

## Contenido

<b>Introducción a la algoritmia .....</b>	<b>5</b>
<i>Programación estructurada .....</i>	<i>9</i>
<i>Dato.....</i>	<i>10</i>
<i>Tipos de datos.....</i>	<i>10</i>
<i>Estructuras de datos simples .....</i>	<i>10</i>
<i>Operadores y operandos.....</i>	<i>11</i>
<i>Operadores .....</i>	<i>11</i>
<i>Operadores de asignación .....</i>	<i>12</i>
<i>Operadores Aritméticos.....</i>	<i>12</i>
<i>Operadores relacionales .....</i>	<i>12</i>
<i>Operadores lógicos .....</i>	<i>13</i>
<i>Prioridad de los operadores .....</i>	<i>13</i>
<i>¿Por qué debo usar la prueba de escritorio? .....</i>	<i>14</i>
<i>Pasos para elaborar una prueba de escritorio.....</i>	<i>14</i>
<i>Acciones Simples .....</i>	<i>15</i>
<i>Acciones compuestas o estructuradas.....</i>	<i>16</i>
<i>Cíclicas o iterativas: .....</i>	<i>18</i>
<b>Introducción a secuencias .....</b>	<b>19</b>
<i>Concepto .....</i>	<i>19</i>
<i>Clasificación.....</i>	<i>20</i>
<i>Clasificación de secuencias.....</i>	<i>21</i>
<i>Uso en el algoritmo.....</i>	<i>22</i>
<i>Definición en el ambiente .....</i>	<i>22</i>
<i>Acciones en el procesamiento.....</i>	<i>22</i>
<b>Subsecuencias .....</b>	<b>23</b>
<i>Concepto .....</i>	<i>23</i>
<i>Subsecuencias enlazadas .....</i>	<i>24</i>
<i>Subsecuencias jerárquicas .....</i>	<i>25</i>
<b>Sub acciones o Sub algoritmos.....</b>	<b>26</b>
<i>Funciones .....</i>	<i>26</i>
<i>Declaración de funciones .....</i>	<i>26</i>
<i>¿Cómo funciona esto?.....</i>	<i>28</i>
<i>Procedimientos.....</i>	<i>28</i>

<i>Declaración de funciones</i> .....	29
<i>Parámetros o argumentos</i> .....	30
<i>Variables locales y globales</i> .....	30
<i>Comunicación con subprogramas: paso de parámetros</i> .....	31
<i>Paso de parámetros</i> .....	31
<i>Paso por valor</i> .....	32
<i>Paso por referencia</i> .....	32
<b>Estructuras de datos</b> .....	<b>32</b>
<i>Concepto de campo</i> .....	32
<i>Registros</i> .....	33
<i>¿Cómo se puede utilizar un registro?</i> .....	34
<i>Como se representan</i> .....	35
<i>Campo clave</i> .....	36
<i>Tipos de claves</i> .....	36
<b>Archivos</b> .....	<b>37</b>
<i>Características generales</i> .....	37
<i>Consistencia y Congruencia de archivos</i> .....	37
<i>Clasificación de archivos</i> .....	38
<b>Procesos con ficheros</b> .....	<b>41</b>
<i>Procesos Individuales</i> .....	41
<i>Genérico</i> .....	42
<i>Generación de informes (Emisores), Listados o Padrones</i> .....	44
<i>Estadísticos</i> .....	45
<i>Corte de control</i> .....	46
<i>Procesos Múltiples</i> .....	48
<i>Mezclas</i> .....	49
<i>Actualización</i> .....	51
<i>Actualización secuencial</i> .....	54
<b>Archivo Indexado</b> .....	<b>63</b>
<i>Implicancia del uso de índices</i> .....	63
<i>Como está constituido físicamente un archivo Indexado</i> .....	63
<i>Grabar versus regrabar</i> .....	64
<i>Comparación entre secuenciales e indexados</i> .....	65
<i>Actualización Indexada</i> .....	65

<i>Actualización Interactiva IN-SITU (al momento)</i> .....	70
<b>Arreglos</b> .....	<b>71</b>
<i>Como lo definimos</i> .....	72
<i>Uso del índice de un arreglo</i> .....	72
<i>Procesos</i> .....	73
<i>Carga de datos en memoria interna</i> .....	73
<i>Recorridos</i> .....	73
<i>Ordenamiento</i> .....	74
<i>Búsqueda</i> .....	78
<b>Listas lineales</b> .....	<b>82</b>
<i>Clasificación</i> .....	82
<i>Pilas</i> .....	83
<i>Colas</i> .....	84
<i>Punteros</i> .....	85
<i>Listas lineales</i> .....	86
<i>Listas lineales simplemente enlazadas</i> .....	86
<i>Operaciones con listas</i> .....	86
<i>Procesos con listas</i> .....	89
<b>Recursividad</b> .....	<b>93</b>
<i>Función recursiva</i> .....	93
<i>¿Por qué escribir programas recursivos?</i> .....	94
<i>Partes de un algoritmo recursivo</i> .....	95
<i>Tipos de recursividad</i> .....	95
<i>Recursión vs Iteración</i> .....	97
<b>Árboles</b> .....	<b>98</b>
<i>Conceptos básicos</i> .....	98
<i>Clasificación de árboles binarios</i> .....	100
<i>Arboles de Expresión</i> .....	101
<i>Recorrido de un árbol binario</i> .....	102
<i>Árboles binarios de búsqueda (ABB)</i> .....	103
<i>Propiedad de los ABB</i> .....	104
<i>Arboles AVL</i> .....	106
<i>Definición de la altura de un árbol</i> .....	107
<i>Factor de equilibrio</i> .....	108

<b>Complejidad logarítmica .....</b>	<b>108</b>
--------------------------------------	------------

## Introducción a la algoritmia

Metodología Para la Solución de Problemas Por Medio de Computadoras

El HARDWARE está formado por componentes físicos de la computadora, y el SOFTWARE está constituido por los diferentes Programas que la hacen funcionar.

**“Programar” es crear ese software (Programa) que indica a la computadora qué, cómo, dónde y cuándo tiene que hacer las cosas.**

Un algoritmo tiene muchas semejanzas con un programa de computadoras, están compuestos por un conjunto finito de acciones. La diferencia es: el lenguaje

### El lenguaje algorítmico

Es aquel por medio del cual se realiza un análisis previo del problema a resolver y se elige un método para resolverlo.

### El lenguaje informático

Es aquel por medio del cual dicho algoritmo se codifica a un sistema comprensible por el ordenador o computadora.

### Un Programa

Es un algoritmo destinado a gobernar un ordenador o computadora (maquina real).

## *¿Qué es un Algoritmo?*

Es una secuencia finita de instrucciones, reglas o pasos que describen de modo preciso las operaciones que una computadora debe realizar para ejecutar una tarea determinada en un tiempo finito.

En este sentido entonces, para crear un algoritmo es necesario:

Comenzar **identificando los resultados esperados**, porque así quedan claros los objetivos a cumplir. **Entender el problema** y la **necesidad** de solución es imprescindible para avanzar.

Luego, **individualizar los datos** con que se cuenta y determinar si con estos datos es suficiente para llegar a los resultados esperados. Es decir, **definir los datos de entrada** con los que se va a trabajar para lograr el resultado.

Finalmente, si los datos son completos y los objetivos claros se intentan **plantear los procesos** necesarios para pasar de los datos de entrada a los datos de salida

En la práctica, un algoritmo es un método para resolver problemas mediante los pasos o etapas siguientes:

**Diseño del algoritmo** que describe la secuencia ordenada de pasos – sin ambigüedades- conducentes a la solución de un problema dado (análisis del problema y desarrollo del algoritmo utilizando alguna técnica por ejemplo pseudocódigo).

**Expresar el algoritmo** como un programa en un lenguaje de programación adecuado. (fase de codificación).

**Ejecución y validación** del programa por la computadora.

Para llegar a la realización de un programa es necesario el diseño previo de un algoritmo indicando cómo hace el algoritmo la tarea solicitada, y eso se traduce en la construcción de un algoritmo. El resultado final del diseño es una solución que debe ser fácil de traducir a estructuras de datos y estructuras de control de un lenguaje de programación específico.

### Características de los algoritmos

Las características fundamentales que debe cumplir todo algoritmo son:

Un algoritmo debe ser **preciso** e indicar el orden de realización de cada paso.

Un algoritmo debe estar bien **definido**. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.

Un algoritmo debe ser **finito**. Si se sigue un algoritmo, se debe terminar en algún momento, o sea debe tener un número finito de pasos.

Un algoritmo debe ser **correcto**: el resultado del algoritmo debe ser el resultado esperado.

Un algoritmo es **independiente** tanto del lenguaje de programación en el que se expresa como de la computadora que lo ejecuta.

### Proceso, Acción y Estado

Proceso: es la unidad más pequeña de trabajo, individualmente planificable por un sistema operativo<sup>1</sup>. Formalmente un proceso es **una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos de los sistemas asociados**.

Acción: Es un acontecimiento producido por un actor que tiene un **tiempo finito** (período), produce un **resultado definido y preciso** y produce un cambio de **estado**.

---

<sup>1</sup> Un sistema operativo es el software encargado de ejercer el control de coordinar el uso del hardware entre diferentes programas de aplicación y los diferentes usuarios. Es un *administrador* de los recursos de hardware del sistema. Por ej. Windows 10, Linux, Android, etc.

Estado: El estado de un sistema está determinado por la observación de los elementos de este en un instante de tiempo dado.

Existen dos estados particulares o especiales cuando hablamos de acciones, ellos son:

- Estado Inicial: es la observación de los elementos en el instante que comienza la acción.
- Estado Final: es la observación de los elementos en el instante en el que termina la acción

A todos los otros estados que no son ni el inicial ni el final pero que están dentro de ambos los llamaremos:

- Estados Intermedios: son los estados observados en el sistema en cualquier instante entre el comienzo de la acción y su finalización. La identificación de estados intermedios permite el reconocimiento de acciones simples que componen a las acciones complejas.

## Flujograma

Es una notación gráfica para implementar algoritmos. Se basa en la utilización de unos símbolos gráficos que denominamos cajas, en las que escribimos las acciones que tiene que realizar el algoritmo.

Las cajas están conectadas entre sí por líneas y eso nos indica el orden en el que tenemos que ejecutar las acciones.

En todo algoritmo siempre habrá una caja de inicio y otra de fin, para el principio y final del algoritmo.

Los símbolos:



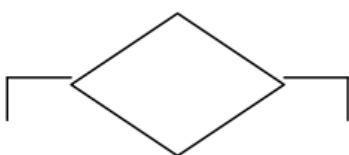
Líneas de flujo: Una línea con una flecha que sirve para conectar los símbolos del diagrama y la flecha indica la secuencia en la que se van a ejecutar las acciones.



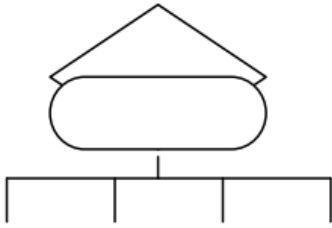
Símbolo de proceso: Indica la acción que tiene que realizar la computadora. Dentro escribimos la acción.



Representa las acciones de entrada y salida. Dentro colocaremos las acciones de lectura y escritura.



Condición: Dentro se va a colocar una condición. Sirve para representar estructuras selectivas y repetitivas y lo que se hace al encontrar ese signo es evaluar la condición que hay dentro tal que según la condición sea verdadera o falsa iremos por caminos distintos.



Principio y fin: Dentro del símbolo ira la palabra inicio o fin del algoritmo.



Subprograma: Dentro se coloca el nombre del subprograma al que se llama.

Conectores: Nos sirven cuando un flujograma no me cabe en una columna de la hoja y hay que seguir en otra columna:

- Si es en la misma hoja:



- Si es en hoja distinta



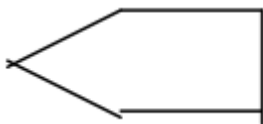
Los conectores se ponen uno donde termina la columna y otra donde empieza.



Es una aclaración para entender mejor el código, pero no es parte del código, no se ejecuta.

Otros símbolos:

- Pantalla: Cuando una salida es por pantalla.



- Teclado: Para representar una entrada por teclado.



- Impresora:





- Entrada/Salida por disco:



## ***Pseudocódigo***

El **pseudocódigo** nació como un lenguaje similar al inglés y era un medio para representar las estructuras de control de **programación estructurada**.

Se considera un primer borrador, igual que el diagrama de flujo, dado que el pseudocódigo tiene que traducirse posteriormente a un lenguaje de programación. Cabe señalar que el pseudocódigo **no puede ser ejecutado** por una computadora.

## **Programación estructurada**

La **programación estructurada** es un conjunto de técnicas y métodos para diseñar y escribir programas utilizando un método científico y no solamente el método de prueba y error. Se basa en dos cuestiones fundamentales:

### **1. Las características de un algoritmo**

- a. Poseer un solo punto de entrada y uno solo de salida
- b. Todas las acciones (sentencias o instrucciones) deben ser accesibles, o sea, que existe al menos un camino que va desde el principio al fin del algoritmo que pasa a través de dicha acción.
- c. No posee ciclos o bucles infinitos. No puede ejecutar una o varias instrucciones una cantidad infinita de veces.

### **2. Teorema de la programación estructurada, que establece que todo algoritmo estructurado puede ser escrito utilizando solamente tres tipos de estructuras de control de secuencia (Repetitivas, Alternativas y secuenciales).**

Las ventajas de utilizar un Pseudocódigo en lugar de un Diagrama de flujo son:

- Permitir **representar en forma fácil** operaciones repetitivas complejas
- Es muy fácil **transformar una solución** de Pseudocódigo a un programa en algún lenguaje de programación.

- Si se siguen las reglas se **puede observar** claramente **los niveles** que tiene cada operación.

Elementos básicos de los algoritmos

## Dato

Se define como la expresión general que describe los objetos con los cuales opera una computadora.

Los datos de entrada se transforman por el programa, después de las etapas intermedias en datos de salida.

## Tipos de datos

En Pseudocódigo vamos a utilizar los siguientes tipos de datos:

### ➤ Numéricos

Son aquellos que representan una cantidad o valor determinado

- Enteros: Es un conjunto finito de los números enteros. Los números enteros son números completos, no tienen componentes fraccionarios o decimales y pueden ser negativos o positivos. Algunos ejemplos son: 20,5,0,-3,-345. El dato que represente la edad de una persona debe ser siempre del tipo entero.
- Reales: Consiste en un subconjunto de los números reales. Estos números siempre tienen un punto decimal y pueden ser positivos o negativos. Algunos ejemplos son: 0,345,12,45,-34,34. Un dato que represente el peso de una persona debe ser del tipo real.

### ➤ Alfanuméricos

Son los datos que representan información textual (palabras, frases, símbolos, etc.)

No representan valor para efectos numéricos, o sea no podemos operar algebraicamente con ellos. Por ejemplo: el nombre de una persona, alguna frase cualquiera, etc.

### • Lógicos

También se le denomina Booleano, es aquel dato que solo puede tomar uno de dos valores: Falso o verdadero. Por ejemplo, se quiere determinar si un número es primo o no.

## Estructuras de datos simples

Las estructuras de datos simples pueden clasificarse en:

- Variables: Son elementos de almacenamiento de datos. Representan una dirección de memoria<sup>2</sup> en donde se almacena un dato, que puede **variar** en el desarrollo del algoritmo. Lo más importante de la *definición de las variables* y la *elección del tipo de datos* asociados es el **significado de la variable**, o su semántica: ya que en base al tipo de datos seleccionados serán las operaciones que podamos realizar con esa variable. Como se la define:

**Nombre\_variable: tipo\_de\_dato**

- Constantes: Una constante al igual que la variable representa una zona de memoria en la cual se almacena un dato. Sin embargo, su contenido **no puede modificarse** durante la ejecución del algoritmo. Se recomienda usar un dato constante, cuando se utiliza un valor definido y estático varias veces dentro de un programa. Recuerda que ese dato ocupa espacio en memoria, por lo que no debemos definir constantes por doquier sin analizar realmente la necesidad de hacerlo.

## Operadores y operandos

Los programas de computadoras se apoyan esencialmente en la realización de numerosas operaciones aritméticas y matemáticas de diferente complejidad.

### Operadores

Un operador es el símbolo que determina el tipo de operación o relación que habrá de establecerse entre los operandos para alcanzar un resultado.

Los operadores nos permiten manipular datos, sean variables, constantes, otras expresiones, objetos, atributos de objetos, entre otros, de manera que podamos:

- a) Transformarlos;
- b) Usarlos en decisiones para controlar el flujo de ejecución de un programa
- c) Formar valores para asignarlos a otros datos

El tipo de datos involucrado en una expresión se relaciona con los operadores utilizados. Ejemplo de una expresión es:  $3 \times 2$ . Donde en esta expresión el símbolo **x** es el **operador** de MULTIPLICACIÓN y los números 3 y 2 se llaman **operandos**.

En síntesis, una expresión es una secuencia de operaciones y operandos que especifica un cálculo. Existen varios tipos de operadores:

---

<sup>2</sup> Memoria RAM-Memoria principal de la computadora, donde residen programas y datos, sobre la que se pueden efectuar operaciones de lectura y escritura

## Operadores de asignación

Es el operador más simple que existe, se utiliza para **asignar un valor a una variable**. El signo que representa la asignación es := y este operador indica que el valor a la derecha del := será asignado a lo que está a la izquierda.

Asignación	Receptor := Emisor (Emisor puede ser constante o variable)
------------	---

La acción de asignar es **destructiva**, ya que el valor que tuviera la variable antes de la asignación se pierde y se reemplaza por el nuevo valor.

## Operadores Aritméticos

Son operadores **binarios** (requieren siempre dos operandos), que realizan las **operaciones aritméticas** habituales.

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División Real
DIV	División entera
MOD	Módulo o resto de la división
^o**	Exponenciación
ABSO	Valor absoluto
TRUNC	Truncado parte entera
REDOND	Redondeo

## Operadores relacionales

Los operadores relacionales sirven para **realizar comparaciones** de igualdad, desigualdad y relación de menor o mayor. Estos operadores sirven para expresar las condiciones en los algoritmos. Proporcionan resultados lógicos.

Operador	Significado
==	Igual a
<>	No igual a

>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que

El formato general para la comparación es:

**Expresión1 operador de relación expresión2**

El resultado de la operación será Verdadero o Falso.

Los operadores de relación se pueden aplicar a varios tipos de datos: enteros, real, alfanumérico.

Las comparaciones entre datos alfanuméricos se realizan utilizando código ASCII

## Operadores lógicos

Son aquellos que permiten la **combinación de condiciones** para formar **una sola expresión lógica**. Utilizan operadores lógicos y proporcionan resultados lógicos también.

Operador	Relación
-	Negación (No)
^	Conjunción (Y)
v	Disyunción (O)

Hay que tener en cuenta que en ciertos casos el segundo operando no se evalúe porque no es necesario (si ambos tienen que ser verdadero y el primero es falso ya se sabe que la condición de que ambos sean verdadero no se va a cumplir).

## Prioridad de los operadores

Los operadores lógicos y matemáticos tienen un **orden de prioridad o precedencia**. Este es un esquema general que indica el orden en que deben evaluarse en la mayoría de los lenguajes de programación:

Prioridad de los operadores aritméticos, relacionales, lógicos y de cadena (de mayor a menor) en pseudocódigo	
()	Paréntesis

$\wedge$	Potencia
*,/,Mod,Not	Multiplicación, División, Resto o módulo de la división, negación
>,<,>=,<=,<>=, Or	Mayor que, Menor que, Mayor o igual que, Menor o igual que, distinto, Conjunción

## *Prueba de escritorio*

Se denomina **prueba de escritorio** a la comprobación que se hace de un algoritmo para saber si está bien realizado. Esta prueba consiste en tomar datos específicos como entrada y seguir la secuencia indicada en el algoritmo hasta obtener el resultado, el análisis de estos resultados indicará si el **algoritmo está correcto** o si por el contrario **hay necesidad de corregirlo** o hacerle ajustes.

La herramienta fundamental que nos permitirá conocer cómo se comporta nuestro algoritmo es la prueba de escritorio y es una muy buena costumbre utilizarla.

### ¿Por qué debo usar la prueba de escritorio?

Las pruebas de escritorio son vitales en la formación de cualquier programador y son muchas las razones para afirmar esto, en este curso solo se mencionarán las 3 principales:

1. La primera razón es que las pruebas de escritorios son una estrategia para buscar errores en un programa.
2. La segunda razón, es que con una prueba de escritorio se puede entender cómo funciona un algoritmo.
3. Finalmente, la razón más importante las pruebas de escritorio desarrollan habilidades cognitivas que son esenciales para toda persona que programe.

### Pasos para elaborar una prueba de escritorio

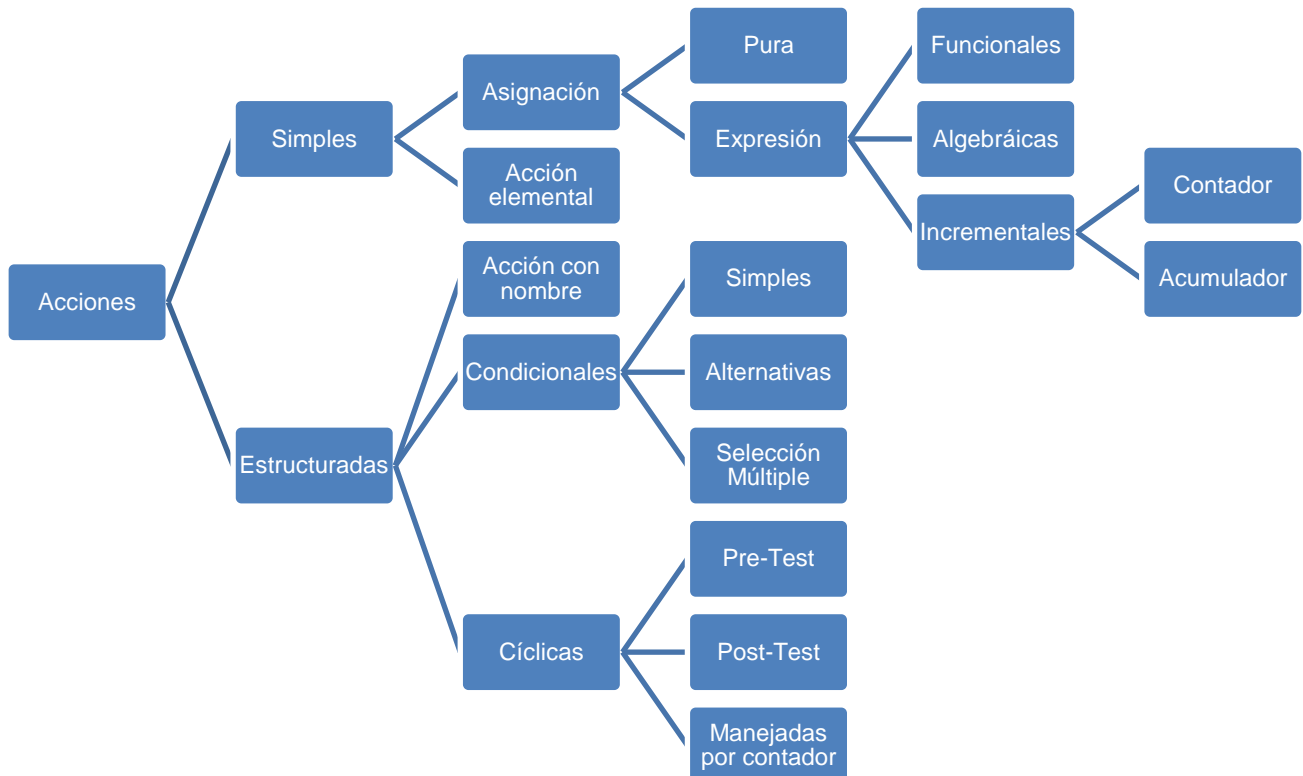
Este trabajo se realiza en base a una tabla cuyos encabezados son las **variables** que se usan en el algoritmo y debajo de cada una de ellas se van colocando los valores que van tomando, paso a paso y siguiendo el flujo indicado por el algoritmo hasta llegar al final. Cada fila de la tabla representará el estado de las variables en un instante determinado.

La prueba consistirá en 2 etapas:

- La primera, en probar inicialmente que el programa funcione correctamente para lo que se elegirá algunos datos fáciles de probar cosa que siempre es posible.

- La segunda, se prueba que ya funciona, se buscarán otros datos (si los hay) que hagan que falle el algoritmo en cuyo caso se habrán de detectar otros errores.
  - Si el algoritmo no falla, podemos concluir que el programa está terminado y revisado, por lo tanto, correcto.

### Clasificación de acciones



### Acciones Simples

Son las que pueden ser realizadas directamente y pueden ser:

- Asignación simple: es la acción que da valor a una expresión a una variable, es la acción de transferir un contenido. El operador de asignación es: “:=”. Ejemplo:

Receptor := Emisor (Emisor puede ser constante o variable). La estructura es:

Asignación pura	Receptor := Emisor (Emisor puede ser constante o variable)
-----------------	---

- Expresión: es la reunión de datos (Constantes y variables) relacionadas mediante operadores.
  - Algebraicas y funcionales: son ecuaciones comunes.

Asignación funcional	Receptor := Función_Interna (Emisor) (Funciones internas: ABSO – SQRT – LN – LOG – EXP – TRUNC – REDOND – SQR – SIN – COS – TAN)
----------------------	---

Asignación algebraica	Receptor := Emisor Operador Emisor (Operadores matemáticos: *, /, +, -, **, DIV, MOD)
-----------------------	--

- Contador: el receptor se incrementa en forma constante

Contador	Receptor := Receptor + 1
----------	--------------------------

Cada contador o acumulador tiene tres momentos:

- Puesta a cero, antes de comenzar el ciclo que lo cuenta.
- Contas/acumular dentro del ciclo que le corresponde.
- Mostrar, debajo del pie del ciclo que lo cuenta.

- Acumulador: el receptor se incrementa de forma variable

Acumulador	Acumulador := Acumulador + Variable
------------	-------------------------------------

- Acciones elementales: Es la ejecución de un acontecimiento elemental. Es una acción que aparece en el algoritmo por su nombre. Ejemplo: Leer() o Escribir()

## Acciones compuestas o estructuradas

Son las que no se pueden realizar directamente sino a través de una descomposición en acciones simples (de menor complejidad)

- Acción con nombre: presenta una estructura secuencial, ya que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es entrada de la siguiente y así sucesivamente hasta el final del proceso. Encierra un grupo de acciones entre la palabra ACCION y la palabra FIN.



Acción con nombre	Acción NOMBRE es Acción 1; ... Acción n; Fin Acción
-------------------	---

- Condicionales

- Simples: Es la ejecución condicional de una acción. La composición condicional permite expresar que no es necesario provocar un cierto acontecimiento más que bajo una determinada condición.

Condicional simple	Si CONDICIÓN entonces Acción 1; Fin Si
--------------------	--

- Alternativa: Es la ejecución alternativa de una entre dos acciones, la composición alternativa permite expresar que debe provocarse un acontecimiento bajo una determinada condición u otro bajo la condición contraria.

Condicional alternativo	Si CONDICIÓN entonces Acción 1; sino Acción 2; Fin Si
-------------------------	---

- Selección múltiple: Permite ejecutar una sentencia u otra, según el valor de una **variable**. Si el valor de la **variable** coincide con algún **valor** o el valor de alguna **expresión**, se ejecuta la sentencia o el grupo de sentencias escritas a continuación.

