UNIDAD 5. INTRODUCCIÓN A LA PROGRAMACIÓN

INTRODUCCIÓN

En oportunidad de hablar sobre las unidades básicas de una computadora, hemos mencionado al programa como elemento esencial. Un programa no es nada más que la descripción de la solución a un problema que se nos ha planteado, descripción que deberá ser hecha de manera tal que sea factible su proceso para la computadora.

Por lo tanto, resolver un problema computacional es una etapa esencial: no habrá programa si antes no hemos analizado y resuelto el problema:

Resolución de problemas computacionales

En general, para obtener la solución de un problema computacional, debemos realizar los siguientes pasos:

1) Formulación del problema: deberá ser clara, concisa, precisa, evitándose todo tipo de ambigüedades. Cuanto mejor esté formulado un problema mayor será la facilidad para encarar su solución.
2) Planteo del Modelo Matemático que resuelve el problema. En esta etapa, debemos distinguir dos aspectos fundamentales:
. Identificación de las variables de entrada o datos con que se cuenta y
. Relaciones entre esas variables y las incógnitas del problema.

3) Descripción de la secuencia de pasos necesaria para llegar a la solución del problema. Una vez obtenida esta secuencia, es aconsejable eliminar de ella los pasos superfluos.
4) Traducción de la secuencia anterior a un lenguaje comprensible para la computadora, o sea, a un lenguaje de alto nivel. Esto es escribir un programa.
5) Verificación del funcionamiento del programa y ejecución de los ajustes necesarios.
6) Aplicación de las variables de entrada al programa y obtención de la solución.

Es evidente que para la ejecución de los pasos 4, 5 y 6 es necesario conocer al menos un lenguaje de programación, por lo tanto, en este curso nos interesará solamente el desarrollo de las tres primeras etapas.

ALGORITMOS

Definición: se denomina tarea elemental a aquella tarea que no puede subdividirse en tareas más pequeñas.

Definición: un sujeto es aquel capaz de realizar tareas elementales.

Definición: un algoritmo es una lista secuencial de tareas elementales que pueden ser realizadas por un sujeto.

Si la secuencia de tareas elementales que escribimos es la que conduce a la obtención de la solución de un problema, la etapa 3 para la resolución de los mismos no es más que la obtención del correspondiente algoritmo.

Un algoritmo debe reunir las siguientes características:

1. No debe ser ambiguo, esto es, cada tarea elemental debe significar una sola cosa.
2. Debe detenerse en algún paso.
3. Debe mostrar el resultado obtenido.

Ejemplos:

1) Formulación: sumar dos números dados A y B.
Modelo Matemático: S = A + B; variables de entrada: A y B; relación: operación suma; variable de salida: S.
Algoritmo:
Paso 1: ingresar un valor para A
Paso 2: ingresar un valor para B
Paso 3: sumar A y B, asignando a S el resultado de la suma
Paso 4: mostrar el valor de S
Paso 5: finalizar

2) Formulación: sumar cinco números dados.
Modelo Matemático: S = S + X; variable de entrada: X; relación: operación suma; variable de salida: S.
Algoritmo:
Paso 1: ingresar el primer valor de X
Paso 2: tomarlo como resultado parcial de S
Paso 3: si ya se sumaron los cinco datos, mostrar el contenido de S y finalizar
Paso 4: dar el próximo valor para X
Paso 5: acumularlo en S
Paso 6: volver al paso 3

3) Formulación: sumar números de una lista de enteros dados, mientras los sumandos sean naturales.
Modelo Matemático: S = S + X mientras X > 0; variable de entrada: X; relación: operación suma; variable de salida: S.
Algoritmo:
Paso 1: asignar cero a S
Paso 2: ingresar el primer valor de X
Paso 3: si el valor no es natural, mostrar el contenido de S y finalizar
Paso 4: acumularlo en S
Paso 5: dar el próximo valor para X
Paso 6: volver al paso 3

Analicemos los ejemplos 2 y 3. En ambos casos hay una cierta cantidad de tareas elementales que deben repetirse. Esta repetición recibe el nombre de ciclo y dentro de él se distinguen cuatro elementos:

1. La asignación inicial a la variable que controla la repetición del ciclo. Esto se llama INICIALIZACIÓN de la variable.
2. El conjunto de tareas elementales que se repite. Este conjunto de tareas se denomina RANGO del ciclo.
3. La pregunta cuya respuesta indica si debe continuarse o no con la ejecución del rango. Este testeo se llama CONDICIÓN de la variable del ciclo.
4. La modificación de la variable que controla la repetición. Esto es la MODIFICACIÓN de la variable del ciclo.

Ahora notemos la siguiente diferencia: el rango del ciclo del ejemplo 2 se repetirá una cantidad pre-establecida de veces (cuatro). En este caso diremos que el ciclo es DIRECTO. En cambio, en el ejemplo 3, no sabemos cuántas veces deberá repetirse el rango puesto que dependerá de los datos de ingreso. Diremos que un ciclo de este tipo se denomina INDIRECTO.

Hasta aquí hemos especificado algoritmos usando el lenguaje natural, que es una herramienta útil para tal fin pero no la única. Su principal inconveniente es la extensión del algoritmo y la diversidad de redacción que podría surgir cuando diferentes personas especifican el mismo algoritmo. Por ello, se han diseñado otras herramientas tales como el **pseudocódigo** que combina expresiones de texto usadas en lenguaje natural con expresiones de texto provenientes de los lenguajes de programación y un mínimo recurso gráfico que facilita la lectura, sin embargo, las ventajas de la herramienta pseudocódigo por sobre el lenguaje natural no son tantas como para adoptarlo y se prefiere una herramienta gráfica que especifica algoritmos a través de **diagramas**.

