cuyos códigos se leyeron; es decir, a la celda identificada con la fila (**depto**) y columna (**mpio**) se le suma el número de personas que hay en esa dirección.

La instrucción 8 simplemente delimita las instrucciones pertenecientes al ciclo; y en la instrucción 9 cerramos el archivo "***censo***".

En la instrucción 10 invocamos al método *ImprimaPorfilas*, el cual produce el informe correspondiente al total de habitaciones de cada departamento.

2.4.7.2 INICIALIZACIÓN DE LOS DATOS DE UNA MATRIZ

En muchas de las situaciones en las cuales se trabaja con una matriz, esta se usa como un grupo de contadores o un grupo de acumuladores. En el ejemplo que vimos anteriormente, la matriz que se utilizo es realmente un conjunto de acumuladores: cada celda tendrá el total de habitantes de cada municipio de cada departamento. Cuando se trabajan acumuladores, estos se deben **inicializar en cero** para poder obtener resultados confiables.

Como en nuestra matriz del censo cada celda es un acumulador, cada celda se debe inicializar en cero. A continuación, presentamos un método con el cual se ejecuta esta tarea:

```
1. PUBLICO ESTATICO VOID Inicialice (mat, m, n)
```

```
2.      VARIABLES: i, j (ENTEROS)
3.              PARA (i= 1, m, 1)
4.                      PARA (j= 1, n, 1)
5.                              mat[i][j] = 0
6.                      FINPARA
7.              FINPARA
8.  FIN(Método)
```

En la instrucción 1 se define el método con sus parámetros: **mat** es la matriz a la cual se le asignaran los valores iniciales a cada una de sus celdas, **m** es el número de filas y **n** es el número de columnas.

En la instrucción 2 se definimos las variables de trabajo de nuestro método. Llamamos a estas variables **i** y **j.**

En la instrucción 3 planteamos un ciclo con la variable **i**. Con este ciclo vamos a recorrer las filas de la matriz: cuando el contenido de **i** sea 1, significa que vamos a recorrer la fila 1; cuando el contenido de **i** sea 2 significa que vamos a recorrer la fila 2, cuando el contenido de **i** sea 3 significa que vamos a recorrer la fila 3, y así sucesivamente.

En la instrucción 4 planteamos un ciclo interno, el cual vamos a controlar con la variable **j.** Con este ciclo vamos a recorrer los datos correspondientes a cada columna, es decir, cuando **i** contenga el valor correspondiente a una fila, **j** variará desde 1 hasta **n**, que será el número de columnas de la matriz**.**

Y, finalmente, en la instrucción 5 le asignaremos cero (0) a cada celda dentro de la matriz.

2.4.7.3 RECORRIDO E IMPRESIÓN POR FILAS

Cuando se trabaja con matrices una de las tareas más importantes es imprimir el contenido de cada una de las celdas. Para imprimir el contenido de cada una de las celdas de la matriz existen dos formas de hacerlo. Una de ellas es imprimiendo primero el contenido de la fila 1, luego el contenido de la fila 2, luego el de la fila 3 y así sucesivamente. Esta forma de recorrer e imprimir la matriz se hace por filas. El método que sigue ejecuta esta tarea:

```
1.  PUBLICO ESTATICO VOID ImprimaPorFila (mat, m, n)
2.      VARIABLES: i, j (ENTEROS)
3.              PARA (i= 1, m, 1)
```

```
4.                      IMPRIMA (“fila:”, i)
5.                      PARA (j= 1, n, 1)
6.                              IMPRIMA (“columna:”, j, “contenido de la celda:”, mat[i][j])
7.                      FINPARA
8.              FINPARA
9.  FIN(Método)
```

En la instrucción 1 se define el método con sus parámetros: **mat** es la matriz que desea recorrer e imprimir por filas; **m** es el número de filas de **mat**; es el número de columnas de **mat.**

En la instrucción 2 se define las variables de trabajo para efectuar el recorrido.

Las instrucciones de ciclo son completamente similares a las desarrolladas en el método *Inicialice*, con el cual inicializamos cada celda de la matriz en cero, y realmente la inicialización del contenido de cada celda de la matriz, se hizo por filas. Es decir, primero asigna ceros a todas las celdas de la fila 1, luego a todas las celdas de la fila 2 y así sucesivamente.

En nuestro método de impresión incluimos una nueva instrucción, la instrucción 6, en la cual colocamos el título que indique que se van a imprimir los datos de la fila **i**.

La instrucción 5, que es la instrucción correspondiente al ciclo interno, es la instrucción en la cual se imprime el contenido de cada celda de una fila **i**. En dicha instrucción imprimimos el mensaje correspondiente a cada columna y el contenido de la celda descrita por **mat[i][j]**.

2.4.7.4 RECORRIDO E IMPRESIONES POR COLUMNAS

Cuando se desea recorrer e imprimir la matriz por columnas hay que hacer una “pequeña gran” variación a nuestro método de recorrido por filas. En el método que sigue presentamos dicha variación:

```
1.  PUBLICO ESTATICO VOID ImprimaPorColumna (mat, m, n)
2.      VARIABLES: i, j (ENTEROS)
3.              PARA (i= 1, n, 1)
4.                      IMPRIMA(“columna:”, i)
5.                      PARA (j= 1, m, 1)
6.                              IMPRIMA (“fila:”, j, “contenido de la celda:”, mat[j][i])
7.                      FINPARA
8.              FINPARA
```

```
9.  FIN(Método)
```

En la instrucción 3, que es la instrucción correspondiente al ciclo externo, a la variable controladora del ciclo **i** la ponemos a variar hasta **n**, y no hasta **m,** ya que con esta variable **i** vamos a recorrer las columnas: primero los datos de la columna 1, luego los datos de la columna 2, luego de la columna 3 y así sucesivamente.

La instrucción 4 produce el mensaje correspondiente a la columna cuyos datos se van a imprimir junto con el número de dicha columna: el contenido de la variable **i**.

En la instrucción 5, que es la instrucción correspondiente al ciclo interno, a la variable controladora **j** la ponemos a variar hasta **m,** y no hasta **n,** ya que con esta variable **j** vamos a recorrer los datos correspondientes a cada columna, y el total de elementos de cada columna es **m,** el número de filas de la matriz**.**

En la instrucción 6 escribimos cada uno de los datos correspondientes a la columna **i**, es decir, el dato de cada fila **j: mat[j][i].** Observe que en el método de recorrido por filas se imprime **mat[i][j]** y en el método de recorrido por columnas se escribe **mat[j][i].**

2.4.7.5 SUMA DE LOS DATOS DE CADA FILA DE UNA MATRIZ

Sumar los datos de cada fila es una operación que se requiere con frecuencia. Si consideramos la matriz del ejemplo del censo: cada celda contiene el número de habitantes de cada municipio dentro de cada uno de los departamentos. Aquí nos interesa conocer el total de habitantes de cada departamento: la suma de los datos de las celdas de la fila 1 da el total de habitantes del departamento 1; la suma de los datos de las celdas de la fila 2 da el total de habitantes del departamento 2; la suma de los datos de las celdas de la fila 3 da el total de habitantes del departamento 3, y así sucesivamente.

Para efectuar esta operación se necesita recorrer la matriz por filas, es decir, usaremos las instrucciones de recorrido de la matriz por filas, las cuales fueron explicadas en el método *ImprimaPorFila* que vimos anteriormente.

A continuación, presentamos un método que suma e imprime el total de los datos de las celdas de cada fila de una matriz:

```
1.  PUBLICO ESTATICO VOID SumaPorFila (mat, m, n)
```

```
2.      VARIABLES: i, j, sf (ENTEROS)
3.              PARA (i= 1, m, 1)
4.                      sf = 0
5.                      PARA (j= 1, n, 1)
6.                              sf = sf + mat[i][j]
7.                      FINPARA
8.                      IMPRIMA (“suma de la fila:”, i, “es:”, sf)
9.              FINPARA
10. FIN(Método)
```

La instrucción 1 define el método con sus parámetros: **mat,** la matriz en la cual hay que efectuar el proceso de suma de los datos de cada fila; **m**, el número de filas de **mat**; y **n**, el número de columnas de **mat.**

En la instrucción 2 definimos las variables de trabajo para efectuar la tarea de sumar e imprimir el total de los datos de las celdas de cada fila: **i**, para recorrer las filas; **j,** para recorrer las celdas de cada fila, es decir las columnas; **sf**, para llevar el acumulado de los datos de las celdas de la fila **i.**

En la instrucción 3 planteamos el ciclo de variación de **i**, para recorrer la matriz por filas.

En la instrucción 4 se inicializa el acumulador de los datos de cada fila en cero. Este acumulador (**sf**) se debe inicializar en cero cada vez que se cambie de fila, ya que, si no hacemos esto, cuando se cambie la variable seguirá incrementando su valor con los datos de la nueva fila.

En la instrucción 5 se plantea el ciclo para recorrer las celdas de cada fila.

En la instrucción 6 se acumula el contenido de la celda **mat[i][j]** (fila i, columna j) en la variable **sf**.

En la instrucción 8 se imprime el total de la suma de los datos de las celdas de la fila **i,** con su correspondiente mensaje.

Veamos un ejemplo. Consideremos la siguiente matriz:

	1	2	3	4	5	6	7
1	3	1	6	2	8	4	5
2	9	7	7	4	1	2	4
3	5	6	2	8	5	4	7
4	3	5	5	4	8	9	3

5	8	4	5	8	8	7	6
6	6	3	5	9	8	4	2
7	2	5	7	5	4	4	1

Al ejecutar por primera vez la instrucción 3, la variable i toma el valor de 1, significa que vamos a recorrer la fila 1.

En la instrucción 4 se inicializa el contenido de sf en cero.

Al ejecutar por primera vez la instrucción 5, la variable j toma el valor de 1. En este instante, el contenido de las variables i y j es 1; es decir, estamos en la posición a_{11}.

Al ejecutar la instrucción 6 le suma a sf el contenido de la celda de la fila 1 columna 1, es decir, 3. El contenido de sf será 3.

Al ejecutar la instrucción 7, el fin de PARA interno, incrementa el contenido de j en 1. La j queda valiendo 2, mientras que la i sigue en 1; ahora nos encontramos en a_{12}.

Ejecuta de nuevo la instrucción 6, en la cual le suma el contenido de la celda fila 1 columna 2, cuyo valor es 1, a la variable sf. El contenido de sf será 4.

De esta forma continúa ejecutando el ciclo interno hasta que el contenido de la variable j sea mayor que 7. Cuando esto sucede, continuo con la instrucción 8, en la cual imprime el contenido de sf, que para la primera fila de nuestro ejemplo es 29.

Luego de escribir el resultado de la suma de la primera fila, llegamos a la instrucción 9, que es el fin del PARA externo. En esta instrucción incrementa automáticamente el contenido de la i en 1 y luego regresa a la instrucción 3 a comparar el contenido de la i con el contenido de la m. Como el contenido de i aún no ha alcanzado el valor de m ejecuta nuevamente las instrucciones 4 a 7.

A la variable sf le asignan nuevamente cero y recorre la fila i sumando el contenido de cada una de las celdas con el ciclo interno y luego imprimiendo el contenido de sf.

Al terminar de ejecutar el ciclo externo finaliza la ejecución del método habiendo escrito la suma de los datos de las celdas de cada fila.

2.4.7.6 SUMA DE LOS DATOS DE CADA COLUMNA DE UNA MATRIZ

Así como en muchas situaciones es necesario sumar los datos de las celdas de cada fila, hay otras situaciones en las cuales lo que se requiere es sumar los datos de las celdas de cada columna. Para ello haremos uso del método *ImprimaPorColuma* que habíamos visto anteriormente.

Para totalizar los datos de las celdas de cada columna de una matriz, basta con plantear los ciclos de recorrido por columnas e insertar las instrucciones propias para manejar el acumulador y producir el informe de total de cada columna. Veamos cómo se plantea dicho método:

```
1.  PUBLICO ESTATICO VOID SumaPorColumna (mat, m, n)
2.      VARIABLES: i, j, sc (ENTEROS)
3.          PARA (i= 1, n, 1)
4.              sc = 0
5.              PARA (j= 1, m, 1)
6.                  sc = sc + mat[j][i]
7.              FINPARA
8.              IMPRIMA (“suma de la columna:”, i, “es:”, tc)
9.          FINPARA
10. FIN(Método)
```

La instrucción 1 define el método con sus parámetros: **mat**, la matriz en la cual hay que efectuar el proceso de suma de los datos de cada columna; **m**, el número de filas de **mat;** y **n,** el número de columnas de **mat.**

En la instrucción 2 definimos las variables de trabajo para ejecutar la tarea de sumar e imprimir el total de los datos de las celdas de cada columna: **i**, para recorrer las columnas; **j**, para recorrer las celdas de cada columna; y **sc**, para llevar el acumulado de los datos de las celdas de la columna **i**.

En la instrucción 3 planteamos el ciclo de variación de la **i,** para recorrer la matriz por columnas.

En la instrucción 4 se inicializa el acumulador de los datos de cada columna en cero. Este acumulador, **sc**, se debe inicializar en cero cada vez que se cambie de columna, ya que, si no hacemos esto, cuando se cambie de columna arrastrará el acumulado de la columna anterior al de la nueva columna.

En la instrucción 5 se plantea el ciclo para recorrer las celdas de cada columna.

En la instrucción 6 se acumula el contenido de la celda **mat[j][i]** (fila j, columna i) en la variable **sc**.

En la instrucción 8 se imprime el total de la suma de los datos de las celdas de la columna **i,** con su correspondiente mensaje.

2.4.7.7 CLASIFICACIÓN DE MATRICES

Las matrices se clasifican según la forma y según algunas características con respecto a la distribución de sus datos.

Según la forma, las matrices pueden ser *rectangulares* o *cuadradas.* Matrices rectangulares son aquellas en las que el número de filas es diferente del número de columnas, y matrices cuadradas son aquellas en las que el número de filas es igual al número de columnas.

Según la distribución de sus datos, existen muchas clases de matrices. Entre ellas podemos destacar las siguientes:

CLASIFICACIÓN DE MATRICES

- **Matriz *simétrica:***

Es una matriz cuadrada en la que el dato de una celda de la fila i, columna j, es igual al dato de la celda de la fila j, columna i; es decir, por ejemplo, que el dato ubicado en la posición a_{13} es igual al dato de la posición a_{31}.

- **Matriz *identidad:***

Es una matriz cuadrada en la que todos los elementos son 0, excepto los de la diagonal principal, que son 1. Los elementos de la diagonal principal son aquellos en los cuales la fila es igual a la columna; es decir, las posiciones a_{11}, a_{22}, a_{33}, …, a_{nn}.

- **Matriz *triangular inferior:***

Es una matriz cuadrada en la cual los elementos que se hallan por encima de la diagonal principal son todos ceros.

- **Matriz *triangular superior:***

Es una matriz cuadrada en la cual los elementos que se hallan por debajo de la diagonal principal son todos ceros.

- **Matriz *diagonal:***

Es matriz una cuadrada en la que todos los elementos que se encuentran por fuera de la diagonal principal son ceros.

Según la clase de matriz que se esté trabajando, las operaciones sobre ellas varían. Por ejemplo, para calcular el determinante de una matriz, esta debe ser cuadrada. Así $a_{ij} = 0$, donde **i** es diferente de **j**.

2.4.7.8 SUMA DE LOS ELEMENTOS DE LA DIAGONAL PRINCIPAL DE UNA MATRIZ CUADRADA

Como un ejemplo de trabajo con matrices cuadradas presentamos un método en el que se suman los elementos de la diagonal principal de una matriz. El siguiente es un método que efectúa dicha tarea:

```
1.  PUBLICO ESTATICO ENTERO SumaDiagPpal (mat, n)
2.      VARIABLES: i, s (ENTEROS)
3.              s = 0
4.              PARA (i= 1, n, 1)
5.                      s = s + mat[i][i]
6.              FINPARA
7.              RETORNE (s)
8.  FIN(Método)
```

En la instrucción 1 se define el método, el cual retornara un dato entero, con sus parámetros: **mat**, la matriz a la cual le vamos a sumar los elementos de la diagonal principal, y **n** el número de filas y columnas de la matriz **mat**.