Condicional de selección múltiple	Según VARIABLE hacer valor 1: acción 1; ... valor n: acción n; otro: acción n+1; Fin Según
-----------------------------------	---

La instrucción **otro caso** ejecuta una o varias instrucciones cuando no se cumple ningún caso de los contemplados más arriba. **Otro caso** debe estar siempre al final (cuando sea necesario, si no se puede omitir)

**Valor 1, Valor2, ValorN:** puede ser tanto valor de cualquier tipo (numérico, alfanumérico, lógico) como así también un rango de valores. Por ejemplo: 0..100.

**Acción1, Acción 2, Acción N:** puede ser una acción simple o un conjunto de acciones.

### Cíclicas o iterativas:

Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan **bucles**. Y cada repetición del bucle se llama **iteración**.

Todo bucle tiene que llevar **asociada una condición**, que es la que va a determinar cuándo se repite el bucle y cuando deja de repetirse.

Hay que prestar atención especial a los **bucles infinitos**, hecho que ocurre cuando la condición de finalización del bucle **no se llega a cumplir nunca**.

- **Pre-Test:** Esta estructura repetitiva “Pre-Test”, es en la que el cuerpo del bucle se repite siempre que se cumpla una determinada condición.

Es una estructura repetitiva del tipo **indefinida e impura**, pues no se conoce la cantidad de veces que se debe repetir el conjunto de instrucciones del bucle y existe un elemento extraño que no debe tratarse.

El conjunto de acciones se ejecuta mientras **la evaluación de la condición** devuelva un resultado **verdadero**, el ciclo **se puede ejecutar 0 o más veces**. Esto ocurre porque si inicialmente la condición no se cumple, el bucle no se ejecuta.

Ciclo Pre-Test (Rango: 0 a n)	Mientras CONDICIÓN hacer Acción 1; ... Acción n; Fin Mientras
-------------------------------	---

- **Post-Test:** Esta estructura es muy similar a la anterior, solo que la ejecución de este texto provoca sucesivamente la ejecución de la acción y a continuación la observación de la condición “**mientras la condición sea falsa**”. Es una estructura **Indefinida y pura** puesto que todos los elementos son tratados de la misma manera. No está implementada en la mayoría de los lenguajes actuales, por lo que se reemplaza su uso con While.

Ciclo Post-Test (Rango: 1 a n)	Repetir Acción 1; ... Acción n; hasta que CONDICIÓN
--------------------------------	---

- Manejada por contador: Esta estructura es un ciclo **definido**, pues se conoce de antemano la cantidad de veces que se iterará.

El final de bucle está controlado por un contador que define la cantidad de veces a iterar, por ello también se conoce como estructura manejada por contador.

La variable “contador” se va incrementando automáticamente de acuerdo al incremento indicado. Si en lugar de incremento, debe ser decremento se indica con el signo “--”. Incremento o decremento diferente a 1 debe indicarse. Se ejecutan las acciones una cantidad preestablecida de veces. Es una estructura **Definida y Pura**.

Ciclo manejado por contador	Para VC:= VI hasta VF, I hacer Acción 1; ... Acción n; Fin Para
-----------------------------	---

La variable “VC” es de tipo numérica y se denomina “Variable de control”, “Vi” y “Vf” son variables de tipo numérica, constantes numéricas o expresiones aritméticas. “Vi” recibe el nombre de “Variable inicial”, “Vf” recibe el nombre de “Variable final”. El ciclo va incrementando la variable de control hasta llegar al final, se hace de uno en uno a menos que se especifique otro caso. “I” en el caso que se desee hacer en forma decreciente se define como “Paso -1”

	PRE - TEST	POST-TEST	MANEJADO X CONT
Se debe conocer anticipadamente el Número de iteraciones	NO	NO	SI
En que momento se verifica la condición	Antes de la ejecución del cuerpo del bucle	Después de la ejecución del cuerpo del bucle	Antes de la ejecución del cuerpo del bucle
Puede el bucle no ejecutarse nunca	SI, Si la condición es falsa la primera vez	NO, Al menos una vez se ejecuta	SI, Si $Vf < Vi$ en FOR_TO o $Vi < Vf$ en FOR_DOWNT0
Se debe modificar el valor de la condición para finalizar el bucle?	SI, haciendo que el valor de la condición sea falsa	SI, haciendo que el valor de la condición sea verdadera	NO, Es automático
Un bucle puede ser infinito	SI	SI	NO
Cuando debe utilizarse?	El Nro de iteraciones es indeterminado y no debe ejecutarse el bucle cuando la condición es falsa la primera vez	El Nro de iteraciones es indeterminado y el bucle se debe ejecutar al menos una vez	El Nro de iteraciones se conoce por adelantado

## Introducción a secuencias

### Concepto

Una secuencia, es un conjunto de datos relacionados entre sí tal como se muestra en la figura a continuación:

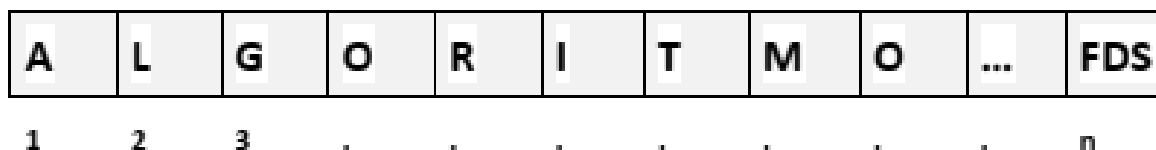
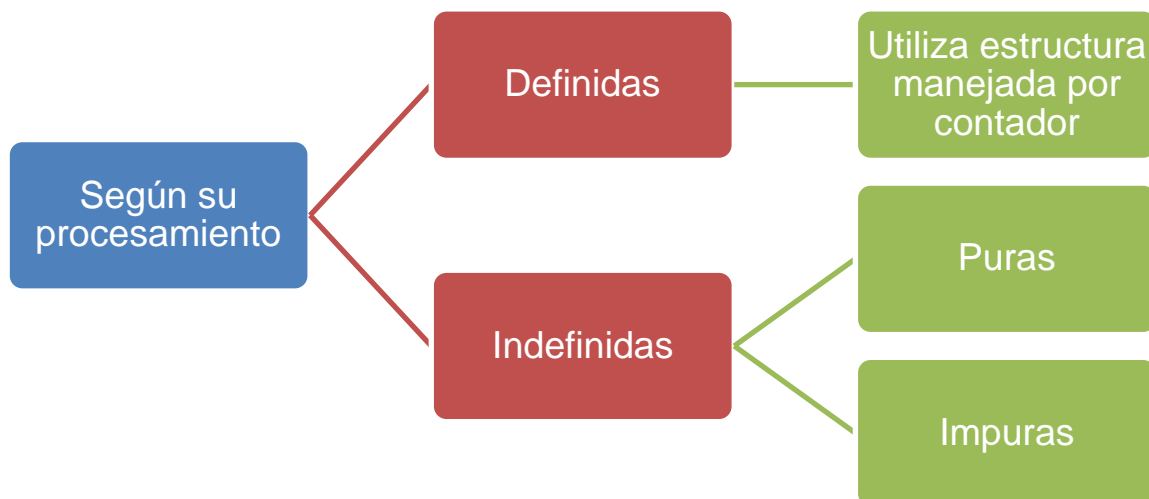


Figura 1. Ejemplo de Secuencia

Que debe cumplir ciertas características:

- Existencia del primer elemento de la secuencia: el acceso a este elemento permite el acceso posterior a los demás elementos de la secuencia.
- Relación de sucesión entre los elementos: todo objeto de la secuencia, salvo el último, precede a uno de los demás elementos, su sucesor. Esta relación entre los elementos de la secuencia permite construir el acceso a todos los elementos de la misma hasta alcanzar el final.
- Finitud: puede ser conocida o no. Puede estar dada por la cantidad de elementos o bien por una marca de fin. Todas las secuencias deben ser finitas por lo que deben estar acotadas por una condición de fin para evitar entrar en un ciclo infinito.
- Existencia del último elemento de la secuencia: debe estar definido un indicador de fin de secuencia el elemento final, que permite detener la enumeración de la secuencia por la observación de la característica de este elemento. En la práctica se utiliza la función No FDS(Sec) para determinar el final del ciclo indefinido impuro Pre-test, esta función pregunta si el elemento que está en la ventana en ese momento “no es el final de la secuencia”.

## ***Clasificación***



## ***Clasificación de secuencias***

Por su contenido:

- Puras: todos los elementos son de la misma especie o tipo. Se caracteriza porque el último elemento es tratado de igual forma que los demás. Este tipo de secuencia es manejada con estructuras puras. (Estructura Post-test).
- Impuras: son aquellas secuencias donde existe un elemento extraño que no es tratado y el cual indica el final de la secuencia. En este caso, la secuencia se maneja con una estructura del tipo Pre-test

Por la cantidad de elementos:

- Definidas: se conoce el número de elementos de la secuencia. Puede ser manejada por una estructura manejada por contador.
- Indefinidas: cuando el número de elementos de la secuencia no es conocido. Por lo tanto, la enumeración de esta secuencia, se utiliza una estructura Pre-Test o Pos-Test.

## *Uso en el algoritmo*

### Definición en el ambiente

Para utilizarla en el algoritmo es necesario definir tal como se ve en la figura siguiente, por un lado la estructura de datos, y por otro la variable que permitirá tratar cada elemento de la secuencia, pues solo se permite acceder de forma individual y secuencial.

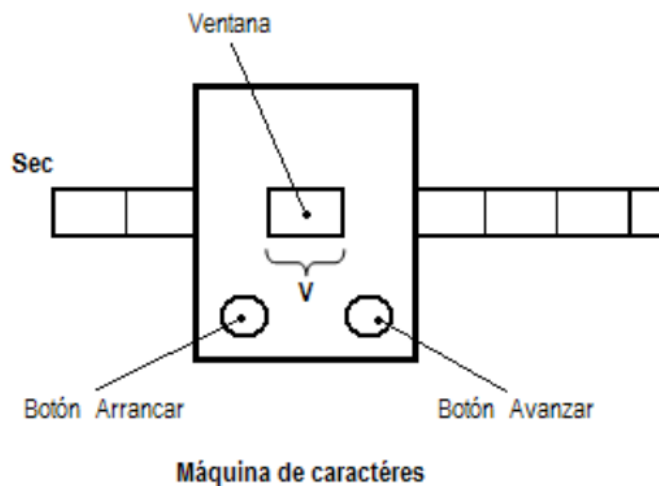
```
1  sec: Secuencia de caracter
2  v:  caracter
```

**Figura 2.** Definición en el ambiente

### Acciones en el procesamiento

La máquina de caracteres es una máquina abstracta capaz de procesar las secuencias de caracteres. Está compuesta por los siguientes elementos:

1. Una ventana: a través de la cual se podrán observar los objetos que componen la secuencia de caracteres, permitiendo el acceso de a uno por vez.
2. Uno botón “Arrancar”: la función de éste es colocar la cinta en la máquina para ser tratada.
3. Un botón “avanzar”: mediante este botón se podrá recorrer la secuencia, elemento a elemento, siempre en la misma dirección (de izquierda a derecha), accediendo a ellos de a uno por medio de la ventana. Luego de arrancar la secuencia, se la debe avanzar la primera vez para que el primer elemento de la secuencia aparezca en la ventana.



Se describe a continuación cada uno de los verbos que se utilizarán en la cátedra para trabajar con tipos de datos de secuencia<sup>3</sup>:

### **ARR(Sec)**

Es el operador que permite iniciar el tratamiento de una secuencia existente.

Solo requiere que se le indique qué secuencia se está por trabajar

### **AVZ(Sec,V)**

Es el operador que permite recuperar el contenido de cada elemento de la secuencia.

Requiere que se le indique la secuencia a la cual se accederá, y la variable que permitirá almacenar el contenido del elemento accedido.

### **Crear(SecNueva)**

Es el operador que permite inicializar una secuencia vacía.

Solo requiere que se le indique qué almacenará desde el inicio los elementos.

### **ESC(Sec,V)**

Es el operador que permite iniciar el tratamiento de una secuencia

Se lo utiliza generalmente para escribir el carácter de una secuencia existente en una secuencia nueva. Esc(SecNueva;V).

### **CERRAR(Sec)**

Es el operador que finaliza el tratamiento de una secuencia nueva o existente. Es importante cerrar siempre las secuencias utilizadas y creadas.

## **Sub-secuencias**

### **Concepto**

---

<sup>3</sup> Se basará la explicación en los elementos definidos en la figura 2

Conjunto de elementos consecutivos, que están incluidos en una secuencia, considerados como un subconjunto de acuerdo con **la definición del problema**.

#### Ejemplos:

**Palabra:** subconjunto de elementos consecutivos de una secuencia de carácter que comienza con un carácter distinto de “ ”(espacio en blanco) y finaliza con otro carácter “ ”(espacio en blanco) o alguna marca específica planteada en el problema.

**Oración:** subconjunto de elementos consecutivos de una secuencia de carácter que comienza con un carácter distinto de “ ” (espacio en blanco) y finaliza con carácter por ejemplo el punto “.” O alguna marca planteada en el problema.

H	O	L	A		B	U	E	N	O
S		D	I	A	S	.	FDS		

**DNI:** subconjunto de 8 elementos consecutivos de una secuencia de carácter. El tipo carácter es dígito numérico. “1” “2” “3” etc.

Las sub-secuencias también deben cumplir todas las características de una SECUENCIA, como ser:

- Existencia del primer elemento de la secuencia
- Relación de sucesión entre elementos
- Finitud
- Existencia del último elemento de la secuencia

Y también se las clasifica de acuerdo con su **contenido** y **procesamiento**.

### ***Relación entre Sub-secuencias***

Las sub-secuencias pueden relacionarse entre sí de acuerdo a como se presentan o definen dentro de la secuencia principal. Pueden existir relaciones de inclusión o pertenencia, estas últimas determinan una relación de jerarquía entre las sub-secuencias.

### ***Sub-secuencias enlazadas***

En este caso las sub-secuencias están identificadas una a continuación de la otra. Para esto se debe definir el principio y fin de cada una.

Por ejemplo:

N	O	M	B	R	E	-	2	5	4
5	6	7	8	9	N	O	M	B	R
E	-	3	1	4	7	5	8	6	9
									FDS



En este caso tenemos una sub-secuencia que hace referencia a un nombre (de una persona por ejemplo) que finaliza con el carácter “-“ y a continuación comienza una sub-secuencia de 8 caracteres numéricos que hacen referencia a un número de DNI. Esta última sub-secuencia es de tipo definida (se conoce la cantidad de veces que va iterar como es DNI son 8 números) por lo que no requiere una marca de fin de secuencia.

La **relación de enlace** existe porque se identifica, salvo al inicio y final de la secuencia, que el inicio de una sub-secuencia indica el fin de la anterior, por tal motivo también podemos decir que el **fin** de una sub-secuencia enlazada indica el **inicio** de la otra.

### Sub-secuencias jerárquicas

En este caso las sub-secuencias están identificadas a través de una relación de pertenencia o inclusión, pueden ser utilizadas para definir jerarquías de sub-secuencias.

Por ejemplo:

En este primer ejemplo podemos ver que identificamos palabras, de acuerdo a la definición del problema.

H	O	L	A		B	U	E	N	O
S		D	I	A	S	.	FDS		

Palabra con contenido: conjunto de caracteres distinto de blanco y de marca. Es una secuencia impura.

Palabra vacía: conjunto de caracteres blancos. Es una secuencia impura.

En este segundo ejemplo identificamos el concepto de oración, definido en el problema. Pero además podemos ver que la misma sub-secuencia está formada por palabras.

H	O	L	A		B	U	E	N	O
S		D	I	A	S	.	FDS		

Y en este caso debemos procesarlas teniendo en cuenta esta relación:



La relación de jerarquía existe porque se identifica una relación de pertenencia o inclusión entre sub-secuencias, tal cómo se mostró en el ejemplo. Podemos concluir en que el inicio y el fin de una sub-secuencia determina el inicio y el fin de otras sub-secuencias incluida en esta.

## Sub-acciones o Sub-algoritmos

La resolución de problemas complejos se facilita considerablemente si se dividen en problemas más pequeños; y la resolución de estos sub-problemas se realiza mediante sub-algoritmos.

Para escribir estos sub-algoritmos usaremos funciones y procedimientos

Los sub-algoritmos, llamados en forma vulgar sub-acciones, son unidades de algoritmo o módulos que están escritos para ejecutar alguna tarea específica.

Dicha tarea tiene la particularidad de que va a ser invocada en varias partes del algoritmo.

Estas sub-acciones se escriben solamente una vez, pero pueden ser referenciados en diferentes puntos del algoritmo, de modo que se puede evitar la duplicación innecesaria del código.

El sub-algoritmo es un algoritmo en sí mismo, ejecutado por la solicitud del algoritmo principal o de otro sub-algoritmo, una n cantidad de veces. Cuando realiza la solicitud, el algoritmo se detiene hasta que el sub-algoritmo deja de realizar su tarea, luego continúa; esto se conoce como control de ejecución.

## Funciones

Una función es un sub-algoritmo que recibe, como argumentos o parámetros, datos de tipo numérico o no numérico, y devuelve un único resultado.

Las funciones incorporadas al sistema se denominan funciones internas, o intrínsecas o predefinidas; las funciones definidas por el usuario se llaman funciones externas.

El algoritmo o programa invoca la función con el nombre de esta última en una expresión seguida de una lista de argumentos que deben coincidir en cantidad, tipo y orden con los de la función que fue definida.

## Declaración de funciones

En pseudocódigo usaremos la siguiente sintaxis:

**Función** nombrefun (lista de parámetros): tipo

...

(Declaraciones locales si fuera necesario > Ambiente)

...

(Acciones de la función)

...

**Nombrefun:=<valor de la función>**

**Fin\_Función**

¡No te olvides!

Debe estar para que la función  
devuelva un valor

**Nombrefun:** es el nombre de la función.

**Lista de parámetros:** es la lista de parámetros formales. **Esta lista NO puede estar vacía.**

**Tipo:** es el resultado que devuelve la función.

Ejemplo:

**ACCION** Despertador es

**Ambiente**

dia : N(1)                    {MEJOR DEFINIRLO COMO RANGO 1..7}  
llueve, hace\_viento : CHAR ;

**FUNCION** **Hace\_buen\_dia** (L: Carácter, H: Carácter): Logico

**SI** (L = 'N') Y (H = 'N') **ENTONCES**

**hace\_buen\_dia**:= VERDAD

**CONTRARIO**

**hace\_buen\_dia**:= FALSO;

**Fin\_Funcion**

**Proceso**

ESCRIBIR (“RIIING !!”)

ESCRIBIR (“¿Qué día es hoy?”)

LEER (dia)

**SI** dia <7 **ENTONCES**

ESCRIBIR (“LEVANTATE, HAY QUE IR A TRABAJAR”)

**CONTRARIO**

ESCRIBIR (“Llueve (S/N)?”)

LEER (llueve)

ESCRIBIR (“Hace viento (S/N)?”)

LEER (hace\_viento)

```

SI hace_buen_dia (llueve, hace_viento) ENTONCES
    ESCRIBIR (“LEVANTATE Y A DISFRUTAR DEL DIA”)
CONTRARIO
    ESCRIBIR (“TRANQUILO, PUEDES SEGUIR TUMBADO”)
Fin_Si
Fin_Si
Fin_Accion

```

### ¿Cómo funciona esto?

El control de ejecución lo toma la función, ejecuta secuencialmente cada una de sus sentencias, y cuando termina de ejecutarse, le devuelve el control al algoritmo llamador, ejecutándose la secuencia inmediatamente siguiente a la de la llamada.

El resultado lo devuelve en el nombre de la función.

Cada vez que se llama a una función desde el algoritmo principal, se establece automáticamente una correspondencia entre los parámetros formales y los reales. Debe haber exactamente el mismo número de parámetros reales que de formales en la declaración de la función, y se presupone una correspondencia uno a uno de izquierda a derecha entre los parámetros formales y reales.

Una llamada a una función implica los siguientes pasos:

- A cada parámetro formal se le asigna el valor real de su correspondiente parámetro actual (REAL se refiere al valor verdadero con el cual va a trabajar el sub-algoritmo, y no al tipo de dato).
- Se ejecuta el cuerpo de acciones de la función y se devuelve el valor de la función al nombre de la función y se retorna al punto de llamada.

### ***Procedimientos***

Un procedimiento es una sub-acción que ejecuta una tarea determinada.

Está compuesto por un conjunto de sentencias, a las que se le asigna un nombre, o identificador.

Los procedimientos deben ser declarados antes de su invocación desde el algoritmo principal y solo se activan o ejecutan desde el algoritmo en que fueron declarados.

Pueden recibir cero o más valores del algoritmo principal que lo llama, y devolver cero o más valores a dicho algoritmo.

Todo procedimiento, al igual que un algoritmo principal, consta de una cabecera, que proporciona su nombre y sus parámetros de comunicación; de declaraciones locales si son necesarias y el cuerpo de sentencias ejecutables.

Las ventajas más destacables de usar procedimientos son:

Facilitan el diseño top-down.<sup>4</sup>

Se pueden ejecutar más de una vez en un algoritmo, con solo llamarlos las veces que así desee.

El mismo procedimiento se puede usar en distintos algoritmos.

Su uso facilita la división de tareas entre los programadores de un equipo.

Se pueden probar individualmente e incorporarlos en librerías o bibliotecas

## Declaración de procedimientos

En pseudocódigo usaremos la siguiente sintaxis:

**Procedimiento** *nombrepro* (*lista de parámetros*) **es**

...

(*Declaraciones locales si fuera necesario > Ambiente*)

...

(*Acciones del procedimiento*)

*Fin\_Procedimiento*

**Nombrepro:** es el nombre del procedimiento.

**Lista de parámetros:** es la lista de parámetros formales. **Esta lista puede estar vacía.**

Ejemplo:

**ACCION** Uno **ES**

**AMBIENTE**

Entero 1 ,entero2 : N(2);

**Procedimiento** **intercambio** (*var* pf1,pf2:N(2))

Aux:N(2); (*variable local uso exclusivo en procedimiento*)

Aux:=pf1;

---

<sup>4</sup> Se trata de ir descomponiendo el problema en niveles o pasos cada vez más sencillos, tal que la salida de una etapa va a servir como entrada de la siguiente.

```
Pf1:=pf2;
```

```
Pf2:= aux
```

**FIN\_PROCEDIMIENTO**

**PROCESO**

```
ESCRIBIR (“introduzca 2 variables enteras:”)
```

```
LEER(entero1, entero2)
```

```
ESCRIBIR (“valores de las variables antes de la llamada”)
```

```
ESCRIBIR (“Entero 1 =”, entero1, ”entero 2 =”,entero2)
```

```
intercambio (entero1,entero2); {llamada al procedimiento} ESCRIBIR  
    (“Valor de las variables después de la llamada”    );
```

```
ESCRIBIR (“entero 1 =”, entero1, ”entero 2 =”, entero2);
```

**FIN\_ACCION**

En el caso del procedimiento, tanto los datos de entrada como de salida van definidos dentro de los argumentos, y en particular para el PASCAL deberá tener la palabra VAR para que se produzca el cambio de dichos valores.

### ***Parámetros o argumentos***

Los parámetros son variables y/o constantes para pasar datos entre algoritmos y sub-algoritmos en ambos sentidos.

Los parámetros que se utilizan en la llamada o invocación la sub-acción se denominan parámetros **actuales, reales o argumentos**, y son los que entregan la información a la función o procedimiento.

Los parámetros que figuran en la definición de la sub-acción se denominan **parámetros formales o ficticios** y se declaran en la *cabecera* de dicha sub-acción.

En una llamada a una sub-acción tiene que verificarse que:

1. El número de parámetros formales debe ser igual al de los actuales.
2. Los parámetros que ocupen el mismo orden en cada una de las listas deben ser compatibles en tipo.

### ***Variables locales y globales***

Las variables utilizadas en los programas principales y subprogramas se clasifican en dos tipos: **locales y globales**.

Una variable local es una variable que está declarada dentro de un subprograma, y se dice que es local al subprograma. Una variable local sólo está disponible durante el funcionamiento del subprograma que la declara, y su valor se pierde una vez que finaliza la ejecución del subprograma.

Las variables declaradas en el programa principal se llaman globales, pueden ser utilizadas en el programa principal y en todos los subprogramas en él declarados.

Si existen dos variables con el mismo nombre, pero una es global y la otra es local, dentro del subprograma tiene prioridad la variable local de igual nombre.

### ***Comunicación con subprogramas: paso de parámetros***

Cuando un programa llama a un subprograma, la información entre ellos se comunica a través de la lista de parámetros, y se establece una correspondencia automática entre los parámetros formales y los reales.

Cada subprograma tiene un encabezamiento, en el cual se indican los parámetros formales, en el momento en que un subprograma realiza la llamada a otro módulo, se indican los parámetros reales.

Los parámetros reales son utilizados por el subprograma en lugar de los parámetros formales.

### ***Paso de parámetros***

Existen diferentes métodos para el paso de parámetros a subprogramas. Es preciso conocer el método adoptado por cada lenguaje, ya que la elección puede afectar a la semántica del código. Los parámetros pueden ser clasificados como:

- Entradas (E): las entradas proporcionan valores desde el programa que llama, y se utilizan dentro del procedimiento.
- Salidas (S): las salidas proporcionan los resultados del subprograma.
- Entradas/Salidas (E/S): un solo parámetro se utiliza para mandar argumentos a un programa y para devolver resultados.

Los métodos más empleados para realizar el paso de parámetros son:

- Paso por valor (parámetro valor)
- Paso por referencia o dirección (parámetro variable)
- Paso por nombre
- Paso por resultado

En la cátedra vamos a utilizar, en mayor parte, el paso de parámetro por valor.

## Paso por valor

Son los parámetros unidireccionales, entregan información a la sub-acción pero no devuelven a la acción principal las modificaciones que estos parámetros hayan tenido en la sub-acción.

Esto se debe a que, en la llamada al subprograma, se le asigna el valor del parámetro real a la variable que representa al parámetro formal correspondiente, dicho de otra forma, se crea una copia del parámetro real.

Los parámetros formales (locales al subprograma) reciben como valores iniciales los valores de los parámetros reales, y con ellos se ejecutan las acciones descritas en el subprograma.

## Paso por referencia

Son los parámetros que están precedidos por una palabra reservada (por ejemplo, en PASCAL es VAR), que indica que el subprograma recibe su valor y si estos parámetros sufren cambios en la sub-acción se los devuelve a la acción principal con dicha modificación.

Se los considera como parámetros bidireccionales o variables, ya que son de Entrada y/o Salida.

### **¿Funciones o procedimientos?**

**Deben utilizarse funciones cuando solo tenga que devolverse un solo valor simple al algoritmo llamador.**

**En todos los demás casos utilizaremos procedimientos.**

## Estructuras de datos

### *Concepto de campo*

Un campo es un conjunto de caracteres capaz de suministrar una determinada información referida a un concepto. Un campo es la **entidad lógica más pequeña**, consiste en un conjunto de byte que conforman un dato.

Al igual que las variables, al definir un campo hay que indicar claramente sus características:

- Nombre: nombre que identifica a ese conjunto de caracteres.
- Tipo: Tipo de caracteres que puede contener (alfanumérico, entero, etc.).
- Tamaño: Cantidad de caracteres que puede contener.



NOMBRE	TIPO	TAMAÑO
dni	:	N (8)

.....Ahora bien, distinguiremos dos tipos de campos: contenido y continente.

Un campo contenido es la unidad lógica más pequeña, consiste en un conjunto de bytes que conforman un dato que no necesita o no puede ser descompuesto.