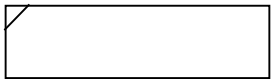
Existen diagramas diseñados por distintos autores y especialmente útiles para determinados fines. En este curso utilizaremos los **diagramas de bloques**, también conocidos como **diagramas N-S**.

ELEMENTOS DE DIAGRAMACIÓN

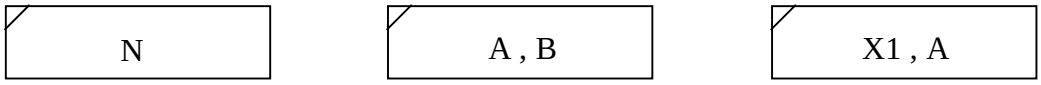
Un diagrama de bloques es la representación gráfica de un algoritmo. Los bloques son las estructuras básicas de los diagramas de bloques y se presentan a continuación:

Bloque de entrada de datos

Se simboliza:



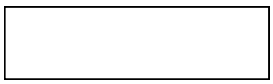
Adentro del bloque se colocan el o los nombres de las variables de entrada, por ejemplo:



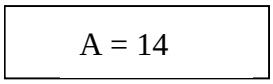
La tarea elemental que simboliza es la siguiente: ingresa un valor para la variable y dicho valor constituye el contenido de la variable cuyo nombre es el que se le asignó en el interior del bloque. Así, en el primer ejemplo, se almacenará en la memoria el valor que se ingrese para la variable N.

Bloque de asignación

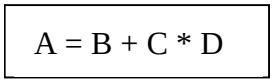
Se simboliza:



Adentro del bloque se coloca la transferencia de valores o expresiones a una variable. La variable que recibe el valor o la expresión debe colocarse a la izquierda del sigo "=" y a su derecha la constante o expresión que le proporciona el valor que se está asignando. Debe cuidarse que todas las variables intervinientes en la expresión de la derecha, tengan un valor dado previamente. Ejemplos:



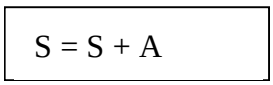
asigna el valor 14 como contenido de A.



asigna a A el resultado del producto C * D más la suma de B (con valores B, C y D previamente definidos)

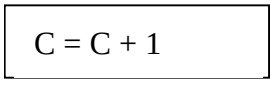
Algunas asignaciones importantes

Acumulador



Acumula en S el valor de A. Es decir, suma A al contenido previo de S y lo almacena nuevamente en S.

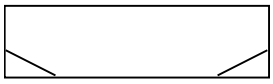
Contador



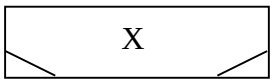
Incrementa en 1 el contenido anterior de C.

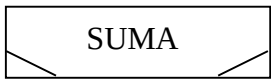
Bloque de salida de resultados

Se simboliza:

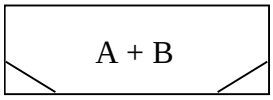


En el interior del bloque se colocan la o las variables cuyos contenidos desean mostrarse. Ejemplos:

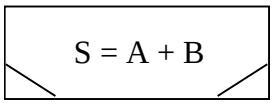




También es correcto:

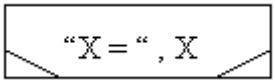


Se mostrará el resultado de la suma de A con B. Sería incorrecto:



pues resultaría ambiguo la ejecución del bloque, ya que no es la misma tarea elemental asignar que mostrar resultado.

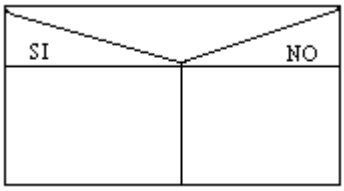
También es posible y muy recomendable, acompañar la salida de los datos con un texto explicativo de aquello que se está mostrando. Aunque la notación para este fin difiere entre distintos lenguajes de programación, adoptaremos las comillas para encerrar toda expresión textual que tenga ese fin. Por ejemplo:



proporcionará la salida: X = 20 (asumiendo que 20 era el contenido de X antes de ejecutarse este bloque).

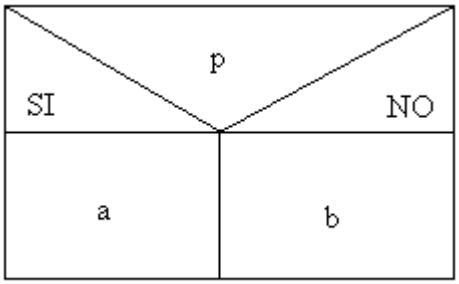
Bloque de alternativa

Se simboliza:

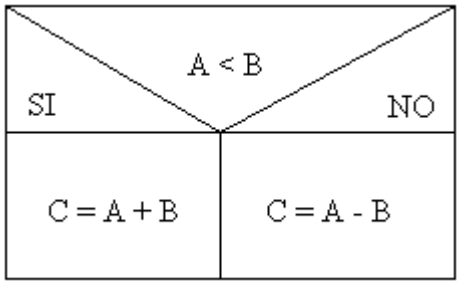


Su tarea es evaluar una proposición lógica de cuyo resultado depende la ejecución de una

tarea o secuencia de tareas. Si el valor de verdad de p es verdadero, se ejecutará la tarea o grupo de tareas indicadas con a; si el valor de verdad de p es falso se ejecutará la tarea o grupo de tareas indicadas con b. Esto es: $p \Rightarrow a \wedge \bar{p} \Rightarrow b$