En la instrucción 2 definimos nuestras variables de trabajo; **i**, variable con la cual recorremos los elementos de la diagonal principal, y **s**, la variable en al cual almacenamos la suma de dichos elementos.

En la instrucción 4 planteamos el ciclo con el cual se recorren los elementos de la diagonal principal. Como ya habíamos dicho, los elementos de la diagonal principal son aquellos en los cuales la fila es igual a la columna; por tanto, solo requerimos una variable para recorrerlos.

En la instrucción 5 acumulamos en la variable **s** dichos elementos, y en la instrucción 7 retornamos el valor de **s**.

Tomemos como ejemplo la siguiente matriz:

								n
		1	2	3	4	5	6	7
	1	**3**	1	6	2	8	4	5
	2	9	**7**	7	4	1	2	4
	3	5	6	**2**	8	5	4	7
	4	3	5	5	**4**	8	9	3
	5	8	4	5	8	**8**	7	6
	6	6	3	5	9	8	**4**	2
n	7	2	5	7	5	4	4	**1**

Al ejecutar nuestro método *SumaDiagPpaL*, el método sumará los valores que aparecen resaltados en la matriz y, retornará en **s** un valor de 29.

2.4.7.9 SUMA DE LOS ELEMENTOS DE LA DIAGONAL SECUNDARIA DE UNA MATRIZ CUADRADA

Como un segundo ejemplo, vamos a construir un método con el que se sumen los elementos de la diagonal secundaria de una matriz cuadrada. Una celda pertenece a la diagonal secundaria si la suma de la fila y la columna que identifican la celda, da como resultado **n+1**, siendo **n** el número de filas y columnas de la matriz. El siguiente método ejecuta dicha terea:

```
1.  PUBLICO ESTATICO ENTERO SumaDiagSec (mat, n,)
2.      VARIABLES: i, j, s (ENTEROS)
3.              s = 0
4.              j = n
5.              PARA (i= 1, n, 1)
6.                      s = s + mat[i][j]
7.                      j = j – 1
8.              FINPARA
9.              RETORNE (s)
10. FIN(Método)
```

En la instrucción 1 definimos el método, el cual retornara un valor entero, con sus parámetros: **mat**, la matriz sobre la cual se va a trabajar, y **n,** el orden de la matriz.

En la instrucción 2 se define las variables de trabajo: **i,** para identificar la fila de una celda de la diagonal secundaria; **j**, para identificar la columna de una celda de la diagonal secundaria; y **s**, la variable en la cual llevaremos la suma de los elementos de la diagonal secundaria.

En la instrucción 3 inicializamos el acumulador en cero y en la instrucción 4 le asignamos a **j** el valor de **n**, es decir, la última columna. La variable **j** la utilizaremos para identificar la columna de un elemento perteneciente a la diagonal secundaria.

En la instrucción 5 planteamos el ciclo con el cual recorremos la matriz. La variable controladora del ciclo, es decir la **i**, la utilizaremos para identificar la fila de un elemento de la diagonal secundaria.

En la instrucción 6 acumulamos en la variable **s** el dato de una celda perteneciente a la diagonal secundaria. Dicha celda la identificamos con la fila **i** y la columna **j.**

En la instrucción 7 le restamos 1 a **j,** la variable con la cual se identifica la columna de un elemento de la diagonal secundaria.

En la instrucción 9 se retorna el contenido de la variable **s**, la cual contiene la suma de los elementos de la diagonal secundaria.

Veamos el ejemplo:

							n
	1	2	3	4	5	6	7
1	3	1	6	2	8	4	**5**
2	9	7	7	4	1	**2**	4
3	5	6	2	8	**5**	4	7
4	3	5	5	**4**	8	9	3
5	8	4	**5**	8	8	7	6
6	6	**3**	5	9	8	4	2
n 7	**2**	5	7	5	4	4	1

Al ejecutar nuestro método *SumaDiagSec* se retorna un valor de 26.

2.4.7.10 INTERCAMBIO DE DOS FILAS

Otra de las operaciones que se requieren con frecuencia cuando se manipulan matrices es intercambiar los datos correspondientes a dos filas dadas. Veamos el método que efectúa dicha tarea:

```
1.  PUBLICO ESTATICO VOID IntercambiarFila (mat, m, n, i, j)
2.      VARIABLES: k, aux (ENTEROS)
3.          PARA (k= 1, n, 1)
4.              aux = mat[i][k]
5.              mat[i][k] = mat[j][k]
6.              mat[j][k] = aux
7.          FINPARA
8.  FIN(Método)
```

En la instrucción 1 definimos el método con sus parámetros: **mat,** la matriz en la cual se efectúa el intercambio; **m**, el número de filas de la matriz **mat; n,** el número de columnas de la matriz **mat; i,** y **j,** las filas cuyos datos hay que intercambiar. El parámetro **mat** es el único parámetro por referencia de este método.

En la instrucción 2 definimos las variables de trabajo: **k**, para recorrer las filas cuyos datos se desean intercambiar, y **aux**, una variable auxiliar para poder efectuar el intercambio.

En la instrucción 3 se plantea el ciclo con el cual se recorren simultáneamente las filas **i** y **j**.

Las instrucciones 4 a 6 son las instrucciones con las cuales se intercambia el dato de la celda de la fila **i** columna **k**, con el dato de la celda de la fila **j** columna **k**. en la instrucción 4 se aguarda en la variable auxiliar **aux** el contenido de la celda fila **i** columna **k**; en instrucción 5 trasladamos el dato de la celda de la fila **j** columna **k** hacia la celda fila **i** columna **k,** y en la instrucción 6 llevamos lo que tenemos en la variable auxiliar **aux** (lo que había en la celda fila **i** columna **k**) hacia la celda de la fila **j** columna **k**.

Las instrucciones del ciclo se ejecutan hasta que el contenido de la variable **k** sea mayor que **n**. Cuando esto suceda se habrán intercambiado los datos de la fila **i** con los datos de la fila **j**.

Veamos el siguiente ejemplo:

		n						
		1	2	3	4	5	6	7
	1	3	1	6	2	8	9	7
	2	4	1	2	8	2	5	7
	3	5	3	7	1	3	5	5
	4	4	3	6	3	2	8	1
	5	4	2	5	4	1	1	6
m	6	1	8	9	7	1	3	9

(*a*)

		n						
		1	2	3	4	5	6	7
	1	3	1	6	2	8	9	7
	2	4	2	5	4	1	1	6
	3	5	3	7	1	3	5	5
	4	4	3	6	3	2	8	1
	5	4	1	2	8	2	5	7
m	6	1	8	9	7	1	3	9

(*b*)

La matriz *a* es la matriz original; antes del intercambio de filas. La matriz *b* es la matriz resultante de intercambiar la fila 2 con la 5.

2.4.7.11 INTERCAMBIO DE DOS COLUMNAS

Así como intercambiar filas es una opción que se requiere con cierta frecuencia en la manipulación de matrices, intercambiar columnas también. El proceso de intercambiar columnas es similar al proceso de intercambiar filas, la diferencia básica es que en vez de recorrer las filas se recorre las columnas que debemos intercambiar. Veamos como efectuar este proceso:

```
1.  PUBLICO ESTATICO VOID  IntercambiarColumna (mat, m, n, i, j)
2.       VARIABLES: k, aux (ENTEROS)
3.                PARA (k= 1, m, 1)
4.                         aux = mat[k][i]
5.                         mat[k][i] = mat[k][j]
6.                         mat[k][j] = aux
7.                FINPARA
8.  FIN(Método)
```

Como ejemplo veamos la siguiente matriz, en la cual se intercambian los datos de la columna 2 con los datos de la columna 5:

		1	2	3	4	5	6	n=7
	1	3	1	6	2	8	9	7
	2	4	1	2	8	2	5	7
	3	5	3	7	1	3	5	5
	4	4	3	6	3	2	8	1
	5	4	2	5	4	1	1	6
m	6	1	8	9	7	1	3	9

(*a*)

		1	2	3	4	5	6	n=7
	1	3	8	6	2	1	9	7
	2	4	2	2	8	1	5	7
	3	5	3	7	1	3	5	5
	4	4	2	6	3	3	8	1
	5	4	1	5	4	2	1	6
n	6	1	1	9	7	8	3	9

(*b*)

La matriz *a* es la matriz original; antes del intercambio de columnas. La matriz *b* es la matriz resultante de intercambiar la columna 2 con la 5.

2.4.7.12 ORDENAR LOS DATOS DE UNA MATRIZ CON BASE EN LOS DATOS DE UNA COLUMNA

Sucede con frecuencia que cuando se manejan datos es una matriz, en una columna se maneja, digamos, un código, y en el resto de la fila se manejan datos correspondientes a ese código.

Supongamos que en cada fila se maneja los siguientes datos: el código de un artículo y los datos correspondientes a las ventas de ese artículo en cada uno de los almacenes que maneja la compañía; como lo vemos en el siguiente ejemplo:

		1	**2**	**3**	**4**	**5**	**6**	**7**	**n=8**
	1	3	1	4	9	7	7	5	6
	2	1	2	5	7	3	4	8	6
	3	6	4	2	5	4	1	1	3
	4	2	9	3	4	2	9	8	9
m	**5**	8	5	8	4	1	9	3	4

Si miramos la matriz anterior, el dato de la columna 1 fila 1 es el código de un artículo (artículo con código 3) y los demás datos de la fila 1 son las ventas del artículo 3 en los diferentes almacenes, es decir: 1, 4, 9, 7, 7, 5, 6.

El dato de la fila 2 columna 1 es el código de un artículo (artículo con código 1) y los demás datos de la fila 2 son las ventas del artículo 1, es decir: 2, 5, 7, 3, 4, 8, 6.

Si se desean conocer las ventas de un artículo, hay que buscar el código del artículo recorriendo la columna 1. Si los datos de la columna 1 no están ordenados, el proceso de búsqueda del código del artículo es bastante dispenso. Si los datos de la columna 1 están ordenados, el proceso de búsqueda será bastante fácil y eficiente.

Ahora veamos el método que ordena los datos de la matriz teniendo como base los datos de una columna previamente seleccionada:

```
1.  PUBLICO ESTATICO VOID OrdenarPorColumna (mat, m, n, c)
2.      VARIABLES: i, j, k (ENTEROS
3.          PARA (i= 1, m-1, 1)
4.              k = 1
5.              PARA (j= i+1, m, 1)
6.                  SI (mat[j][c] < mat[k][c])
7.                      k = j
8.                  FINSI
9.              FINPARA
10.             OrdenarPorColumna.IntercambiarFilas (mat, m, n, i, k)
11.         FINPARA
12. FIN(Método)
```

El método que utilizamos para efectuar el ordenamiento es el método de selección que vimos en vectores.

En la instrucción 1 definimos el método con sus parámetros: **mat**, la matriz cuyos datos se desean ordenar; **m,** el número de filas de la matriz **mat**; **n**, el número de columnas de **mat;** y **c,** la columna que se toma como base para ordenar los datos de la matriz. Recuerde que el parámetro **mat** debe ser por referencia.

En la instrucción 2 definimos las variables de trabajo para ejecutar el proceso de ordenamiento: **i**, para identificar a partir de cual fila faltan datos por ordenar; **j**, para determinar en cual fila se halla el menor dato, de los datos de la columna **c;** y **k,** la variable en la cual almacenamos la fila que tiene el menor dato de la columna **c.**

Las instrucciones 3 a 9 de este método son similares a las instrucciones 3 a 9 del método *OrdenamientoAscendenteSeleccion*, con el cual ordenamos un vector. La única diferencia es que

en la instrucción 6 comparamos el dato de la fila **j** columna **c**, con el dato de la fila **k** columna **c,** para determinar en cual fila se halla el menor dato de la columna **c**. Al terminar el ciclo interno se ejecuta la instrucción 10, en la que se invoca el método para intercambiar los datos de la fila **i** con los datos de la columna **k**, el cual fue desarrollado previamente.

En la instrucción 11 se finaliza la instrucción PARA y se incrementa la **i** en 1 y continúa con el proceso de ordenamiento hasta que el contenido de la variable **i** sea igual al contenido de la variable **m.**

En la siguiente figura se muestra como quedan los datos de la matriz después de ejecutar el proceso de ordenamiento con base a la columna 1.

								n
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	3	1	4	9	7	7	5	6
2	1	2	5	7	3	4	8	6
3	6	4	2	5	4	1	1	3
4	2	9	3	4	2	9	8	9
m 5	8	5	8	4	1	9	3	4

(*a*)

								n
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	3	1	4	9	7	7	5	6
2	1	2	5	7	3	4	8	6
3	6	4	2	5	4	1	1	3
4	2	9	3	4	2	9	8	9
m 5	8	5	8	4	1	9	3	4

(*b*)

La matriz *a* es la matriz original, mientras que la matriz *b* es la misma matriz, pero ordenada por la primera columna.

2.4.7.13 TRANSPUESTA DE UNA MATRIZ

Dada una matriz **A** se define la matriz transpuesta de **A,** y la expresaremos como $\mathbf{A^{t}}$, a la matriz que resulta de intercambiar las filas por las columnas de **A,** de tal forma que, si **A** es una matriz de **m** filas y **n** columnas, su transpuesta resulta de **n** filas y **m** columnas.

Consideremos la siguiente figura, en la cual se presenta una matriz con su correspondiente transpuesta. Como se puede observar la matriz original **A** tiene 5 filas y 7 columnas. La matriz $\mathbf{A^{t}}$ tiene 7 filas y 5 columnas.

A=		1	2	3	4	5	6	n = 7
	1	3	1	6	2	8	4	9
	2	7	7	2	5	1	5	4
	3	4	1	3	8	1	2	8
	4	6	3	2	1	5	5	3
m	5	4	1	2	2	1	3	7

A^t=		1	2	3	4	M = 5
	1	3	7	4	6	4
	2	1	7	1	3	1
	3	6	2	3	2	2
	4	2	5	8	1	2
	5	8	7	1	5	1
	6	4	5	2	5	3
n	7	9	4	8	3	7

Cada elemento de la fila **i** columna **j** de la matriz **A** queda ubicado en la fila **j** columna **i** de la matriz $\mathbf{A^t}$. El dato que en la matriz **A** se hallaba en la fila 2 columna 5 queda en la fila 5 columna 2, y así sucesivamente.

Ahora veamos el método con el cual se construye la transpuesta de una matriz:

```
1.  PUBLICO ESTATICO VOID Transpuesta (A, m, n, At)
2.      VARIABLES: i, j (ENTEROS)
3.          PARA (i= 1, m, 1)
4.              PARA (j= 1, n, 1)
5.                  At[j][i] = A[i][j]
6.              FINPARA
7.          FINPARA
8.  FIN(Método)
```

En la instrucción 1 definimos el método con sus parámetros: **A,** la matriz a la cual le calcularemos la transpuesta; **m**, el número de filas de **A**; **n**, el número de columna de **A**; **At**, la variable en la cual quedara almacenada la transpuesta de **A**. El parámetro **At** debe ser por referencia.

En la instrucción 2 se definen las variables de trabajo **i** y **j**, con las cuales se recorre la matriz **A** y se construye **At.**

En la instrucción 3 y 4 se plantean los ciclos para recorrer la matriz **A** por filas.

En la instrucción 5 se construye la transpuesta de **A**; asignando a **At** fila **j** columna **i** el contenido de **A** fila **i** columna **j**.

Al terminar de ejecutar los ciclos de recorrido de **A,** por filas, en **At** queda la transpuesta de **A.**

2.4.7.14 SUMA DE DOS MATRICES

La suma de dos matrices **A** y **B** consiste en crear una nueva matriz **C** en la cual cada elemento de **C** es la suma de los correspondientes elementos de las matrices **A** y **B.** Por tanto, para poder sumar dos matrices, estas tienen que tener las mismas dimensiones, es decir, igual número de filas y columnas. Simbólicamente la suma de dos matrices es expresa así:

C[i][j] = A[i][j] + B[i][j]

El siguiente es un método con el cual se suman dos matrices:

```
1.  PUBLICO ESTATICO VOID SumarMatriz (A, m, n, B, p, q, C)
2.      VARIABLES: i, j (ENTEROS)
3.          SI ((m != p) v (n != q))
4.              IMPRIMA ("las matrices no se pueden sumar")
5.          SINO
6.              PARA (i= 1, m, 1)
7.                  PARA (i= 1, n, 1)
8.                      C[i][j] = A[i][j] + B[i][j]
9.                  FINPARA
10.             FINPARA
11.         FINSI
12. FIN(Método)
```

En la instrucción 1 se define el método con sus parámetros: **A** y **B,** las matrices que se van a sumar; **m y n**, las dimensiones de la matriz **A**; **p** y **q**, las dimensiones de la matriz **B**; y **C**, la matriz resultante. El parámetro **C** es un parámetro por referencia.