Un campo continente, en cambio está formado por varios campos contenidos y es la unidad de información que referencia a una entidad en un registro.

Ejemplos:

El campo documento que arriba definimos, es un campo contenido, (no se justica dividirlo) pero si quisiéramos definir fecha de nacimiento

En este caso, tendríamos un campo continente, formado por los campos contenidos año, mes y día.

Fecha-nac = registro

año: N(4)

mes: 1..12

día: 1..31

Fin\_Registro

Como no existe una palabra para definir agrupamiento de campos contenidos, abusaremos de la palabra "registro" para definir campos continentes.

debemos forzar a los campos a tener una longitud fija

Los datos (nombre, dirección, estado) en su gran mayoría, los datos alfanuméricos, tienen longitudes variables.

Pero nosotros podemos pensar en establecer una medida fija para cada uno de los campos, por ejemplo, en el registro "PERSONA", definido abajo, cada campo tiene una longitud y el tamaño total del registro para cada entidad siempre será de 65 bytes (30+20+15)

PERSONA = registro

NOMBRE-APELLIDO: AN(30)

DOMICILIO: AN(20)

LOCALIDAD: AN(15)

Fin registro

## *Registros*

Un registro es una estructura de datos que contiene información referida a una entidad y está compuesta de un número fijo de componentes llamados “campos”, donde cada uno de ellos viene definido con un nombre y un tipo. Es una estructura estática, compleja y se almacena en memoria interna. Los campos que contiene pueden ser continentes y contenidos.

Ejemplo:

Fecha = registro

año: N(4)

mes: 1..12

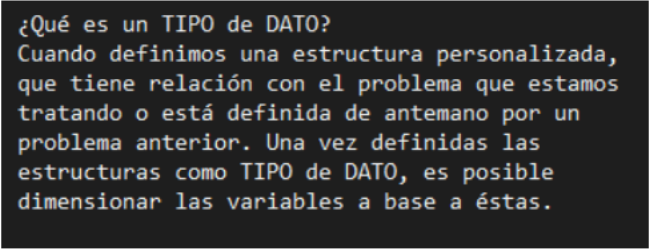
día: 1..31

Fin\_Registro

El registro debe definirse en el ambiente para ser usado en el algoritmo

**¿Porque usamos un símbolo de = en la definición?**

**Para definir un TIPO de DATO**



¿Qué es un TIPO de DATO?  
Cuando definimos una estructura personalizada, que tiene relación con el problema que estamos tratando o está definida de antemano por un problema anterior. Una vez definidas las estructuras como TIPO de DATO, es posible dimensionar las variables a base a éstas.

*¿Cómo se puede utilizar un registro?*

Cada uno de los campos de un registro puede ser usado usando la sintaxis {registro}.{campo} - esto se denomina SELECTOR :

Una vez creado el tipo, es posible crear variables de tipo Fecha:

var nacimiento: Fecha;

hoy: Fecha;

Cada uno de los campos de un registro puede ser usado utilizando la sintaxis {registro}.{campo} - esto se denomina SELECTOR :

nacimiento.día := 24;

nacimiento.mes := 11;

nacimiento.ano := 1986;

Cuando es obligatorio usar el selector. Cuando existe ambigüedad en los nombres de las variables.

¿Cómo es esto?

Si en el registro tuviéramos la necesidad de trabajar con dos campos fechas, fecha de ingreso y fecha de nacimiento, por ejemplo. las definimos:

```
fecha-ing, fecha-nac: registro
```

```
anio: N(4)
```

```
mes: 1..12
```

```
dia: 1..31
```

```
Fin_Registro
```

Y cuando nos referimos al campo año deberemos aclarar a que registro corresponde, es decir:

```
fecha-nac.anio <> fecha-ing.anio
```

Al asignar un registro entero a otro del mismo tipo, todos los valores son copiados automáticamente, uno por uno:

```
hoy:= nacimiento; {dia, mes y anio son copiados}
```

El registro es una unidad. Al ser una estructura está compuesta por elementos, pero debe ser tratada como una sola entidad.

## Como se representan

Existen varias formas de representar los registros, las cuales se utilizan según en qué paso del diseño de nuestro sistema/programa nos encontremos.

- Definición arbórea o jerárquica: esta es la forma inicial de representarlos y suele utilizarse como forma borrador, pues es la más fácil de modificar a mano alzada. Consiste en una serie de nodos que se vinculan mediante flechas, y donde la lectura se realiza siguiendo las reglas jerárquicas de arriba-abajo y de izquierda derecha.
- Definición gráfica: esta es una forma lineal, que usualmente la utiliza el analista/ingeniero para definir la estructura cuando le suministra el planteo del problema al programador.
- Definición por nivel o literaria: esta es la forma de representar los datos en el ambiente del algoritmo. Obviamente es la que utiliza el programador.

## REPRESENTACIÓN DE REGISTROS

Lineal, gráfica o esquemática	Jerárquica o arbórea	Literaria						
<p>Celdas continuas, donde cada dato tiene su cuadro.</p> <p>Fecha</p> <table border="1"> <tr> <td>Día</td><td>Mes</td><td>Año</td></tr> <tr> <td></td><td></td><td></td></tr> </table>	Día	Mes	Año				<p>Cada nodo representa una entidad y se relacionan mediante líneas de flujos.</p> <pre> graph TD     Fecha[Fecha] --&gt; Día[Día]     Fecha --&gt; Mes[Mes]     Fecha --&gt; Año[Año]         </pre>	<p>Cada elemento ocupa un renglón de código. La subordinación está dada por el uso de sangrías.</p> <p>REG = registro</p> <p>Campo 1: tipo 1</p> <p>...</p> <p>Campo n: tipo n</p> <p>Fin registro</p>
Día	Mes	Año						

### *Campo clave*

Un campo clave (key) es aquel que identifica al registro y lo diferencia de los otros registros.

Debe ser **UNICO**, es decir debe ser diferente para cada registro. De todos los campos o datos siempre se elige a uno como campo clave.

Un campo **clave simple** es aquel que está formado por un campo contenido, es decir, no se encuentra subdividido.

Un campo **clave compleja** es aquel que está formado por un campo continente, es decir, se encuentra subdividido en campos contenidos.

### Tipos de claves

Los campos claves son importantes, pues serán los campos por los que normalmente se ordenen los registros dentro de los archivos.

Debido a ello, podemos distinguir dos tipos:

- Clave principal o primaria: es el dato que por su importancia jerárquica se encuentra al inicio del registro.
- Clave secundaria: es el dato que si bien tiene importancia no identifica fehacientemente al registro dentro del archivo y por lo tanto no es el campo inicial de la definición. Puede darse que hasta aparezca repetido.

#### Ejemplos:

En el sistema de Alumnado, el legajo del alumno es su campo clave principal en la definición del registro, y el número del documento es un campo clave secundaria. ¿Porqué?

Si un alumno está cursando dos carreras a la vez, en el archivo existirán dos registros, uno por cada carrera, en los cuales los números de legajo serán distintos, pero como pertenecen a la misma persona, el número de documento se repetirá en cada definición.

## Archivos

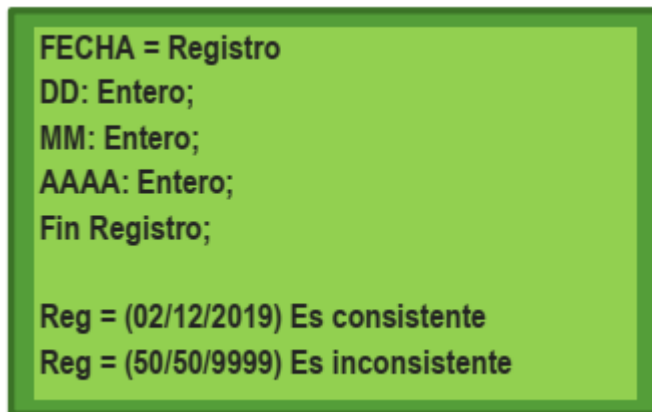
Un archivo o fichero es un conjunto de datos estructurados en una colección de entidades elementales o básicas denominadas registros, que son de igual tipo y constan a su vez de diferentes entidades de nivel más bajo denominadas campos.

### *Características generales*

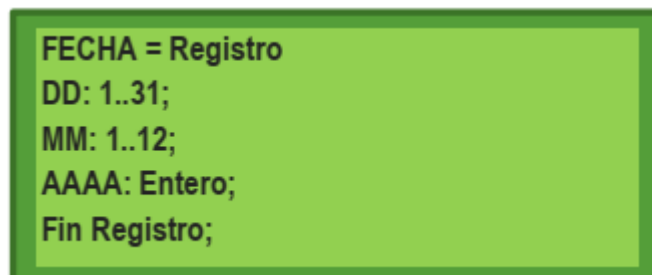
1. Un archivo está siempre almacenado en **memoria externa**, su proceso se realiza en la memoria interna. La información almacenada es permanente.
2. Existe **independencia** de los datos respecto de los algoritmos que los utilicen.
3. Todo algoritmo intercambia datos con el archivo y la unidad básica de entrada/salida es el registro. Los datos extraídos o almacenados en el archivo son los de un registro completo.
4. Los archivos en memoria externa permiten una gran capacidad de almacenamiento.

### *Consistencia y Congruencia de archivos*

La Consistencia de un archivo es la propiedad que verifica la validez del dato almacenado con su definición en el ambiente.



Cuando hablamos de Consistencia Automática hacemos referencia a la definición de límite para los datos por ejemplo Rango, Conjunto.



La Congruencia de un archivo es la propiedad que verifica la validez de los datos entre sí.

Puede ser:

- **Fina:** Validación entre datos en archivos distintos (Datos de un campo, con datos de otro archivo, por ejemplo: un DNI en un archivo de ALUMNOS, validado con un archivo de PADRON)
- **Gruesa:** Validación entre datos en un mismo registro (Por ejemplo: mes/día/año -> 31/02/2019 ... sería consistente pero no congruente)

```
FECHA = Registro
DD: 1..31;
MM: 1..12;
AAAA: Entero;
Fin Registro;

Reg = (31/02/2019) Es consistente, pero no congruente.
```

NOTA1: si es consistente, no indica que sea congruente.

NOTA2: si no es consistente, tampoco es congruente.

Organización: Es la manera en cómo van a ser almacenados los datos dentro de un fichero. La organización es permanente, ya que una vez definida no puede cambiarse. La forma de almacenamiento nace y muere con el archivo.

Acceso: Es la manera en la que se van a leer (recuperar) los registros de un fichero.

Que verbos usar

ABRIR E/ (ARCHIVO)

LEER(ARCHIVO, REGISTRO)

ABRIR /S (ARCHIVO2)

GRABAR(ARCHIVO2, REGISTRO2) o ESCRIBIR(ARCHIVO2, REGISTRO2)

REGISTRO2:= REGISTRO

FDA(ARCHIVO) \\ Función que verifica si es el fin del archivo

## ***Clasificación de archivos***

1. De acuerdo con sus elementos se encuentran:

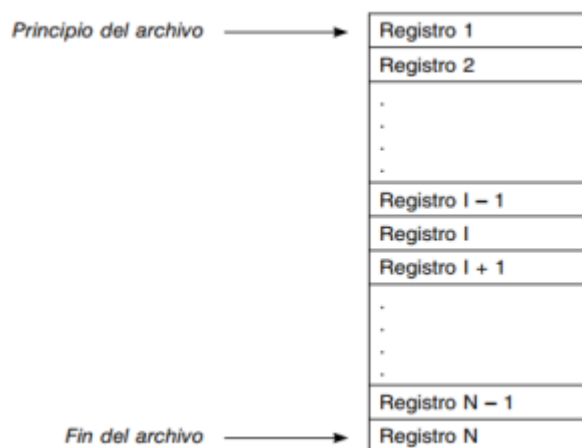
- Archivos de entrada: compuestos por registros almacenados en un dispositivo magnético y sobre los cuales se producirá la operación de lectura. Cuando se da la orden de apertura de estos archivos, si el sistema operativo no los encuentra se aborta el programa.
- Archivos de salida: estos archivos contienen información que se la proporciona desde la computadora. Sobre ellos se producirá la operación de grabar. Cuando se da la orden de apertura de estos archivos, si el sistema operativo no los encuentra los crea, y si existen se coloca al inicio, sobre escribiendo en el mismo.
- Archivos históricos: está compuesto por registros que se fueron almacenando desde el inicio de la vida del sistema y con información actualizada.
- Archivos de movimiento: esta clase de archivos se utilizan para suministrar información que permitirá actualizar a los archivos históricos u otros del sistema.
- Archivos TEMPORALES: estos se crean en el momento en que se ejecuta algún programa y se borran una vez que finaliza la ejecución, son auxiliares.

## 2. De acuerdo a los datos que almacenan se encuentran:

- ASCII: en este tipo de archivo los datos son almacenados a través de un simple texto. Esto permite intercambiar a los datos que contienen, así como también para crear archivos que el propio usuario pueda modificar.
- Binario: esta clase de archivos, en cambio, almacena información en un lenguaje al que sólo la propia computadora comprende, por ejemplo: colores, sonidos, imágenes u órdenes. Estos archivos son de menor peso que los anteriores.

## 3. De acuerdo a como se crearon: (ORGANIZACION)

- Archivos secuenciales: Un archivo con organización secuencial es una sucesión de registros almacenados consecutivamente (continuidad física). Por tal motivo, para acceder a un registro  $n$  dado es obligatorio pasar por todos los  $n - 1$  registros que le preceden. En este formato, el tamaño de memoria del archivo es el mínimo necesario.



- Archivos directos: en estos, cada registro se almacena en forma ordenada según uno de sus campos, (CLAVE o índice), se denominarán Relativos o Indexados. La continuidad en los mismos es lógica, no necesariamente tiene relación con el orden de carga.

En el caso de los Relativos, estos mantienen un orden posicional, y ya nacen con el pronóstico del tamaño que van a necesitar en todo su período de vida.

En el caso de los Indexados, están formados por un área de índices (automáticamente ordenada por el valor de la clave) y un área de datos, a la cual solo se accede desde el área de índices. Este tipo de organización ocupa más espacio de memoria que los formatos anteriores.

	CLAVE	DIRECCIÓN
Área de índices	15	010
	24	020
	36	030
	54	040
	.	.
	.	.
	240	090

	CLAVE	DATOS
Área principal	010	15
	011	
	012	
	.	
	.	
	019	
	020	24
	021	
	.	
	.	
	029	
	030	36
	031	
	.	

- De acuerdo con el mecanismo que se utiliza para acceder a los archivos se encuentran los siguientes:
  - Acceso secuencial: los registros se organizan de manera secuencial de manera tal que para leer/ grabar uno, se deben haber leído/grabado los anteriores según sea el



orden (físico o lógico). Un acceso secuencial sobre un archivo de organización secuencial implica la lectura/grabación manteniendo la continuidad física.

Este tipo de acceso es aplicable a todo tipo de archivos, lo que varía es a qué tipo de continuidad nos referimos.

- Acceso Directo: que puede ser
  - Acceso Puntual o al Azar; se accede indirectamente a los registros por su clave, mediante consulta secuencial al área de índices (tabla que contiene la clave y la dirección relativa de cada registro), y posterior acceso directo al registro.
  - Acceso Dinámico o mixto; es cuando se accede a los archivos mediante un acceso puntual y luego se continúa en forma secuencial.

## Procesos con ficheros

### 1 - FICHEROS SECUENCIALES

TIPO	PROCESO	CARACTERÍSTICAS
<b>INDIVIDUALES</b>  Un proceso es individual cuando existe un único Fichero de Entrada y 1 o ningún Fichero de Salida.	Genérico	Es el proceso de carga o generación, tiene como objetivo crear un archivo consistente y de ser posible, congruente grueso.
	Emisión	Tiene como objetivo la salida impresa de datos. Son listadores cuando emiten listados como se ingresan sin más que títulos. Se considera padrón cuando los datos ingresados están ordenados y se emiten además totales finales.
	Estadísticos	Recorrido del archivo para contabilizar elementos, utilizando una tabla (memoria interna) y al finalizar emitir un cuadro de resumen.
	Corte De Control	Son padrones, pero poseen totales parciales. Es requisito obligatorio que el archivo de entrada esté ordenado por clave compleja.
<b>MÚLTIPLES</b>  Un proceso es múltiple cuando existen 2 o más Ficheros de Entrada y 1 o más Ficheros de Salida.	Mezcla	<b>Ficheros de entrada:</b> por lo menos dos. <b>Ficheros de salida:</b> uno (resultado de la combinación de los dos de entrada).
	Actualización Secuencial Por Lotes	Cero o más registros del fichero movimiento por cada registro del maestro
	Actualización Secuencial Unitaria	Cero o un (como máximo) registro del fichero movimiento por cada registro del maestro.

### Procesos Individuales

El proceso simple o unitario es aquel en el que interviene un solo fichero de entrada, pudiendo tener o no uno de salida. La característica fundamental es que el control de finalización del proceso va a estar dado por el fin de fichero de entrada. "Hay un fichero que controla el fin del proceso".

Puede tratarse de cuatro diferentes procesos:

1. Genérico
  - a. Creación
  - b. Duplicación.

2. Generación de informes (Emisores), Listados o Padrones
3. Estadísticos.
4. Corte de Control.

## Genérico

Es el proceso de carga o generación, tiene como objetivo crear un archivo consistente y de ser posible, congruente grueso.

La carga es un proceso que consiste en la escritura o grabación de todos los registros que van a formar parte del archivo, en un soporte determinado. Los datos son ingresados a través de periféricos de entrada.

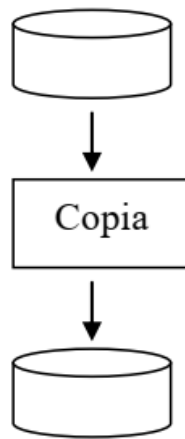
Esquema de Carga
<p>Algoritmo</p> <p>    Abrir_S (Arch)</p> <p>    Tratar (cond)</p> <p>    Mientras Cond hacer</p> <p>        Tratar (Reg)</p> <p>        Escribir(Arch, Reg)</p> <p>        Tratar (cond)</p> <p>    Fin Mientras</p> <p>    Cerrar(Arch)</p> <p>Fin Acción</p>

La generación consiste en crear un archivo a partir de otro, por medio de un tratamiento del archivo de entrada.

Esquema de Generación
<p>Algoritmo</p> <p>    Abrir_/E(Arch1)</p> <p>    Obtener datos(Reg1)</p> <p>    Abrir_/S(Arch_X)</p> <p>    Mientras no ultimo dato(Arch1) hacer</p> <p>        Tratar (Arch1)</p> <p>        Escribir(Arch_X, Reg_X)</p> <p>        Obtener datos(Reg1)</p> <p>    Fin Mientras</p> <p>    Cerrar(Arch1)</p> <p>    Cerrar(Arch_X)</p> <p>Fin Acción</p>

Copia consiste en crear un nuevo archivo como duplicación de otro existente. La copia puede realizarse en el mismo o en diferente soporte de información.

Un caso particular de esta es la impresión (copia en impresora) de un archivo.



Características de Carga o Generación: Los datos de salida deben tener consistencia y congruencia gruesa.

La **consistencia** es verificar que los datos cumplan con las definiciones y si es posible con la mayor acotación. Con “verificar los datos” nos referimos a que si por ejemplo definimos un fichero de alumnos:

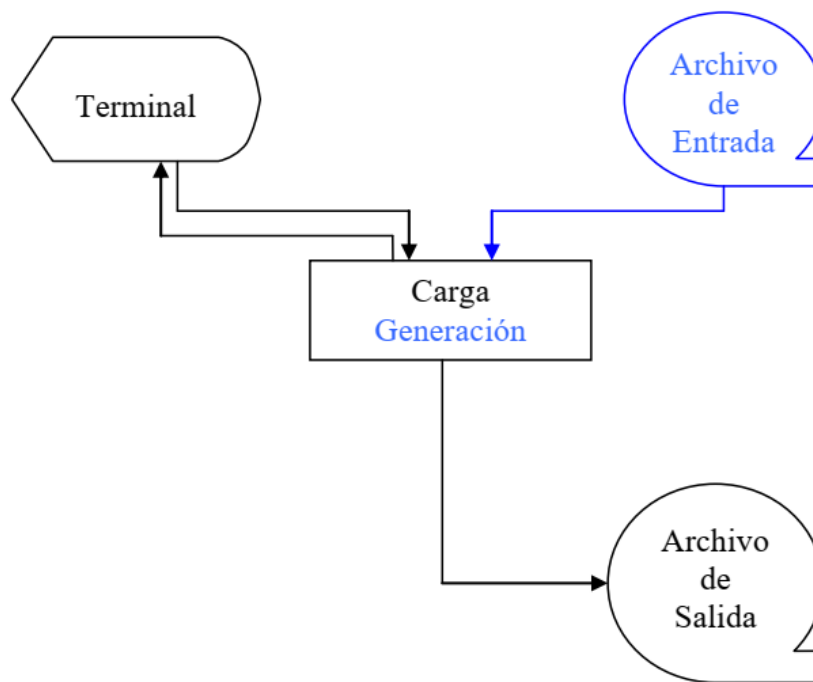
Alumnos (Archivos secuencial)

Nº Legajo: N(5)	Apellido y Nombre: AN(20)	DNI: N(8)
-----------------	---------------------------	-----------

No podemos encontrarnos con un DNI de tipo alfanumérico, y con la mayor acotación se refiere a que si decimos que DNI es N(8) no podemos encontrar un DNI con más de 8 dígitos.

En cambio, la **congruencia** es la relación correcta entre datos (se necesitan dos datos para comparar).

Por ej.: No debería existir un Alumno sin Nº de legajo, o un legajo sin el nombre ni el DNI. No siempre se puede verificar la congruencia.



Siempre que un problema nos pide crear un archivo es un proceso genérico. Puede tener ningún archivo de entrada y muchas terminales. Lo que se busca de este proceso es justamente crear un archivo consistente y en lo posible congruente grueso.

### Generación de informes (Emisores), Listados o Padrones

En este proceso siempre hay un archivo de entrada el cual es mostrado a través de **pantalla o por impresora**. Se busca una salida voluminosa.

La generación de informes: es una de las aplicaciones que más se utilizan en la programación de ordenadores. Es de general conocimiento el gran número de informes que debe manejar el ejecutivo empresarial si quiere disponer en todo momento de una información completa y, además, actualizada.

Listados: es el proceso por el cual un archivo o parte de él es presentado, por lo general, en papel por medio de una **impresora**. También los listados suelen presentarse por pantalla antes de ser impresos para corroborar la inexistencia de discrepancias en los datos. Es un proceso más simple que el informe ya que, sólo se limita a mostrar información.

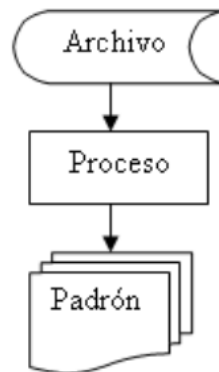
Padrones: es el proceso por el cual un archivo o parte de él es presentado, en papel por medio de una impresora.

El padrón es igual que el informe, pero la **diferencia** fundamental es que en este listado los datos previamente están ordenados, y la generalidad es que al final del proceso se emitan totales generales.

A la hora de confeccionar los programas generadores de informes hay que tener en cuenta dos aspectos importantes.

Características generales de este proceso:

- Se debe asegurar que el archivo sea consistente, y en lo posible congruente.
- El ordenamiento depende del objetivo del padrón.



## Estadísticos

Un informe estadístico es un conjunto de resultados cuantitativos que se obtienen de un proceso sistemático de **recopilación, tratamiento y divulgación de datos primarios** obtenidos sobre hechos que son relevantes para el estudio de fenómenos demográficos, sociales y económicos

### Cuadro estadístico

Es un **arreglo de filas y columnas** dispuestas de modo tal que se puedan presentar y organizar adecuadamente para comparar e interpretar las características de dos o más variables.

La presentación de un cuadro tiene mucha importancia para el usuario por lo tanto un buen diseño transmite la calidad de los procedimientos previos realizados en la recopilación, elaboración y análisis de datos.

1. Poner a 0 la estadística
2. Iniciar el procesamiento del archivo
3. Leer el registro
4. De acuerdo al valor actualizar la matriz. Por ejemplo:  $A[\text{Reg.BD}, \text{Reg.Prob}] := A[\text{Reg.BD}, \text{Reg.Prob}] + 1$
5. Verificar si se requiere actualizar filas/columnas totalizadoras
6. Continuar el procesamiento del archivo hasta el final
7. Mostrar la matriz y los datos específicos que se soliciten

### Características:

- Es el único proceso que no pide ningún requisito para el archivo de entrada, el cual se recorre para contabilizar elementos (utilizando un arreglo o vector);
- Busca emitir tabla de valores (se asemeja a una matriz);

- Muestra los resultados al final del proceso;
- Debe mostrar una alerta;
- Existe un fuerte proceso de cálculo

Por ejemplo, si tenemos un archivo con todos los alumnos de la sección de ISI, entonces. Acumula o contabiliza distintos elementos, pero SOLO MUESTRA LOS RESULTADOS AL FINAL DEL PROCESO, normalmente armando una matriz resultado, esa matriz resultado una vez que nosotros la tenemos la podemos listar o mostrar por pantalla o grabar en algún programa especial y quizás armar un gráfico tipo estadístico.

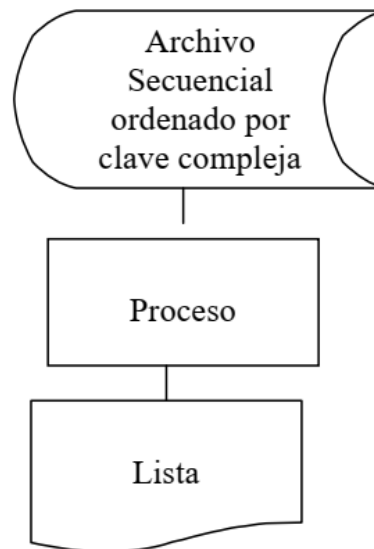
### Corte de control

Dentro de la generación de informes se puede producir lo que se llama ruptura de control, que consiste en la modificación de una línea de detalles del informe debido a que los datos dejan de estar relacionados entre sí, porque varía el valor de alguno de los campos claves. Un corte en la secuencia se da a partir de una condición que controla por cuál de las opciones debe seguir. Este puede hacerse mediante una marca (bandera, flag, señal), o comparando variables de control.

Cuando uno de los campos que se consideran claves en el proceso difiere en un valor del que tuvo en la etapa anterior, el algoritmo efectúa un corte llamado de nivel que se realiza con una sub-acción la cual realiza las acciones necesarias para tratarlo.

Las acciones más comunes en este tipo de proceso (generalmente manejo de cuantificadores) son la de presentar subtotales, los cuales son acumulados para emitir un total general al final del proceso. Una vez hecho esto se procede a actualizar el resguardo con el valor actual del campo clave para que el proceso no entre en un bucle infinito.

La característica fundamental de este proceso es que los datos del fichero deben estar ordenados por una clave compleja, además de ser deben ser consistentes y congruentes.