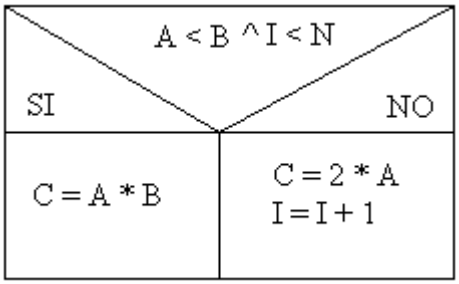


En el siguiente ejemplo:



Si $A < B \Rightarrow C = A + B$; si $A \geq B \Rightarrow C = A - B$

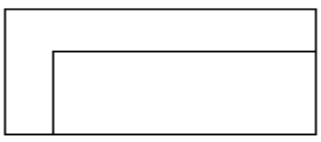
La proposición p puede ser compuesta, por ejemplo:



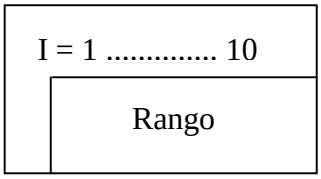
Si $A < B \wedge I < N \Rightarrow C = A * B$; si $A \geq B \vee I \geq N \Rightarrow C = 2 * A \wedge I = I + 1$

Bloque de ciclos

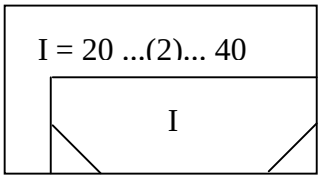
Se simboliza:



En la parte superior se colocan los valores inicial y final que toma la variable que controla el ciclo cuando se trata de DIRECTO o INCONDICIONADO o la proposición lógica que se evalúa cuando se trata de ciclo INDIRECTO o CONDICIONADO. En el rectángulo interno más pequeño se ubica el rango del ciclo. Por ejemplo:

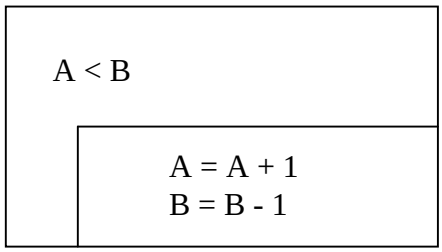


indica que el rango del ciclo incondicionado debe repetirse diez veces. Este tipo de ciclos admite que el paso entre uno y otro valor que toma la variable que lo controla sea distinto de 1. Así, la funcionalidad del siguiente ciclo incondicionado será:



mostrar los números naturales pares entre 20 y 40 ambos inclusive.

En los ciclos condicionados, su rango se ejecutará mientras el valor de verdad de la proposición lógica que se evalúa sea verdadero. Por ejemplo:



Funcionará así: mientras A sea menor que B, se ejecutarán las tareas previstas en el rango que son: incrementar A en 1 y decrementar B en 1. Es importante destacar que, si al evaluar la condición del ciclo, el contenido de A es igual o mayor que B, el rango del ciclo no se ejecutará nunca.

PRUEBA DE ESCRITORIO

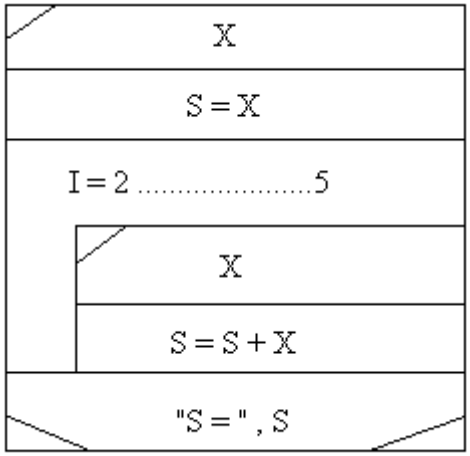
Una vez confeccionado el diagrama, resulta sumamente conveniente realizar una prueba de su funcionamiento. No olvidemos que este diagrama es un paso previo a la elaboración del programa, de manera que ahorraremos mucho tiempo y esfuerzo si tenemos una idea de su corrección desde el punto de vista lógico. De otro modo, deberemos corregir después el programa ante un fallo en su verificación.

Para controlar la lógica del diagrama, seleccionamos un conjunto apropiado de datos que le

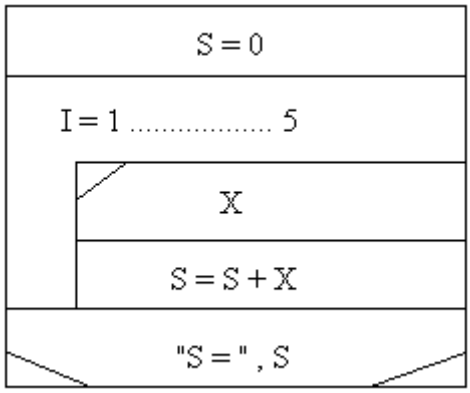
será asignado a la o las variables de entrada, recorremos el diagrama con dichos valores y controlamos si el resultado obtenido es o no la salida esperada.

A continuación veremos una colección de ejemplos que serán útiles para ilustrar la mecánica de procedimiento de las estructuras privilegiadas que presentamos anteriormente junto a los bloques, esto es: la **secuencia**, la **alternativa (o selección)** y la **iterativa (o de repetición)**. Para cada ejemplo se construirá el diagrama de bloques de una posible solución al problema, asumiendo que, en general, son numerosos y variados los algoritmos que resuelven un mismo problema. En algunos de los ejemplos se finalizará con prueba de escritorio. Se recomienda al lector completar la elaboración de las pruebas de escritorio no incluidas aquí.