En la instrucción 2 definimos las variables de trabajo: **i** y **j**, para recorrer las matrices **A** y **B** por filas.

En la instrucción 3 se controla que las matrices **A** y **B** se pueden sumar. Para que dos matrices se puedan sumar deben obtener las mismas dimensiones (**m==p** y **n==q**). Por tanto, si el número de filas de **A**, que es **m**, es diferente del número de filas de **B,** que es **p**, o el número de columnas de **A**, que es **n**, es diferente del número de columnas de **B**, que es **q**, las matrices no se podrán sumar y se ejecutará la instrucción 4, en la cual produce el mensaje apropiado y finalizaría sin ejecutar otra acción.

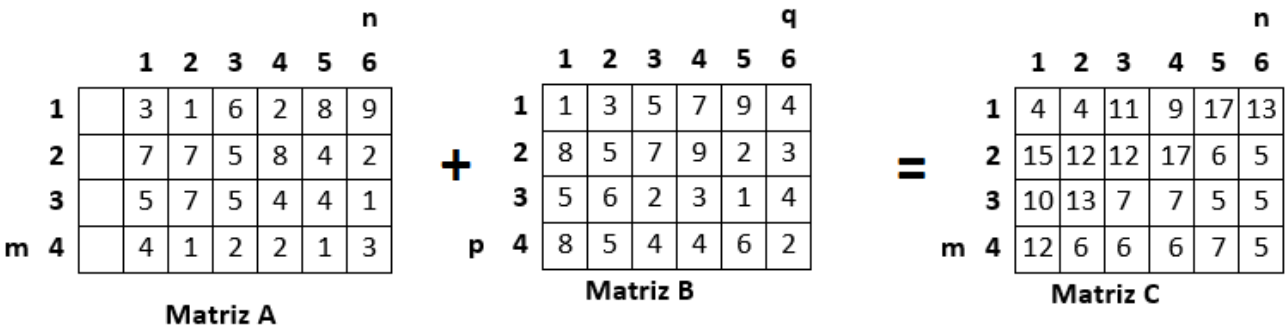
Si la condición de la instrucción 3 es falsa significa que las dos matrices **A** y **B** tienen las mismas dimensiones y por consiguiente continuará ejecutando la instrucción 6.

En las instrucciones 6 y 7 planteamos los ciclos para recorrer las matrices por filas.

En la instrucción 8 se le asigna a la celda **ij** de **C** el resultado de sumar el contenido de la celda **ij** de **A** con el contenido de la celda **ij** de **B**.

Al termina de ejecutar los ciclos en la matriz **C** queda el resultado de sumar los datos de la matriz **A** con los datos de la matriz **B.**

Veamos un ejemplo de suma de matrices:



Observe que cada celda de la matriz **C** contiene la suma de sus correspondientes posiciones de las matrices **A** y **B**.

2.4.7.15 MULTIPLICACIÓN DE MATRICES

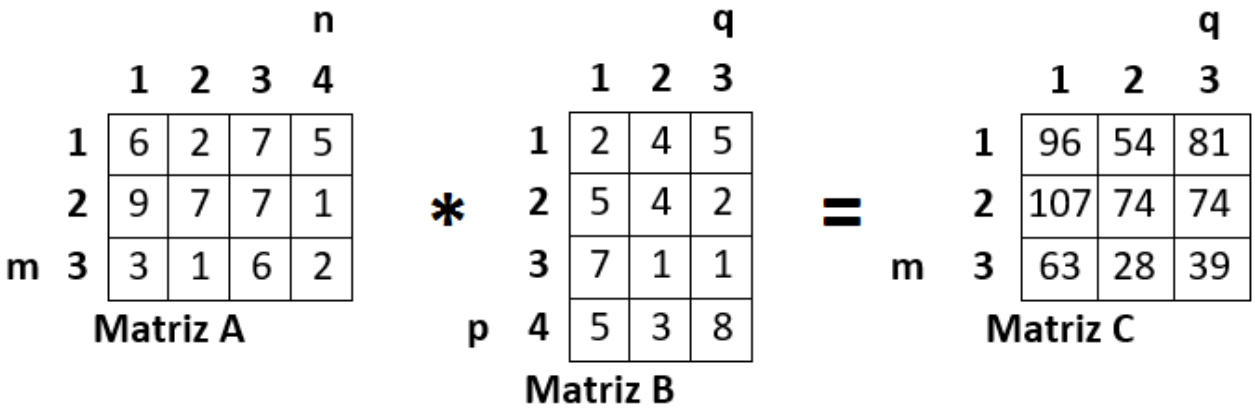
Sea **A** una matriz de dimensiones **m*n,** siendo **m** el número de filas y **n** el número de columnas. Sea **B** una matriz de dimensiones **p*q,** siendo **p** el número de filas y **q** el número de columnas.

Para poder efectuar el producto de **A** y **B**, en estas dos matrices se debe cumplir que **n==p**; es decir, el número de columnas de **A** (**n**) debe ser igual al número de filas de **B** (**p**). Si **n** es diferente de **p** no se podrá efectuar el producto **A** y **B**.

La matriz resultante **C** tendrá dimensiones **m*q,** siendo **m** el número de filas de la matriz **A** y **q** el número de columnas de la matriz **B**.

Cada elemento **C[i][j]** de la matriz resultante es la sumatoria de productos de los elementos **A[i][k] * B[k][j]**, con **k** desde 1 hasta **n**.

Consideramos como ejemplo las matrices **A** y **B** de la siguiente figura. El producto de **A*B** es la matriz **C,** que también se presenta en la misma figura:



En nuestro ejemplo, la matriz **A** tiene dimensión **3*4,** es decir, **m** vale 3 y **n** vale 4; y la matriz **B** tiene dimensión **4*3,** es decir, **p** vale 4 y **q** vale 3**.** Por consiguiente, el producto **A*B** se puede efectuar ya que **n==p**, y la dimensión de la matriz resultante son **3*3,** es decir, 3 filas y 3 columnas.

Simbólicamente, cada elemento de la matriz resultante (C) se describe así:

$$C_{ij} = \sum_{K=1}^{n} A_{i,k} * B_{k.j}$$

Es decir, para multiplicar la matriz **A** por la matriz **B** de nuestro ejemplo, y obtener como resultado la matriz **C**, lo hacemos de la siguiente forma:

$C_{11} = A_{11}*B_{11} + A_{12}*B_{21} + A_{13}*B_{31} + A_{14}*B_{41}$

$C_{12} = A_{11}*B_{12} + A_{12}*B_{22} + A_{13}*B_{32} + A_{14}*B_{42}$

$C_{13} = A_{11}*B_{13} + A_{12}*B_{23} + A_{13}*B_{33} + A_{14}*B_{43}$

$C_{21} = A_{21}*B_{11} + A_{22}*B_{21} + A_{23}*B_{31} + A_{24}*B_{41}$

$C_{22} = A_{21}*B_{12} + A_{22}*B_{22} + A_{23}*B_{32} + A_{24}*B_{42}$

$C_{23} = A_{21}*B_{13} + A_{22}*B_{23} + A_{23}*B_{33} + A_{24}*B_{43}$

$C_{31} = A_{31}*B_{11} + A_{32}*B_{21} + A_{33}*B_{31} + A_{34}*B_{41}$

$C_{32} = A_{31}*B_{12} + A_{32}*B_{22} + A_{33}*B_{32} + A_{34}*B_{42}$

$C_{33} = A_{31}*B_{13} + A_{32}*B_{23} + A_{33}*B_{33} + A_{34}*B_{43}$

Veamos el método que se encarga de realizar la multiplicación entre matrices:

```
1.  PUBLICO ESTATICO VOID MultiplicarMatriz (A, m, n, B, p, q, C)
2.      VARIABLES: i, j, k (ENTEROS)
3.          SI (n != p)
4.              IMPRIMA ("las matrices no se pueden multiplicar")
5.          SINO
6.              PARA (i= 1, m, 1)
7.                  PARA (j= 1, q, 1)
8.                      C[i][j] = 0
9.                      PARA (k= 1, n, 1)
10.                         C[i][j] = C[i][j] + A[i][k] * B[k][j]
11.                     FINPARA
12.                 FINPARA
13.             FINPARA
14.         FINSI
15. FIN(Método)
```

En la instrucción 1 se define el método con sus parámetros: **A** y **B,** las matrices a multiplicar; **m** y **n**, las dimensiones de la matriz **A**; **p** y **q**, las dimensiones de la matriz **B**; y **C**, la matriz resultante del producto, recuerde que **C** debe ser un parámetro por referencia.

En la instrucción 2 definimos las variables de trabajo: **i,** para recorrer las filas de la matriz **A**; **j**, para recorrer las columnas de la matriz **B**; y **k**, para recorrer simultáneamente las columnas de **A** y las filas de **B.**

En la instrucción 3 se controla que se pueda efectuar el producto entre **A** y **B. S**i el número de columnas de **A,** que es **n**, es diferente del número de filas de **B**, que es **p,** el producto no se podrá efectuar y, por tanto, ejecuta la instrucción 4, con la cual se produce el mensaje apropiado y finalizará sin ejecutar más acciones.

Si la condición de la instrucción 3 es falsa, significa que el producto si se puede ejecutar y, por tanto, continúa ejecutando la instrucción 6.

En la instrucción 6 se plantea el ciclo con el cual se recorre la matriz **A** por filas.

En la instrucción 7 se plantea el ciclo con el cual se recobra la matriz **B** por columnas.

En la instrucción 8 se inicializa en cero el contenido de la celda fila **i** columna **j** de la matriz **C**. Esta inicialización es necesaria ya que dicha posición funciona como un acumulador.

En la instrucción 9 se plantea el ciclo con el cual se recorren las celdas de la fila **i** de **A** y las celdas de la columna **j** de **B**. Para un valor cualquiera de **k** (variable controladora del ciclo de la instrucción 9), se efectúa el producto del contenido de la celda **ik** de **A,** con el contenido de la celda **kj** de **B** y se acumula este producto en la celda **ij** de **C**.

Al terminar el ciclo más interno (el ciclo de la instrucción 9), se pasará a otra celda de la matriz **C**; que en ese ejemplo se hace por filas.

En las matrices se utilizan las mismas acciones que en los vectores (borrar, insertar, buscar, etc.), la diferencia radica en que, para realizar alguna de estas acciones en una matriz, se necesitará un ciclo adicional para podernos desplazar en las filas y columnas.

2.4.8 EJERCICIO DE APRENDIZAJE

En cada uno de los temas hay ejercicios de Aprendizaje

2.4.9 TALLER DE ENTRENAMIENTO

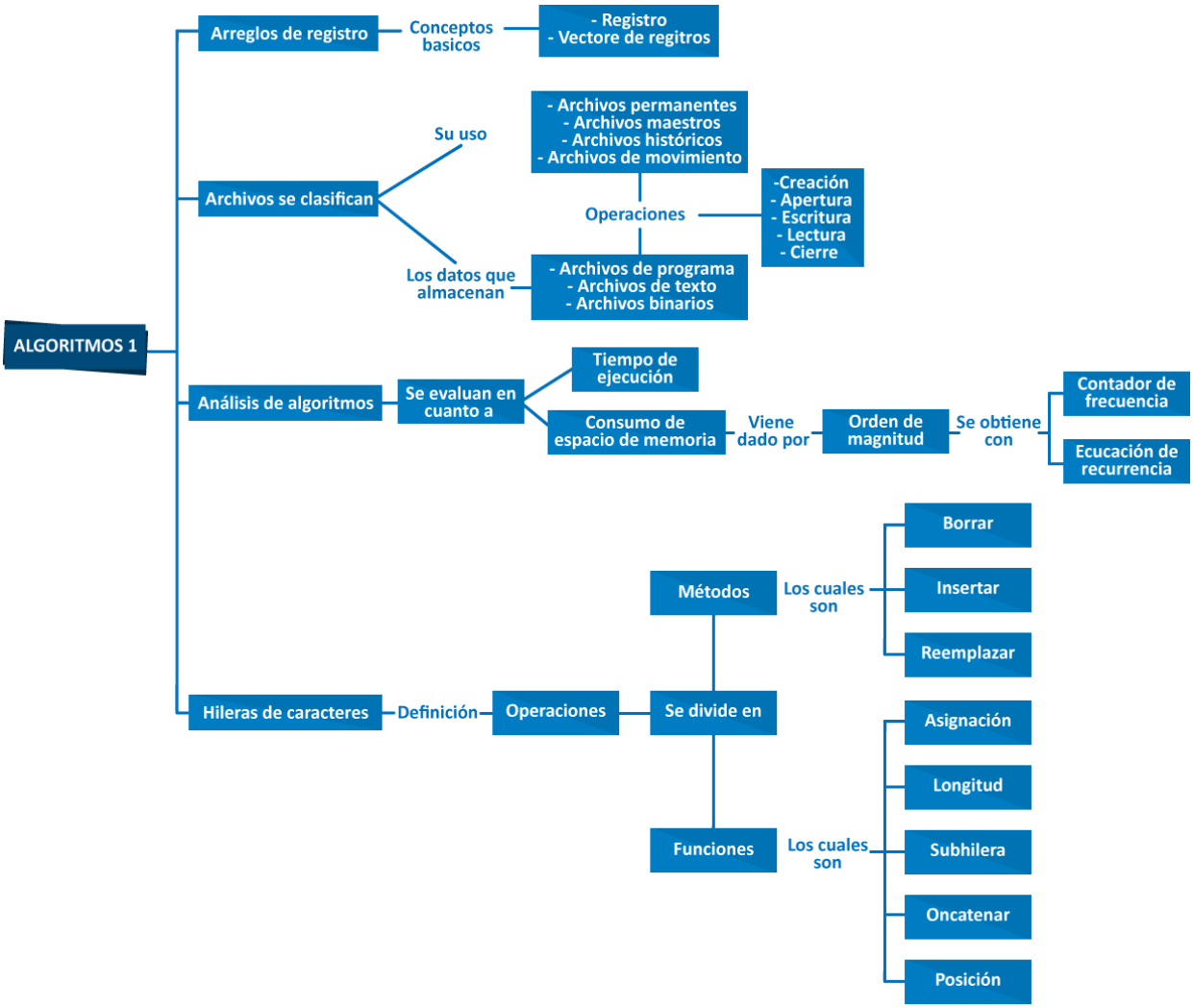
1. Elabore un método que imprima los datos que se hallan en las posiciones impares de un vector de n elementos.
2. Elabore un método que calcule y retorne cuántas veces se halla un dato d, entrado como parámetro, en un vector de n elementos.
3. Elabore un método que intercambie los datos de un vector así: el primero con el segundo, el tercero con el cuarto, el quinto con el sexto, y así sucesivamente.
4. Elabore un método que intercambie los datos de un vector así: el primero con el tercero, el segundo con el cuarto, el quinto con el séptimo, el sexto con el octavo, y así sucesivamente.

5. Se tiene un vector cuyos datos se hallan ordenados en forma descendente. Elabore un método que procese dicho vector, de tal manera que no quede con datos repetidos, es decir, si hay algún dato repetido, sólo debe quedar uno de ellos.
6. Elabore un método que sustituya un valor x por un valor y en un vector cuyos datos se hallan ordenados ascendentemente, de tal manera que el vector siga conservando su orden creciente. La función debe regresar como resultado 1 si se hizo la sustitución, 0 si x no se encontraba en el arreglo y -1 si el arreglo estaba vacío.
7. Un tablero de damas es un arreglo de 8 filas por 8 columnas. Uno (1) representa la presencia de una ficha blanca en el tablero, dos (2) representa la presencia de una ficha negra en el tablero, y cero (0) representa la ausencia de ficha. Elabore un método que determine e imprima: el número de fichas blancas en el tablero, el número de fichas negras y el número de espacios vacíos.
8. Elabore un método que calcule y retorne en el programa principal la suma de los elementos que están por encima de la diagonal principal en una matriz cuadrada de orden n.
9. Elabore un método que calcule y retorne en el programa principal la suma de los elementos que están por debajo de la diagonal secundaria en una matriz cuadrada de orden n.
10. Elabore un método que lea una matriz de n*m elementos y un valor d. Luego modifique la matriz multiplicando cada elemento por el valor d.