Esquema de corte de Control
<b>Algoritmo</b> <b>Iniciar adquisición</b> <b>Iniciar contadores</b> <b>ResN:=claveN; Res2:=clave2; Res1:=clave1;</b> <b>Mientras no FDA(Arch) hacer</b> <b>Tratar corte</b> <b>Tratar registro</b> <b>Leer (Arch, R)</b> <b>Fin mientras</b> <b>Corte_N</b> <b>Emitir totales generales</b> <b>Fin acción</b>

<b>Tratar corte</b>
<b>Si claveN <math>\diamond</math> ResN entonces</b> <b>Corte_N</b> <b>Sino Si clave2 <math>\diamond</math> Res2 entonces</b> <b>Corte_2</b> <b>Sino Si clave1 <math>\diamond</math> Res1 entonces</b> <b>Corte_1</b> <b>Fin si</b> <b>Fin si</b> <b>Fin si</b>

Subacción corte_N {Acciones de corte}	
<b>Algoritmo</b>	
<b>Corte_(N – 1)</b>	{Ejecutar el corte de nivel inmediato inferior(si existe)}
<b>Emitir totales de nivel</b>	{Emitir los totales del nivel o bien la acción que corresponda al corte}
.	
<b>Acumular para nivel superior</b>	
<b>Inicializar contadores de nivel</b>	{Poner a cero los totalizadores de nivel}
<b>Resguardar claves</b>	{Actualizar el campo e resguardo}
<b>Fin acción</b>	

Corte de control es un padrón, pero debe estar **ordenado por clave compleja y el problema me tiene que exigir cantidades totales parciales**. Se pide que liste algún corte, Clave compleja.

Es una acción, un programa donde se pide que tenga el dato de una clave compleja (tiene varios campos), además el problema nos debe pedir ciertos **totales en particular**. Es decir, es necesario que el problema nos pida subtotales de cada clave. Se muestra y se acumula en el nivel superior excepto el último en el que solo se emiten los totales.

### **Diferencia entre corte y padrón**

Si se solicita que un archivo esté ordenado por cierto campo, entonces tenemos un padrón. Un proceso Corte de Control es un padrón con totales parciales. Es necesario, para hacer el corte, que el archivo esté ordenado por clave compleja. El total final se obtiene con el padrón. Los totales parciales, con el corte de control.

## **Procesos Múltiples**

¿Cuándo es un proceso múltiple? Cuando se tiene más de un archivo en la entrada mínimo dos. Podemos tener procesos múltiples falsos

Como mínimo dos ficheros de entrada puedo no tener un fichero de salida, pero generalmente hay uno o más ficheros de salida, lo más general es que se genere un fichero de salida.

- Se aplica la “técnica de apareo”.
- Las estructuras de los ficheros deben tener un elemento común: la “clave de apareo” o “campo clave”.
- Los ficheros deben estar ordenados por el “campo clave”.

Tipos de procesos:

- Mezcla
  - Ficheros de entrada: por lo menos dos.
  - Ficheros de salida: uno (resultado de la combinación de los dos de entrada).



- Actualización Secuencial Por Lotes
  - Cero o más registros del fichero movimiento por cada registro del maestro
- Actualización Secuencial Unitaria
  - Cero o un (como máximo) registro del fichero movimiento por cada registro del maestro.

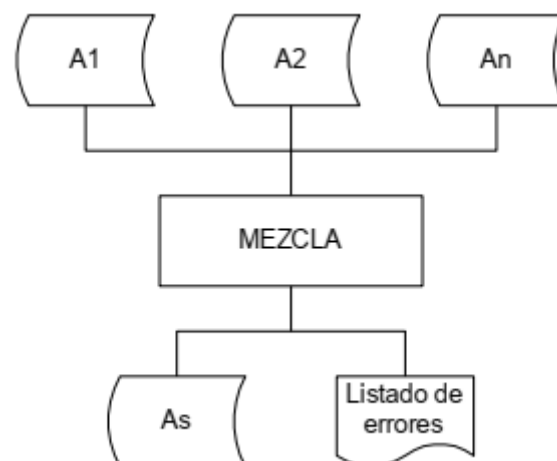
## Mezclas

En los procesos de mezclas se arman parejas (apareo), que pueden ser de forma directa homogénea o Heterogénea indirecta, esto hace referencia al tipo de información que tiene cada archivo. Las mezclas directas generalmente son automáticas, lo hace el sistema operativo no es necesario programar. Para hacer una mezcla heterogénea es necesario que el archivo comparta una misma clave y estar ordenado, ambos archivos que intervengan en una mezcla tienen que estar **ordenados por la misma clave**. Con que haya un solo campo distinto vamos a tener una mezcla indirecta.

Si es Heterogénea no se puede predecir la cantidad ni el formato del archivo de salida.

En una mezcla lo importante es como hago, que archivo es el más importante que va a manejar el tipo de mezcla. Puede ser ciclos (pre-test) incluyente o excluyente.

Incluyente vamos a tener todos los registros de los archivos. En excluyente vamos a tener solamente dentro del ciclo principal a los registros comunes y fuera de él todos aquellos registros que nos quedan fuera del ciclo principal todos aquellos otros registros que quedan en los archivos. Debemos mirar el planteo del problema para elegir el tipo de ciclo que vamos a usar, si me piden comunes nunca vamos a usar el incluyente, voy a usar el excluyente.



### Tipos de Mezcla:

	DIRECTA	INDIRECTA
Formato de los registros de los Ficheros de Entrada	igual	distinto
Formato de los registros del Fichero de Salida	igual al de los ficheros de entrada	igual a alguno de los ficheros de entrada o una combinación de estos
Cantidad de registros del Fichero de Salida	es igual a la sumatoria de las cantidades de los registros de los ficheros de entrada: $n$ $\# As = \sum_{i=1} \# Ai$	no es posible predecir

Ciclos de apareo:

- Incluyente

```
MIENTRAS (Clave1 <> HV) o (Clave2 <> HV) o .... (ClaveN <> HV) HACER
    PROCESO
Fin Mientras.
```

HV (high value) si el archivo está ordenado de menor a mayor, pero si tuviéramos ordenado el archivo de mayor a menor debemos colocar LV (low value). El valor HV es un carácter más del tamaño del campo, la máquina comprime ese número para que entre en el tamaño que definimos.

- Excluyente

MIENTRAS NO FDA (Arch 1) y NO FDA(Arch 2) HACER  
PROCESO de registros comunes  
Fin Mientras.

[illegible]

Fin mientras.

MIENTRAS NO FDA (Arch 2) HACER  
 PROCESO de Registros del Arch 2  
 Fin Mientras.

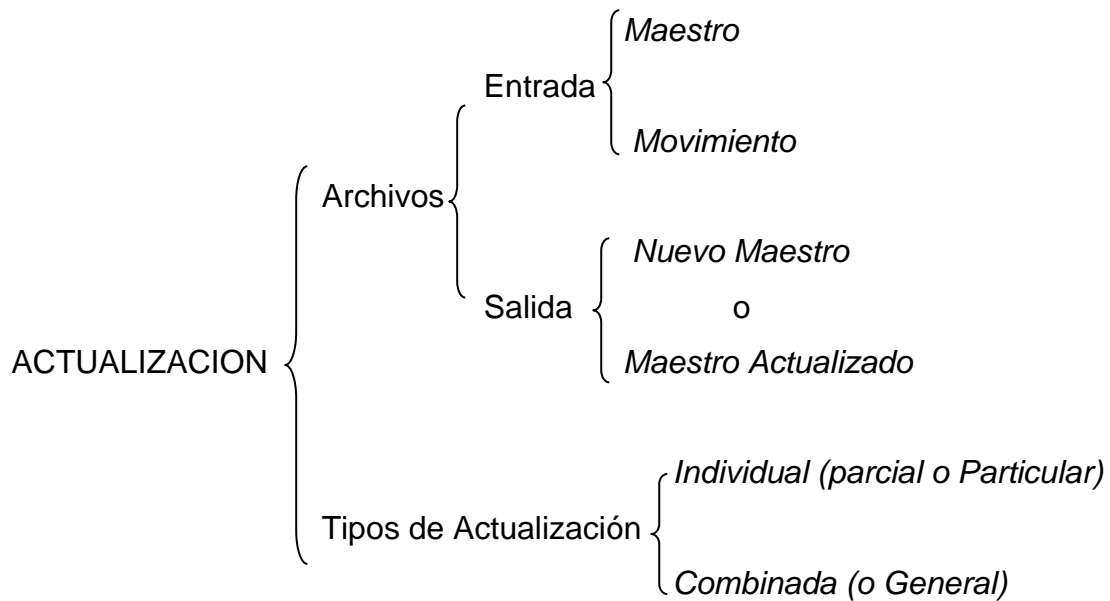
Si hay más de 2 ficheros se necesitarán más ciclos, además del ciclo principal.  
Por ej: para 3 ficheros se necesitarán

7 ciclos:

- 1 - Condición: NO FDA (Arch 1) y NO FDA (Arch 2) y NO FDA (Arch 3) - Ciclo principal que procesa registros comunes.
- 2 - Condición: NO FDA (Arch 1) y NO FDA (Arch 2)
- 3 - Condición: NO FDA (Arch 1) y NO FDA (Arch 3)
- 4 - Condición: NO FDA (Arch 2) y NO FDA (Arch 3)
- 5 - Condición: NO FDA (Arch 1)
- 6 - Condición: NO FDA (Arch 2)
- 7 - Condición: NO FDA (Arch 3)

### Actualización

Significa incorporar, modificar o eliminar información de un archivo mayor (maestro) y se utilizan como mínimo dos de entrada y uno de salida.



Según la cantidad de movimientos se clasifican en:

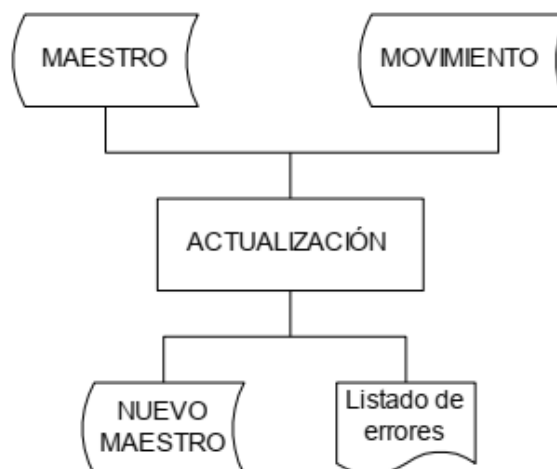
- Unitaria: Un movimiento por cada registro maestro.
- Por lotes: Varios movimientos por cada registro del maestro.

Según su organización el archivo puede ser:

- Secuencial: El archivo maestro es secuencial.
- Indexada: El archivo maestro es indexado.

Ficheros de entrada: por lo menos dos, MAESTRO Y MOVIMIENTOS.

Ficheros de salida: como mínimo uno.





#### Tipos de Actualización:

POR LOTES	UNITARIA
Varios registros de movimiento para un registro maestro.	Un registro movimiento para un registro maestro.

Tipos de Ficheros Maestros: Es el archivo de estructura mayor y es el que va a sufrir actualización. Por su periodicidad puede ser:

- Histórico: Contiene información desde la implantación del sistema en la organización o desde el nacimiento de la entidad. Es muy difícil de mantener ya que crece día a día.
- Común o Normal: Contiene parte de la información del maestro histórico. Es la información que la entidad necesita rutinariamente.

Tipos de Ficheros de Movimientos: Es el archivo de estructura menor o igual al del maestro, y debe contener como mínimo el campo clave de apareo, para poder realizar la actualización.

- Altas: Incorporación de un nuevo registro al archivo maestro manteniendo la unicidad de claves en general.
- Modificación: Corresponde a la existencia de un registro del maestro en el cual se desea modificar parte o todo su tipo contenido, exceptuando su campo clave
- Bajas: Corresponde a la eliminación física o lógica de un registro del maestro.
- Baja Física: Consiste en eliminar totalmente contenido y clave del registro del archivo maestro.
- Baja Lógica: Consiste en señalar o marcar el registro como eliminado, pero sin eliminarlo del maestro. La baja lógica requiere un código de estado que es un campo que se le agrega al registro.

#### Tipos de Actualización:

- INDIVIDUAL (parcial o particular): un archivo de movimientos individual es aquel que contiene movimientos de un solo tipo.
- GENERAL O COMBINADA: un archivo de movimientos general contiene registros de los tres tipos de movimientos: Altas, Bajas y Modificaciones.

### Actualización secuencial

Proceso Diferido o Batch: Es batch porque los movimientos están en un archivo. Los errores se corrigen cada cierto periodo de tiempo. Se corrigen normalmente antes de la próxima actualización.

Es un proceso seguro ya que puede ser cuantificado; se puede decir si fue bueno o malo y donde se produjo el error.

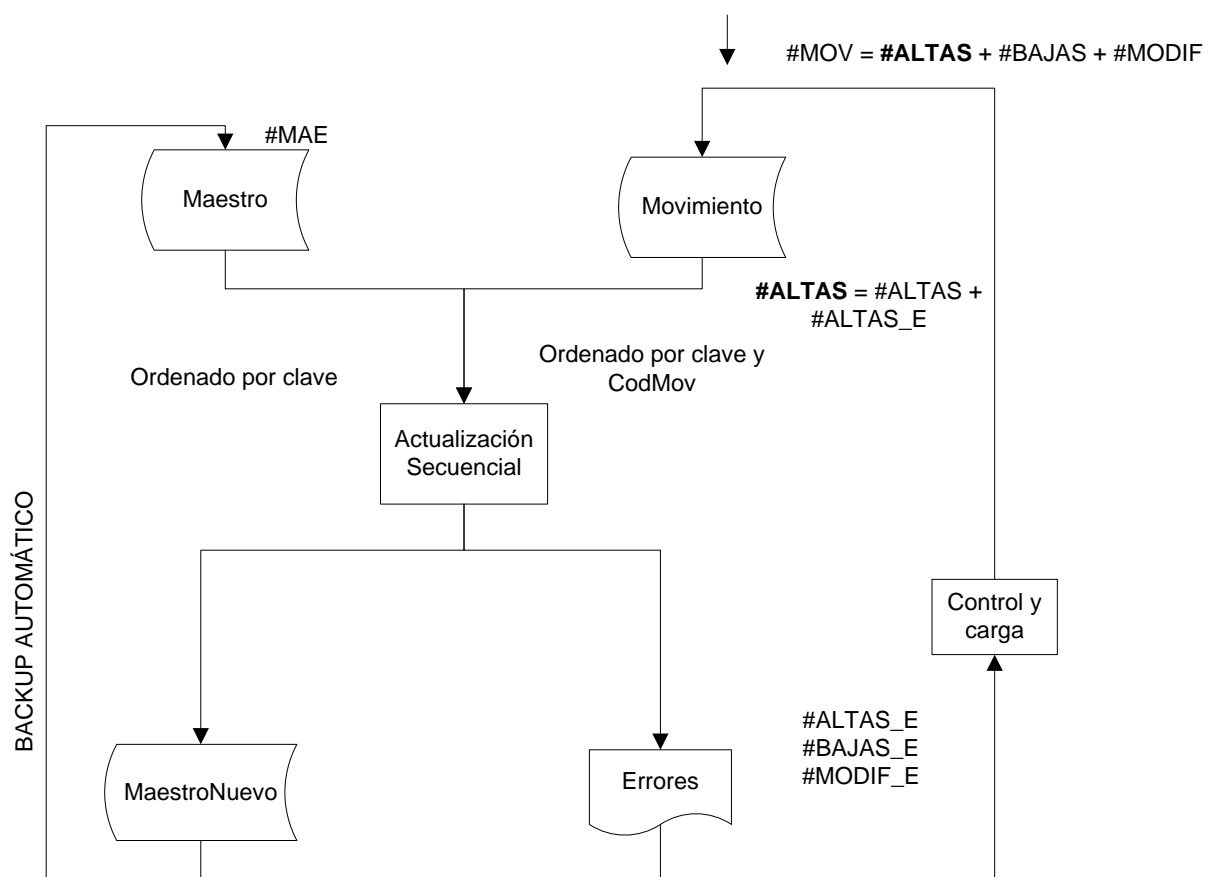
El maestro actualizado es correcto si:

$$\#Reg. MAE\_ACTUAL = \#Reg. MAESTRO + \#ALTAS - \#BAJAS$$

$$\#ERRORES = \#ALTASE + \#BAJASE + \#MODIFICACIONESE$$

$$\#Reg. MOVIMIENTO = \#ALTASCI + \#BAJAS CI + \#MODIFICACIONES CI$$

Esquema de actualización secuencial



## Referencias

#MAE: número de registros existentes en el archivo maestro.

#MOV: número de registros existentes en el archivo movimiento, compuesto por:

#ALTAS + #BAJAS + #MODIF

#MN: número de registros grabados en el nuevo archivo maestro, compuesto por:

#MAE - #BAJAS + #ALTAS

#ALTAS: cantidad de altas correctas.

#BAJAS: cantidad de bajas correctas.

#MODIF: cantidad de modificaciones correctas.

#ALTAS\_E: cantidad de altas erróneas

#BAJAS\_E: cantidad de bajas erróneas.

#MODIF\_E: cantidad de modificaciones erróneas.

Características: Lento, seguro, diferido, batch.

### Ejemplo de actualización unitaria

#### Utilizando ciclo incluyente

ACCION ACT\_UNIT ES

Algoritmo

Abrir\_Archivos

Leer\_Maestro

Leer\_Movimiento

MIENTRAS (Clave\_Mae <> High Value) o (Clave\_Mov <> High Value) HACER

    SI Clave\_Mae = Clave\_Mov ENTONCES

        // Movimiento

            Proceso\_iguales

    SINO

        SI Clave\_Mae < Clave\_Mov ENTONCES

            // Maestro sin movimiento

                Reg\_sal:= Reg\_mae

                ESCRIBIR (Arch\_sal, Reg\_sal)

                Leer\_Maestro

        SINO

            // Movimiento sin Maestro

                Proceso\_distintos

        Fin\_si;

    Fin\_Si;

Fin\_mientras;

Cerrar Archivos



FACCIÓN.

Acción Leer\_Movimiento es

LEER(Arch\_mov, Reg\_mov)

SI FDA(Arch\_mov)

ENTONCES Clave\_mov := High\_value

FSI;

FAcción.

Acción Leer\_Maestro es

LEER(Arch\_mae, Reg\_mae)

SI FDA(Arch\_mae)

ENTONCES Clave\_mae := High\_value

FSI;

FAcción.

Acción Proceso\_iguales es

SI Reg\_mov.Cod\_mov = "ALTA"

ENTONCES ESCRIBIR ("Error alta no existe")

Reg\_sal:= Reg\_mae

ESCRIBIR (Arch\_sal, Reg\_sal)

SINO

SI Reg\_mov.Cod\_Mov = "MODIF" ENTONCES

Proceso\_modif\_maestro

Reg\_sal:= Reg\_mae

ESCRIBIR (Arch\_sal, Reg\_sal)

SINO \* eliminación lógica \*

Marcar\_registro

Reg\_sal:= Reg\_mae

ESCRIBIR (Arch\_sal, Reg\_sal)

Fsi;

Fsi;

Leer\_Maestro

Leer\_Movimiento

FAcción.

Acción Proceso\_distintos es

SI Reg\_mov.Cod\_Mov = "BAJA" ENTONCES

```

        ESCRIBIR ("Error baja no existe")
SINO
    SI Reg_mov.Cod_Mov = "MODIF" ENTONCES
        ESCRIBIR("Error modificación no existe")
    SINO * Asigna campo por campo, porque Reg_sal y Reg_mov tienen
distinto formato *
        Reg_sal.clave:= Reg_mov.clave
        Reg_sal.campo1:= Reg_mov.campo1
        Reg_sal.campo2:= Reg_mov.campo2
        .....
        Reg_sal.campoN:= Reg_mov.campoN
        Reg_sal.Marca_baja:= " "
        ESCRIBIR (Arch_sal, Reg_sal)
    Fsi;
Fsi;
Leer_Movimiento
FAcción.
Acción Proceso_modif_maestro es
    Si Reg_Mov.campo1 <> " " entonces
        Reg_mae.campo1:= Reg_mov.campo1
    fsi;
    Si Reg_Mov.campo2 <> " " entonces
        Reg_mae.campo2:= Reg_mov.campo2
    fsi;
    Si Reg_Mov.campo3 <> ""
        entonces Reg_mae.campo3:= Reg_mov.campo3
    fsi;
    * ... y así sucesivamente para todos los campos del registro...*
FAcción.
Acción Marcar_registro es
    Reg_mae.Marca_baja:= "*" * en vez de asterisco, se puede asignar la
fecha del día, o cualquier otro dato, según el problema *
FAcción.

```

Algoritmo Actualización por lotes

Utilizando ciclo incluyente, se considera un archivo de movimientos consistente y se realizan bajas lógicas.

Consideraciones sobre el ambiente:

ACCION ACT\_LOT ES

AMBIENTE

Formato\_Clave = Registro

clave1: ...

clave2: ...

...

claven: ...

fin registro

formato\_Maestro = Registro

clave: formato\_clave

campo1: ...

campo2: ...

...

campon: ...

Marca\_baja: ...

fin registro

formato\_Movimiento = Registro

clave: formato\_clave

campo1: ...

campo2: ...

...

campon: ...

cod\_mov: ("A", "M", "B")

fin registro

mae, mae\_sal: archivo de formato\_Maestro ordenado por clave

reg\_sal, reg\_mae, aux: formato\_Maestro

```
mov: archivo de formato_movimiento ordenado por clave
reg_mov: formato_movimiento
```

#### ALGORITMO

Abrir\_Archivos

Leer\_Maestro

Leer\_Movimiento

MIENTRAS (reg\_mov.clave <> High\_Value) o (reg\_mae.clave <> High\_Value)

HACER

SI reg\_mae.clave < reg\_mov.clave ENTONCES

//Maestro sin Movimiento

reg\_sal:= reg\_mae

ESCRIBIR (mae\_sal, Reg\_sal)

Leer\_Maestro

SINO

SI reg\_mae.clave = reg\_mov.clave ENTONCES

//Movimiento

aux: = reg\_mae

MIENTRAS (reg\_mae.clave = reg\_mov.clave) HACER

Proceso\_Movim

Leer\_Movimiento

Fin Mientras

reg\_sal: = Aux

ESCRIBIR(mae\_sal, reg\_sal)

Leer\_Maestro

SINO

// Movimiento sin Maestro ~ 1 Alta y 0-1 Modif. y/o Bajas

// Asigna campo por campo, porque Aux y Reg\_mov tienen distinto

formato

Aux.clave := Reg\_mov.clave

Aux.campo1:= Reg\_mov.campo3

Aux.campo2:= Reg\_mov.campo4

.....

Aux.campon:= Reg\_mov.campon

```

        Aux.Marca_baja:= ""
        Leer_Movimiento
        MIENTRAS (aux.clave = reg_mov.clave)) HACER
            Proceso_Movim
            Leer_Movimiento
        FMientras
        reg_sal:= aux
        ESCRIBIR (mae_sal, reg_sal)
    FIN SI
FIN SI
FIN MIENTRAS

```

```

CERRAR(mae)
CERRAR(mae_sal)
CERRAR(mov)

```

Subacción Leer\_Movimiento es

```

    LEER(mov, Reg_mov)
    SI FDA(mov) ENTONCES
        reg_mov.clave := High_value
    FSI;
FAcción

```

Subacción Leer\_Maestro es

```

    LEER(mae, reg_mae)
    SI FDA(mae) ENTONCES
        reg_mae.clave : = High_value
    FSI;
FAcción

```

Subacción Proceso\_Movim es

```

//Comparar la información del Registro de Movimientos y, de acuerdo a los
//valores que tengan, alterar los contenidos del Registro Auxiliar (Aux).
    SI reg_mov.Cod_Mov = "M" ENTONCES

```

```

//Modificación
    Proceso_modif_maestro
SINO
    SI reg_mov.Cod_Mov = "B" ENTONCES
        //eliminación lógica
        Marcar_registro
    Fsi
Fsi
FAcción

```

```

Subacción Proceso_modif_maestro es
    Si Reg_Mov.campo1 <> "" entonces
        // No se actualizan los campos claves.
        Aux.campo1 := Reg_mov.campo1
    fsi;
    Si Reg_Mov.campo2 <> "" entonces
        Aux.campo2 := Reg_mov.campo2
    fsi;
    .....
    //... y así sucesivamente para todos los campos del registro...
    .....
    Si Reg_Mov.campon <> "" entonces
        Aux.campon := Reg_mov.campon
    fsi;
FAcción.

```

```

Subacción Marcar_registro es
//en vez de asterisco, se puede asignar la fecha del día,
//o cualquier otro dato, según el problema
    Aux.Marca_baja:= "*"
FAcción.

```

## Archivo Indexado

Es la organización de archivos que incluye índices en el almacenamiento de los registros; de esta forma nos será más fácil buscar un registro determinado sin necesidad de recorrer todo el archivo.

Un campo (o un grupo de campos) del registro denominado CLAVE es utilizado como campo de índice. Por ejemplo, en una aplicación bancaria, podría existir un archivo de registros que describiesen a las sucursales. Por lo que sería adecuado indexar el archivo en base al nombre de la sucursal, para proporcionar información de una sucursal en particular a través de consulta interactiva.

### *Implicancia del uso de índices*

a.- La colocación de un listado al inicio del archivo: para la identificación del contenido.

b.- La presentación de un segundo índice: para reflejar la información de cada punto principal del índice anterior.

c.- La actualización de los índices: Cuando se insertan y eliminan archivos, es preciso actualizar los índices para evitar contratiempos actualizando un archivo.

d.- La organización de un índice: Nos evita examinar archivo por archivo para recuperar algún registro buscado; por lo tanto, ahorraríamos tiempo si tenemos una adecuada organización de los índices.

### *Como está constituido físicamente un archivo Indexado*

En la vista física de un archivo indexado, observaremos dos áreas: el área de índices y el área de datos.

El área de índices consiste en un listado de todos los valores del campo clave de los registros en el archivo, junto con la posición del registro correspondiente en el almacenamiento masivo (área de datos).

Área de índices	CLAVE	DIRECCIÓN
	15	010
	24	020
	36	030
	54	040
	.	.
	.	.
	.	.
240	090	

		CLAVE	DATOS
Área principal	010	15	
	011		
	012		
	.		
	.		
	.		
	019		
	020	24	
	021		
	.		
	.		
	.		
	029		
	030	36	
	031		

El área de índices está SIEMPRE ORDENADA de menor valor de clave a mayor. Mientras que como se observa en la figura en el área de Datos los registros de información no guardan ningún orden, se han ido almacenando de acuerdo con las disponibilidades libres que tenía la memoria en el momento que se generó la orden de grabación.

Los índices apoyan las aplicaciones para acceder selectivamente a registros individuales, en lugar de buscar a través de toda la colección de registros en secuencia.

Verbos para archivos indexados

<b>SECUENCIAS</b>	<b>ARCH SECUENCIAL</b>	<b>ARCH INDEXADO</b>
ARRANCAR (SEC)	ABRIR (ARCH)	ABRIR (ARCH)
AVANZAR (SEC, V)	LEER (ARCH, REG)	CLAVE:= Información
CREAR (SEC)	ABRIR /S (ARCH)	LEER (ARCH, REG)
VS:= V	Asignar a cada campo del registro los valores	ABRIR /S (ARCH)
AVANZAR (SEC, VS)	GRABAR (ARCHS, REGS)	GRABAR (ARCHS, REGS)
----	-----	ABRIR E/S (ARCH)
----	-----	CLAVE:= Información
----	-----	LEER (ARCH, REG)
----	-----	Modificar los datos
----	-----	RE GRABAR (ARCH, REG)
		BORRAR (ARCH, REG)

## Grabar versus regrabar

Cuando un archivo, sin importar a que organización se refiera, es abierto como archivo de salida, es seguro que durante el proceso existirá la orden de grabar en el mismo.