Ejemplo 1:
Formulación: sumar cinco números dados.
Modelo Matemático: S = S + X; variable de entrada: X; relación: operación suma; variable de salida: S.
Diagrama:



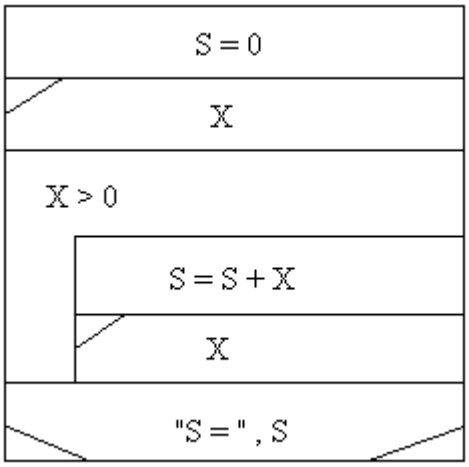
Tengamos en cuenta que hemos recalcado la necesidad de omitir los pasos superfluos que pueda contener un algoritmo. Observamos que, como se trata de un ciclo incondicionado, podemos eliminar la asignación del primer valor de X a S y acumular directamente en ella los cinco datos, inicializando S en cero. Así, resultará optimizado:



Prueba de escritorio:
Tomemos la siguiente lista como los sucesivos valores de X: 10, 30, -2, 4, 8

S	I	X	Salida
~~0~~	~~1~~	~~10~~	
~~10~~	~~2~~	~~30~~	
~~40~~	~~3~~	~~-2~~	
~~38~~	~~4~~	~~4~~	
~~42~~	~~5~~	8	
50	6		S = 50

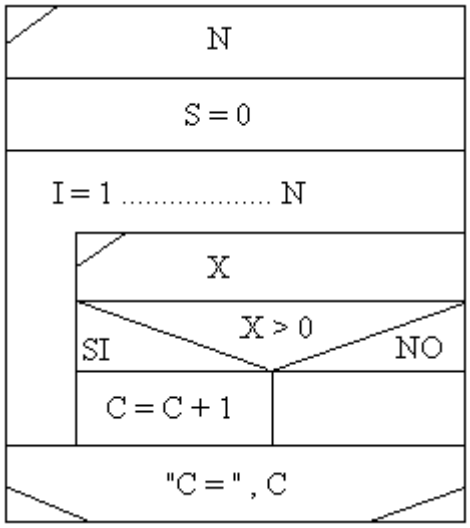
Ejemplo 2:
Formulación: sumar números de una lista de enteros dados, mientras los sumandos sean naturales.
Modelo Matemático: S = S + X mientras X > 0; variable de entrada: X; relación: operación suma; variable de salida: S.



Prueba de escritorio:
Tomemos la siguiente lista como los sucesivos valores de X: 13, 3, -2, 4, -8, 9, 21, -4

S	X	X > 0	Salida
~~0~~	~~13~~	V	
~~13~~	~~3~~	V	
16	-2	F	
			S = 16

Ejemplo 3:
Formulación: dados N números, determinar la cantidad de ellos que son positivos.
Modelo Matemático: C = C + 1 si X > 0; variables de entrada: N y X; relación: operación suma; variable de salida: C.
Diagrama:



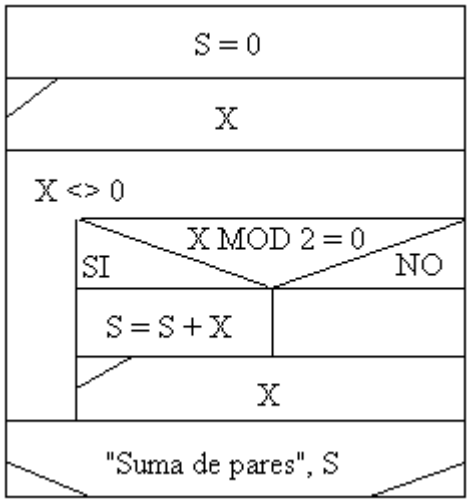
Prueba de escritorio:
Tomemos la siguiente lista como valores de X: 7, 21, -19, 10, -7, 21

N	C	I	X	X>0	Salida
6	~~0~~	~~1~~	~~7~~	V	
	~~1~~	~~2~~	~~21~~	V	
	~~2~~	~~3~~	~~-19~~	F	
		~~4~~	~~10~~	V	
	~~3~~	~~5~~	~~-7~~	F	
		~~6~~	21	V	
	4	7			C = 4

Ejemplo 4:
Formulación: dada una lista de números naturales, sumar los pares.
Modelo Matemático: S = S + X si X es par; variable de entrada: X; relación: operación suma; variable de salida: S.

Para distinguir los valores de X que son pares de los que no lo son, necesitaremos efectuar una división para observar el resto entre X y 2. Si ese resto es 0, indica que X es múltiplo de 2, o sea, es par; si ese resto es 1, indica lo contrario. Afortunadamente, la mayoría de los lenguajes de programación de alto nivel poseen incorporada una función que retorna el resto de la división entera entre dos valores z y t, denominada MOD.
Por definición: $\forall$ z, t $\in$ Z; z MOD t = z – t * [z / t]
La formulación del problema no indica la cantidad de datos que se ingresarán para su procesamiento, sin embargo, sabemos que esos datos son números naturales, por lo que aprovecharemos el conocimiento de la *característica* de los datos, ya que no disponemos de la *cantidad* de ellos. Usaremos un elemento denominado **bandera de fin de lista**, que definiremos como aquel valor que no cumple con al menos una de las propiedades que poseen los elementos de la lista de datos de ingreso. En este ejemplo, como esa lista está integrada por números naturales, podemos utilizar como bandera de fin de lista cualquier número no natural, por ejemplo, el cero.