TIPS

Los vectores y las matrices son estructuras de almacenamiento que nos permiten guardar gran volumen de Información en memoria RAM.

3 UNIDAD 3 MANEJO DE REGISTROS, ARCHIVOS, ANÁLISIS DE ALGORITMOS Y MANEJO DE HILERAS DE CARACTERES



- **Apertura de un archivo:** Permite relacionar una variable tipo archivo con el nombre que el archivo tendrá en el medio de almacenamiento, llamado nombre externo. Este es el nombre con el cual se guarda el archivo en el medio de almacenamiento, y su extensión depende de la clase de información que se va a guardar.

- **Archivos binarios**: Estos archivos almacenan información en un lenguaje descifrable solo por el computador. La información que puedan almacenar es diversa: colores, sonidos, imágenes, órdenes o datos discretos, llamados archivos de datos.

- **Archivos de movimiento**: Son temporales y su función principal es captar información para actualizar los archivos maestros.

- **Archivos de programa:** Son los programas fuente que se escriben en un determinado lenguaje de programación, donde cada instrucción es un registro.

- **Archivos de texto o ASCII (American Standard Code for Information Interchange)**: Estos almacenan cualquier carácter del código ASCII: palabras, frases, párrafos, y se suelen crear y mantener mediante programas que procesan o editan texto.

- **Archivos históricos:** Son archivos maestros que ya no se utilizan y cuya función es solo de consulta para quienes desean obtener información sobre el pasado.

- **Archivos maestros:** La información almacenada en esta clase de archivos maneja el estado o situación de los registros de una determinada base de datos, y su actualización o los cambios de los datos de los registros se hacen periódicamente.

- **Archivos permanentes:** Son archivos rígidos, ya que la información que almacenan cambia muy poco.

- **Archivos:** Se definen como estructuras de almacenamiento a nivel externo, permitiendo realizar diferentes operaciones sobre ellos.

- **Arreglos de registros**: Estructuras de almacenamiento de información a nivel de memoria interna, y que permite realizar diferentes operaciones a nivel de búsqueda, inserción, borrado entre otras.

- **Cierre del archivo:** Permite proteger el área donde reside el archivo y actualiza el directorio del medio de almacenamiento, reflejando el nuevo estado del archivo. Adicionalmente, destruye las conexiones físicas y lógicas que se construyeron cuando se abrió el archivo.

- **Creación del archivo:** Permite definir una variable dentro del programa que represente el archivo que se quiere manipular.

- **Escritura en el archivo:** Permite almacenar la información i en el archivo que se está manipulando.

- **Lectura de un archivo:** Permite extraer la información del archivo y almacenarla en memoria principal. La información que se transporta a la memoria puede ser cualquier tipo de variable u objetos almacenados en el archivo.

- **Registro:** Es una estructura de datos compuesta. Se puede decir que un registro es un conjunto de campos variables relacionados que, en general, puede pertenecer a tipos de datos diferentes, llamados componentes del tipo registro, donde todas las componentes pueden manipularse bajo un solo nombre de variable.

- **Vectores de registro:** Otra importancia que tienen los arreglos de registros es que cada uno de sus elementos está formado por varios campos pertenecientes a diferentes tipos de datos, a diferencia de los otros tipos de arreglos en los que sus elementos son de un solo tipo; aunque cabe anotar que, por ejemplo, un vector puede ser de tipo registró.

3.1.1 OBJETIVO GENERAL

- Reconocer las estructuras de registros, archivos, análisis de algoritmos y manejo de hileras de caracteres y las diferentes operaciones que se pueden realizar con cada una de ellas.

3.1.2 OBJETIVOS ESPECÍFICOS

- Identificar las estructuras tipo registros y las diferentes operaciones a realizar entre ellas

- Reconocer de forma aclara las estructuras tipo archivos y las diferentes operaciones que se pueden realizar entre ellas.

- Medir la eficiencia en cada uno de los algoritmos.

- Identificar las operaciones que se pueden realizar con hileras de caracteres.

3.1.3 PRUEBA INICIAL

- ¿Qué es un registro y las partes que lo conforman?
- ¿Qué representan los archivos en la programación y porque tienen tanta importancia en el almacenamiento de datos?
- ¿Cómo se mide la eficiencia de un algoritmo?
- ¿Qué son las hileras de caracteres?

3.2 TEMA 1 DEFINICIÓN DE REGISTROS Y OPERACIONES

➢ ***Video: " Pascal - 029 - Array de registros (Lista estatica) | TutorialesNET"***

➢ ***" Ver Video":*** https://www.youtube.com/watch?v=RDtKhXxhfSY"

Se define los **arreglos de registros** como **estructuras de almacenamiento de información a nivel de memoria interna**, y que permite realizar diferentes operaciones a nivel de **búsqueda, inserción, borrado entre otras**.

Los **archivos** se definen como **estructuras de almacenamiento a nivel externo**, permitiendo realizar diferentes operaciones sobre ellos.

Un registro es una estructura de datos compuesta. Se puede decir que un registro es un conjunto de campos variables relacionados que, en general, puede pertenecer a tipos de datos diferentes, llamados componentes del tipo registro, donde todas las componentes pueden manipularse bajo un solo nombre de variable.

Por ejemplo, si se tiene un registro de datos compuestos por los campos: **cédula, nombre, deducción y salario**, podemos representarlo de la siguiente forma:

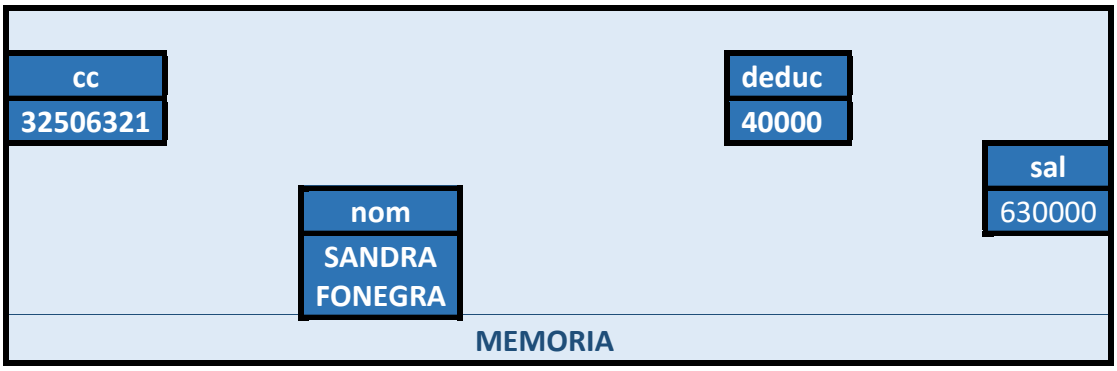
32506321	**SANDRA FONEGRA**	**40000**	**630000**

Como puede notarse, dentro del registro solo hay datos (información); es el programa quien coloca nombres a cada dato para poderlos almacenar en la memoria de la computadora.

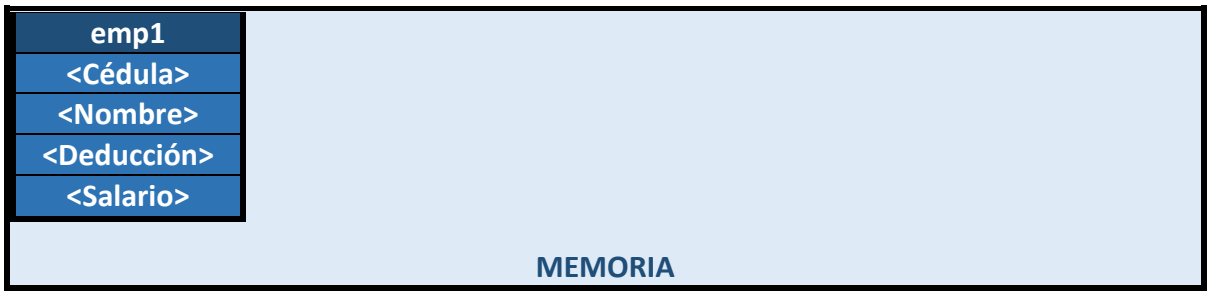
Si la información del empleado se fuera a tratar en forma individual y los nombres de variables seleccionados para cada campo fuera: **cc, nom, deduc y sal;** al ejecutar la instrucción:

LEA: **cc, nom, deduc, sal**

Ocurrirá en memoria lo siguiente:



La información de la empleada está dispersa en la memoria. La idea de usar registro es agrupar toda la información en una misma área de memoria bajo un solo nombre. Si se toma la determinación de que **emp1** es una variable de ***tipo registro***, o sea, almacenar en el área asignada toda la información de un empleado, gráficamente se puede mirar de la siguiente manera:



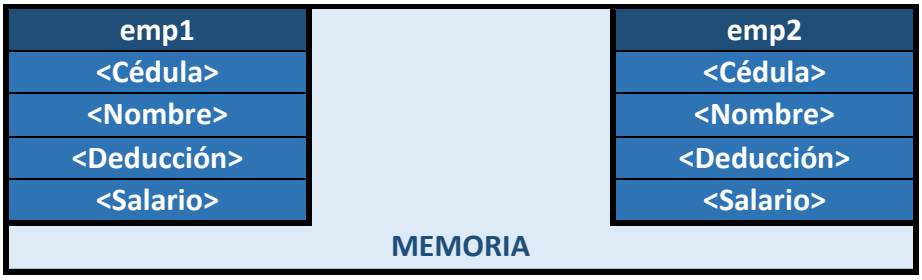
Para que **emp1** sea tomada como una variable tipo registró, se debe definir como tal; es decir, se define a la variable como ***registro*** y se definen los demás campos que habrá en dicho registro. Para nuestro ejemplo, la variable registró quedaría así:

```
1.  CLASE Ejemplo
2.    METODO PRINCIPAL ()
3.          empleado (Registro):
4.          cc, nom (Alfanuméricos)
5.          deduc, sal (Reales)
6.          VARIABLES: emp1, emp2 (Empleado)
7.          Ejemplo.Llenar (emp1)
```

```
8.    end(Método)
9.  end(Clase)
```

En el anterior ejemplo se ha definido un dato de tipo registro llamado ***empleado*** y además se define una variable que es de tipo ***empleado***; esto quiere decir que cualquier variable que sea de tipo ***empleado*** puede almacenar los valores de las componentes del registro, es decir, que en **emp1** se almacenan: **cc, nom, deduc y sal.**

En la estructura anterior se define a **emp1** y **emp2** como variables tipo registro, por lo tanto, pueden almacenar todos los valores de los campos que conforman al tipo de dato empleado, y su representación interna seria:



Las variables tipo registro **emp1** y **emp2** son variables compuestas; para referenciar a cada una de sus componentes hay que clasificarlas, o sea, decir en forma explícita a cuál de las componentes del registro nos vamos a referir. Dicha clasificación se hace así: **variable_tipo_regristro.componente.**

Veamos un método de cómo llenaríamos la información de un empleado, después de haber definido la variable de tipo registro ***empleado*** y a la variable de tipo ***empleado*:**

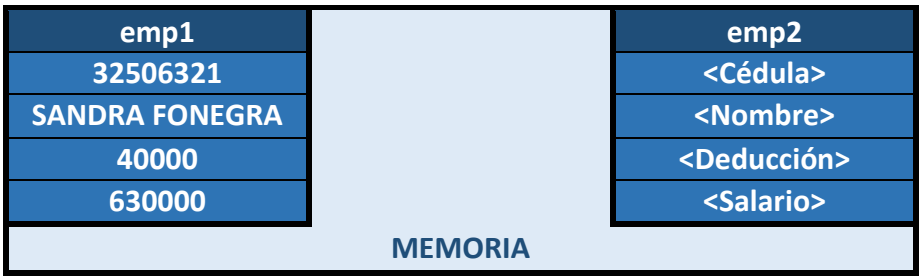
```
1.    PUBLICO ESTATICO VOID Llenar (Reg)
2.              IMPRIMA (“Ingrese la cédula”)
3.              LEA (Reg.cc)
4.              IMPRIMA (“Ingrese el nombre”)
5.              LEA (Reg.nom)
6.              IMPRIMA (“Ingrese la deducción”)
7.              LEA (Reg.deduc)
8.          IMPRIMA (“Ingrese el salario”)
9.          LEA (Reg.sal)
```

10. End (método)

El parámetro **emp1** es un parámetro por referencia.

En la instrucción 3 se lee la cédula que se almacenará en el registro. Igualmente en las instrucciones 5, 7, y 9 donde se almacenará el **nombre, deducción y salario del empleado**, respectivamente.

Después de ejecutar el método anterior y llenarlo con los datos que teníamos anteriormente, nuestra memoria estaría de la siguiente forma:



Si hay necesidad de mover la información de una variable tipo registro a otra, se puede hacer componente a componente, o moviendo toda la información al mismo tiempo. Por ejemplo:

emp2.cc = emp1.cc
emp2.nom = emp1.nom
emp2.deduc = emp1.deduc
emp2.sal = emp1.sal
Lo cual sería lo mismo que tener: **emp2 = emp1.**

Un ***arreglo de registro*** es una estructura de datos de gran utilidad e importancia en la programación de aplicaciones, ya que muchas veces es deseable, desde el punto de vista de la lógica del programa en particular, mantener disponible en memoria principal una serie de registro con información del mismo tipo para su proceso; dichos registros no estarán dispersos; harán parte de un arreglo, lo que facilita su manejo. Un arreglo de registros es muy parecido a un archivo de datos, se diferencian en que los arreglos de registros residen en memoria principal y los archivos en memoria auxiliar, por lo tanto, estos últimos tienen existencia permanente.

3.2.1 OPERACIONES CON ARREGLOS DE REGISTROS

Los **arreglos de registros** permiten manejar información en memoria interna, lo cual **permite realizar las operaciones de:**

- **Búsqueda**

Que permite verificar si un dato se encuentra en la estructura de almacenamiento,

- **Inserción**

Que permite ingresar un nuevo dato en el arreglo de registro,

- **Borrado**

Que permite borrar un dato del arreglo de registro entre otras operaciones que se pueden realizar con estas estructuras.

Otra importancia que tienen los arreglos de registros es que cada uno de sus elementos está formado por varios campos pertenecientes a diferentes tipos de datos, a diferencia de los otros tipos de arreglos en los que sus elementos son de un solo tipo; aunque cabe anotar que, por ejemplo, **un vector puede ser de tipo registro.** Veamos un ejemplo en donde almacenamos los datos de un registro en un vector y luego imprimimos dicho vector:

```
1.  CLASE RegVector
2.      METODO PRINCIPAL()
3.          R (REGISTRO):
4.          cc, nom (Alfanuméricos)
5.          deduc, sal (Reales)
6.          VARIABLES: vec[], emp (R)
7.              i, n (Entero)
8.              IMPRIMA (Ingrese el número de registros")
9.              LEA (n)
10.   para (i=1, n, 1)
11.         RegVector.Llenar (emp)
12.         vec[i] = emp
13.   endpara
14.   para (i= 1, n, 1)
15.       IMPRIMA (vec[i].cc)
16.       IMPRIMA (vec[i].nom)
17.       IMPRIMA (vec[i].deduc)
18.      IMPRIMA (vec[i].sal)
19.   endpara
20.  end(Método)
21. end(Clase)
```

En la instrucción 3 definimos a **R** de tipo ***registro*** con los campos: **cc, nom, deduc y sal**; y en la instrucción 6 definimos nuestras variables, en donde utilizamos 2 variables de tipo ***R***: vec[] y emp.

En la instrucción 9 leemos a **n**, para conocer la cantidad de registros que almacenaremos y, en la instrucción 10, definimos nuestro ciclo desde 1 hasta **n** para llenar el vector con los registros.

En la instrucción 11 llamamos al método Llenar para leer los datos que guardaremos en el vector. Enviamos como parámetro a la variable **emp**.

En la instrucción 12 llevamos al vector en posición **i** lo que se guardó en **emp**.