Para proceder a grabar, previamente se deben asignar a cada uno de los campos del registro de salida, los contenidos correspondientes. Estos contenidos se pueden obtener de distintos lugares dentro del proceso (archivos de entrada, cálculos, etc.) y generalmente no se asignan todos juntos a último momento sino, que se van trasladando a medida que se van obteniendo. La orden de grabar el registro se debe efectuar cuando la lógica del procedimiento de obtención de los datos ya haya finalizado, es decir, el registro ya contiene todos los datos necesarios.

El proceso de grabación implica que la máquina, buscará generar un NUEVO registro en el archivo correspondiente. Si el archivo de salida es de organización secuencial, esto significará agregar a continuación del último registro emitido, dado que este tipo de organización obliga a la continuidad física.

¿Qué puede ocurrir? de no existir espacio físico continuo disponible, se producirá un error, y la máquina emitirá un aviso de ello, y en muchos casos una interrupción definitiva del programa.



De no haber inconvenientes, se procede a grabar el registro y continúa con la lógica del programa.

Solo para las **organizaciones Directas**, es posible utilizar además del verbo grabar el verbo Regrabar. Para poder hacer uso del mismo se debe haber abierto el archivo como de entrada/salida.

En este último caso, la apertura del archivo E/S, hace que tanto se puedan crear registros nuevos mediante el verbo grabar, como habiendo previamente leído un registro, producir modificaciones sobre algunos de los contenidos en los campos del mismo y entonces la orden de salida será Regrabar.

Como el registro ya está leído, el espacio de memoria está asignado, por lo que esta orden no genera error por falta de espacio, dado que se reescribe sobre el mismo.

Entonces ¿Que puede ocurrir?, el error de producirse se deberá a un problema de lógica (error nuestro), estamos intentando regrabar sin haber previamente leído, o (muy raro) que se haya perdido el enlace con el área de lectura.

### Comparación entre secuenciales e indexados.

Un archivo secuencial ocupa en memoria solo el tamaño que necesita para guardar los registros de información.

Un archivo indexado ocupa el espacio de memoria para guardar los registros de información (área de Datos) y además un espacio más para poder almacenar los índices (área de índices).

Un archivo secuencial no necesariamente está ordenado, en cambio un archivo indexado siempre está ordenado según el índice.

Si quisiéramos efectuar un listado total de un archivo, a igualdad de volumen de información (cantidad de registros), el tiempo de recorrido de un archivo secuencial, sería más rápido que el de un archivo indexado... pero cuando queremos hacer la búsqueda de un registro en particular, la velocidad de acceso de un indexado es superlativa con respecto a la de un archivo secuencial.

### ***Actualización Indexada***

En una actualización indexada el archivo Maestro se encuentra justamente indexado por la clave principal (ordenado automáticamente por clave).

Las novedades si bien podrían venir en un archivo tal como ocurre en la actualización secuencial, no es la forma más común; normalmente lo que existe es una terminal con lo que denominamos, un usuario inteligente.

¿Qué significa inteligente?, que este usuario tiene ciertos privilegios que le permiten actuar según los permisos que le otorga el privilegio.

Principalmente este usuario puede cargar los datos y corregirlos durante la carga, por lo que esto implica una **interactividad** con el programa de actualización. A mayor nivel de permisos, mayor será el riesgo de seguridad, dado que el usuario estará trabajando sobre el archivo Maestro y todo lo que realice afectará directamente a dicho archivo, esto implica una actualización **In Situ**.

Cada dato ingresado por el usuario está precedido por el valor de clave, lo que permite acceder directamente al registro correspondiente en el archivo Maestro (acceso puntual o al azar).

¿Qué puede hacer el usuario?, según los privilegios, podrá incorporar un nuevo registro al archivo (proceso de ALTA), podrá modificar la información de un registro existente en el archivo (proceso de MODIFICACION) o podrá eliminar / marcar un registro (proceso de BAJA).

¿Por qué elegir este tipo de proceso?

Bien, hemos visto que este es un proceso inseguro, ¿qué justificará entonces que lo usemos?

¡¡ Velocidad !! en el proceso, imagínate si fueras un usuario parado ante una terminal bancaria y quisieras saber tu saldo de la cuenta de ahorro, y el proceso fuera secuencial.... El archivo Maestro debería recorrerse hasta encontrar tu número de usuario, para darte la información de tu saldo, si eres el usuario nro. 1, obviamente no vas a notar diferencia, pero si tu número de usuario es el 198734562, ¿cuánto tiempo tardará en leer los datos previos hasta llegar a tu clave?....

Y una vez que vos te retiras de allí, ¿cómo hará para tratar el próximo usuario que se acerque a la terminal?

¡Claro !, por ser secuencial, ¡¡deberá cerrar el archivo y volver al inicio...!! ORGANIZACIÓN SECUENCIAL solo admite ACCESO SECUENCIAL.

Ahora, como el archivo es indexado puedo hacer ACCESO PUNTUAL, basta con informar la clave para que se acceda a la información correspondiente a dicha clave y no importa si el usuario anterior fue una clave menor o mayor.

Esta rapidez en el tratamiento de la información permite que la interactividad con el usuario se pueda dar en forma eficiente, y este es el motivo de elegir este proceso.

¿Cómo mejoramos el proceso de actualización indexada?

Si bien este proceso de actualización es el más veloz, su característica de inseguro hace que pensemos en algún "artilugio", o proceso extra que nos permita de alguna manera corregir esta falencia.

Normalmente, de no existir una forma más económica de asegurarnos de que este proceso no rompa con la característica de "correcto" del maestro indexado, adosamos un proceso en paralelo de actualización secuencial. ¿Por qué?, justamente por la característica de seguridad del proceso secuencial.

¿Cómo es esto? Sin que el usuario se dé cuenta, por cada acción "correcta" del usuario sobre el archivo maestro se graba la información de cuál ha sido dicha acción, en un archivo secuencial de novedades. En un determinado tiempo el proceso indexado se para, y se efectúa el proceso secuencial tomando como archivo maestro la copia de seguridad que se tiene del maestro previo a todo el proceso (esto es para la primera vez); y el archivo de novedades generado con las acciones "correctas" del usuario.

Se ejecuta el procedimiento y al finalizar se compara el archivo maestro resultante de este último proceso con el que se tiene del proceso indexado, si son iguales (no importa que uno tenga organización secuencial y el otro indexada, igual se pueden comparar), quiere decir que es fiable el proceso indexado y se vuelve a restablecer el sistema para el usuario.

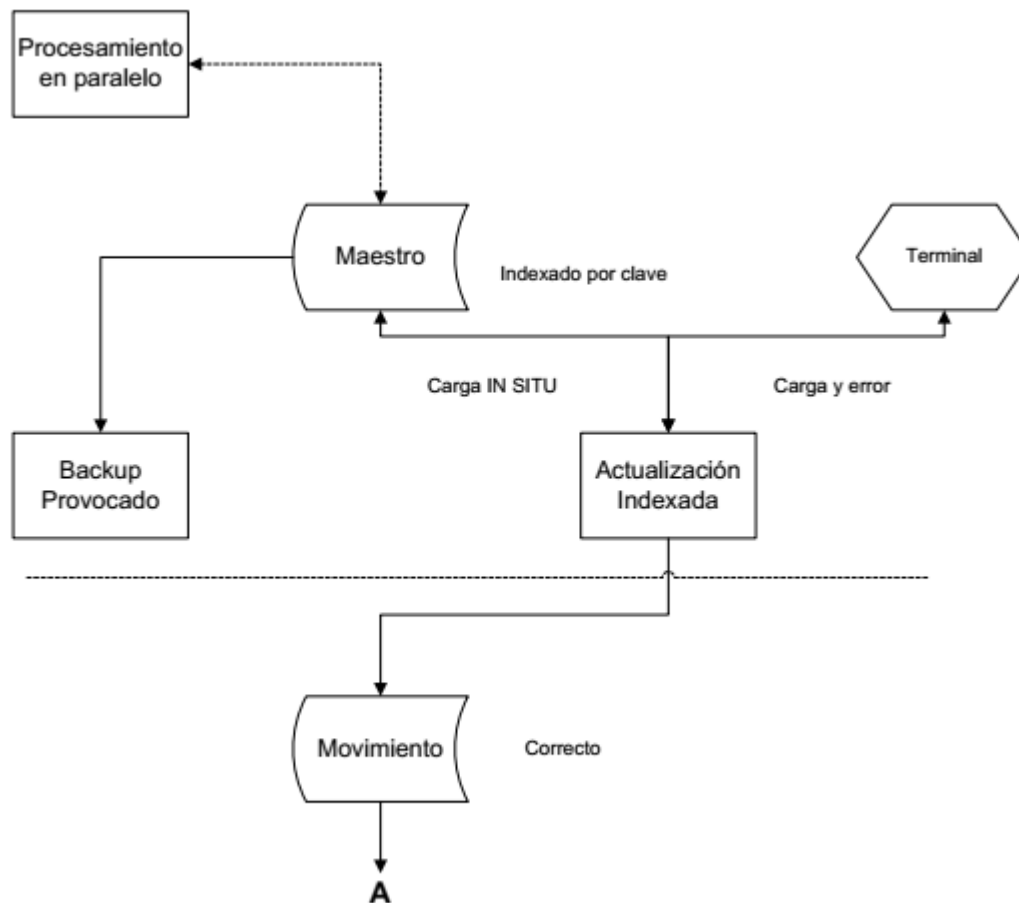
¿Se dio cuenta de esto el usuario?, no, normalmente en procesos con terminales remota, aparece el mensaje de "disculpe, estamos en tareas de mantenimiento", etc.

La ventaja de este paralelismo, además de asegurarnos que no estamos perdiendo la condición de correctitud del archivo indexado es que también nos genera la copia de seguridad (archivo maestro de salida del proceso secuencial) en forma automática, otra característica buena que tenía la actualización secuencial.

De haber diferencias, se para el sistema y se somete a procesos de auditoria de la información, para validar los datos del maestro indexado.

¿En qué momento se efectúa este proceso en paralelo?, en horarios que no afectan al normal proceso de la información, por ejemplo, madrugada, feriados, o períodos que se determinan estadísticamente como los más convenientes.

## Esquema de actualización indexada



Características: interactivo, rápido e inseguro.

- Archivo maestro indexado por clave
- Se ingresan movimientos por terminal

ACCION ACT\_INDEX\_UNIT ES

Ambiente

Valido= ("A", "B", "M")

Arch\_mae: archivo de reg\_mae indexado por clave\_mae.

Cod\_mov: AN(1)

Clave: N(2)

Algoritmo

Abrir E/S (arch\_mae)

Escribir ("Por favor ingrese la clave a procesar y el código de movimiento (A: incorporaciones, B: bajas, M: modificaciones) Para finalizar ingrese cualquier otra letra.")

Leer (clave, cod\_mov)

```

MIENTRAS cod_mov en valido hacer
    Reg_mae.clave_mae := clave
    Leer (arch_mae, Reg_mae)
    SI No existe ENTONCES
        SI cod_mov = "B" ENTONCES
            ESCRIBIR ("Error baja no existe")
        SINO
            SI cod_mov = "M" ENTONCES
                ESCRIBIR ("Error modificación no existe")
            SINO ** Ingresar por teclado los datos correspondientes a la
nueva clave **
                Leer (Reg_mae.campo1)
                Leer (Reg_mae.campo2)
                .....
                Leer (Reg_mae.campon)
                ESCRIBIR (arch_mae, Reg_mae)
            FSI;
        FSI;
    SINO
        SI cod_mov = "A"
            ENTONCES ESCRIBIR ("Error clave existe, alta no es posible")
        SINO
            SI cod_mov = "M" ENTONCES ** Ingresar por teclado los datos
que se desean modificar **
                Leer( campo1)
                SI campo1 <> " " ENTONCES
                    Reg_mae.campo1 := campo1
                FSI
                Leer(campo2)
                SI campo2 <> " " ENTONCES
                    Reg_mae.campo2 := campo2
                FSI
                .....
                Leer(campon)

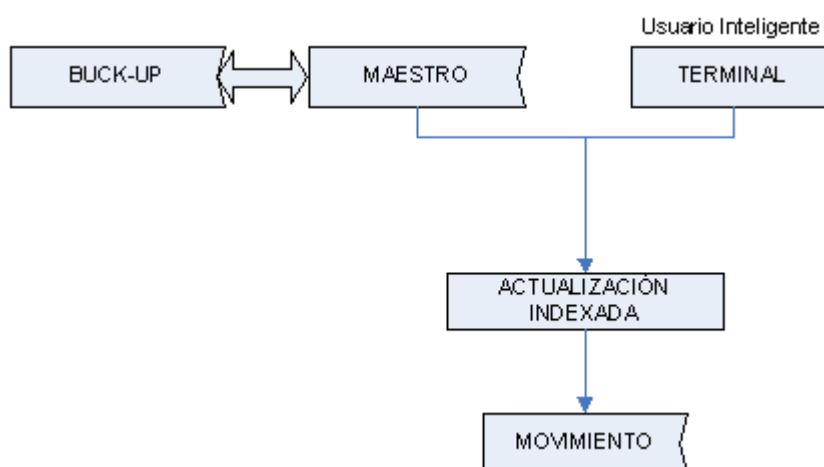
```

```

    SI campON <> " " ENTONCES
        Reg_mae.campoN := campON
    FSI
    RE-ESCRIBIR (arch_mae, Reg_mae)
SINO **{baja lógica}**
    Marcar Registro Maestro
    RE-ESCRIBIR(arch_mae, Reg_mae)
    ** o {baja física}**
    BORRAR (arch_mae, Reg_mae)
FSI;
FSI;
FSI;
    Escribir ("Por favor ingrese la clave a procesar y el código de
movimiento (A: incorporaciones, B: bajas, M: modificaciones) Para finalizar
ingrese cualquier otra letra.")
    Leer (clave, cod_mov)
Fin_Mientras
Cerrar (Arch_mae)
FACCIÓN.

```

### ***Actualización Interactiva IN-SITU (al momento)***



Se cuenta con una Terminal (PROCESO INTERACTIVO) ya que no existe un archivo de movimiento. Como es una Terminal, aquí se encuentra un usuario inteligente (tiene poder para

decidir). Quien corrige los errores es el usuario. Este proceso termina cuando la Terminal termina de funcionar.

#### Cuadro Comparativo

Archivos Secuenciales	Archivos Indexados
Poseen acceso secuencial y para acceder al último elemento se debe acceder a todos los anteriores.	Poseen acceso secuencial y directo a cualquier registro.
Son estructuras estáticas, que una vez creadas no se pueden ampliar. No hace falta dar su longitud.	Son estructuras dinámicas que pueden crecer o reducirse (no hace falta dar su longitud al crearlos)
Los registros se almacenan secuencialmente en memoria con adyacencia física.	Los registros de estos archivos se almacenan en memoria con adyacencia lógica y no física.
Para insertar un registro hay que crear un archivo nuevo	

## Arreglos

Un arreglo es una estructura de datos de memoria interna de característica estática que se visualiza como un conjunto de elementos con un cierto orden.

Es un conjunto de... Porque está formado por datos del mismo tipo agrupados bajo un mismo nombre y a los cuales solo se accede indicando el nombre del conjunto y la posición que ocupa dentro del conjunto.

Está en memoria interna significa que habrá que cargarle el contenido y si el mismo se quiere guardar al final del proceso en forma definitiva habrá que almacenarlo externamente.

Es estática porque se declara en el ambiente del algoritmo con un tamaño definido y no se lo puede alterar durante el proceso de ejecución.

Tiene orden significa que podremos crear agrupamientos que originarán lo que llamaremos dimensiones, pero que respetan el concepto de jerarquías.

Los arreglos se clasifican de acuerdo con el número de dimensiones que tiene:

- Unidimensionales (Vector): es un arreglo unidimensional, tipo de datos estructurado que está formado por una colección finita y ordenada de datos del mismo tipo.
- Bidimensionales (Matrices o tablas): es un espacio de memoria que permite almacenar una colección de elementos, todos del mismo tipo, pero sus elementos no están organizados linealmente, sino que su organización está acorde con una dimensión no unitaria.

- Multidimensionales (tres o más dimensiones)

### *Como lo definimos*

Nombre\_del\_tipo: arreglo [tipo\_indice] de tipo\_de\_dato\_base

Nombre\_del\_tipo: nombre válido

Tipo\_dato\_indice: debe ser de tipo ordinal, carácter o lógico, un tipo enumerado o sub-rango. Existe un elemento por cada valor del tipo índice.

Tipo\_dato\_base: describe el tipo de cada elemento del vector, todos los elementos de un vector son del mismo tipo. Según cuestionario se pueden declarar como de cualquier tipo y cualquier tamaño.

### *Uso del índice de un arreglo*

Cada referencia a un arreglo incluye el nombre y el índice encerrado en corchetes: el índice determina que elemento se procesa.

Nombre: arreglo [limitinf...limitsup] de tipo

limitinf...limitsup indica el valor inicial del rango ordenado de los valores que permitirán acceder a los elementos, la cantidad de elementos que tendrá el arreglo.

Permiten acceder a los elementos del arreglo

¿Cómo se definen?

1	2	3		80
DNI	DNI	DNI	..	DNI

A: arreglo [1..80] de N(8)

Acceso variable: Nombre [variable] Ej.: A[i]

PARA i:=1 A 80 HACER

LEER (NRO)

A[i]:=NRO

FIN\_PARA

Acceso Puntual: Nombre [valor posicional] Ej.: A[i]. Se debe escribir el nombre del arreglo y el valor posicional.

LEER (NRO)

A[1]:=NRO

LEER (NRO)

A[2]:=NRO



•  
•

LEER (NRO)

A[80]:=NRO

## ***Procesos***

Los procesos para estas estructuras son:

### **Carga de datos en memoria interna**

### **Recorridos**

De acuerdo con el orden establecido en el ambiente:

- Recorrido normal: Es aquel que permite posicionarse en cada elemento de la estructura siguiendo el orden de su definición. Es el recorrido más rápido porque es tal como lo guarda la memoria (secuencial, ordenamiento físico).
- Recorrido antinatural (anormal): Es aquel que permite posicionarse en cada elemento de la estructura no respetando el orden de su definición. Es el más lento no sigue la naturaleza de cómo se guardó la información.

Suponga que se declara el arreglo de la siguiente manera:

Cifras: arreglo [1..10] de entero

X: 1..10

Si quisiéramos recorrer el arreglo en la forma más rápida:

PARA X:=1 A 10 HACER

Tratar\_Arreglo

FIN\_PARA

Si quisiéramos recorrer el arreglo en forma descendente

PARA X:= 10 A 1, -1 HACER

Tratar\_Arreglo

FIN\_PARA

Supongamos una matriz definida como

Persona: arreglo [1..100, "a".."z"] de AN(20)

X: 1..100

Y: "a".."z"

Para recorrer por columnas

```

PARA Y:="a" a "z" HACER
    PARA X:=1 A 100 HACER
        Tratar_Arreglo
    FIN_PARA
FIN_PARA

Debo recorrer por filas invertida
PARA X:=100 A 1, -1 HACER
    PARA Y:="a" a "z" HACER
        Tratar_Arreglo
    FIN_PARA
FIN_PARA

Debo recorrer en la forma normal
PARA X:=1 A 100 HACER
    PARA Y:="a" a "z" HACER
        Tratar_Arreglo
    FIN_PARA
FIN_PARA

```

En fin, los recorridos implican observar el contenido de cada celda siguiendo una secuencia que puede ser total (recorrido total) o solo un subconjunto, pero contiguo (recorrido parcial), pero también puede seguir el sentido de la definición de su dimensión y entonces se denomina recorrido normal o natural o cualquier otra variación no natural o anormal.

## Ordenamiento

Consiste en disponer de un conjunto (estructura) de datos en algún determinado orden con respecto a uno de los campos de elementos del conjunto. Puede ser ordenación interna cuando están en un arreglo u ordenación externa cuando están en un archivo.

Una lista se dice que está ordenada por la clave k si la lista está en orden ascendente o descendente en relación con esa clave.

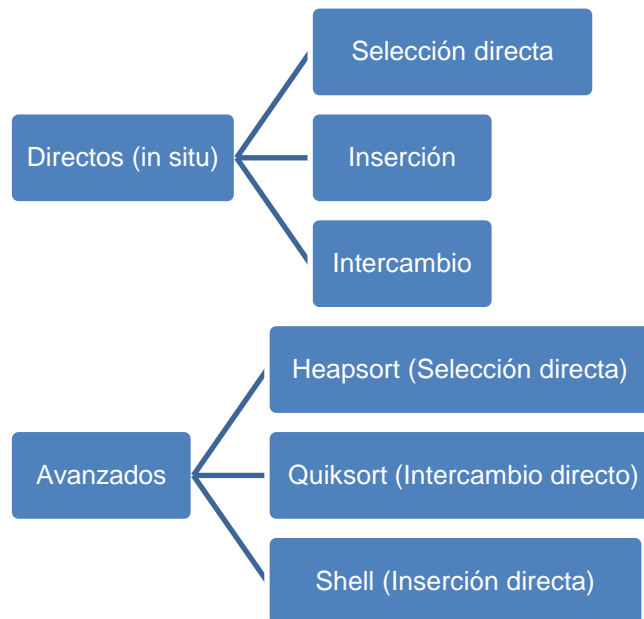
Dos criterios a tener en cuenta para determinar cuál es el mejor algoritmo:

- Tiempo de menor ejecución en computadora
- Menor número de instrucciones

Los tipos de ordenamiento son:

1. Directos: son sobre el mismo arreglo, lista pequeña, algoritmo sencillo.

2. Indirectos o avanzados: Se utilizan para ordenar archivos, memoria externa. No se utiliza en la práctica porque existen comandos u órdenes del sistema operativo.



### Selección directa

Si el vector está ordenado al revés es más rápido que el de inserción, se los puede utilizar en vectores muy grandes que no están tan desordenados.

Este método se basa en buscar el elemento menor del vector y colocarlo en la primera posición. Luego se busca el segundo elemento más pequeño y se lo coloca en la segunda posición, y así sucesivamente.

El método se basa en encontrar en cada pasada del arreglo el elemento con clave mínima (o máxima depende del planteo del ejercicio) y ubicarlo en la posición correspondiente a la pasada intercambiando los elementos. Es decir, selecciona el menor (o mayor) del arreglo no ordenado y se lo ubica en el arreglo ordenado. Existen dos vectores uno desordenado de N elementos y otro ordenado de 0 elementos.

... Algoritmo

PARA  $i:=1$  a  $(N-1)$  HACER

$X:= A[i]$  {Arreglo auxiliar para guardar el elemento}

$MIN:=i$  {Variable del tipo ordinal para guardar la posición del elemento}

    PARA  $j:=(i+1)$  A  $N$  HACER

        SI  $A[j]<X$  ENTONCES {si el elemento que está delante es menor guardamos en las variables auxiliares definidas}

$MIN:=j$

$X:=A[j]$

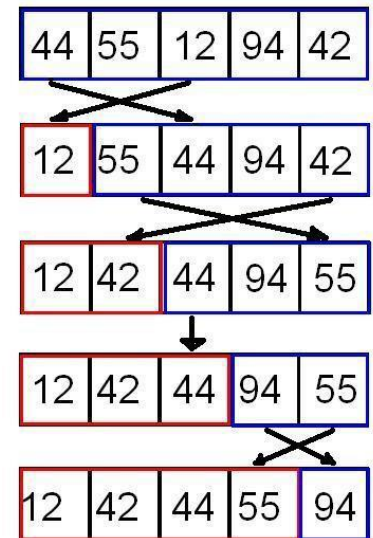
        FIN\_SI

    FIN\_PARA

$A[MIN]:= A[i]$

$A[i]:=X$

FIN\_PARA



Si necesitamos ordenar solo una parte conviene más este método.

Este método, así como el de inserción suponen el peor de los casos, es decir el caso en que el arreglo esté muy desordenado. En el caso que los elementos queden ordenados prematuramente el método seguirá trabajando hasta llegar al último elemento.

### Inserción directa

Este método consiste en insertar un elemento en el vector en una parte ya ordenada del vector y comenzar de nuevo con los elementos restantes. Por ser utilizado generalmente por jugadores de cartas se le conoce también por el método de la baraja.

Los ítems están divididos conceptualmente en un arreglo destino  $\{A[1] \dots A[i-1]\}$  y un arreglo origen  $\{A[i] \dots A[n]\}$ . En otras palabras, existen dos vectores uno desordenado de  $N-1$  elementos otro ordenado de 1 elemento. En cada caso empezando con  $i=2$  e incrementando  $i$  de uno en uno, se toma el elemento  $i$  de la secuencia origen y se transfiere a la secuencia destino insertándolo en el lugar adecuado. ¿Cómo sabemos cuál es el sitio apropiado? Como el arreglo destino está ordenado podemos “dejar caer” el elemento mientras este sea menor que cada uno de los elementos del arreglo destino.

Cuando insertamos un elemento en el arreglo ordenado los elementos que queden por encima de este deberán correrse de posición.

...Algoritmo

PARA  $i:=2$  A  $N$  HACER

$X:= A[i]$

$J:=i-1$

MIENTRAS  $((J>0) \text{ Y } (X<A[J]))$  HACER

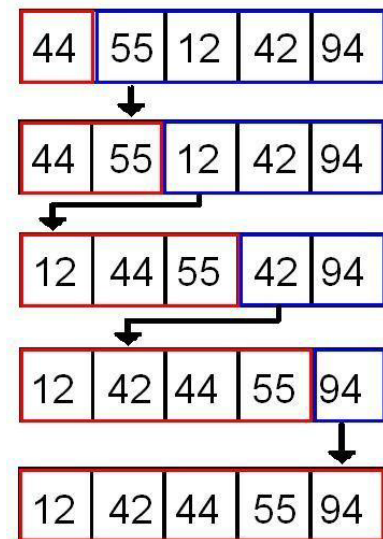
$A[J+1]:= A[J]$

$J:= J-1$

FIN\_MIENTRAS

$A[J+1]:=X$

FIN\_PARA



Observaciones: debemos controlar que  $J>0$  para no exceder el límite inferior del arreglo, cuando el elemento a insertar en el arreglo ordenado es menor a todos ellos.

Siempre se supone que el primer elemento está ordenado, para no tener que perder el tiempo comparándolo consigo mismo.

Hasta que no termina no se sabe en qué lugar quedan los elementos. Hay un recorrido de  $N-1$  vez.

### Intercambio directo (método de la burbuja)

La principal característica de este método es el intercambio entre pares de ítems. Se basa en el principio de comparar e intercambiar pares de ítems adyacentes hasta que todos estén ordenados.

Consignas de partida: Se tiene un vector desordenado de  $N$  elementos. Se comparan de a pares de elementos adyacentes. El proceso finaliza cuando en un recorrido total del arreglo no se produjeron cambios.

...Algoritmo

BANDERA:= FALSO

MIENTRAS NO BANDERA HACER

    BANDERA:= VERDADERO

    PARA J:=1 A (N-1) HACER

        SI  $A[J] > A[J+1]$  ENTONCES

$X := A[J]$

$A[J] := A[J+1]$

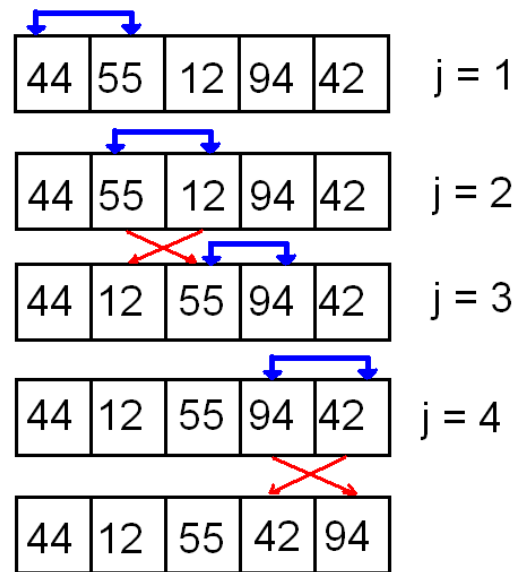
$A[J+1] := X$

        BANDERA:= FALSO

    FIN\_SI

FIN\_PARA

FIN\_MIENTRAS



Observaciones: El método realiza sólo las pasadas necesarias + una extra para verificar que el arreglo quede efectivamente ordenado. Esto lo diferencia de los otros métodos. En velocidad es peor que los otros.