Diagrama:



Pruebas de escritorio:
Prueba 1: lista: 3, 1, 6, 12, 5, 0 (bandera de fin de lista)

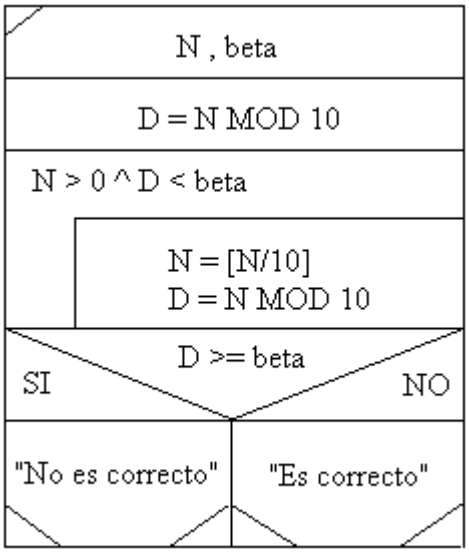
S	X	X <> 0	X MOD 2 = 0	Salida
~~0~~	~~3~~	V	F	
	~~1~~	V	F	
	~~6~~	V	V	
~~6~~	~~12~~	V	V	
18	~~5~~	V	F	
	0	F		Suma de pares 18

Prueba 2: lista: 1, 3, 9, 13, 0 (bandera de fin de lista)

S	X	X <> 0	X MOD 2 = 0	Salida
0	~~1~~	V	F	
	~~3~~	V	F	
	~~9~~	V	F	
	~~13~~	V	F	
	0	F		Suma de pares 0

Ejemplo 5:
Formulación: Dado un número natural, indicar si está correctamente expresado en un sistema de numeración de base β.
Modelo Matemático: Todo dígito del número natural debe ser menor a β; variables de entrada: N y beta.

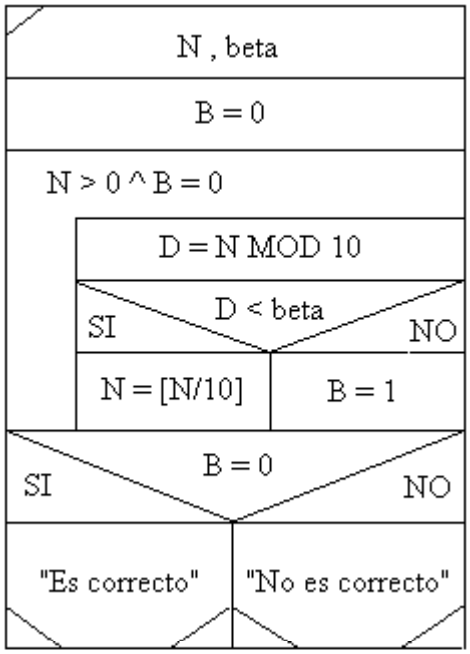
Diagrama:



Prueba de escritorio: Si N = 385 y beta = 8

N	beta	D	N > 0	D < beta	D >= beta	Salida
~~385~~	8	~~5~~	V	V		
38		8	V	F	V	No es correcto

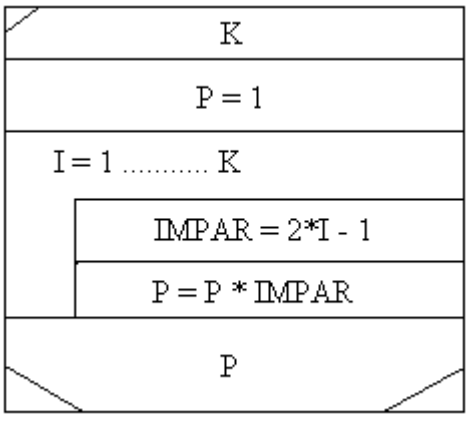
Ejemplo 6:
Idem al ejemplo 5.
Diagrama: La variante que se presenta a continuación incluye un elemento denominado **bandera de control**, su función es la de tomar un valor inicial bajo ciertas condiciones y modificar ese valor inmediatamente tales condiciones dejan de observarse. En nuestro ejemplo, la bandera tendrá un valor inicial que se mantendrá mientras se observe la condición de que los sucesivos dígitos que componen al número N sean menores que la base beta. Inmediatamente se constate lo contrario, la bandera modifica su valor. Este elemento, aparte de proporcionarnos una variable que da cuenta de algún acontecimiento de interés, actúa como variable de control para el ciclo condicionado que ejecuta la acción de analizar cada dígito de N. Cuando ocurre que uno de ellos no es menor que beta, por efecto del nuevo valor de la bandera de control, el ciclo deja de ejecutarse, evitando repeticiones inútiles a los fines de la salida requerida por el algoritmo.