En las instrucciones 15 a 18 imprimimos el contenido de los vectores, es decir, imprimimos los registros que hemos almacenado.

Ejercicios de entrenamiento

Pautas para desarrollar los siguientes ejercicios: Puedes crear una estructura tipo vector de registro, con los campos correspondientes al enunciado en el ejercicio, o puede suponer que la estructura ya está creada y puedes trabajar sobre ella, realizando las operaciones que te pide cada uno de los ejercicios mencionados.

1. Una compañía distribuye N productos a distintos comercios de la ciudad. Para ello almacena en un arreglo toda la información relacionada a su mercancía:

   - Clave
   - Descripción
   - Existencia
   - Minino a mantener de existencia
   - Precio unitario

   Escriba un programa que pueda llevar a cabo las siguientes operaciones:

   a) Venta de un producto: se deben actualizar los campos que correspondan, y verificar que la nueva existencia no esté por debajo del mínimo. (Datos: clave, cantidad vendida)

b) Reabastecimiento de un producto: se deben actualizar los campos que correspondan. (Datos: clave, cantidad comprada.)

c) Actualizar el precio de un producto. (Datos: clave, porcentaje de aumento.)

d) Informar sobre un producto: se deben proporcionar todos los datos relacionados a un producto. (Dato: clave.)

2. Una inmobiliaria tiene información sobre departamentos en renta. De cada departamento se conoce:

- Clave: es un entero que identifica al inmueble.
- Extensión: superficie del departamento, en metros cuadrados.
- Ubicación: (excelente, buena, regular, mala)
- Precio: es un real.
- Disponible: VERDADERO si está disponible para la renta y FALSO si ya está rentado.

Diariamente acuden muchos clientes a la inmobiliaria solicitando información.

Escriba un programa capaz de realizar las siguientes operaciones sobre la información disponible:

a) Liste los datos de los departamentos disponibles que tengan un precio inferior o igual a un cierto calor P.

b) Liste los datos de los departamentos que tengan una superficie mayor o igual a un cierto valor E y una ubicación excelente.

c) Liste el monto de la renta de todos los departamentos alquilados.

d) Llega un cliente solicitando rentar un departamento. Si existe un departamento con una superficie mayor o igual a la deseada, con un precio y una ubicación que se ajustan a las necesidades del cliente, el departamento se renta. Actualizar los datos que correspondan.

e) Se vence un contrato, si no se renueva, actualizar los datos que correspondan.

f) Se ha decidió aumentar las rentas en X%. Actualizar los precios de las rentas de los departamentos no alquilados.

3.2.2 ARCHIVOS

En la mayoría de aplicaciones por computador se requiere que la información que se almacene permanezca para poder usarla en proceso posteriores. Para ello necesitamos otros medios de almacenamiento como los archivos que se guardan en la memoria auxiliar **(USB, discos, DVD, memoria flash, entre otros)** a través de dispositivos que no necesitan estar conectados constantemente al computador; de este modo solo se recurre a ellos cuando se necesita usar la información que tienen almacenada. El hecho de que estos aparatos residan fuera de la memoria principal y guarden la información de forma permanente permite que los datos sirvan para diferentes ejecuciones del mismo programa y sean usados por programas diferentes. Desde luego, también se puede guardar mucha más información que en la memoria principal.

Un archivo es, entonces,

Una estructura lógica donde se almacena en forma permanente información con las siguientes ventajas:

- ✓ **Almacenamiento de grandes volúmenes,**
- ✓ **Persistencia de esta en medios diferentes a la memoria principal del computador, y**
- ✓ **Posibilidad de que diferentes programas puedan acceder a ella.**

3.2.2.1 CLASIFICACIÓN DE LOS ARCHIVOS SEGÚN SU USO

Clasificación según su uso	Características
Archivos permanentes	Son **archivos rígidos**, ya que la información que almacenan cambia muy poco. **Se usan para extraer información** que se utiliza en otros **archivos o procesos**: por ejemplo, el archivo que contiene toda la información sobre los salarios de los empleados de una empresa en años posteriores.
Archivos maestros	**La información almacenada en esta clase de archivos maneja el estado o situación de los registros de una determinada base de datos, y su actualización o los cambios de los datos de los registros se hacen periódicamente**. Por ejemplo, un archivo que almacene el inventario de los bienes de una empresa.

Archivos históricos	Son **archivos maestros** que ya no se utilizan y cuya **función es solo de consulta** para quienes desean obtener información sobre el pasado. Por ejemplo, la información contenida en las hojas de vida académica de los estudiantes que terminaron su carrera en la década pasada.
Archivos de *movimiento	Son **temporales y su función principal es captar información para actualizar los archivos maestros.** Sus registros muestran las **transacciones o movimientos** que se producen durante un determinado periodo. Cuando el **archivo maestro** se actualiza, usando los **archivos de movimiento**, estos últimos pierden su validez y se pueden destruir. Por ejemplo: los cambios de aumento de salarios, deducciones y sobresueldos producidos durante un determinado periodo.

3.2.2.2 CLASIFICACIÓN DE LOS ARCHIVOS SEGÚN LOS DATOS QUE ALMACENAN

Clasificación de los archivos según los datos que almacenan	Características
Archivos de programa	Son los **programas fuente** que se escriben en un **determinado lenguaje de programación**, donde cada instrucción es un registro. **Estos archivos se pueden cargar del medio magnético a memoria, compilar, ejecutar, imprimir y volver a guardar** donde estaban o en otro medio de almacenamiento distinto.
Archivos de texto o ASCII (American Standard Code for Information Interchange)	Estos **almacenan cualquier carácter del código ASCII: palabras, frases, párrafos,** y se suelen crear y mantener mediante programas que procesan o editan texto.
Archivos binarios	Estos archivos **almacenan información en un lenguaje descifrable solo por el computador**. La información que puedan almacenar es diversa: ***colores, sonidos, imágenes, órdenes o datos discretos, llamados archivos de datos.***

Antes de hablar sobre los archivos de texto, definiremos algunas operaciones básicas que se deben tener en cuenta cuando se trabaja con archivos.

TIPS

Cuando se quiere guardar información para uso posteriores el almacenamiento permanente es fundamental.

3.3 TEMA 2 MANEJO DE ARCHIVOS DE DATOS Y OPERACIONES

- ***Video: "PCurso Python. Archivos externos I"***
- ***" Ver Video": https://www.youtube.com/watch?v=V87m9SltcI8"***

3.3.1.1 OPERACIONES CON ARCHIVOS

Los archivos son estructuras de almacenamiento de datos a nivel externo, lo cual permite manejar información para procesos posteriores, donde se puede manejar las diferentes operaciones sobre su conjunto de datos, estas operaciones son: **insertar, ordenar, buscar, actualizar, borrar, sobre uno o más datos.**

3.3.1.1.1 OPERACIONES BÁSICAS CON ARCHIVOS

A continuación, se definen las **acciones básicas con archivos:**

- **Creación del archivo:**

Permite **definir una variable dentro del programa que represente el archivo que se quiere manipular.** Esta **variable** es conocida como nombre interno del archivo y se define con la siguiente instrucción:

ARCHIVO a

- **Apertura de un archivo:**

Permite relacionar una **variable tipo archivo** con el nombre que el archivo tendrá en el **medio de almacenamiento**, llamado **nombre externo**. Este es el nombre con el cual se guarda el archivo en el medio de almacenamiento, y su extensión depende de la clase de información que se va a guardar. También es posible darle una **ruta que indique donde quedara guardado el archivo; si**

no se especifica la ruta, el archivo se guarda dónde está el programa que lo crea. La sintaxis de apertura es:

ABRIR a (“nombreExterno.extensión”)

Con esta instrucción **se genera una conexión entre el nombre interno a y el nombre externo del archivo**, donde **a** es la **variable tipo archivo** con la que se manejara el nombre externo del archivo en el programa, **nombre-externo** es el nombre que se le dará al archivo en el medio de almacenamiento y la **extensión** depende de la información que se quiere almacenar.

Esta instrucción crea la variable **a** como **variable tipo archivo y hace las conexiones físicas y lógicas con el nombre externo**. Al abrir el archivo la variable tipo archivo **a** toma el valor verdadero si el archivo se pudo abrir o falso si el archivo no se pudo abrir o no existe. **Esto nos permite preguntar si el archivo se abrió o no**. Al abrir un archivo hay que especificar el modo de apertura: **de salida, de entrada, entrada y salida, adicionar más información, etcétera.**

- **Escritura en el archivo:**

Permite **almacenar** la información **i** en el archivo que se está manipulando. La instrucción es:

ESCRIBA a (i)

Esta instrucción pasa la información **i** que reside en **memoria al archivo** cuyo nombre externo está relacionado con **a**.

- **Lectura de un archivo:**

Permite **extraer la información del archivo y almacenarla en memoria principal.** La información que se transporta a la memoria puede ser cualquier **tipo de variable u objetos** almacenados en el archivo. La sintaxis es:

LEA a (var)

La variable var puede representar cualquier tipo de dato: las más comunes son **variables de texto, estructuras y objetos.**

- **Cierre del archivo:**

Permite proteger el área donde reside el archivo y actualiza el directorio del medio de almacenamiento, reflejando el nuevo estado del archivo. Adicionalmente, **destruye las conexiones físicas y lógicas** que se construyeron cuando se abrió el archivo. La sintaxis es:

CERRAR a

Cuando se ejecuta la instrucción se adiciona un registro después del último grabado, donde coloca una marca de fin de archivo (**EOF – END OF FILE**).

3.3.1.2 ARCHIVO DE TEXTO

También se les conoce como archivos llanos o planos por el hecho de almacenar solo texto, es decir que almacenan caracteres, palabras, frases o párrafos y no tienen formato: contienen solo texto. Los caracteres que se almacenan pueden ser cualquiera del conjunto de caracteres de ASCII. Cada línea de texto es un registro, y la variable tipo archivo asignada en la definición del archivo se usa para fluir información desde un medio de entrada hacia la memoria auxiliar donde este reside y para extraer información desde el archivo hacia un medio externo de salida. Un ejemplo de un registro (línea de texto) seria:

Este es un archivo de texto que almacena un flujo de caracteres.

3.3.2 EJERCICIO DE APRENDIZAJE

Ejercicio de aprendizaje

Elaborar una clase que genere un archivo te texto con los campos cedula y nombres, imprima el contenido del archivo y adicione más registros al final del archivo.

1. Identificación de datos, acciones y limitaciones

- Datos de entrada: Cédula, nombre.
- Datos de salida: La información almacenada en el archivo.
- Acciones: crearArchivo, imprimirArchivo y adicionarRegistroAlFinal.

2. Definición de clases

- Clase seleccionada: Archivo.

- Definición del constructor de la clase: Archivo (): no recibe argumentos.
- Diagramación de la clase:

Archivo
-+Archivo() +crearArchivo() +imprimirArchivo() +adicionarRegistroAlFinal()

- **Explicación de los métodos: crearArchivo:** se crea un archivo de texto que solicite al usuario la información que se desea almacenar. El archivo se guarda en la carpeta donde está el programa, pero se puede establecer una ruta para que se almacene en cualquier otro lugar del computador. Se le pregunta al usuario si el archivo esta creado o no. En caso que el archivo este creado, lo redirige a la opción **ADICIONAR REGISTRO AL FINAL.** El archivo tiene como nombre interno "archivo" y como nombre externo "nombres.txt". la información que se guarda por línea es:

CÉDULA: 1158377694 NOMBRE: MAXIMILIANO DEL TORO

El usuario decide cuanta información desea almacenar. Cuando este termine de entrar la información, se cierra el archivo.

- **ImprimirArchivo:** haciendo uso de un ciclo MIENTRAS, se recorre e imprime el archivo línea por línea, hasta cuando se llegue a la marca de fin de archivo (OEF).
- **AdicionarRegistroAlFinal:** se abre el archivo en modo escritura, se le solicitan los datos por ingresar al usuario, se escriben al final del archivo y posteriormente se cierra.

3. **Definición del método principal:** Se lee la opción seleccionada por el usuario y se invoca el método correspondiente.

   Definición de variables:

   - **Archivo:**Oobjeto de la clase Archivo para invocar los métodos.

- **Opción:** Opción del menú seleccionada por el usuario.

Ejercicio de aprendizaje

```
1.  CLASE Archivo
2.      METODO PRINCIPAL()
3.          PUBLICO VOID CrearArchivo()
4.              ENTERO cedula
5.              CARÁCTER nombre
6.              CARÁCTER sw
7.              ESCRIBA: “¡SI EL ARCHIVO ESTA CREADO SERA DESTRUIDO!
8.                  ¿ESTA CREADO? (S/N):”
9.              LEA: sw
10.             si (sw==”N”) ENTONCES
11.                 CREAR archivo (“nombres.txt”)
12.                 ESCRIBA: “DIGITE LA CEDULA:”
13.                 LEA: cedula
14.                 mientras (cedula!=0) HAGA
15.                     ESCRIBA: “DIGITE EL NOMBRE:”
16.                     LEA: nombre
17.                     ESCRIBA archivo (“CEDULA:”, cedula, “\t”, “NOMBRE:”,
18.                             nombre)
19.                     ESCRIBA: “DIGITE NUEVA CEDULA O CERO PARA TERMINAR:”
20.                     LEA: cedula
21.                 end_mientras
22.                 CERRAR archivo
23.             end_si
24.         end_crearArchivo
25.         PUBLICO VOID imprimirArchivo()
26.             CADENA línea
```

```
27.         ABRIR archivo (“nombres.txt”)
28.         si (archivo) ENTONCES
29.             mientras (archivo<>EOF) HAGA
30.                 LEA archivo(línea)
31.                 ESCRIBA: línea
32.             end_mientras
33.             CERRAR archivo
34.         si_no
35.                 ESCRIBA: “EL ARCHIVO NOMBRES.TXT NO EXISTE O NO SE PUDO
36.                             ABRIR”
37.         end_si
38.         end_imprimirArchivo
39.         PUBLICO VOID  adicionarRegistroAlFinal()
40.                 ENTERO cedula
41.                 CARÁCTER nombre
42.                 ABRIR archivo (“nombres.txt”)
43.                 si (archivo)
44.                     ESCRIBA: ”DIGITE LA CEDULA:”
45.                     LEA: cedula
46.                     ESCRIBA: “DIGITE EL NOMBRE:”
47.                     LEA: nombre
48.                     ESCRIBA archivo (cedula+”\t”+ nombre)
49.                     CERRAR archivo
50.                 si_no
51.                 ESCRIBA: “EL ARCHIVO NOMBRES.TXT NO EXISTE O NO SE PUDO ABRIR”
52.                 end_si
53.         end_adicionarRegistroAlFinal
54.         METODO PRINCIPAL()
```

```
55.                 ENTERO opción
56.                 Archivo archivo
57.                 HACER
58.                         ESCRIBA: ”1: CREAR EL ARCHIVO”
59.                         ESCRIBA: ”2: IMPRIMIR CONTENIDO DEL ARCHIVO”
60.                         ESCRIBA: ”3: ADICIONAR REGISTRO AL FINAL”
61.                         ESCRIBA: ”4: SALIR DEL MENU”
62.                         ESCRIBA:” DIGITE OPCION:”
63.                         LEA: opción
64.                         CASOS DE (opción)
65.                                 CASO 1: archivo.crearArchivo()
66.                                 CASO 2: archivo.imprimirArchivo()
67.                                 CASO 3: archivo.adicionarRegistroAlFinal()
68.                          end_CASOS
69.                 MIENTRAS (opción!=4)
70.           end_PRINCIPAL
71.     end(Método)
72. end(Clase)
```

3.3.3 EJERCICIOS DE ENTRENAMIENTO

Pautas para desarrollar los siguientes ejercicios: Para desarrollar estos ejercicios, debes de tener en cuenta, declarar una clase y dentro de esta clase declarar las variables o atributos que va a tener el archivo que se desea crear.

1. Elaborar una clase que forme un archivo de datos que contenga en cada registro: cédula, nombre, horas trabajadas, valor de hora trabajada y porcentaje de deducción. La clase debe producir un informe (archivo de texto) con la cedula, el nombre y el salario devengado (ordenado ascendentemente por el campo cédula).

2. Elaborar una clase que cree un archivo de datos con la siguiente información por registro: cedula, apellidos y nombre y cuatro notas con sus respectivos porcentajes. Luego de

creado el archivo se pide mostrar un listado ordenado por apellido, insertar un nuevo estudiante, la calificación definida de cada estudiante y visualizar el contenido del archivo.