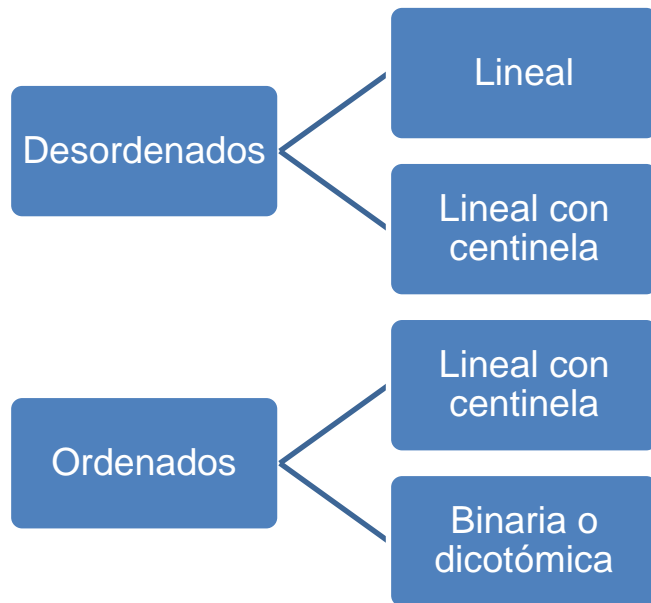
### Ordenamientos avanzados

- Heapsort: El ordenamiento por montículos, está basado en el método de selección directa.
- Quicksort: El ordenamiento más rápido, método óptimo para clasificar arreglo. Está basado en el método de intercambio directo.
- Shell: es una mejora de la ordenación por inserción, donde se van comparando elementos distantes al tiempo que se los intercambian si corresponde. A medida que se aumentan los pasos el tamaño de los saltos disminuye, por esto mismo es útil tanto como si los datos desordenados se encuentran cercanos o lejanos.

Es bastante adecuado para ordenar listas de tamaño moderado debido a que su velocidad es aceptable y su codificación es bastante sencilla.

### Búsqueda

Se clasifican según como se encuentra el arreglo



### Lineal

El arreglo  $\{A[i] \dots A[N]\}$  está desordenado o no se conoce su situación.

Se ingresa el elemento a buscar y se recorre **totalmente** el arreglo. En cada paso, se compara el elemento ingresado con el contenido en la posición  $i$  y de encontrarse la similitud se acciona según la consigna y se continúa con el recorrido hasta alcanzar la posición  $N$  del arreglo. ¿Qué ocurre si el elemento se encuentra repetido en el arreglo? El proceso encuentra **todos** y en cada ocurrencia del elemento buscado con los valores del arreglo el proceso me permite actuar. El proceso **no** finaliza hasta terminar el recorrido del arreglo.

...Algoritmo

Ingresa (dato)

$J := 0$

PARA  $i := 1$  A  $N$  HACER

SI  $(DATO = A[i])$  ENTONCES

Tratar\_Exito

$J := J + 1$

FIN\_SI

FIN\_PARA

SI  $J = 0$  ENTONCES

Tratar\_fracaso

FIN\_SI

Observaciones:

Como debemos recorrer todo el arreglo es más conveniente hacer uso de un recorrido natural.

Este el único método para búsquedas múltiples.

#### Lineal con centinela (para arreglos desordenados)

El método se basa en efectuar un recorrido manual del arreglo y en el momento de hallar lo buscado terminar con el recorrido. Es decir, se utiliza un ciclo indefinido el valor posicional se realiza mediante una variable contador.

...Algoritmo

Ingresar (dato)

X:=1

MIENTRAS (X<N) Y (dato<>A[x]) HACER

    X:=X+1

FIN\_MIENTRAS

SI dato=A[X] ENTONCES

    Tratar\_exito

CONTRARIO

    Tratar\_fracaso

FIN\_SI

Observaciones: Este método, así como el lineal suponen un caso de fracaso en la búsqueda el recorrido total del arreglo. Además, como finaliza al encontrar lo buscado, **no** permite hallar elementos repetidos. En caso de éxito, dado que no necesita recorrer totalmente el arreglo es más veloz que el lineal.

#### Lineal con centinela (para arreglos ordenados)

El método se basa en efectuar un recorrido manual del arreglo y en el momento de hallar lo buscado, o en su defecto el inmediato mayor (orden creciente) /inmediato menor (orden decreciente) y terminar con el recorrido. Nuevamente se utiliza un ciclo indefinido con la posición mediante variable contador.

...Algoritmo

Ingresar (dato)

X:=1

MIENTRAS (X<N) Y (dato<A[x]) {o dato>A[x] depende de cómo está ordenado}

    X:=X+1



```

FIN_MIENTRAS
SI dato = A[x] ENTONCES
    Tratar_exito
CONTRARIO
    Tratar_fracaso
FIN_SI

```

Observaciones: Este método solo cambia en la condicional del ciclo la desigualdad (<>) por una simple menor o mayor depende del orden igual que en su homólogo para desordenado, como finaliza al encontrar lo buscado, **no** permite hallar elementos repetidos. Al estar el arreglo ordenado es aún más veloz que su homologo desordenado.

### Búsqueda binaria o dicotómica

Este método tiene su aplicabilidad en arreglos ordenados de gran cantidad de elementos. **No** permite hallar elementos repetidos. En caso de éxito, dado que va reduciendo su área de búsqueda siempre en un 50% es más veloz que el lineal.

Proceso: se definen los valores del rango del arreglo como variables independientes (límite inferior o izquierdo y límite superior o derecho). Se calcula el valor posicional medio del arreglo como el promedio de los límites. Se verifica si el elemento buscado se encuentra en el medio del arreglo (éxito) o caso contrario se determina sobre que sub-vector se continúa la búsqueda (sub-vector de la derecha o de la izquierda).

```

...Algoritmo
INGRESAR (dato)
Izq:= 1
Der:= N
Medio:= (Izq + Der) DIV 2
MIENTRAS (Izq<Der) Y dato <> A[Medio] HACER
    SI dato<A[Medio] ENTONCES
        Der:= Medio-1
    CONTRARIO
        Izq:= Medio+1
    FIN_SI
    Medio:=(Izq + Der) DIV 2
FIN_MIENTRAS

```

SI dato = A[Medio] ENTONCES

Tratar\_exito

CONTRARIO

Tratar\_Fracaso

FIN\_SI

Observaciones: este método es rápido en forma proporcional al tamaño del vector. Cuanto más grande el vector mejor será la performance del método. Si el vector es pequeño, es más conveniente por la poca cantidad de acciones a ejecutar, el método lineal con centinela para ordenados.

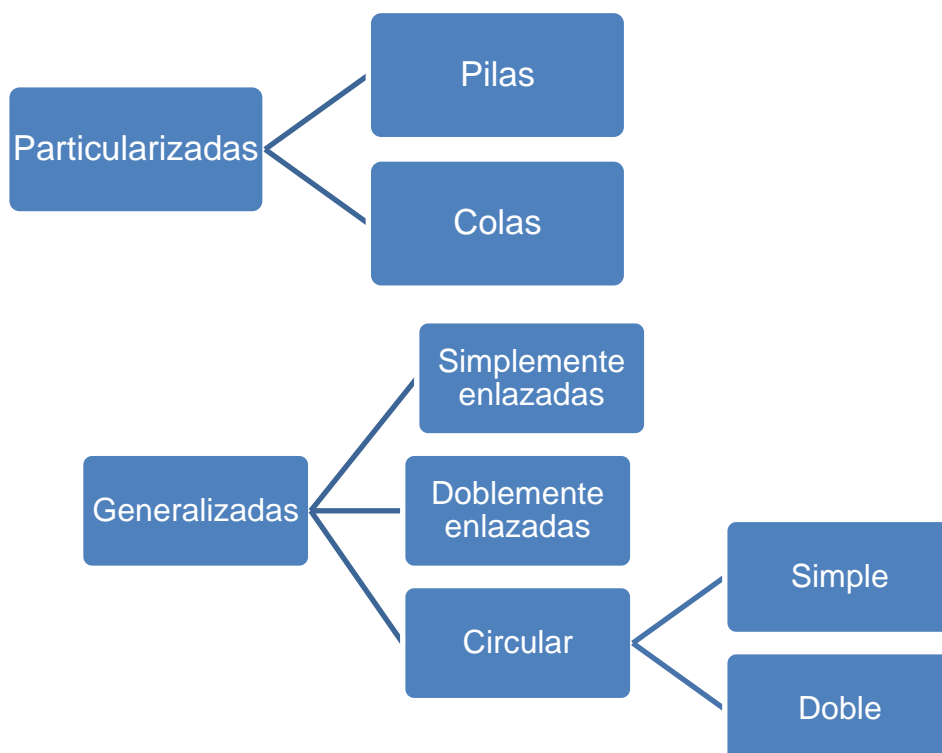
## Listas lineales

Una lista lineal es una colección, originalmente vacía, de elementos de cualquier tipo no necesariamente consecutivos en memoria, que durante la ejecución del programa puede crecer o decrecer elemento a elemento según las necesidades previstas en el mismo (estructura dinámica)

Según la definición dada surge una pregunta: si los elementos no están consecutivos en memoria, ¿Cómo pasaremos desde un elemento al siguiente cuando recorramos la lista?

La respuesta es que cada elemento debe almacenar información de donde está el siguiente elemento o el anterior, o bien ambos.

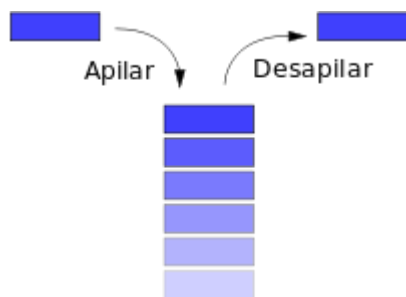
## Clasificación



En función de la información que cada elemento de la lista almacene respecto a la localización de sus antecesores y/o predecesores, las listas pueden clasificarse en: Listas simplemente enlazadas, listas circulares, listas doblemente enlazadas y listas circulares doblemente enlazadas.

## *Pilas*

Una pila (stack en inglés) es una lista ordinal o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (Last In First Out, último en entrar, primero en salir) que permite almacenar y recuperar datos. Esta estructura se aplica en multitud de ocasiones en el área de informática debido a su simplicidad y ordenación implícita de la propia estructura.



Para el manejo de los datos se cuenta con dos operaciones básicas: apilar (push), que coloca un objeto en la pila, y su operación inversa, retirar (o desapilar, pop), que retira el último elemento apilado.

En cada momento sólo se tiene acceso a la parte superior de la pila, es decir, al último objeto apilado (denominado TOS, Top of Stack). La operación retirar permite la obtención de este elemento, que es retirado de la pila permitiendo el acceso al siguiente (apilado con anterioridad), que pasa a ser el nuevo TOS.

Por analogía con objetos cotidianos, una operación apilar equivaldría a colocar un plato sobre una pila de platos, y una operación retirar a retirarlo.

Una pila típica es un área de la memoria de los computadores con un origen fijo y un tamaño variable. Al principio, el tamaño de la pila es cero. Un puntero de pila, por lo general en forma de un registro de hardware, apunta a la más reciente localización en la pila; cuando la pila tiene un tamaño de cero, el puntero de la pila apunta al origen de la pila.

Las dos operaciones aplicables a todas las pilas son:

- Una operación apilar, en el que un elemento de datos se coloca en el lugar apuntado por el puntero de pila, y la dirección en el puntero de pila se ajusta por el tamaño de los datos de partida.

- Una operación desapilar: un elemento de datos en la ubicación actual apuntado por el puntero de pila es eliminado, y el puntero de pila se ajusta por el tamaño de los datos de partida.

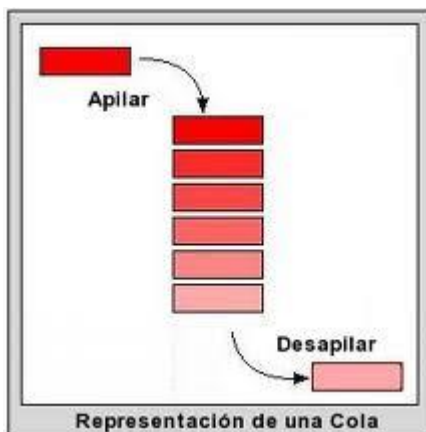
**Al tener un único punto de acceso, tanto para insertar como para extraer un elemento, siempre accederemos al último elemento insertado (el único visible), si es que la estructura no está vacía.**

En informática las pilas se usan muchísimo. Casi todos los entornos de computación de tiempo de ejecución de memoria utilizan una pila especial. Quizá la más famosa de todas sea la conocida como pila de llamadas o call stack. Gracias a la existencia de esta pila los programas pueden guardar los puntos de retorno al llamar a subrutinas. También, en varios lenguajes, se usan pilas para guardar los datos que pasamos como parámetros. Esto se hace en forma automática y transparente para nosotros, y ayuda a usar la memoria disponible con eficiencia.

Pila es una forma importante de apoyar llamadas anidadas o a funciones recursivas.

## *Colas*

Una cola (también llamada fila) es una estructura de datos, caracterizada por ser una secuencia de elementos en la que la operación de inserción push se realiza por un extremo y la operación de extracción pop por el otro. También se le llama estructura FIFO (del inglés First In First Out), debido a que el primer elemento en entrar será también el primero en salir.

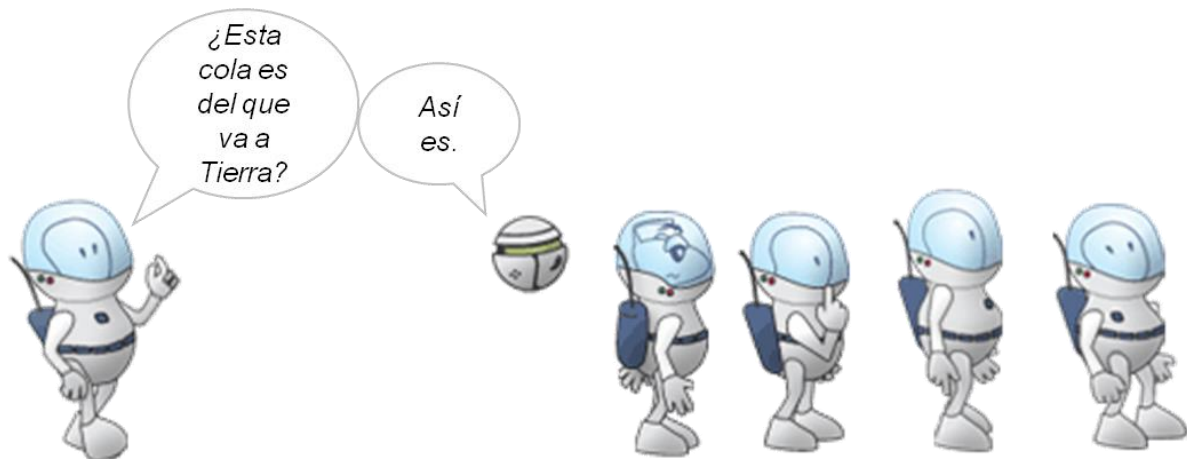


Las colas se utilizan en sistemas informáticos, transportes y operaciones de investigación (entre otros), dónde los objetos, personas o eventos son tomados como datos que se almacenan y se guardan mediante colas para su posterior procesamiento.

La particularidad de una estructura de datos de cola es el hecho de que sólo podemos acceder al primer y al último elemento de la estructura. Así mismo, los elementos sólo se pueden eliminar por el principio y sólo se pueden añadir por el final de la cola.

Ejemplos de colas en la vida real serían: personas comprando en un supermercado, esperando para entrar a ver un partido de béisbol, esperando en el cine para ver una película, una pequeña peluquería, etc. La idea esencial es que son todas líneas de espera.

Fueron pensadas para mantener y manejar elementos respetando siempre y directamente su orden de llegada. Las colas tienen un punto de inserción de elementos y otro para la extracción de estos y están en extremos opuestos (cabecera-final, frente-fondo, o primero-último).



## ***Punteros***

Un puntero es una variable de memoria que **referencia una región de memoria** en otras palabras es una variable cuyo valor es una dirección de memoria. La variable “p” de tipo puntero contiene una dirección de memoria en la que se encuentra almacenado un nodo de información (campo continente)

Definición

P: Puntero a Nodo

Nodo: Registro

Dato: (es la información a almacenar)

Prox: puntero a Nodo

Fin\_Registro

Objetivo: manejar datos alojados en la zona de memoria dinámica o heap, bien sean datos elementales, estructuras u objetos pertenecientes a una clase

La asignación dinámica de la memoria es la asignación de almacenamiento de memoria para utilización por parte de un programa de computador durante el tiempo de ejecución de ese programa.

Un objeto asignado dinámicamente permanece asignado hasta que es desasignado explícitamente, o por el programador

## Listas lineales

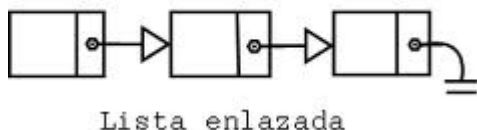
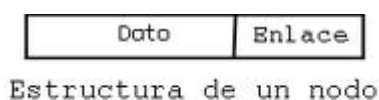
Al contrario que las pilas y las colas, las listas lineales pueden acceder a una zona de memoria de forma aleatoria, ya que cada trozo de información lleva un enlace al siguiente elemento de la cadena.

Una lista enlazada requiere una estructura de datos compleja.

Se clasifican de acuerdo con la cantidad de punteros tanto externos como internos y a la existencia o no del concepto de nulidad

### Listas lineales simplemente enlazadas

Una lista lineal simplemente enlazada es una colección de objetos (elementos de una lista), cada uno de los cuales contiene datos o un puntero a los datos, y un puntero al siguiente objeto en la colección (elemento de la lista). Gráficamente puede representarse de la forma siguiente:



Para construir una lista lineal, primero tendremos que definir el tipo de los elementos que van a formar parte de esta. Por ejemplo, cada elemento de la lista puede definirse como una estructura de datos con dos o más miembros, de los cuales uno será un puntero al elemento siguiente y el resto corresponderá con el área de datos. El área de datos puede ser de un tipo predefinido o de un tipo definido por el usuario.

El valor **NULL**, puntero nulo, permite crear listas de objetos finitas. Así mismo, suponiendo que “p” apunta al principio de la lista, diremos que dicha lista está vacía si p vale NULL.

### Operaciones con listas

Las operaciones que podemos realizar con listas incluyen fundamentalmente las siguientes:

- crear un nodo: para ello contamos con el verbo NUEVO que va acompañado por el argumento que representa al puntero externo, o al puntero auxiliar. Ejemplo: Nuevo(Prim), Nuevo(p)
- Insertar un elemento en una lista: esta operación diferirá en si queremos trabajar con la filosofía de pila o de cola o si la lista está ordenada. En las dos primeras será simplemente un agregado, pero en la tercera, implicará un recorrido hasta encontrar el lugar y luego efectuar el cambio de los enlaces.
- Buscar un elemento en una lista: esta operatoria tendrá un recorrido hasta el final, si la lista no está ordenada, o un método con centinela si la lista está ordenada; procesos muy similares a los vistos en la teoría de arreglos.
- Borrar un elemento de una lista: esto implica encontrar el elemento y en cambiar los enlaces para posteriormente aplicar el verbo BORRAR(puntero) o DISPONER(puntero)
- Recorrer los elementos de una lista: El proceso implica partir del puntero externo, el cual se lo asigna a un puntero auxiliar e ir avanzando mediante la asignación sucesiva de los punteros internos(siguietes) al puntero externo, hasta que se alcance el concepto de nulidad en el puntero.
- Borrar todos los elementos de una lista: Para esta operatoria se deberá ir posicionando en cada elemento y borrarlo en forma individual, para al final cuando la estructura nodal esté vacía recién proceder a anular el puntero externo.

Supongamos una lista lineal apuntada por “Prim”. Para insertar un elemento al principio de la lista, primero se crea el elemento y después se reasignan los punteros.

Esta operación básica nos sugiere cómo crear una lista. Para ello, y partiendo de una lista vacía, no tenemos más que repetir la operación de insertar un elemento al comienzo de una lista tantas veces como elementos deseemos que tenga dicha lista.

La secuencia de acciones sería:

Nuevo (q)

q^.siguiente := prim

q^.dato:= contenido

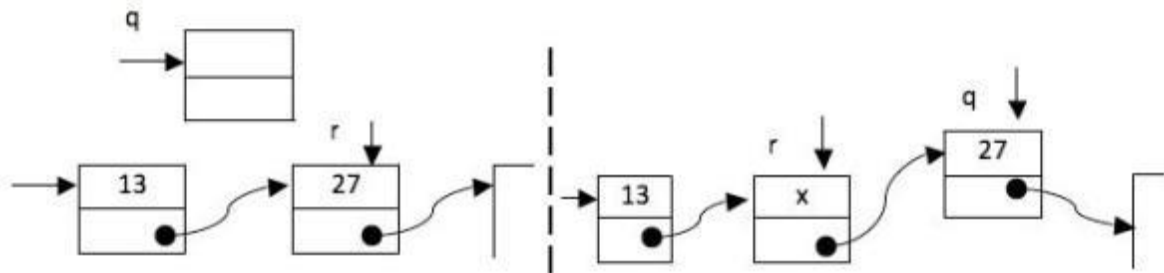
prim:= q

Para poder insertar un elemento al final de la lista (filosofía de encolar) como la dirección del puntero externo es la del primero cargado, deberemos recorrer toda la lista hasta llegar al último y luego agregar el nuevo elemento.

Esta forma de inserción se agiliza si cada vez que guardamos un elemento resguardamos su dirección, así no deberemos volver a recorrer toda la lista para los próximos agregados.

Además, esta forma mantiene el orden de carga de los elementos tal como se fueron ingresando.

Si tuviéramos que insertar un nodo apuntado por q entre los nodos cuyos contenidos son 13 y 27, y el puntero de posición r esta apuntado al valor 27, tal como lo muestra la figura:



*Inserción en la lista antes del elemento apuntado por r*

La inserción de un elemento en la lista antes de otro elemento apuntado por r, se hace insertando un nuevo elemento detrás del elemento apuntado por r, intercambiando previamente los valores del nuevo elemento y del elemento apuntado por r.

Borrar todos los elementos de una lista equivale a liberar la memoria asignada a cada uno de los elementos de esta. Supongamos que queremos borrar una lista, cuyo primer elemento está apuntado por prim.

Observa que antes de borrar el elemento individual debemos resguardar la dirección del siguiente elemento, porque si no perderíamos el resto de la lista. Y, ¿por qué perderíamos el resto de la lista? Porque se pierde la única referencia que nos da acceso a la misma.

La secuencia de operaciones es la siguiente:

```
p:= prim
mientras p <> nil hacer
    a:= p^.siguiente
    BORRAR (p)
    p:= a
fin mientras
Prim:= nil
```



## Procesos con listas<sup>5</sup>

### Carga apilada

La carga apilada significa que ingresaremos los datos en la lista de manera tal que cuando los recuperemos a través del puntero externo (cabecera), estaremos accediendo al último cargado. (recuperamos en forma inversa)

Acción Carga es

```
Prim, p: puntero a nodo;  
    Nodo: registro  
        dato: {definición del contenido}  
        prox: p;  
    fin
```

Valor: {debe ser del mismo tipo que dato}

Algoritmo

Ingresar (valor)

Prim := nil

Mientras valor <> tope hacer

    Nuevo (p)

    p^.Dato := valor

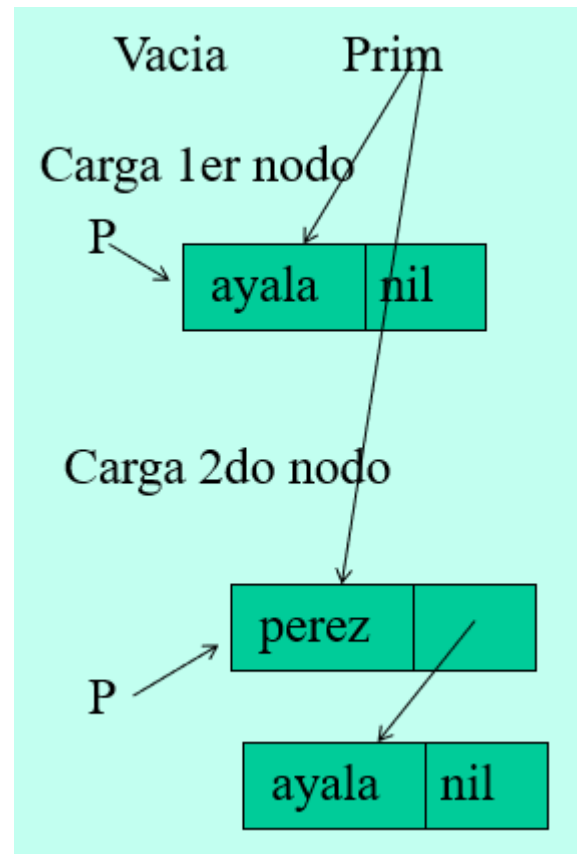
    p^.Prox := prim

    prim := p

    Ingresar (valor)

Fin mientras

Fin acción



### Carga encolada

La carga encolada significa que ingresaremos los datos en la lista de manera tal que cuando los recuperemos a través del puntero externo (cabecera), estaremos accediendo al primero cargado. (mantenemos el orden de la carga)

<sup>5</sup> Los ejemplos de los siguientes procesos son de listas simplemente enlazadas, se aplican también a los otros tipos de listas considerando las características de cada una ellas.