Prueba de escritorio: Si N = 385 y beta = 8

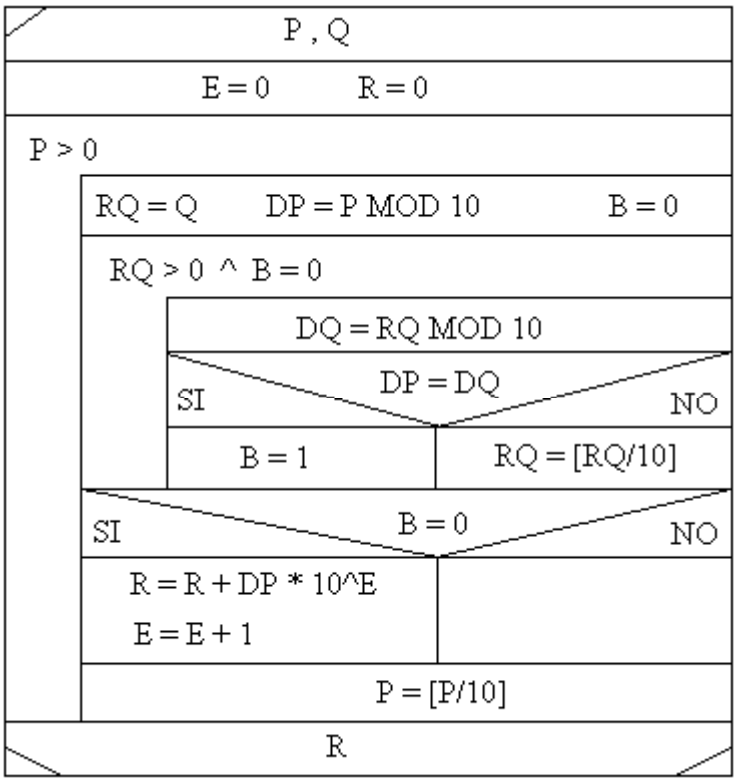
N	beta	B	N > 0	B = 0	D	D < beta	Salida
~~385~~	8	~~0~~	V	V	~~5~~	V	
38			V	V	8	F	
		1	V	F			No es correcto

Ejemplo 7:
Formulación: multiplicar los K primeros número naturales impares.
Modelo Matemático: IMPAR = 2*I – 1; I = 1, ..., K ; P = P * IMPAR; variable de entrada: K; relación: operación producto; variable de salida: P
Diagrama:



Se deja al lector la elaboración de la prueba de escritorio.

Ejemplo 8:
Formulación: dados dos números naturales P y Q, mostrar el número R que contenga los dígitos de P que no son dígitos de Q, ordenados según se encuentran en P. Si esto no es posible, mostrar R = 0.
Diagrama:



Observemos la utilidad de la variable RQ. Esta variable actúa como un **resguardo** del contenido de otra variable, en nuestro ejemplo la variable Q. El resguardo puede alterar su valor por efecto de la asignación RQ = [RQ/10]. En una siguiente iteración, donde es necesario contar con el valor inicial de Q, éste se consigue por copia de dicho valor a su resguardo, en la asignación RQ = Q.

Pruebas de escritorio:

Prueba 1: Si P = 1.237 ; Q = 728 entonces R = 13

P	Q	E	R	P > 0	RQ	DP	B	RQ>0	B=0	DQ	DP=DQ	Salida
~~1237~~	~~728~~	~~0~~	~~0~~	V	~~728~~	~~7~~	~~0~~	V	V	~~8~~	F	
					~~72~~			V	V	~~2~~	F	
					~~7~~			V	V	~~7~~	V	
							~~1~~	V	F			
~~123~~				V	~~728~~	~~3~~	~~0~~	V	V	~~8~~	F	
					~~72~~			V	V	~~2~~	F	
					~~7~~			V	V	~~7~~	F	
		~~1~~	~~3~~		~~0~~			F	V			
~~12~~				V	~~728~~	~~2~~	~~0~~	V	V	~~8~~	F	
					~~72~~			V	V	~~2~~	V	
							~~1~~		F			
~~1~~				V	~~728~~	1	0	V	V	~~8~~	F	
					~~72~~			V	V	~~2~~	F	
					~~7~~			V	V	7	F	
					0			F	V			
0		2	13	F								13

Prueba 2: Si P = 339 ; Q = 78.656 entonces R = 339. Se deja al lector la elaboración de esta prueba de escritorio.

Prueba 3: Si P = 876 ; Q = 7.768 entonces R = 0

P	Q	E	R	P > 0	RQ	DP	B	RQ>0	B=0	DQ	DP=DQ	Salida
~~876~~	~~7768~~	0	0	V	~~7768~~	~~6~~	~~0~~	V	V	~~8~~	F	
					~~776~~			V	V	~~6~~	V	
							~~1~~	V	F			
~~87~~				V	~~7768~~	~~7~~	~~0~~	V	V	~~8~~	F	
					~~776~~			V	V	~~6~~	F	
					~~77~~			V	V	~~7~~	V	
							~~1~~	V	F			
~~8~~				V	~~7768~~	8	~~0~~	V	V	8	V	
							1	V	F			
0												0

En todos los ejemplos desarrollados hasta aquí, ha ocurrido que la asignación de un valor a una variable con un contenido asignado previo, produce la **eliminación** del contenido anterior. Esto sucede por la identificación de la variable con una única dirección de memoria, tema tratado oportunamente cuando se describió el modo de acceso de la misma.