TIPS

El manejo de operaciones con archivos es de gran utilidad en la PROGRAMACIÓN.

3.4 TEMA 3 ANÁLISIS DE ALGORITMOS

- ***Video: "PCurso Python. Archivos externos I"***
- ***" Ver Video":*** *https://www.youtube.com/watch?v=V87m9SltcI8"*

Cuando se trabaja con procesos en línea y los algoritmos son utilizados con un grado alto de frecuencia, **se deben manejar algoritmos muy eficientes**, para ello hay que medir la eficiencia del algoritmo con su orden de magnitud, para tener certeza de que los algoritmos si son eficientes.

3.4.1 EVALUACIÓN DE ALGORITMOS

La evaluación de algoritmo consiste en medir la eficiencia de un algoritmo en cuanto a consumo de memoria y tiempo de ejecución.

Anteriormente, cuando existían grandes restricciones tecnológicas de memoria, **evaluar un algoritmo en cuanto a consumo de recursos de memoria era bastante importante**, ya que dependiendo de ella los esfuerzos de programación y de ejecución sería grande. **En la actualidad,** con el gran desarrollo tecnológico del hardware, **las instrucciones de consumo de memoria han pasado a un segundo plano**, por lo tanto, el aspecto de evaluar un algoritmo en cuanto a su consumo de memoria no es relevante.

En cuanto al **tiempo de ejecución**, a pesar de las altas velocidades de procesamiento que en la actualidad existen, **sigue siendo un factor fundamental**, especialmente en algoritmos que tienen que ver con **el control de procesos en tiempo real y en aquellos cuya frecuencia de ejecución es alta.**

Procedamos entonces a ver como **determinar el tiempo de ejecución de un algoritmo**. Básicamente **existen dos formas de hacer esta medición:**

- Una que llamamos **a posteriori,** y
- Otra que llamamos a **priori**.

Consideremos la primera forma, a posteriori: el proceso de elaboración y ejecución de un algoritmo consta de los siguientes pasos:

- Elaboración del algoritmo.
- Codificación en algún lenguaje de programación.
- Compilación del programa.
- Ejecución del programa en una maquina determinada.

Cuando se ejecuta el programa, los sistemas operativos proporcionan la herramienta de informar cuánto tiempo consumió la ejecución del algoritmo.

Esta forma de medición tiene algunos inconvenientes: al codificar el algoritmo en algún lenguaje de programación estamos realmente midiendo la eficiencia del lenguaje de programación, ya **que no es lo mismo un programa codificado en FORTRAN, que en PASCAL o que en C**; al ejecutar un programa en una máquina determinada estamos introduciendo otro parámetro, el cual es la eficiencia de la máquina, ya que no es lo mismo ejecutar el programa en un XT, en un 386, en un 486, en un Pentium, es un AS400, es un RISC, etc.

En otras palabras, con esta forma de medición estamos es evaluando la calidad del compilador y de la máquina, más no la del algoritmo, que es nuestro interés.

Lo que nos interesa es medir la eficiencia del algoritmo por el hecho de ser ese algoritmo, independiente del lenguaje en el que se haya codificado y de la máquina en la cual se ejecute.

Veamos entonces la forma de medir la ejecución de un **algoritmo a priori.** Para ello necesitamos definir dos conceptos básicos que son: **Contador de frecuencias y Orden de magnitud.**

3.4.2 CONTADOR DE FRECUENCIAS

El contador de frecuencias:
Es una expresión algebraica que identifica el número de veces que se ejecutan las instrucciones de un método.

Para ilustrar como determinar esta expresión consideramos los siguientes ejemplos:

MÉTODO PRINCIPAL1()

1. LEA (a, b, c)	1
2. x = a + b	1
3. y = a + c	1
4. z = b * c	1
5. w = x / y – z	1
6. IMPRIMA (a, b, c, w)	1
Contador de frecuencias	**6**

MÉTODO PRINCIPAL2()

1. LEA (n)	1
2. s = 0	1
3. i = 1	1
4. MQ (i <= n)	n+1
5. s = s + 1	n
6. i = i + 1	n
7. endMQ	n
8. IMPRIMA (n, s)	1
Contador de frecuencias	**4n+5**

En todos los métodos las instrucciones están numeradas secuencialmente. Al frente de cada instrucción aparece un número o una letra que indica el número de veces que se ejecuta esa instrucción en el método.

En el **método 1** cada instrucción se ejecuta solo una vez. El total de veces que se ejecutan todas las instrucciones es 6. Este valor es el contador de frecuencias para el método 1.

En el **método 2** las instrucciones 1, 2, 3 y 8 se ejecutan solo una vez, mientras que las instrucciones 5, 6 y 7 se ejecutan **n** veces cada una, ya que pertenece a un ciclo, cuya variable controladora del ciclo se inicia en uno, tiene un valor final de **n** y se incrementa de a uno, la instrucción 4 se ejecuta una vez más ya que cuando **i** sea mayor que **n** de todas formas hace la pregunta para luego salir del ciclo. Por tanto, el número de veces que se ejecutan las instrucciones del método es **4n + 5**. Esta expresión es el contador de frecuencias para el método 2.

Veamos otros ejemplos de métodos con su contador de frecuencias respectivo

METODO PRINCIPAL3()

1.	LEA (m, n)	1
2.	s = 0	1
3.	i = 1	1
4.	MQ (i <= n)	n+1
5.	t = 0	n
6.	j = 1	n
7.	MQ (j <= m)	n*m+n
8.	t = t + 1	n*m
9.	j = j + 1	n*m
10.	endMQ	n*m
11.	s = s + t	n
12.	i = i + 1	n
13.	endMQ	n
14.	IMPRIMA (n, m, s)	1
	Contador de frecuencias	**4(n*m) +7n+5**

MÉTODO PRINCIPAL4()

1.	LEA (n, m)	1
2.	s = 0	1
3.	i = 1	1
4.	MQ (i <= n)	n+1
5.	t = 0	n
6.	j = 1	n
7.	MQ (j <= n)	n2+n
8.	t = t + 1	n2
9.	j = j + 1	n2
10.	endMQ	n2

11.	s = s + t	n
12.	i = i + 1	n
13.	endMQ	n
14.	IMPRIMA (n, m, s)	1
Contador de frecuencias		**4n2+7n+5**

MÉTODO PRINCIPAL5()

1.	LEA (n, m)	1
2.	s = 0	1
3.	i = 1	1
4.	MQ (i <= n)	n+1
5.	s = s + 1	n
6.	i = i + 1	n
7.	endMQ	n
8.	IMPRIMA (n, s)	1
9.	t = 0	1
10.	i = 1	1
11.	MQ (i <= m)	m+1
12.	t = t + 1	m
13.	i = i + 1	m
14.	endMQ	m
15.	IMPRIMA (m, t)	1
Contador de frecuencias		**4m+4n+9**

De una manera análoga se obtuvieron los contadores de frecuencias para los métodos 3, 4 y 5.

Ejercicios de entrenamiento

Pautas para desarrollar los siguientes ejercicios: Las pautas que debes utilizar para hallar el contador de frecuencia, es mirar el número de veces que se ejecuta cada instrucción en el algoritmo, recuerda que el contador de frecuencia es una expresión algebraica, que sale del número de veces que se ejecuta cada instrucción.

Calcule el contador de frecuencias de los siguientes métodos:

```
1.    PUBLICO ESTATICO VOID Programa1 (n)
2.        s = 0
3.        PARA (i= 1, n, 1)
4.            PARA (j= 1, i, 1)
```

```
5.                      PARA (k= 1, j, 1)
6.                              s= s + 1
7.                      endPARA
8.              endPARA
9.        endPARA
10. end(Método)
```

```
1.      CLASE Programa2
2.       METODO PRINCIPAL()
3.        LEA (n)
4.        s = 0
5.        i = 2
6.        MQ (i <= n)
7.              s = s + 1
8.              i = i * i
9.        endMQ
10.            IMPRIMA (n, s)
11.      end(Método)
12.     end(Clase)
```

```
1.      CLASE Programa3
2.       MÉTODO PRINCIPAL()
3.        LEA (n)
4.        s = 0
5.        PARA (i= 1, n, 1
6.              t = 0
7.              PARA (j= 1, i, 1)
8.                      t = t + 1
9.              endPARA
10.             s = s + t
11.       endPARA
12.       IMPRIMA (n, s)
13.  end(Método)
14. end(Clase)
```

3.4.3 ORDEN DE MAGNITUD

EL ORDEN DE MAGNITUD
Es el concepto que define la eficiencia del método en cuanto a tiempo de ejecución. Se obtiene a partir del Contador de Frecuencias.

Para obtener el orden de magnitud seguimos los siguientes pasos:

- Se eliminan los coeficientes, las constantes y los términos negativos de los términos resultantes;
- Si son dependientes entre sí, se elige el mayor de ellos y este será el orden de magnitud de dicho método, de lo contrario el orden de magnitud será la suma de los términos que quedaron.
- Si el contador de frecuencias es una constante, **como en el caso del método 1, el orden de magnitud es 1 y lo representamos como O (1).**

Teniendo presente esto, los órdenes de magnitud de los métodos que vimos anteriormente son:

Método 1: O(1)
Método 2: O(n)
Método 3: O(n*m)
Método 4: O(n^2)
Método 5: O(n+m)

Es bueno aclarar lo siguiente:
En los ejemplos, el valor de n es un dato que se lee dentro del programa. En el caso más amplio, n podrá ser el número de registros que haya que procesar en un archivo, el número de datos que haya que producir como salida, o quizá otro parámetro diferente; es decir, hay que poner atención en el análisis que se haga del método, de cuál es el parámetro que tomamos como n.

En el ambiente de sistemas, **los órdenes de magnitud más frecuentes** para los métodos que se desarrollan son:

ORDEN DE MAGNITUD	REPRESENTACION
Constante	O(1)
Lineal	O(n)
Cuadrático	$O(n^2)$
Cubico	$O(n^3)$
Logarítmico	$O(\log_2(n))$
Semilogarítmico	$O(n \log_2(n))$
Exponencial	$O(2^n)$

Si quisiéramos clasificar estas órdenes de magnitud **desde la más eficiente hasta la menos eficiente**, tendríamos el siguiente orden:

1.	Constante	O(1)
2.	Logarítmico	$O(\log_2(n))$
3.	Lineal	O(n)
4.	Semilogarítmico	$O(n \log_2(n))$
5.	Cuadrático	$O(n^2)$
6.	Cúbico	$O(n^3)$
7.	Exponencial	$O(2^n)$

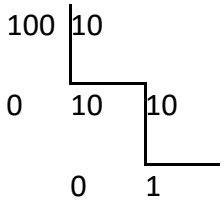
Hasta aquí hemos visto **métodos cuyo orden de magnitud es constante, lineal y cuadrático**. De igual manera que se hizo el método con orden **de magnitud cuadrático** se puede hacer uno **cúbico**; en el cuadrático se necesitaron 2 ciclos anidados, controlados por la misma variable, para el cúbico se necesitarían 3 ciclos con las mismas condiciones.

Pasemos a tratar **métodos con orden de magnitud logarítmicos**. Empecemos definiendo lo que es un logaritmo.

La **definición clásica del logaritmo** de un número **x** es: es el exponente al cual hay que elevar un número llamado base para obtener **x.**

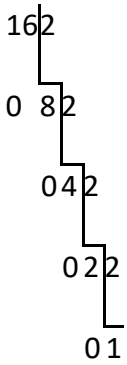
Con fines didácticos, presentamos otra definición de logaritmo: logaritmo de un numero **x** es el número de veces que hay que dividir un número, por otro llamado base, para obtener como cociente uno (1).

Por ejemplo, si tenemos el número 100 y queremos hallar su logaritmo en base 10, habrá que dividirlo por 10 sucesivamente hasta obtener como cociente uno (1).



Hubo que dividir el 100 dos veces por 10 para obtener 1 en el cociente, por tanto el logaritmo en base 10 de 100 es 2 (**$\log_{10}$ (100) = 2**).

Si queremos hallar el logaritmo en base 2 de 16 habrá que dividir 16 sucesivamente por 2 hasta obtener un cociente de 1. Veamos:



Es decir, hubo que dividir el 16, cuatro (4) veces por dos (2) para obtener un cociente de 1, por lo tanto, el logaritmo en base 2 de 16 es 4 (**$\log_2$ (16) = 4**).

Planteemos un **ejemplo práctico de algoritmo** en el cual se presenta esta situación.

Consideremos el caso de un vector en el cual tenemos los datos ordenados ascendentemente:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	2	4	7	9	11	15	17	28	31	36	43	50	58	62	69

Si nos interesa buscar el dato **x = 85** y decir en cual posición del vector se encuentra hay dos formas de hacerlo:

Una, es **hacer una búsqueda secuencial** a partir del primer elemento e ir comparando el contenido de esa posición con el dato **x** (85), el proceso terminará cuando lo encontremos o cuando hayamos recorrido todo el vector y no lo encontremos. Un pequeño método que haga ese proceso es el siguiente:

Sea **n** el número de elementos del vector y **x** el dato a buscar.

```
1.  i = 1
2.  MQ ((i <= n) & (V[i] != x))
3.          i = i + 1
4.  endMQ
5.  SI (i <= n)
6.          Pos = i
7.  SINO
8.          IMPRIMA ("el dato no existe")
endSI
```

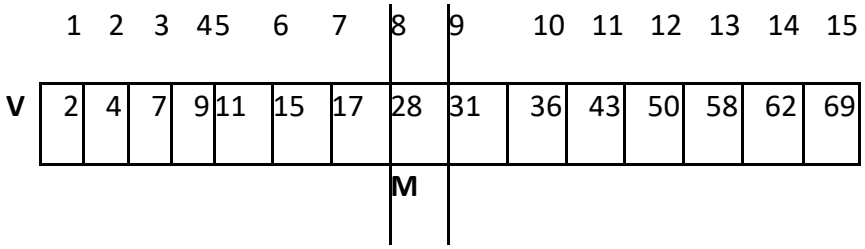
En dicho método tenemos planteado un ciclo, el cual, en el peor de los casos, se ejecuta **n** veces, es decir, cuando el dato **x** no esté en el vector. Por tanto el orden de magnitud de ese método es **O(n).**

En general, cuando se efectúa una búsqueda, **el peor caso es no encontrar lo que se está buscando**.

En nuestro ejemplo hubo que hacer 15 comparaciones para poder decir que el 85 no está en el vector.

Si el vector hubiera tenido quinientos mil de datos y hubiéramos tenido que buscar un dato que no estaba, hubiéramos tenido que haber hecho quinientas mil comparaciones para poder decir que el dato buscado no se encontró.

Ahora, si aprovechamos la característica de que los datos en el vector están ordenados podemos plantear **otro método de búsqueda**:



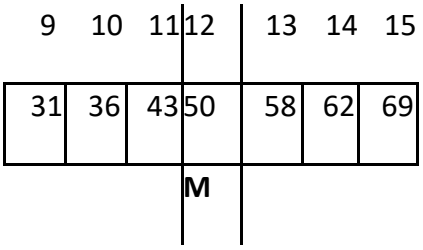
Comparamos el dato **x** con el dato del elemento de la mitad del vector, (llamémoslo **M**). Pueden suceder tres casos: **V(M) == x; V(M) > x; V(M) < x**

- Si $V(M) == x$ se ha terminado la búsqueda.
- Si $V(M) > x$ significa que el dato x, si se encuentra en el vector, estará en la primera mitad del vector.
- Si $V(M) < x$ significa que, si el dato x se encuentra en el vector, estará en la segunda mitad del vector.

En nuestro ejemplo, para seguir buscando el dato x (85) lo haremos desde la posición 9 hasta la posición 15.

Es decir, con unas comparaciones hemos eliminado la mitad de los datos sobre los cuales hay que efectuar la búsqueda. En otras palabras, hemos dividido la muestra por dos (2).

El conjunto de datos sobre el cual hay que efectuar la búsqueda queda así:



De la mitad que quedo escogemos nuevamente el elemento de la mitad y repetimos la comparación planteada anteriormente. Con esta comparación eliminamos nuevamente la mitad de los datos que nos quedan, es decir, dividimos nuevamente por dos.