Acción Carga es

Prim, p, a: puntero a nodo;

Nodo: registro

dato: { definición del contenido}

prox: p;

fin

Valor: {debe ser del mismo tipo que dato}

Algoritmo

Ingresar (valor)

Prim:= nil

Mientras valor <> tope hacer

nuevo (p)

p^.Dato := valor

p^.Prox:= nil

Si prim = nil entonces

prim:= p

contrario

a^. Prox:= p

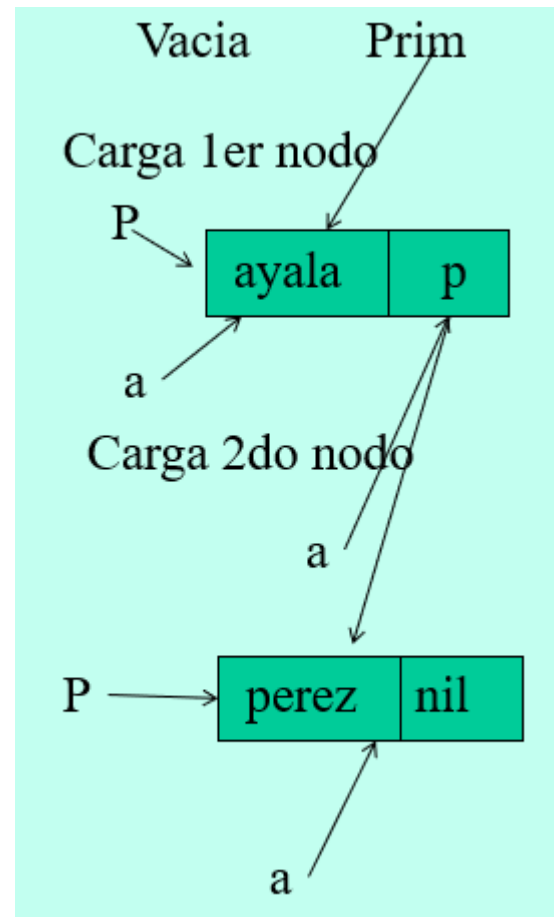
Fin si

a:= p

Ingresar (valor)

Fin mientras

Fin acción

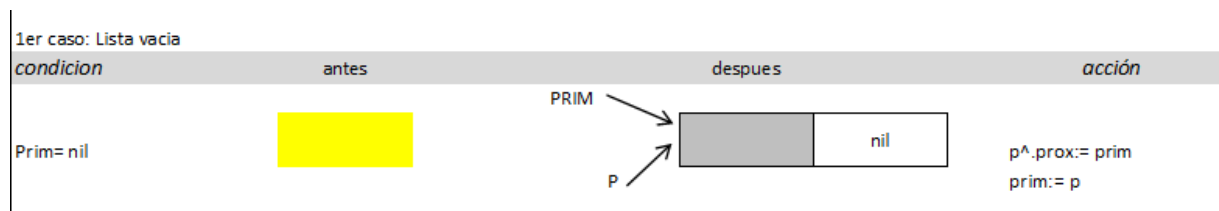


### Carga ordenada

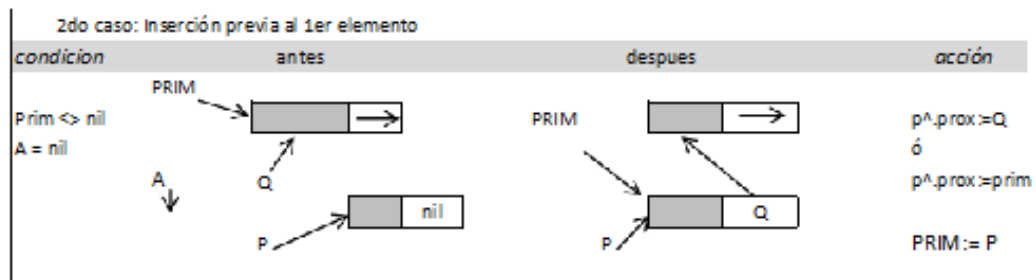
La carga ordenada implica una búsqueda de la posición donde insertar el nuevo elemento. Recuperaremos los datos, a través del puntero externo (cabecera), según el orden que hayamos establecido.

En este proceso deberemos contemplar varias situaciones:

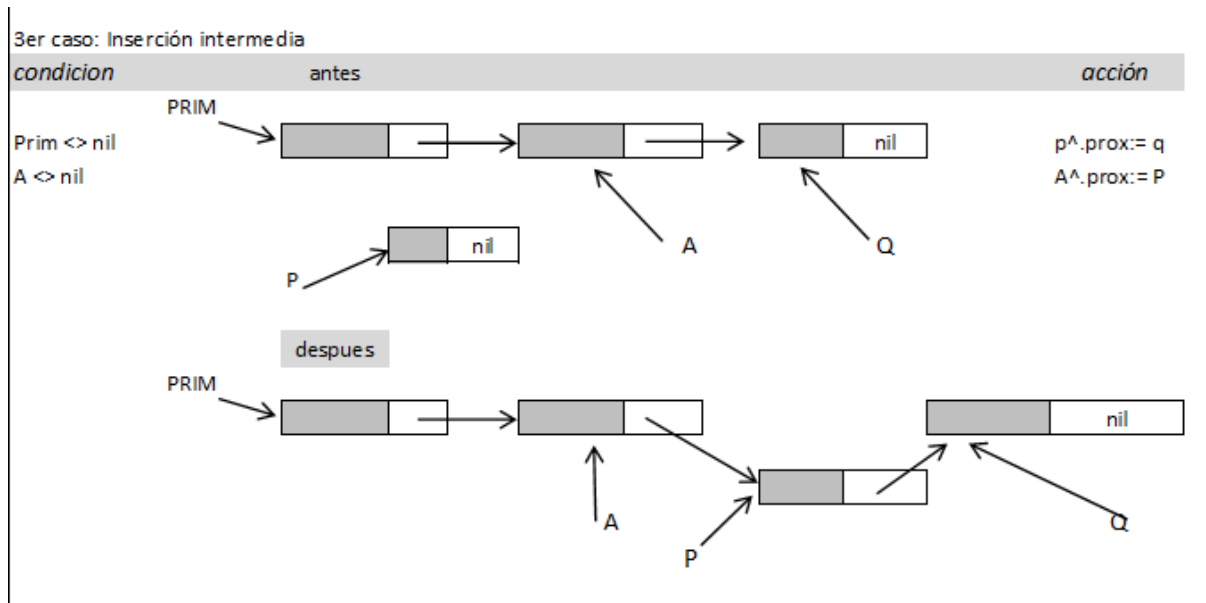
- Lista vacía



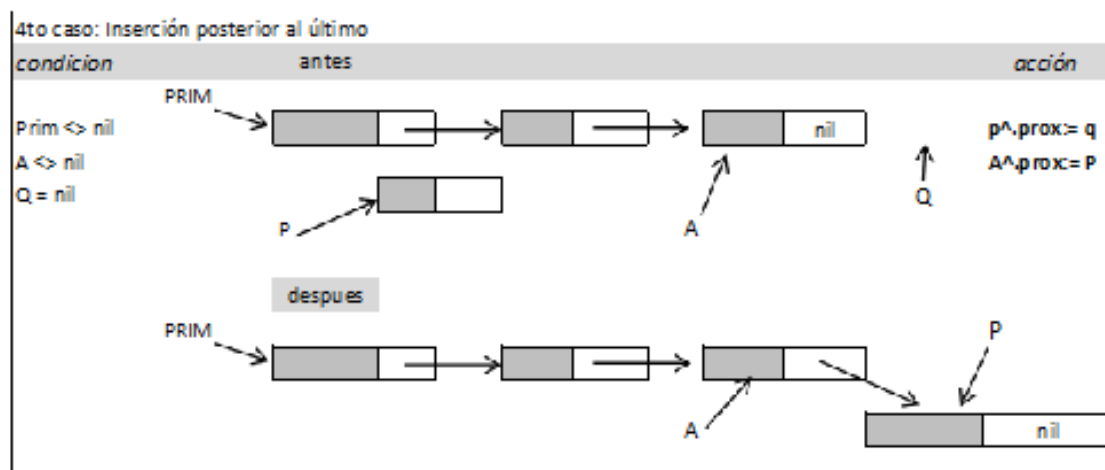
- Inserción previa al 1er nodo



- Inserción intermedia



- Inserción posterior al último nodo



Accion carga ordenada es

Ambiente

Prim, P, Q, A: puntero a nodo

Nodo: registro

dato: ....

prox: puntero a nodo

fin

Valor: (igual que dato)

Algoritmo

Prim:= nil; A:= nil;

ingresar (valor)

Mientras valor <> tope hacer

Nuevo (P)

P^.dato:= valor

Q:= prim

Mientras (Q<> nil) y (Q^.dato< valor) hacer

A:= Q

Q:= Q^.prox

Fin\_mientras

Si A=nil entonces

p^.prox:=Prim ;

Prim:= P

Contrario

p^.prox:= q

a^.prox:= p

Fin\_si

Ingresar (valor)

Fin mientras

Fin Acción

El proceso de eliminar un elemento es la lista tiene en forma similar a la carga, varias situaciones.

Previamente debemos efectuar una búsqueda y de acuerdo en donde encontremos el elemento deberemos considerar.

- Eliminar el primer elemento
- Eliminar un elemento intermedio
- Eliminar el último elemento

## Recursividad

La recursividad es un concepto fundamental en matemáticas y en computación.

Es una alternativa diferente para implementar estructuras de repetición (ciclos). Los módulos hacen llamadas recursivas.

Se puede usar en toda situación en la cual la solución pueda ser expresada como una secuencia de movimientos, pasos o transformaciones gobernadas por un conjunto de reglas no ambiguas.

### *Función recursiva*

Las funciones recursivas se componen de:

- Caso base: una solución simple para un caso particular (puede haber más de un caso base). La secuenciación, iteración condicional y selección son estructuras válidas de control que pueden ser consideradas como enunciados.

NOTA: Regla recursiva Las estructuras de control que se pueden formar combinando de manera válida la secuenciación, iteración condicional y selección también son válidos.

- Caso recursivo: una solución que involucra volver a utilizar la función original, con parámetros que se acercan más al caso base. Los pasos que sigue el caso recursivo son los siguientes:
  - El procedimiento se llama a sí mismo
  - El problema se resuelve, resolviendo el mismo problema, pero de tamaño menor
  - La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará.

Ejemplo: Factorial

Escribe un programa que calcule el factorial (!) de un entero no negativo. He aquí algunos ejemplos de factoriales:

$$0! = 1$$

$$1! = 1$$

$$2! = 2 \quad \rightarrow 2! = 2 * 1!$$

$$3! = 6 \quad \rightarrow 3! = 3 * 2!$$

$$4! = 24 \quad \rightarrow 4! = 4 * 3!$$

$$5! = 120 \quad \rightarrow 5! = 5 * 4!$$

Usando una estructura iterativa

Accion factorial es

Ingresar (n)

fact := 1

para i := 1 a n hacer

fact := i \* fact

fin para

Escribir (“factorial de “, n, “ es “, fact)

Fin acción

Ahora la estructura recursiva

Accion factorial (int n)

si n = 0 entonces

factorial:=1

contrario

factorial:=n\* factorial(n-1)

Finsi

Fin acción

El planteo formal sería:

$$N! = \begin{cases} 1 & \text{si } N = 0 \text{ (base)} \\ N * (N - 1)! & \text{si } N > 0 \text{ (recursión)} \end{cases}$$

Un razonamiento recursivo tiene dos partes: la base y la regla recursiva de construcción. La base no es recursiva y es el punto tanto de partida como de terminación de la definición.

### *¿Por qué escribir programas recursivos?*

Son más cercanos a la descripción matemática.

Generalmente más fáciles de analizar

Se adaptan mejor a las estructuras de datos recursivas.

Los algoritmos recursivos ofrecen soluciones estructuradas, modulares y elegantemente simples.


Debemos usar la recursividad para simplificar el código y cuando la estructura es recursiva, ejemplo: árboles.

NO debemos usar recursividad cuando los métodos usen arreglos largos o cuando el método cambia de manera impredecible los campos o cuando las iteraciones sean la mejor opción.


### *Partes de un algoritmo recursivo*

- ❑ Un algoritmo recursivo genera la repetición de una o más instrucciones (como un bucle).
  - Como cualquier bucle puede crear un bucle infinito.
  - Es necesario establecer una condición de salida para terminar la recursividad.
- ❑ Para evitar un bucle infinito, un algoritmo recursivo tendrá:
  - Caso trivial, caso base o fin de recursión.
    - ✓ La función devuelve un valor simple sin utilizar la recursión ( $0! = 1$ ).
  - Parte recursiva o caso general.
    - ✓ Se hacen llamadas recursivas que se van aproximando al caso base.

```
entero : función Recursiva(...)
...
inicio
...
    devolver(Recursiva(...))
...
fin_función
```



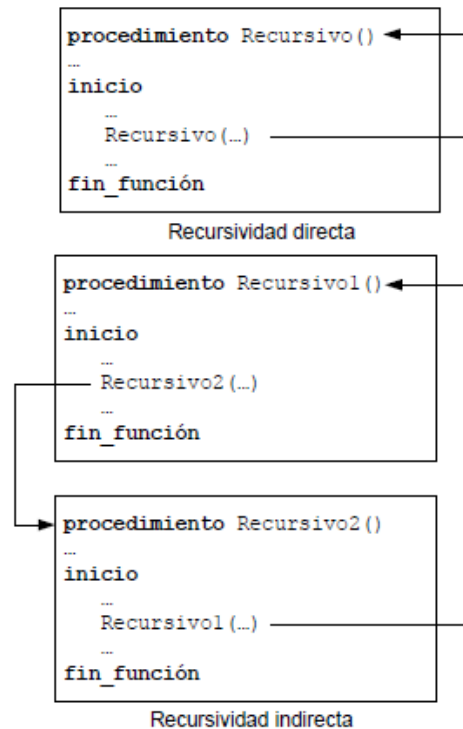
```
entero : función Recursiva(...)
...
inicio
...
    si ... entonces
        //Caso base
        devolver(...)
    si_no
        //Parte recursiva
        devolver(Recursiva(...))
    fin_si
...
fin_función
```



### *Tipos de recursividad*

❑ Según el subprograma al que se llama, existen dos tipos de recursión:

- **Recursividad simple o directa.**
  - ✓ La función incluye una referencia explícita a sí misma.
- **Recursividad mutua o indirecta.**
  - ✓ El módulo llama a otros módulos de forma anidada y en la última llamada se llama al primero.



❑ Según el modo en que se hace la llamada recursiva la recursividad puede ser:

- **De cabeza.**
  - ✓ La llamada se hace al principio del subprograma, de forma que el resto de instrucciones se realizan después de todas las llamadas recursivas.
    - Las instrucciones se hacen en orden inverso a las llamadas.
- **De cola.**
  - ✓ La llamada se hace al final del subprograma, de forma que el resto de instrucciones se realizan antes de hacer la llamada.
    - Las instrucciones se hacen en el mismo orden que las llamadas.
- **Intermedia.**
  - ✓ Las instrucciones aparecen tanto antes como después de las llamadas.
- **Múltiple.**
  - ✓ Se producen varias llamadas recursivas en distintos puntos del subprograma.
- **Anidada.**
  - ✓ La recursión se produce en un parámetro de la propia llamada recursiva.
  - ✓ La llamada recursiva utiliza un parámetro que es resultado de una llamada recursiva.



```

procedimiento f(valor entero: n)
""
inicio
  si n>0 entonces
    f(n-1)
  fin_si
instrucción A
instrucción B
fin_procedimiento

```

Recursividad de cabeza

```

procedimiento f(valor entero: n)
""
inicio
  instrucción A
  instrucción B
  si n>0 entonces
    f(n-1)
  fin_si
fin_procedimiento

```

Recursividad de cola

```

procedimiento f(valor entero: n)
""
inicio
  instrucción A
  si n>0 entonces
    f(n-1)
  fin_si
  instrucción B
fin_procedimiento

```

Recursividad de intermedia

```

procedimiento f(valor entero: n)
""
inicio
  ...
  si n>0 entonces
    f(n-1)
  fin_si
  si n<5 entonces
    f(n-2)
  fin_si
  ...
fin_procedimiento

```

Recursividad múltiple

```

entero función f(valor entero: n)
""
inicio
  ...
  si n>0 entonces
    devolver(f(n-1)+f(n-2))
  fin_si
fin_función

```

Recursividad anidada

Cuando un procedimiento incluye una llamada a sí mismo se conoce como recursión directa.

Cuando un procedimiento llama a otro procedimiento y éste causa que el procedimiento original sea invocado, se conoce como recursión indirecta.

NOTA: Cuando un procedimiento recursivo se llama recursivamente a sí mismo varias veces, para cada llamada se crean copias independientes de las variables declaradas en el procedimiento.

## ***Recursión vs Iteración***

### **Repetición**

Iteración: ciclo explícito

Recursión: repetidas invocaciones a método

### **Terminación**

Iteración: el ciclo termina o la condición del ciclo falla

Recursión: se reconoce el caso base

En ambos casos podemos tener ciclos infinitos

Hay que considerar que resulta más positivo para cada problema la elección entre eficiencia (iteración) o una buena ingeniería de software, la recursión resulta normalmente más natural.

## Árboles

Estructura de datos homogénea, no lineal y dinámica

HOMOGÉNEA porque cada elemento es del mismo tipo de dato, DINÁMICA porque tiene la posibilidad de cambiar su forma y cantidad de nodos en tiempo de ejecución y NO LINEAL porque cada elemento puede tener más de un sucesor.

Los árboles tienen una definición recursiva, un árbol es una estructura compuesta por un dato y varios árboles

ÁRBOL: es un conjunto dinámico con una estructura estrictamente jerárquica

Ejemplos:

- árbol genealógico
- estructura de apartados y sub-apartados de un libro
- estructura sintáctica de una frase

## *Conceptos básicos*

Nodo: cada uno de los elementos de un árbol

Nodo padre: el único antecesor directo de un nodo

Nodo hijo: descendiente directo de un nodo

Raíz: antecesor común de todos los nodos del árbol, que a su vez no tiene antecesores

Hoja o nodo terminal: nodo sin descendencia

Sub-árbol: conjunto formado por un nodo y todos sus descendientes

árbol nulo: árbol sin nodos

Camino: secuencia de nodos  $n_1; n_2; \dots; n_k$  tal que  $n_i$  es el nodo padre de  $n_{i+1}$ , para  $1 \leq i < k$

Longitud de un camino: número de nodos que lo forman, salvo el nodo inicial

Altura de un nodo: longitud del camino más largo de ese nodo a una hoja

Altura de un árbol: altura del nodo raíz

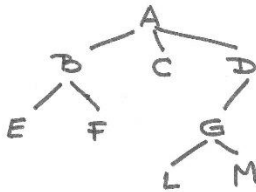
Profundidad o nivel de un nodo: longitud del camino de la raíz a ese nodo

Grado de un nodo: número de hijos de dicho nodo

Grado de un árbol: máximo de los grados de sus nodos

Un árbol es una estructura no lineal en la que cada nodo puede apuntar a uno o varios nodos. También, se suele dar una definición recursiva: un árbol es una estructura compuesta por un dato y varios árboles.

En relación con otros nodos si miramos el ejemplo



¿El nodo 'A' es padre de 'B', 'C' y 'D', además , 'L' y 'M' son hijos de 'G'?

Su respuesta:

El nodo 'A' es padre de 'B', 'C' y 'D', además, 'L' y 'M' son hijos de 'G'.

Comentario:

¡Bien!, has entendido la relación.

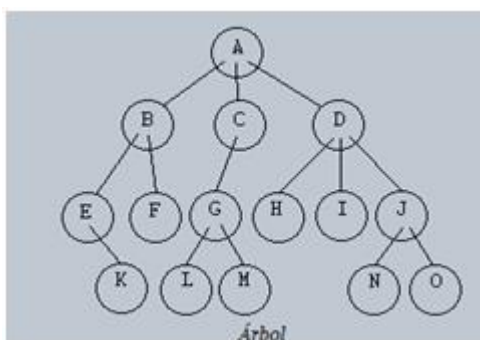
En cuanto a la posición dentro del árbol:

Nodo raíz: nodo que no tiene padre. Este es el nodo que usaremos para referirnos al árbol.

En el ejemplo, ese nodo es el 'A'.

Nodo hoja: nodo que no tiene hijos. En el ejemplo hay varios: 'F', 'H', 'I', 'K', 'L', 'M', 'N' y 'O'.

Nodo rama: aunque esta definición apenas la usaremos, estos son los nodos que no pertenecen a ninguna de las dos categorías anteriores. En el ejemplo: 'B', 'C', 'D', 'E', 'G' y 'J'.



En el ejemplo, nodo raíz es el 'A'.

Comentario:

Es correcto, A es el nodo raíz

En el ejemplo, nodos hojas hay varios: 'F', 'H', 'I', 'K', 'L', 'M', 'N' y 'O'.

Comentario:

Es correcto nodos hojas hay varios

árbol N-ARIO: es un árbol de grado  $n \geq 1$

Se define recursivamente:

1. Un árbol formado por un único nodo es un árbol n-ario
2. Un árbol n-ario se puede construir enraizando  $k \leq n$  arboles n-arios a un único nodo  $r$

El número máximo de nodos en el nivel  $i$ -ésimo de un árbol n-ario es  $n^i$

Un árbol Unario es un tipo degenerado de árbol n-ario, equivalente a una lista.

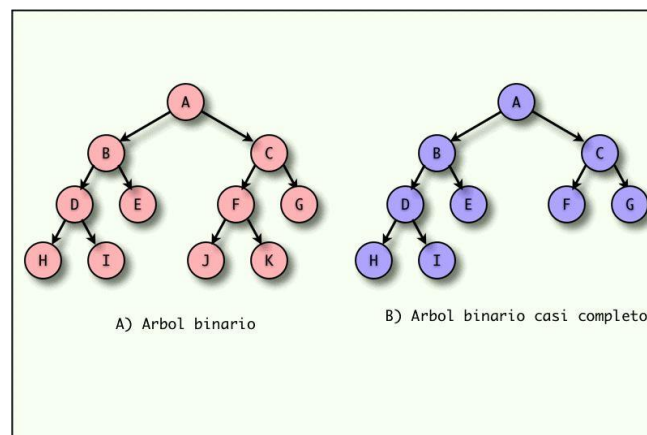
En particular nos interesan los árboles binarios.

Ésta es una estructura muy usada en numerosas aplicaciones.

Su nombre y demás terminología afín proviene de los árboles de la botánica, pero sobre todo de los árboles genealógicos.

Un árbol binario puede adoptar una de estas dos formas: o bien es un árbol vacío o bien es un nodo que, además de un elemento, tiene dos subárboles (de ahí el vocablo “binario”).

Así, un árbol no vacío tiene una raíz (el elemento), un subárbol izquierdo y un subárbol derecho.



## ***Clasificación de árboles binarios***

Existen cuatro tipos de árbol binario:

### **A. B. DISTINTO**

Se dice que dos árboles binarios son distintos cuando sus estructuras son diferentes.

Ejemplo:



#### A. B. SIMILARES

Dos árboles binarios son similares cuando sus estructuras son idénticas, pero la información que contienen sus nodos es diferente. Ejemplo:



#### A. B. EQUIVALENTES

Son aquellos árboles que son similares y que además los nodos contienen la misma información. Ejemplo:



#### A. B. COMPLETOS

Son aquellos árboles en los que todos sus nodos excepto los del último nivel, tiene dos hijos; el subárbol izquierdo y el subárbol derecho

Para calcular cuántos nodos serán necesarios como máximo almacenar, se utiliza la siguiente fórmula:  $N = 2^h - 1$ , siendo h el número de niveles que posee el árbol

### ***Arboles de Expresión***

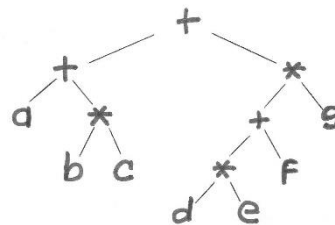
Son aquellos árboles binarios en donde las hojas son operandos, como constantes o nombres de variables y los nodos contienen operadores.

Este árbol en particular debe ser binario porque todas sus operaciones son binarias. Hay que tener en cuenta que un nodo puede tener un solo hijo.

ejemplo:  $(a + b * c) + ((d * e + f) * g)$

En estos tipos de árboles podemos efectuar tres evaluaciones (recorridos) que originan notaciones denominadas:

- infija (recorrido en-orden) permite visualizar la expresión dada sin los separadores: a, +, b, \*, c, +, d, \*, e, +, f, \*, g
- prefija (recorrido en pre-orden) permite visualizar la expresión dando prioridad a los operadores: +, +, a, \*, b, c, \*, +, \*, d, e, f, g
- posfija (recorrido en pos-orden) permite visualizar la expresión dando prioridad a los operandos: a, b, c, \*, +, d, e, \*, f, +, g, +, +



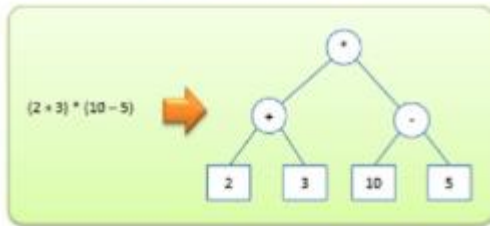
### ***Recorrido de un árbol binario***

Hay tres maneras de recorrer un árbol: en-orden, pre-orden y post-orden. Cada una de ellas tiene una secuencia distinta para analizar el árbol como se puede ver a continuación:

- EN-ORDEN
  1. Recorrer el subárbol izquierdo en en-orden.
  2. Examinar la raíz.
  3. Recorrer el subárbol derecho en en-orden.
- PRE-ORDEN
  1. Examinar la raíz.
  2. Recorrer el subárbol izquierdo en pre-orden.
  3. Recorrer el subárbol derecho en pre-orden.
- POST-ORDEN
  1. Recorrer el subárbol izquierdo en post-orden.

2. Recorrer el subárbol derecho en post-orden.
3. Examinar la raíz.

Si para la siguiente expresión y su representación:



Se la evaluara de la siguiente forma:

<p>PASO 1:</p>	<p>PASO 2:</p>
<p>PASO 3:</p>	<p>¿QUE TIPO DE RECORRIDO ESTÁ IMPLICITO?</p> <p>pre-orden, en-orden o post-orden</p> <p>escribi todo con minúsculas y el guion en el medio.</p>

Su respuesta :

post-orden

Comentario:

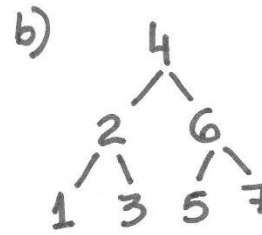
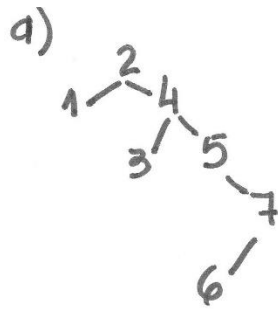
MUY BIEN!!

## Árboles binarios de búsqueda (ABB)

Como su nombre lo indica son árboles binarios cuya información ha sido organizada de manera de facilitar la búsqueda de un elemento cualquiera que pueda estar almacenado en el árbol. Para que un árbol binario sea un ABB debe cumplirse, para todo nodo del árbol, que todos los elementos alojados en el hijo izquierdo sean menores que el alojado en el propio nodo, y que éste a su vez sea menor que todos los alojados en el hijo derecho.

elementos izquierda < raíz < elementos derecha

ejemplos: tanto el ejemplo a como el b son correctos.



Veamos algo muy curioso de este tipo de árbol....

¿Los recorridos de las dos representaciones son distintos?

Su respuesta:

No, son iguales.

Comentario:

Bien, ahora deberías preguntarte porque ocurre esto.

### Propiedad de los ABB

Esta interesante propiedad de que si se listan los nodos del ABB en orden nos da la lista de nodos ordenada, permite definir un método de ordenación similar al Quicksort con el nodo raíz jugando un papel similar al del elemento de partición del Quicksort, aunque con los ABB hay un gasto extra de memoria mayor debido a los punteros.

La propiedad de ABB hace que sea muy simple diseñar un procedimiento para realizar la búsqueda. Para determinar si  $k$  está presente en el árbol la comparamos con la clave situada en la raíz " $r$ ". Si coinciden la búsqueda finaliza con éxito, si  $k < r$  es evidente que  $k$ , de estar presente, ha de ser un descendiente del hijo izquierdo de la raíz, y si es mayor será un descendiente del hijo derecho.

Hay que notar la diferencia entre este procedimiento y el de búsqueda binaria. En éste podría pensarse en que se usa un árbol binario para describir la secuencia de comparaciones hecha por una función de búsqueda sobre el vector.