Variable indizada

Es posible disponer que, a una misma variable, le sean asignadas tantas posiciones de memoria distintas como sean necesarias, a fin de almacenar en cada una de ellas un valor distinto. Para ello, aparte del nombre de la variable, debe indicarse cuál de sus elementos es el del tratamiento en particular, lo que haremos agregando un subíndice, que indicará la posición de un contenido específico.
Esto es, si la variable X dispondrá de n direcciones de memoria diferentes, lo indicaremos:

$X = (X_1, X_2, ... X_n)$

En X_1 se almacenará uno de los valores de X, el de posición 1; en X_2 se almacenará otro de los valores de X, el de posición 2 y así sucesivamente.

Para indicar este tratamiento especial, identificaremos a cada una de ellas colocando el respectivo subíndice entre paréntesis, por ejemplo: X(1), X(2), X(n)

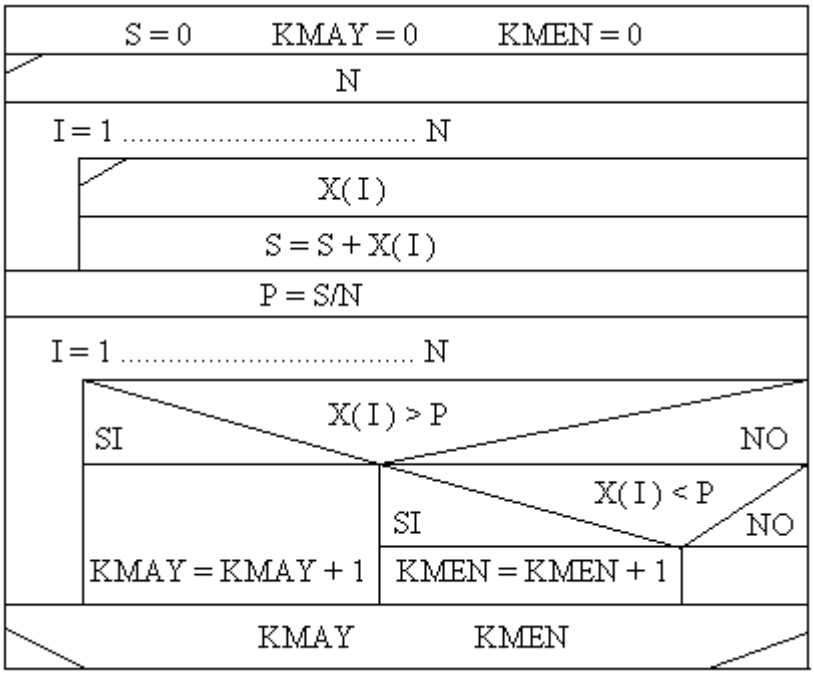
En la notación X(i) → X es el nombre de la variable indizada
i es el nombre de la variable que indica la posición

Teniendo en cuenta que cada elemento de una variable indizada ocupa un lugar de memoria diferente, su uso supone más requerimiento de memoria que las variables simples, por lo que, las variables indizadas deberán usarse solamente en los casos en que los problemas no admiten solución con las variables simples. Veamos las aplicaciones continuando con los ejemplos:

Ejemplo 9:
Formulación: dada una lista de N números, indicar cuántos son mayores y cuántos son menores al promedio de los números dados.
Modelo Matemático: $P = (X_1 + X_1 + ... + X_N) / N$; variables de entrada: N (variable simple) y X (variable indizada);
relación: Si $X(i) > P \Rightarrow KMAY = KMAY + 1$, Si $X(i) < P \Rightarrow KMEN = KMEN + 1$; variables de salida: KMAY, KMEN.

Expliquemos primeramente la razón de haber adoptado a la variable X como variable indizada. La formulación del problema requiere que se ingresen los datos, los cuales deben ser sumados como primer paso para la obtención del promedio P. Una vez obtenido éste, cada uno de esos mismos datos debe someterse a comparación con P para incrementar, cuando corresponda, los contadores KMAY y KMEN previstos. Si el sucesivo ingreso de los datos se hubiese efectuado a través de una variable simple, al finalizar el proceso de ingreso sólo dispondríamos del último de ellos almacenado en la memoria. ¿Qué hacer entonces para volver a disponer de ellos?, ¿ingresarlos nuevamente?. Es fácil imaginar la inconveniencia de pedirle a nuestro potencial usuario del programa que se elaboraría siguiendo este algoritmo, ¡que ingrese nuevamente los N datos que el programa le solicitó hace sólo un instante!. La variable indizada X alojará cada uno de los datos en una posición diferente de memoria, así, todo proceso que sea necesario hacerse involucrando estos datos, será posible con la sola invocación del nombre de la variable indizada X y el nombre de la variable usada para denotar cualquiera de sus elementos, almacenados en una posición determinada.

Diagrama:



Prueba de escritorio: lista de valores de X() = (3, 7, 20, 2, 18, 4)

S	KMAY	KMEN	N	I	X( )	P	X(I) > P	X(I) < P	Salida
~~0~~	~~0~~	~~0~~	6	~~1~~	X(1) = 3				
~~3~~				~~2~~	X(2) = 7				
~~10~~				~~3~~	X(3) = 20				
~~30~~				~~4~~	X(4) = 2				
~~32~~				~~5~~	X(5) = 18				
~~50~~				~~6~~	X(6) = 4				
54				~~7~~		9			
				~~1~~			F	V	
		~~1~~		~~2~~			F	V	
		~~2~~		~~3~~			V		
	~~1~~			~~4~~			F	V	
		~~3~~		~~5~~			V		
	2			~~6~~			F	V	
		4		7					2 4