Si continuamos este proceso, con cuatro comparaciones podremos afirmar que el dato **x** no está en el vector.

Hemos reducido el número de comparaciones de 15 a 4.

Cuatro (4) es el logaritmo en base 2 de 15, aproximándolo por encima.

Si **N** fuera 65000 posiciones en el vector, con 16 comparaciones podremos decir que un dato no se encuentra en un conjunto de 65000 datos.

Como puede observarse, **la ganancia en cuanto al número de comparaciones es grande**.

Un algoritmo de búsqueda que funcione con esta técnica tiene orden de magnitud logarítmico en base dos, ya que la muestra de datos se divide sucesivamente por dos.

Consideremos ahora, métodos sencillos cuyo orden de magnitud es logarítmico:

```
1.  n = 32
2.  s = 0
3.  i = 32
4.  MQ (i > 1)
5.          s = s + 1
6.          i = i / 2
7.  endMQ
IMPRIMA (n, s)
```

En el método anterior, la variable controladora del ciclo es **i,** y dentro del ciclo, **i** se divide por dos (2). Si hacemos un seguimiento detallado tendremos:

- **La primera vez que se ejecuta el ciclo, la variable i sale valiendo 16.**
- **La segunda vez que se ejecuta el ciclo, la variable i sale valiendo 8.**
- **La tercera vez que se ejecuta el ciclo, la variable i sale valiendo 4.**
- **La cuarta vez que se ejecuta el ciclo, la variable i sale valiendo 2.**
- **La quinta vez que se ejecuta el ciclo, la variable i sale valiendo 1 y termina el ciclo.**

Es decir, las instrucciones del ciclo se ejecutan 5 veces.

Cinco (5) es el logaritmo en base dos (2) de **n** (32), por consiguiente, cada instrucción del ciclo se ejecuta un número de veces igual al logaritmo en base dos de **n**. El contador de frecuencias y el orden de magnitud de dicho método se presentan a continuación:

1. n = 32	1
2. s = 0	1
3. i = 32	1
4. MQ (i > 1)	log2 n + 1
5. s = s + 1	log2 n
6. i = i / 2	log2 n
7. endMQ	log2 n
8. IMPRIMA (n, s)	1
Contador de frecuencias	**4 log2 n + 5**
Orden de magnitud	**O(log2 (n))**

Eliminando constantes y coeficientes el orden de magnitud es $O(\log_2 n)$.

Consideremos este nuevo método:

1. n = 32	1
2. s = 0	1
3. i = 1	1
4. MQ (i < n)	log2 n + 1
5. s = s + 1	log2 n
6. i = i * 2	log2 n
7. endMQ	log2 n
8. IMPRIMA (n, s)	1

En este método la variable controladora del ciclo **i** se multiplica por dos dentro del ciclo.

Haciendo un seguimiento tendremos:

- **En la primera pasada, la variable i sale valiendo 2.**
- **En la segunda pasada, la variable i sale valiendo 4.**
- **En la tercera pasada, la variable i sale valiendo 8.**
- **En la cuarta pasada, la variable i sale valiendo 16.**
- **En la quinta pasada, la variable i sale valiendo 32 y el ciclo se termina.**

Las instrucciones del ciclo se ejecutan 5 veces, por tanto, **el orden de magnitud del método es logarítmico en base 2 (O($\log_2$ n)).**

En general, **para determinar si un ciclo tiene orden de magnitud lineal o logarítmica, debemos fijarnos cómo se está modificando la variable controladora del ciclo**: Si esta se modifica mediante sumas o restas el orden de magnitud del ciclo es lineal, si se modifica con multiplicaciones o divisiones el orden de magnitud del ciclo es logarítmico.

Consideremos ahora el siguiente método:

```
1. n = 81
2. s = 0
3. i = 81
4. MQ (i > 1)
5. s = s + 1
6. i = i / 3
7. endMQ
8. IMPRIMA (n, s)
```

Aquí, la variable controladora del ciclo **i** se divide por tres dentro del ciclo. Haciéndole el seguimiento tendremos:

- **En la primera pasada, la variable i sale valiendo 27.**
- **En la segunda pasada, la variable i sale valiendo 9.**
- **En la tercera pasada, la variable i sale valiendo 3.**
- **En la cuarta pasada, la variable i sale valiendo 1, y termina el ciclo.**

Las instrucciones del ciclo se ejecutaron 4 veces. Cuatro es el logaritmo en base 3 de 81. Por consiguiente, el orden de magnitud de dicho método es logarítmico en base 3 (O ($\log_3$ n)).

En general, si tenemos un método:

```
1. n = 81
2. s = 0
3. i = 81
4. MQ (i > 1)
```

```
5.  s = s + 1
6.  i = i / X
7.  endMQ
8.  IMPRIMA (n, s)
```

Aquí, la variable controladora del ciclo se divide por **x.** El número de veces que se ejecutan las instrucciones del ciclo es logaritmo en base **x** de **n** (O ($\log_x$ n)), por tanto, el orden de magnitud es logaritmo en base **x.**

Para obtener métodos con orden de **magnitud semilogarítmico** basta con que un ciclo logarítmico se ubique dentro de un ciclo con orden de magnitud **n** (O(n)) o viceversa. Por ejemplo:

```
1.  LEA (n)                    1
2.  s = 0                      1
3.  i = 1                      1
4.  MQ (i <= n)                n+1
5.      t = 0                  n
6.      j = n                  n
7.      MQ (j > 1)             n*log2n +n
8.          t = t + 1          n*log2n
9.          j = j / 2          n*log2n
10.     endMQ                  n*log2n
11.     IMPRIMA (t)            n
12.     s = s + t              n
13.     i = i + 1              n
14. endMQ                      n
15. IMPRIMA (n, s)             1
```

Contador de frecuencias **4nlog2n+8n+5**

Orden de magnitud **O(n log2n)**

En el método anterior el orden de magnitud es **semilogarítmico.**

Digamos que la idea es elaborar métodos lo más eficientes posible. Consideremos el siguiente problema:

Elaborar un método que lea un entero positivo **n** y que determine la suma de los enteros desde 1 hasta **n.**

Una solución es la siguiente:

```
1. LEA (n)
2. s=0
3. i = 1
4. MQ (i <= n)
5.       s = s + i
6.       i = i + 1
7. endMQ
8. IMPRIMA (n, s)
```

El anterior método lee un dato entero **n** (**n>0**), calcula e imprime la suma de los enteros desde 1 hasta **n,** y el orden de magnitud de éste es **O(n)**.

Matemáticamente hablando, **la sumatoria de los enteros desde 1 hasta n, se representa así:**

$$S = \sum_{i=1}^{i=n} i$$

Esta representación, convertida en **fórmula matemática**, quedaría de la siguiente forma:

$$\frac{n * (n + 1)}{2}$$

Conociendo esta fórmula, podemos elaborar otro método que ejecute exactamente la misma tarea que el anterior. Veamos cómo sería:

```
1. LEA (n)
2. s = n * (n + 1) / 2
3. IMPRIMA (n, s)
```

Éste método tiene orden de **magnitud lineal (O(1))**. Es decir, tenemos dos métodos completamente diferentes que ejecutan exactamente la misma tarea, uno con orden de magnitud **O(n)** y el otro con orden de magnitud **O (1).**

Lo anterior no significa que toda tarea podrá ser escrita con **métodos O(1),** no, sino que de acuerdo a los conocimientos que se tengan o a la creatividad de cada cual, se podrá elaborar métodos más eficientes.

Este ejemplo, el cual es válido, desde el punto de vista de desarrollo de métodos, **es aplicable también a situaciones de la vida real**. Seguro que alguna vez usted tuvo algún problema, y encontró una solución, y salió del problema, aplicando la solución que encontró, y sin embargo, **al tiempo**, dos o tres meses después, pesando en lo que había hecho, encontró que pudo haber tomado **una mejor determinación** y que le habría ido mejor.

La enseñanza es que cuando se nos presente un problema debemos encontrar como mínimo dos soluciones y luego evaluar cuál es más apropiada.

3.4.4 EJERCICIOS DE ENTRENAMIENTO

Pautas para desarrollar los siguientes ejercicios: Recuerda que, primero se debe hallar el contador de frecuencia que es una expresión algebraica, luego de obtener el contador de frecuencia se pasa a hallar el orden de magnitud, que es el que define la eficiencia del algoritmo. **Recuerda** que, para hallar el orden de magnitud se eliminan los coeficientes, las constantes y los términos negativos de los términos resultantes

Determine el orden de magnitud de los siguientes métodos, a los cuales en un ejercicio pasado les calculamos el contador de frecuencias:

```
1.  PUBLICO ESTATICO VOID Programa1 (n)
2.          s = 0
3.          PARA (i= 1, n, 1)
4.                  PARA (j= 1, i, 1)
5.                          PARA (k= 1, j, 1)
6.                                  s = s + 1
7.                          endPARA
8.                  endPARA
9.          endPARA
10. end(Método)



1.  CLASE Programa2
2.     MÉTODO PRINCIPAL()
3.          LEA (n)
```

```
4.        s = 0
5.        i = 2
6.        MQ (i <= n)
7.                s = s + 1
8.                i = i * i
9.        endMQ
10.       IMPRIMA (n, s)
11.   end(Método)
12. end(Clase)
```

```
1.  CLASE Programa3
2.     MÉTODO PRINCIPAL()
3.        LEA (n)
4.        s = 0
5.        PARA (i= 1, n, 1
6.                t = 0
7.                PARA (j= 1, i, 1)
8.                        t = t + 1
9.                endPARA
10.               s = s + t
11.       endPARA
12.       IMPRIMA (n, s)
13.    end(Método)
14. end(Clase)
*
```

4. Elabore los siguientes métodos y determine el contador de frecuencias y el orden de magnitud dichos métodos:

   a) Realizar un método que calcule la suma de la siguiente serie aritmética para **n** términos:

      2, 5, 8, 11, 14,…, n

   b) Realizar un método que calcule la suma de la siguiente serie aritmética para **n** términos:

      2, 4, 8, 16, 32, 64,…n

TIPS

Asignar información a las variables es importantísimo porque son los valores con los cuales se va a trabajar.

3.5 TEMA 4 MANEJO DE HILERAS DE CARACTERES

- ***Video: " Programación en C - MANEJO DE CADENAS"***
- ***" Ver Video":*** https://www.youtube.com/watch?v=BKnIjerMHGA"

3.5.1 RELACIÓN DE CONCEPTOS

- **Asignación:** Consiste en asignarle a una variable una hilera de caracteres.
- **Borrar:** Este método consiste, a partir de la posición **i** de la hilera **x**, borra **j** caracteres.
- **Concatenar:** Consiste en unir una hilera a partir de otra.
- **Hileras:** Es un tipo de dato construido con base al tipo primitivo char.
- **Insertar:** Este método consiste en insertar una hilera **x**, a partir del carácter **i**, de la hilera y.
- **Longitud:** Consiste en verificar la cantidad de caracteres de la cual se compone la hilera.
- **Posición:** Consiste en determinar, a partir de cual posición de una hilera **x**, se encuentra una hilera **y**; si es que se encuentra.
- **Reemplazar:** Este método consiste que a partir de la posición **i** de la hilera **x**, reemplaza **j** caracteres por una copia de la hilera **y**.
- **Subhilera**: Parte que se toma de una hilera.

DEFINICIÓN

Las hileras es un objeto de datos compuestos, **construidos con base en el tipo primitivo char**, su mayor aparición fue cuando empezaron a aparecer los procesadores de textos.

OPERACIONES CON HILERAS

Las operaciones que podemos hacer con cadenas son las operaciones de **asignación, longitud, subhilera, concatenar, insertar, borrar, reemplazar y posición en una cadena.**

Consideremos ahora las **operaciones básicas** que se pueden ejecutar con hileras**. Definamos una clase hilera con sus operaciones:**

```
1.  CLASE Hilera
2.      Publico
3.          entero longitud()
4.          hilera subHilera(entero i, entero j)
5.          hilera concat(hilera s)
6.          void inserte(hileras, entero j)
7.          void borre(entero i, entero j)
8.          void replace(entero i, entero j, hilera s)
9.          entero posición(hilera s)
10. end(Clase)
```

3.5.2 ASIGNACIÓN

En el lado izquierdo tendremos el nombre de la variable a la cual se le asigna una hilera dada. En el lado derecho se tendrá una hilera o una variable tipo hilera. Por ejemplo, si S, T, U y V son variables tipo hilera podemos tener las siguientes instrucciones:

```
S = "abc"
T = "def"
U = T
V = ""
```

En el primer ejemplo la variable **S** contendrá la hilera “abc”.

En el segundo ejemplo la variable **T** contendrá la hilera “def”.

En el tercer ejemplo la variable **U** contendrá la hilera “def”.

En el cuarto ejemplo la variable **V** contendrá la hilera vacía.

3.5.3 FUNCIÓN LONGITUD

Forma general: longitud(S)

Esta función retorna el número de caracteres que tiene la hilera **S**. por ejemplo, si tenemos S = “mazamorra” y ejecutamos la instrucción:

n = S. Longitud ()

La variable **n** quedara valiendo 9.

3.5.4 FUNCIÓN SUBHILERA

Forma general: T = S.subHilera(i, j), con $1 \leq i \leq i + j - 1 \leq n$

Siendo **n** la longitud de la hilera S.

Esta función, a partir de la posición **i** de la hilera **S**, extrae **j** caracteres creando una nueva hilera y dejando intacta la hilera original S. por ejemplo, si tenemos S = “mazamorra” y ejecutamos la instrucción:

T = S.subHilera(4, 5)

La variable T quedara valiendo “amorr”.

3.5.5 FUNCIÓN CONCATENAR

Forma general: U = S.Concat(T)

Esta función crea una copia de la hilera S y a continuación agrega una copia de la hilera T. las hileras S y T permanecen intactas. Por ejemplo, si S = “nacio” y T “nal” y ejecutamos la instrucción:

U = S.concat(T)

La variable U quedara conteniendo la hilera “nacional”.

3.5.6 MÉTODO INSERTAR

Forma general: S.inserte(T, i)

Este método inserta la hilera T a partir del carácter de la posición **i** de la hilera S. por ejemplo, si tenemos S = “nal” y T = “ciona” y ejecutamos la instrucción:

S.inserte(T, 3)

La hilera S quedara valiendo “nacional”.

Es necesario resaltar que en este caso la hilera S queda modificada mientras que la hilera T permanece intacta.

3.5.7 MÉTODO BORRAR

Forma general: S.borre(i, j)

Este método, a partir de la posición **i** de la hilera S borra **j** caracteres. Por ejemplo, si S = “amotinar” y ejecutamos la instrucción:

S.borre(4, 4)

La hilera S quedara valiendo “amor”.

Es bueno anotar también que la hilera S queda modificada.

3.5.8 MÉTODO REEMPLAZAR

Forma general: S.replace(i, j, T)

Este método, a partir de la posición **i** de la hilera S reemplaza **j** caracteres por una copia de la hilera T. por ejemplo, si la hilera S = “abcdf” y la hilera T = “xyz” y ejecutamos la instrucción:

S.replace(3, 2, T)

La hilera S quedara valiendo “abxyzef”.

Este método modifica la hilera S, mientras que la hilera T permanece intacta.

3.5.9 FUNCIÓN POSICIÓN

Forma general: m = S.posicion(T)

Esta función determina a partir de cual posición de la hilera S se encuentra la hilera T, si es que se encuentra la hilera T en la hilera S. en caso de no encontrar la hilera T en la hilera S retornara cero. Por ejemplo, si S = “mazamorra” y T = “amor” y ejecutamos la instrucción:

m = posición(T)

La variable **m** quedara valiendo 4.

Las operaciones aquí descritas nos proporcionan las facilidades para manipular hileras en cualquier lenguaje de programación.

Es importante agregar que lenguajes de programación como C++ y java traen definida la clase hilera (String) en la cual se implementa una gran cantidad de variaciones correspondientes a estas operaciones.

Ejercicios de aplicación

Con las operaciones mencionadas en el numeral anterior, se pueden realizar muchos ejemplos de aplicación, por ejemplo: cuando se tiene un texto y se solicita determinar cuál es la frecuencia de cada una de las letras del alfabeto español, es decir, cuantas veces aparece la letra “p”, cuantas veces aparece la letra “o” y así sucesivamente.