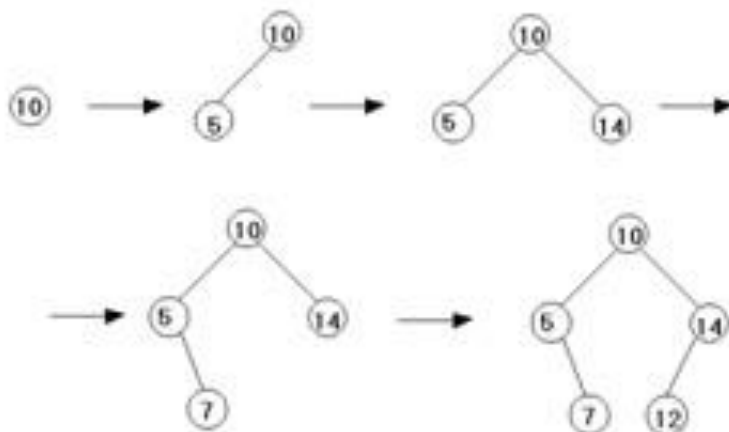
En cambio, en los ABB se construye una estructura de datos con registros conectados por punteros y se usa esta estructura para la búsqueda.

El procedimiento de construcción de un ABB puede basarse en un procedimiento de inserción que vaya añadiendo elementos al árbol. Tal procedimiento comenzaría mirando si el árbol es vacío y de ser así se crearía un nuevo nodo para el elemento insertado devolviendo como árbol resultado un puntero a ese nodo.



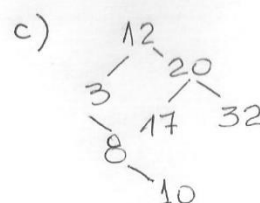
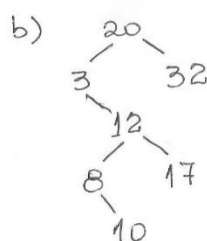
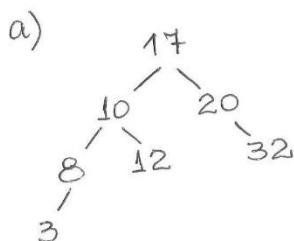
Si el árbol no está vacío se busca el elemento a insertar (con la regla de búsqueda arriba mencionada) sólo que al encontrar un puntero NULL durante la búsqueda, se reemplaza por un puntero a un nodo nuevo que contenga el elemento a insertar.

Supongamos que queremos construir un ABB a partir del conjunto de enteros {10,5,14,7,12} aplicando reiteradamente el proceso de inserción. El resultado es el que muestra la figura.



Si se deseara construir un árbol ABB con los siguientes datos {20,3,12,17,8,10,32}

¿Cuál de las figuras es la correcta?, teniendo en cuenta el orden en que fueron ingresando los datos.



Su respuesta:

opción b)

Comentario:

Bien, es la correcta para esa entrada de datos.

Si recorriéramos las tres representaciones anteriores, en sentido en orden.

Su respuesta:

Los tres recorridos son iguales por la propiedad de los ABB, y solo son distintas sus representaciones por el orden de las entradas de los datos.

Comentario:

MUY BIEN

## *Arboles AVL*

Un árbol AVL es un árbol binario de búsqueda que cumple con la condición de que la diferencia entre las alturas de los subárboles de cada uno de sus nodos es, como mucho 1.

La denominación de árbol AVL viene dada por los creadores de tal estructura (Adelson-Velskii y Landis).

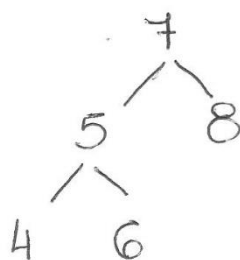
Recordamos que un árbol binario de búsqueda es un árbol binario en el cual cada nodo cumple con que todos los nodos de su subárbol izquierdo son menores que la raíz y todos los nodos del subárbol derecho son mayores que la raíz.

La propiedad de equilibrio que debe cumplir un árbol para ser AVL asegura que la profundidad del árbol sea  $O(\log(n))$ , por lo que las operaciones sobre estas estructuras no deberán recorrer mucho para hallar el elemento deseado.

Como se verá (unidad de complejidad), el tiempo de ejecución de las operaciones sobre estos árboles es, a lo sumo  $O(\log(n))$  en el peor caso, donde  $n$  es la cantidad de elementos del árbol.

Sin embargo, y como era de esperarse, esta misma propiedad de equilibrio de los árboles AVL implica una dificultad a la hora de insertar o eliminar elementos: estas operaciones pueden no conservar dicha propiedad.

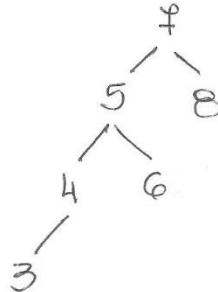
Figura1. Árbol AVL de enteros



A modo de ejemplificar esta dificultad, supongamos que al árbol AVL de enteros de Figura 1 le queremos agregar el entero 3. Si lo hacemos con el procedimiento normal de inserción de árboles binarios de búsqueda el resultado sería el árbol de Figura 2 el cual ya no cumple con la

condición de equilibrio de los árboles AVL dado que la altura del subárbol izquierdo es 3 y la del subárbol derecho es 1.

Figura 2. Árbol que no cumple con la condición de equilibrio de los árboles AVL.



Esto obligará a un proceso (una serie de rotación de los nodos), para poder restaurar la propiedad de equilibrio de un árbol AVL luego de agregar o eliminar elementos.

A los efectos de evitar este proceso de restauración, y para conseguir esta propiedad de equilibrio, la inserción y el borrado de los nodos se ha de realizar de una forma especial. Los árboles AVL más profundos son los árboles de FIBONACCI.

### ***Definición de la altura de un árbol***

Sea  $T$  un árbol binario de búsqueda y sean  $T_i$  y  $T_d$  sus subárboles, su altura  $H(T)$ , es:

0 si el árbol  $T$  contiene solo la raíz

$1 + \max(H(T_i), H(T_d))$  si contiene más nodos

Definición formal de árbol AVL

Un árbol vacío es un árbol AVL

Si  $T$  es un árbol no vacío y  $T_i$  y  $T_d$  sus subárboles, entonces  $T$  es AVL si y solo si:

$T_i$  es AVL

$T_d$  es AVL

$|H(T_i) - H(T_d)| \leq 1$

## ***Factor de equilibrio***

Cada nodo, además de la información que se pretende almacenar, debe tener los dos punteros a los árboles derecho e izquierdo, igual que los árboles binarios de búsqueda (ABB), y además el dato que controla el factor de equilibrio.

El factor de equilibrio es la diferencia entre las alturas del árbol derecho y el izquierdo:

$FE = \text{altura subárbol derecho} - \text{altura subárbol izquierdo};$

Por definición, para un árbol AVL, este valor debe ser -1, 0 ó 1.

Si el factor de equilibrio de un nodo es:

0 -> el nodo está equilibrado y sus subárboles tienen exactamente la misma altura.

1 -> el nodo está desequilibrado y su subárbol derecho es un nivel más alto.

-1 -> el nodo está desequilibrado y su subárbol izquierdo es un nivel más alto.

Si el factor de equilibrio  $|Fe| \geq 2$  es necesario reequilibrar.

## **Complejidad logarítmica**

Un algoritmo será más “eficiente” comparado con otro, siempre que consuma menos recursos, como el tiempo y espacio de memoria necesarios para ejecutarlo.

Para nuestro análisis tendremos en cuenta el coste (tiempo y recursos) que consume un algoritmo para encontrar la solución, ofreciéndonos la posibilidad de comparar distintos algoritmos que resuelven un mismo problema.

El uso eficiente de los recursos suele medirse en función de dos parámetros: el espacio, es decir, la memoria que utiliza, y el tiempo, lo que tarda en ejecutarse. Ambos representan los costes que supone encontrar la solución al problema planteado mediante un algoritmo. Además, dichos parámetros van a servir para comparar los algoritmos entre sí, permitiendo determinar el más adecuado entre varios para la solución de un problema en particular. Solo tomaremos en cuenta la eficiencia temporal para el desarrollo del análisis.

El tiempo de ejecución de un algoritmo va a depender de diversos factores, como son: los datos de entrada que le son suministrados, la calidad del código generado por el compilador para crear el programa, la naturaleza y rapidez de las instrucciones máquina del procesador concreto que ejecute el programa, y la complejidad intrínseca del algoritmo.

La unidad de tiempo a la que debe hacer referencia el estudio del tiempo no puede ser expresada en segundos o en otra unidad de tiempo concreta, pues no existe un ordenador

estándar al que puedan hacer referencia todas las medidas. Denotaremos por  $T(n)$  el tiempo de ejecución de un algoritmo para una entrada de tamaño  $n$ .

Teóricamente  $T(n)$  debe indicar el número de instrucciones ejecutadas por un ordenador idealizado. Debemos buscar, por tanto, medidas simples y abstractas, independientes del ordenador a utilizar. Para ello, es necesario acotar de alguna forma la diferencia que se puede producir entre distintas implementaciones de un mismo algoritmo, ya sea del mismo código ejecutado por dos máquinas de distinta velocidad, como de dos códigos que implementan el mismo método. Esta diferencia es la que acota el siguiente principio:

**Principio de Invarianza:** Dado un algoritmo y dos implementaciones suyas  $I_1$  e  $I_2$ , que tardan  $T_1(n)$  y  $T_2(n)$  segundos respectivamente, el Principio de Invarianza afirma que existe una constante real  $c > 0$  y un número natural  $n_0$  tales que para todo  $n \geq n_0$  se verifica que  $T_1(n) \leq cT_2(n)$ .

Es decir, el tiempo de ejecución de dos implementaciones distintas de un algoritmo dado no va a diferir más que en una constante multiplicativa.

Suelen estudiarse tres casos para un mismo algoritmo: caso peor, caso mejor y caso medio.

El caso mejor corresponde a la traza (secuencia de sentencias) del algoritmo que realiza menos instrucciones. Análogamente, el caso peor corresponde a la traza del algoritmo que realiza más instrucciones. El caso medio es el cociente entre la suma de los tiempos del caso mejor y peor y dos.

A la hora de medir el tiempo, siempre lo hacemos en función del número de operaciones elementales que realiza dicho algoritmo, entendiendo por operaciones elementales (en adelante OE) aquellas que el ordenador realiza en tiempo acotado por una constante. Así, consideraremos OE a las operaciones aritméticas básicas, asignaciones a variables de tipo predefinido por el compilador, los saltos (llamadas a funciones y procedimientos, retorno desde ellos, etc.), las comparaciones lógicas y el acceso a estructuras indexadas básicas, como los vectores y matrices. Cada una de ellas contabilizará como 1 OE. El tiempo de ejecución de un algoritmo va a ser una función que mide el número de operaciones elementales que realiza el algoritmo para un tamaño de entrada dado.

Ingresar (Suma)	(* 1 *)
Mientras Suma > 0 hacer	(* 2 *)
Si (Suma MOD 2 = 0) ^ (Suma < 100) entonces	(* 3 *)
Escribir('Número Válido')	(* 4 *)
Sino	-
Escribir('Número No Válido')	(* 5 *)

Fsi

Suma:= Suma - 1

-  
(\* 6 \*)

FinMientras

La línea 1 tiene 1 OE.

La línea 2 tiene 2 OE, por el mientras y la comparación.

La línea 3 tiene 4 OE, es la operación de MOD, la comparación de ese resultado con cero, la operación “^” y por último la comparación entre Suma y 100.

La línea 4 y 5 tienen 1 OE cada uno.

La línea 6 tiene 2 OE, pues es la asignación y la operación de resta.

Las Cotas de Complejidad son los valores máximos y mínimos que tarda un algoritmo en ejecutarse, es decir, el tiempo máximo y el tiempo mínimo de un algoritmo. En el ejemplo anterior: El tiempo menor y por tanto la cota inferior será 3, pues la acción de ingresar toma 1T y la comparación que no se cumple del Mientras para que no se ejecuten las acciones siguientes. El tiempo peor, o sea, la cota superior será 1 + la sumatoria desde “Suma” hasta n-1 de (2+7) +2, El primer uno es del ingreso de la variable, luego las operaciones del ciclo del mientras se irán ejecutando desde el valor que tenga “Suma” hasta Suma-1, o para no confundir, tomamos “n” como el valor inicial de “Suma”, las operaciones del ciclo son 7 más la de comparar “Suma” con cero en el mientras, y se le suma un dos más al final por la última comparación que realiza el ciclo para terminar. En otras palabras, sería:  $1 + 9 \cdot \text{Suma} + 2$ .

## Anexo 1: Taller Pascal

### *¿Qué es un editor de texto? ¿Qué es una IDE?*

Editor de Texto:

Es un programa.

Permite crear y modificar archivos de “Texto Plano”.

IDE:

Entorno de desarrollo integrado.

Podríamos decir que es un: “Editor de textos con esteroides”.

Posee las características de: Auto completado, sombreado de palabras claves, etc.

Algunas IDEs poseen un compilador o intérprete incorporado.

Ejemplos:

Sublime Text, Visual Studio, Eclipse, Notepad++, Geany, Atom, VIM, etc...

### *¿Qué es un Compilador? ¿Qué es FPC?*

Compilador:

Es un traductor, que convierte el código fuente en un código de máquina.

El archivo generado puede ser ocupado en arquitecturas similares.

Ejemplo:

FreePascal, DevPascal, TurboPascal, etc.

FPC:

Es un compilador de Pascal.

Traduce archivos “.pas” a un archivo ejecutable para el sistema operativo.

### *Estructura de Pascal*

El orden de la Estructura en pascal es la siguiente:

- Nombre del programa (program name)
- Importando librerías (uses CRT)
- Declaración de los Tipos de datos (Type)
- Declaración de las Constantes (Const)
- Declaración de las Variables (Var)
- Funciones y Procedimientos (Function y Procedures)
- Llamada principal del sistema (Begin ... end. )

```

program {Nombre del programa}
uses {Nombres de las librerías que se van a ocupar separados por coma}
const {Constantes Globales que se ocuparan}
var {Variables Globales que se van a ocupar}
function {Declaración de las funciones, si las hubiera}
{ Variables Locales de la función }
begin
...
end;

procedure { Declaración de los Procedimientos, si los hubiera}
{ Variables Locales del procedimiento }
begin
...
end;

begin { Comienzo del programa principal}
...
end. { Fin del programa principal }

```

## ***Comentarios en pascal***

Los comentarios siempre son útiles en la programación.

Sirven para que otras personas puedan entender el código, como así también para los mismos programadores (después de 3 meses...)

En Pascal podemos hacer comentarios de las siguientes maneras:

- // esto es para comentar una sola línea.
- { esto también es una línea comentada }
- {\* esto es para hacer un bloque de comentarios en pascal \*}

## ***Acciones Simples***

Palabras reservadas en pascal

and	array	begin	case	const
-----	-------	-------	------	-------



div	do	downto	else	end
file	for	function	<del>goto</del>	if
in	<del>label</del>	mod	nil	not
of	or	packed	procedure	program
record	repeat	set	then	to
type	until	var	while	with

### Asignación

Al igual que en pseudocódigo ocupamos el símbolo :=

a := 5;

//la variable a ahora tiene el valor 5.

### Escribir en la consola:

Para escribir por consola se puede ocupar la acción:

write('algún texto');

Si queremos que se realice un salto de línea después de lo que ingresamos podemos ocupar

writeln('algún texto');

También se puede escribir variables, o realizar operaciones:

writeln('Sumando 2 con 5, el resultado es: ', 2 + 5);

### Leer desde la consola:

Para leer desde el teclado podemos ocupar:

read(variables) o readln(variables);

Se puede leer 1 o más variables

Se recomienda usar el readln(variable);

## ***Tipos de Datos Básicos en Pascal***

Tipos de datos numéricos a utilizar:

Nombre	Menor Valor	Mayor Valor
shortint	-128	127
integer	-32768	32767
longint	-2147483648	2147483647
real	$2,9 \times 10^{(-39)}$	$1,7 \times 10^{(38)}$
double	$5,0 \times 10^{(-307)}$	$1,7 \times 10^{(307)}$

Otros tipos de datos:

boolean: Puede tomar dos valores: True (verdadero) o False (falso)

char: almacena un solo carácter cualquiera (ej.: 'a', 'A', '4', '@', ...)

string: almacena una cadena de caracteres (por defecto hasta 255) podemos definir un

string dándole la longitud que necesitemos (hasta 255):

a : string[10];

b : string[56];

c : string;

d : string[256]; esto sería un error

## ***Operadores en pascal***

Operadores Aritméticos:

Tomando: a := 5; b := 8

Operador	Descripción	Uso
+	Suma dos variables	a + b da: 13

-	Resta de la primera variable el contenido de la segunda	a - b da: -3
*	Multiplica las dos variables	a * b da: 40
/	Divide la variable por la segunda	b / a da: 1,6
%	Módulo de la división entera	8 % 5 da: 3

#### Operadores Relacionales:

Tomando: a := 5; b := 8

Operador	Descripción	Uso
=	Operador de igualdad	a = b da: falso
<>	Operador de distintos	a <> b da: verdadero
> o >=	mayor o mayor igual que	a > b da: falso
< o <=	menor o menor igual que	a < b da: verdadero

#### Operadores lógicos:

Tomando a := verdadero y b := falso:

Operador	Descripción	Uso
----------	-------------	-----

and	operador lógico y	a and b da: falso
or	operador lógico o	a or b da: verdadero
not	operador de negación	not b da: verdadero

#### Precedencia de los operadores en Pascal

Operador	Precedencia
not	Primero
* / div mod and	
+ - or	
= <> < <= > >=	Último

### ***Bloques en Pascal***

Los bloques en pascal comienzan con un begin y termina con un end

begin

// esto es el contenido de un bloque

end;

Existen algunos casos (como en el case, en el repeat o en los bloques de una única línea) donde no es necesario ocupar esto. Lo vamos a ver más adelante

### ***Condicionales***

Condicional simple en Pseudocodigo

si condición entonces

// código por la condición verdadera

fin si

### Condicional simple en Pascal

```
if condición then  
begin  
// código por la condición verdadera  
end;
```

Al igual que con pseudocódigo, en pascal para la condición podemos ocupar variables, operadores relacionales u operadores lógicos.

Por ej.:  $A = B$ ,  $A < (B+5)$ , bandera, etc.

En pascal si el código que vamos a realizar tiene una única línea podemos no armar el bloque con begin y end. (vemos el ejemplo más adelante)

### Condicional Alternativo en Pseudocódigo

```
si condición entonces  
// código por la condición verdadera  
sino  
// código por la condición verdadera  
fin si
```

### Condicional Alternativo en Pascal

```
if condición then  
begin  
// código por la condición verdadera  
end; //No es necesario el ;  
else  
begin  
// código por la condición verdadera  
end;
```

Al igual que en el condicional simple, si los bloques de código son de una única línea, no es necesario armar el bloque.

```
if i >= 0 then  
begin  
writeln('es positivo!');  
end  
else
```

```
begin
writeln('es negativo!');
end;
```

```
if i >= 0 then
writeln('es positivo!')
else
writeln('es negativo!');
```

#### Condicional Múltiple en Pseudocódigo

según variable hacer

```
  op1: //código para opción 1
  op2: //código para opción 2
  op3: //código para opción 3
otro
  // código para opción no presente
fin según
```

#### Condicional Múltiple en Pascal

```
case variable of
  op1: //código para opción 1
  op2: //código para opción 2
  op3: //código para opción 3
else
  // código para opción no presente
```

end;

#### Ejemplo

```
case x of
  -100..-1: writeln('el numero es menor a 0');
  0: begin
    writeln('Puedo hacer otra cosa!');
    writeln('el numero es igual a 0');
  end;
  1..100: writeln('el numero es mayor a 0');
else
```

```
writeln('el numero esta fuera del rango');  
end;
```

## *Ciclos*

Pre-test en Pseudocodigo

Mientras CONDICIÓN(variable) hacer

    //código a iterar...

    //evento que realiza (o no) el cambio de la variable...

Fin Mientras

pre-test en Pascal

while CONDICIÓN(variable) do

begin

    //código a iterar...

    //evento que realiza (o no) el cambio de la variable...

end;

Post-test en Pseudocodigo

Repetir

    //evento que realiza (o no) el cambio de la variable...

    //código a iterar...

Hasta que CONDICIÓN(variable)

Post-test en Pascal

repeat

    //evento que realiza (o no) el cambio de la variable...

    //código a iterar...

until CONDICIÓN(variable);

Ejemplo pre-test

y := 10

while y > 0 do

begin

    write(y, ' ');

    y := y - 1;

end;

Ejemplo post-test

```

y := 10
repeat
  y := y - 1;
  write(y, ' ');
until y <= 0;

```

Manejado por contador en Pseudocodigo

```

Para i := inicio hasta fin hacer
  //código a iterar...
Fin Para

```

Manejado por contador en Pascal

```

for i := inicio [to|downto] fin do
begin
  //código a iterar...
end;

```

Ejemplo

```

for i := 10 downto 1 do
begin
  writeln(i);
end;

```

```

for i := 1 to 10 do
begin
  writeln(i);
end;

```

## ***Subacciones***

En Pascal tenemos 2 tipos de “subacciones”:

Funciones (function)

Procedimientos (procedures)

La diferencia es:

Las Funciones devuelven un valor, como por ejemplo readKey() que lee la tecla que se apretó y devuelve cuál es la tecla,



Los Procedimientos no devuelven un valor, como por ejemplo `writeln()` que escribe por pantalla un valor que se le paso como parámetro.

#### Parámetro:

Los parámetros son variables y/o constantes para pasar datos entre programas y subprogramas en ambos sentidos.

Los parámetros que se utilizan en la llamada o invocación al subprograma se denominan parámetros actuales, reales o argumentos, y son los que entregan la información al subprograma.

Los parámetros que la reciben en el subprograma se denominan parámetros formales o ficticios y se declaran en la cabecera del subprograma.

#### Pase de variables:

Por valor: Se genera una copia de la variable en la subacción, de esta forma, al modificar dicha variable dentro de la subacción no sufre cambios la variable original.

Por referencia: Dentro de la subacción tenemos la variable que fue ocupada, cualquier modificación de la variable dentro de la subacción repercute en la variable.

En Pascal todas las variables son pasadas por valor, si necesitamos una variable que su paso sea por referencia, tendremos que usar la palabra reservada **var** en la definición del parámetro de la subacción.

## Procedimientos

El esqueleto de un Procedimiento en Pascal es la siguiente:

```
procedure Nombre(Parametro1: type1; Parametro2: type2; ...);  
var (Variables Locales);  
  
begin  
    ...  
    // CÓDIGO  
    ...  
end;
```

Ahora con un ejemplo, esta función realiza un intercambio del valor de la variable de x a la de y.

```

procedure cambiar(var x, y: integer);
var
    aux: integer;
begin
    aux := x;
    x:= y;
    y := aux;
end;

```

¿Como se llama a este procedimiento en nuestro código?

```
cambiar(a, b);
```

## Funciones

El esqueleto de una función en Pascal es la siguiente:

```

function Nombre(Parametro1: tipo1; Parametro2: tipo2; ...):
Tipo_de_parametro_devuelto;
var (Variables Locales);

begin
    ...
    // CÓDIGO
    ...
    Nombre := resultado;
end;

```

Ahora con un ejemplo, esta función toma 2 número (enteros) y devuelve el mayor.

```

function maximo(n1, n2: integer): integer;
begin
    if n1 > n2 then
        maximo := n1
    else
        maximo := n2;
end;

```

¿Como se llama a esta función en nuestro código?

```
begin
```

```

a:= 14;
b:= 100;
writeln(maximo(a, b));
end.

```

Ejemplo:

Crear una “subacción” que permita Validar los datos que el usuario ingresa (en este caso simplemente ver que este dentro del rango solicitado).

```

function validador(tipo: string; min, max: integer): integer;
var
    aux: integer;  salir: boolean;
begin
    salir := false;
repeat
    writeln('Ingrese ', tipo, ' que esté entre ', min, ' y ', max);
    readln(aux);
    if (min > aux) or (max < aux) then
        writeln('Valor ingresado fuera del rango solicitado.')
    else
        salir := true;
until salir;
validador:= aux;
end;

```

Como se llama en el algoritmo:

```

edad:= validador('edad', 4, 140);
peso:= validador('peso', 10, 300);
altura:= validador('altura', 10, 240);

```

## ***Arreglos***

El formato para definir un tipo array es:

nombre\_array = array [tipo subíndice] of tipo

nombre\_array identificador válido

tipo subíndice puede ser de tipo ordinal: boolean o char, un tipo enumerado o un tipo subrango.

Existe un elemento por cada valor del tipo subíndice

tipo describe el tipo de cada elemento del vector; todos los elementos de un vector son del mismo tipo.

Como ya dijimos anteriormente, los arreglos son estructuras de datos, por lo tanto las mismas deben ser declaradas. Esta operación se realiza en la sección “Type” de un programa en Pascal.

formato

type

nombre\_del\_tipo = array[tipo\_subindice \* ] of tipo;

\* debe ser de tipo ordinal: boolean, char, enumerado o subrango

Luego de la declaración del tipo, se declara la variable.

formato

var

nombre\_variable: nombre\_del\_tipo;

Ejemplos de declaraciones:

Ej1: type

Valores = array[ -10..10 ] of real;

var

precios: valores;

Ej2: const Max= 500;

Type

T\_Texto = array[ 1..Max ] of char;

Var

Texto: T\_Texto;

Ej3: const

longitud = 40;

altura = 30;

type

horizontal = 1..Longitud;

T\_Linea = Array [ horizontal ] of char;

var

Linea: T\_Linea

Ej4:

type

```
DiasSemana = (Lunes, Martes, Miercoles,Jueves, Viernes, Sabado, Domingo );
```

```
T_Dias = array [DiasSemana] of integer;
```

var

```
Dias: T_Dias;
```

## Operaciones

Asignación de valores

```
Texto[3] := 'a';
```

```
Precios[0] := 23.50;
```

Los índices de un arreglo pueden ser: entero, lógico, caracter, enumerado o subrango.

Con la siguiente declaración:

type

```
T_Notas = array [1..30] of integer;
```

var

```
Notas: T_Notas;
```

Lectura de un vector

```
for i:= 1 to 30 do
```

```
    read(Notas[i] );
```

Escritura de un vector

```
for i:= 1 to 30 do
```

```
    writeln(Notas[i] );
```

Con la siguiente declaración:

Type

```
T_Notas = array [1..30] of integer;
```

Var

```
Notas, Aux_Notas: T_Notas;
```

Copia de vectores

```
for i:= 1 to 30 do
```

```
    Aux_Notas[i]:= Notas[i];
```

```
o bien: Aux_Notas:=Notas;
```

Cargar 10 elementos en un vector, sumarlos y mostrar el resultado por pantalla.

```
Program Ejemplo1; {Version 2}
type
sumandos = array[1..10] of integer;
var
    suma, i : integer;
    vec_sumandos : sumandos;
begin
    suma := 0;
    for i:= 1 to 10 do
    begin
        read(vec_sumandos[i] );
        suma:= suma +vec_sumandos[i];
    end;
    writeln (´La suma de los números es´, suma);
end.
```

Ej2- Dados 50 números enteros, obtener el promedio de ellos. Mostrar por pantalla dicho promedio y los números ingresados que sean mayores que el mismo.

```
Program Ej2;
const
    max = 50;
type
    t_numeros = array[1.. max] of integer;
var
    suma, i : integer;
    promedio: real;
    numeros : t_numeros;
begin
    suma := 0;
    for i:= 1 to max do
    begin
        read(numeros[i] )
        suma:= suma +numeros[i];
```

```
end;  
Promedio:= suma/max;  
writeln ('El promedio es ',Promedio');  
for i := 1 to 50 do  
    if numeros[i] > promedio  
    then  
        writeln ('El número', numeros[i], 'es mayor al promedio');  
end.  
end.
```