3.5.10 APLICACIÓN DE LA CLASE HILERA

Pasemos a considerar **ejemplos de aplicación** con las funciones anteriormente definidas.

Empecemos considerando un ejercicio en el cual nos solicitan determinar cuál es la frecuencia de cada una de las letras del alfabeto español en un determinado texto dado, es decir, cuantas veces se aparece la “a” cuantas veces aparece la “b” y así sucesivamente.

Llamemos **texto** la variable en la cual hay que hacer dicha evaluación. Para desarrollar dicho método utilizaremos una variable llamada **alfabeto**, que es una variable tipo hilera y que definiremos así:

alfabeto = “abcdefghijklmnñopqrstuvwxyz”

Utilizaremos un vector de 27 posiciones, que llamaremos **frecuencia**, en cada una de las cuales guardaremos el número de veces que se encuentre alguna letra del alfabeto definido. Es decir, a la letra “a” le corresponde la posición 1 del vector de frecuencias, a la letra “b” la posición 2, a la letra “c” la posición 3 y así sucesivamente.

Nuestro método consistirá en recorrer los caracteres de la variable **texto**, buscando cada carácter en la variable **alfabeto** utilizando la función **posición**; cuando la encuentre sumaremos 1 a la posición correspondiente a esa letra en el vector de **frecuencia**.

Nuestro método es:

```
1.  void frecuenciaLetras(hilera alfabeto)
2.      entero m, i, j, n
3.      hilera car
4.      m = alfabeto.longitud()
5.      for (i = 1; i <= m; i++) do
6.          frecuencia[i] =0
7.      end(for)
8.      n = longitud()
9.      for (i = 1; 1 <= n; i++) do
10.         car = subHilera(i, 1)
11.         j = alfabeto.posicion(car)
12.         if (j != 0) then
13.             frecuencia[j] = frecuencia[j] + 1
14.         end(if)
15.     end(for)
16.     for (i = 1; i <= m; i++) do
```

```
17.        letra = alfabeto.subHilera(i, 1)
18.        write(letra, frecuencia[i])
19.    end(for)
20. end(Método)
```

Consideremos ahora un método en el cual nos interesa determinar la frecuencia de cada palabra que aparezca en un texto. Para desarrollar dicho método debemos, **primero que todo**, identificar cada una de las palabras del texto. Para lograr esta identificación hay que definir cuales caracteres se utilizan como separadores de palabras. Estos caracteres pueden ser cualquier símbolo diferente de letra, es decir, la coma, el punto, el punto y coma, los dos puntos, el espacio en blanco, etc.

Nuestro método manejara las siguientes variables:

Variables	Características
Texto	Variable en la cual **se halla almacenado** el texto a procesar.
Palabras	Variables **tipo vector** en la cual almacenaremos cada palabra que se identifique en el texto.
Frecuencia	Variable **tipo vector** en la cual almacenaremos **el número de veces** que se encuentre cada palabra del texto. Una palabra que se encuentre en la posición **i** del vector **palabra**, en la posición **i** del vector **frecuencia** se hallara el número de veces que se ha encontrado.
palabraHallada	Variable en la cual almacenaremos cada palabra que se **identifique** en el texto.
k	Variable para contar **cuantas palabras diferentes** hay en el texto. La inicializamos en 1. El total de las palabras diferentes encontradas será **k – 1.**

n	Variable que **guarda el número de caracteres** en el texto.
Alfabeto	Es una **hilera enviada como parámetros**, la cual contiene los **símbolos** con los cuales se construyen las palabras.

Nuestro método es el siguiente:

```
1.  void frecuenciaPalabras(hilera alfabeto)
2.      entero i, k, j, p, n, frecuencia[1000]
3.      hilera car, palabras[1000]
4.      k = 1
5.      for (i = 1; i <= 1000; i++) do
6.          frecuencia[i] = 0
7.      end(for)
8.      n = longitud()
9.      i = 1
10.     while i <= n do
11.         car = subHilera(i, 1)
12.         p = alfabeto.posicion(car)
13.         while (i < n and p == 0) do
14.             i = i + 1
15.             car = subHilera(i, 1)
16.             p = alfabeto.posicion(car)
17.         end(while)
18.         j = i
```

```
19.         while (i < n  and  p != 0) do
20.             i = i + 1
21.             car = subHilera(i, 1)
22.             p = alfabeto.posicion(car)
23.         end(while)
24.         palabraHallada = subHilera(j, i - j)
25.         palabras[k] = palabraHallada
26.         j = 1
27.         while (palabras[j] != palabraHallada) do
28.             j = j + 1
29.         end(while)
30.         if (j == k) then
31.             frecuencia[k] = 1
32.             k = k + 1
33.         else
34.             frecuencia[j] = frecuencia[j] + 1
35.         end(if)
36.     end(while)
37.     k = k – 1
38.     for (i = 1; i <= k; i++) do
39.         write(palabras[i], frecuencia[i])
40.     end(for)
41. end(Método)
```

En la instrucción 2 y 3 **se definen las variables** con las cuales se va a trabajar.

En las instrucciones 5 a 7 se **inicializan los contadores** de palabras en 0.

En las instrucción 8 se determina la **longitud del texto** a analizar **(el que invoco el método).**

En la instrucción 9 se **inicializa la variable i en 1**. Utilizamos la variable i para recorrer el texto, carácter por carácter, e ir identificando las diferentes palabras en él.

Las instrucciones 10 a 36 conforman el **ciclo principal del método**.

En las instrucciones 11 a 17 se **omiten** los caracteres que no conforman una palabra valida. Los caracteres que conforman una palabra valida son los que pertenecen al alfabeto. Cada carácter del texto se busca en **la hilera alfabeto** (instrucciones 12 y 16): si no lo encuentra significa que no conforma palabra y por lo tanto se desecha. Del ciclo de las instrucciones 13 a 17 se sale cuando encuentre un carácter que pertenezca al alfabeto. Esto significa que a partir de esa posición comienza una palabra. Dicha posición la guardaremos en una variable que llamamos **j** (instrucción 18).

En el ciclo de las instrucciones 19 a 23 se determinan hasta cual posición llega una palabra de este ciclo se sale cuando encuentre un carácter que no pertenezca al alfabeto; por consiguiente, la palabra será **la subhilera desde la posición j hasta la posición i – 1**, la cual extrae con la instrucción 24.

En la instrucción 25 **se almacena la palabra hallada** en la posición **k** del vector palabras. Con el fin de determinar si la palabra almacenada en la posición **k** del vector es primera vez que aparece o ya estaba, la buscamos en el vector de palabras (instrucciones 26 a 29) con la certeza de que la encontraremos: si se encuentra en la posición **k** significa que es la primera vez que aparece y por lo tanto le asignamos 1 al contador de esa palabra (instrucción 31) e incrementamos la variable **k** en 1; si encontró la palabra en una posición diferente a la posición **k** significa que dicha palabra ya se había encontrado y por lo tanto incrementamos su respectivo contador en 1 (instrucciones 34).

Por último, en las instrucciones 37 a 40 se describen las diferentes palabras encontradas con su respectivo contador.

Planteemos ahora un logaritmo que considero interesante: **reemplazar una palabra por otra en un texto dado.**

Utilizaremos tres variables:

Variables	Características
texto	Variable que contiene el **texto** donde hay que hacer el **reemplazo**.
Vieja	Variable que contiene **la hilera** que hay que **reemplazar.**
Nueva	Variable que contiene la **hilera** que **reemplazara la hilera vieja**.

Para entender el desarrollo de este método debemos entender lo siguiente: si tenemos un texto **S = "aabcdabcde"** y **T = "abc"** y ejecutamos la instrucción

m = S.posicion(T)

La variable **m** queda valiendo 2. Esto significa que a partir de la posición 2 de la hilera S se encontró la hilera T, es decir, nuestro método retorna el sitio donde encuentra la primera ocurrencia de la hilera que se está buscando. Fíjese que en nuestra hilera S la hilera "abc" se halla en la posición 2 y en la 6.

Si ejecutamos la siguiente instrucción (fíjese que se está buscando la hilera T a partir de la posición 5 de S):

m = S.subHilera(5, 6).posicion(T) **(formula 1)**

La variable **m** también queda valiendo 2. ¿Por qué? La razón es que la hilera **T** la buscara en la hilera S.subHilera(5, 6), la cual es: "babcde".

La pregunta es: ¿está realmente la segunda ocurrencia de la hilera T en la posición 2 de S? la respuesta es no.

Si planteamos una utilización de la función posición como en la **fórmula 1** y queremos determinar en cual posición de **S** comienza la subhilera T, al resultado obtenido habrá que sumarle **i – 1,** siendo i el primer parámetro de la función subHilera. En nuestro caso **i = 5.**

Llamemos **posini** la variable a partir de la cual se inicia la búsqueda de una hilera en una subhilera obtenida con la función subhilera.

Nuestro método es:

```
1.  void reemplazarViejaPorNueva(hilera vieja, hilera nueva)
2.      v = vieja.longitud()
3.      n = nueva.longitud()
4.      t = longitud()
5.      posini = 1
6.      sw = 1
7.      while (sw == 1) do
8.          p = subHilera(posini, t – posini + 1)
9.          posvieja = p.posicion(vieja)
10.         if (posvieja > 0) then
11.             posreal = posvieja + posini – 1
12.             replace(posreal, v, nueva)
13.             posini = posreal + n + 1
14.             t = longitud()
15.             if(posini > t + v 1) then
16.                 sw = 0
17.             end(if)
18.         else
19.             sw = 0
20.         end(if)
```

21. end(while)

22. end(Método)

3.5.11 TALLER DE ENTRENAMIENTO

Pautas para desarrollar los siguientes ejercicios: Para desarrollar estos ejercicios debe tener en cuenta, todas las funciones y métodos vistos en el tema. Recuerda que, puedes utilizar una estructura tipo vector o una estructura lista ligada; si lo requiere, para dar solución a los problemas.

1. Elabore un método para determinar si una hilera dada constituye un palíndromo o no. Un palíndromo es una hilera que se lee igual de izquierda a derecha que de derecha a izquierda; por ejemplo: radar, reconocer, abba. Su método debe retornar verdadero si cumple la condición, falso de lo contrario.

2. Elabore un método que determine si una palabra tiene más vocales que consonantes o no. Su método debe retornar verdadero si cumple la condición; de lo contrario, debe retornar falso.

3. Elabore un método que invierta todos los caracteres de una hilera dada. Por ejemplo, si la hilera es “amor”, al ejecutar su método debe quedar la hilera “roma”.

4. Elabore un método que determine si una palabra tiene las cincos vocales o no. Su método debe retornar verdadero si cumple la condición; de lo contrario, debe retornar falso.

4 PISTAS DE APRENDIZAJE

- **Traer a la memoria** La división entre cero no está definida en ningún campo numérico. Cuando en el numerador hay un número diferente de cero y en el denominador está el cero se dice que el resultado no existe; si en el numerador y en el denominador está el cero, se dice que el resultado es indefinido.

- **Traer a la memoria** Cuando se suma dos números, si los signos son iguales, se suma los números y se conserva el signo que tienen; si los signos son contrarios, se restan y se conserva el signo del número mayor.

- **Tenga presente** El orden en que se efectúan operaciones es: Primero potencias o raíces, luego multiplicaciones o divisiones y por último sumas y restas.

- **Tenga presente** que todo lo que está entre paréntesis () tiene mayor prioridad

- **Tenga presente** que los métodos son acciones y siempre deben de ir en un verbo en infinitivo

- **Tenga presente** que la clase siempre debe de ser en singular

- **Tenga presente** que las matrices son estructuras de almacenamiento de datos, donde se manejan dos subíndices, uno para fila y otro para columna; y el orden en cualquier operación es primero el subíndice de fila y luego el subíndice de columna.

- **Traer a memoria** los arreglos se consideran estructuras estáticas

- **Tenga presente** que los ciclos se utilizan siempre y cuando una instrucción se necesita repetir más de una vez

- **Recuerde que,** a la hora de intercambiar datos dentro de un vector o una matriz, primero debe guardar el contenido de una celda a dentro de una variable auxiliar, para no perder

lo que hay en a; luego a la celda a le lleva lo que tiene la celda b; y, por último, a la celda b le lleva lo que guardó en la variable auxiliar. Así por ejemplo:

- aux = vec[i] vec[i] = vec[k] vec[k] = aux

- **Recuerde que** los parámetros por referencia son aquellas variables que se modificarán dentro del subprograma y volverán al programa principal con su valor modificado.

- **Recuerde que** cuando se invoca un método void, el control de ejecución retorna a la instrucción siguiente del método que lo activo.

- **Recuerde que** cuando se invoca funciones que retornan, la dirección de retorno es la misma instrucción que las invoca.

- **Tenga presente** que los métodos deben ser lo más simple que se pueda.

- **Tenga presente** que por seguridad se recomienda declara las variables como privadas o protegidas.

- **Tenga presente** que las variables que hacen parte de una expresión tipo lógico deben tener un valor previo asignado

5 GLOSARIO

- **Algoritmo.** Lista de instrucciones para resolver un problema abstracto, es decir, que un número finito de pasos convierten los datos de un problema (entrada) en una solución (salida).
- **Aplicación.** Es un programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajo.
- **Booleano.** Es un tipo de dato lógico que puede representar valores de lógica binaria; es decir, valores que representen falso o verdadero.
- **Ciclo o bucle.** Es una instrucción que se ejecuta repetidas veces dentro de un algoritmo, hasta que la condición asignada a dicho ciclo deje de cumplirse.
- **Condición.** Es una instrucción o grupo de instrucciones que determinan si se realizará la siguiente instrucción o no.
- **Decremento.** Es una disminución que se le realiza a una determinada variable.
- **Factorial.** Se llama factorial de n al producto de todos los números naturales desde 1 hasta n.
- **Incremento.** Es el aumento que sufre una determinada variable.
- **Instrucción.** Es un conjunto de datos insertados en una secuencia estructurada o específica que el procesador interpreta y ejecuta.
- **Iteración.** Es la repetición de una serie de instrucciones dentro de un algoritmo.
- **Límite inferior.** Es el máximo valor que puede tomar una variable dentro de un ciclo PARA.
- **Límite superior.** Es el mínimo valor que puede tomar una variable dentro de un ciclo PARA.
- **Secuencia.** Es una serie de pasos lógicos que tienen coherencia y un orden determinado.
- **Switch o switche.** Generalmente, se utiliza en la condición de un ciclo MIENTRAS QUE o de un ciclo HAGA MIENTRAS QUE. Si la condición se cumple, se ejecutan las instrucciones

que hay dentro de dicho ciclo; de lo contrario, salta hasta el final del ciclo. También suele usarse como condición en las estructuras de decisión.

- **Variables.** Son espacios reservados en la memoria que, como su nombre indica, pueden cambiar de contenido a lo largo de la ejecución de un programa.

6 BIBLIOGRAFÍA

Este capítulo recomienda al estudiante las fuentes de consulta bibliográficas y digitales para ampliar su conocimiento, por lo tanto, deben estar en la biblioteca digital de la Remington. Utilice la biblioteca digital http://biblioteca.remington.edu.co/es/ *para la consulta de bibliografía a la cual puede acceder el estudiante.*

FUENTES BIBLIOGRÁFICAS

- *Agilar, J. (2008). Fundamentos de programación . Madrid: MCGRAW-HILL.*
- *Florez , R. (2011). Algoritmia básica. Medellín: Imprenta Universidad de Antioquia.*
- *Oviedo, E (2015). Lógica de Programación Orientada a Objetos. 1ª .ed. Bogotá: Ecoe Ediciones: Universidad de Antioquia.*
- *Villalobos, J. (2006). Fundamentos de programación. Naucalpan de Juarez: Prentice Hall.*

FUENTES DIGITALES O ELECTRÓNICAS

- Foundation, T. G. (2012). Estructuras de datos: listas enlazadas, pilas y colas. Obtenido de Listas enlazadas.: http://www.calcifer.org/documentos/libroglib-lists-queues.html
- Alejandro, J. (s.f.). Resolución de Problemas y Algoritmos. Obtenido de 2015: http://cs.uns.edu.ar/~wmg/rpa15lz/downloads/Clases%20Teoricas/ale-2015-RPA11(Archivos-de-texto).pdf