Variable Indizada Bidimensional

A menudo, los datos que deben procesarse se presentan dispuestos en una tabla. Su estructura sigue el siguiente esquema:

$$\begin{vmatrix} A_{11} & A_{12} & A_{13} & . & . & A_{1N} \\ A_{21} & A_{22} & A_{23} & . & . & A_{2N} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ A_{M1} & A_{M2} & A_{M3} & . & . & A_{MN} \end{vmatrix}$$

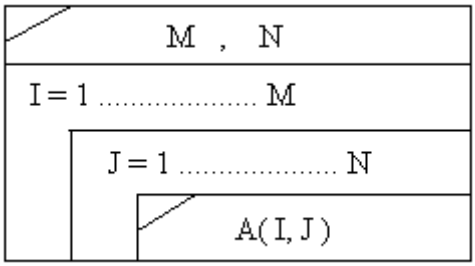
La tabla posee M filas y N columnas. Cada elemento de la tabla ocupa una posición de fila y columna que se distinguen por los subíndices i y j del elemento A_{ij}. Por ejemplo, el elemento A_{23} es el ubicado en fila 2, columna 3.

Siguiendo la notación adoptada para las variables indizadas unidimensionales, indicaremos al elemento A_{ij} como A(i,j). Así:

En la notación A(i,j) → A es el nombre de la variable indizada bidimensional
i es el nombre de la variable que indica la posición de fila
j es el nombre de la variable que indica la posición de columna

Ejemplo 10:
Formulación: ingresar los elementos de una tabla A de M filas y N columnas.
Modelo Matemático: ingreso de A(i,j) para i = 1, ..., M y j = 1, ..., N; variables de entrada: M, N, A (variable indizada bidimensional).
A continuación se presenta el algoritmo requerido, haciendo el recorrido de la tabla "por filas", es decir, iniciando en la primera fila, se ingresarán los sucesivos valores de la tabla correspondientes a esa fila para continuar con la segunda y así sucesivamente. Dejamos al lector el diseño del diagrama para efectuar un ingreso de tabla recorriéndola "por columna" y la prueba de escritorio del siguiente

Diagrama:



ALGUNOS MÉTODOS DE CLASIFICACIÓN Y BÚSQUEDA

Dos operaciones muy importantes en programación de computadoras son: *ordenación* y *búsqueda*; son esenciales para un gran número de programas de proceso de datos y se estima que en estas operaciones las computadores insumen gran parte de su tiempo. La búsqueda y ordenación son también procesos que las personas encuentran habitualmente en su vida diaria. Consideremos, por ejemplo, el proceso de encontrar una palabra en el diccionario o un nombre en una guía telefónica. La búsqueda de un elemento específico se simplifica considerablemente por el hecho de que las palabras en el diccionario y los nombres en la guía telefónica, están *ordenados* o *clasificados* en orden alfabético.

A continuación se presentan estos temas siguiendo el esquema que se detalla:

- ✓ Introducción que clarifica el *modus operandus* del método.
- ✓ Ejemplo de procesamiento seguido por el algoritmo en cada paso de ejecución.

- ✓ Diagrama (no para todos los métodos). No se especifica la entrada de datos ni la salida de resultados.
- ✓ Análisis de eficiencia del algoritmo

Ordenación

Se considera ordenar como el proceso de reorganizar un conjunto dado de objetos en una secuencia especificada. El objetivo de este proceso es facilitar la búsqueda subsiguiente de los elementos del conjunto ordenado.

Ordenación interna y ordenación externa

Distinguiremos a la ordenación interna de la externa porque en el primer caso, la estructura de datos a ordenar corresponden a una variable indizada que se almacena en la memoria interna rápida, de acceso aleatorio, de la computadora. En la ordenación externa, en cambio, la estructura de datos a ordenar están en almacenamientos externos más lentos basados en dispositivos mecánicos como los discos y cintas.

Ordenación interna

La ordenación o clasificación de datos (*sort* en inglés) es una operación consistente en disponer un conjunto –estructura- de datos en algún determinado orden con respecto a uno de los campos de elementos del conjunto. Por ejemplo, cada elemento del conjunto de datos de una guía telefónica tiene un campo nombre, un campo dirección y un campo número de teléfono; la guía está dispuesta en orden alfabético de nombres. En terminología de ordenación el elemento por el cual está ordenado un conjunto de datos se denomina *clave*.

Una lista se dice que está ordenada por la clave k si la lista está en orden ascendente o descendente con respecto a esta clave. La lista está en orden ascendente si.

$i < j \Rightarrow k(i) <= k(j)$

y se dice que está en orden descendente si:

$i > j \Rightarrow k(i) >= k(j)$

para todos los elementos de la lista. A lo largo de este tema se establecerá orden ascendente.

Los algoritmos de numeración son numerosos, por ello se debe prestar especial atención en su elección. ¿Cómo se sabe cuál es el mejor algoritmo?. La *eficiencia* es el factor que mide la calidad y rendimiento de un algoritmo. El mejor criterio para medir la eficiencia de un algoritmo es aislar operaciones específicas trascendentes en la ordenación y contar el número de veces que se realizan.

Criterio de eficiencia para algoritmos de ordenación

La principal condición a imponer sobre los métodos de ordenación interna es la utilización económica de la memoria disponible. Esto implica que las permutaciones de posición de sus

elementos deben realizarse utilizando el espacio ocupado por la variable indizada; esto se denomina ordenación *in situ*.

Se utilizará como medida de eficiencia el número C de comparaciones efectuadas entre claves y M de movimientos (transferencias) de elementos. El algoritmo de ordenación A será más eficiente que el B si requiere menor número de comparaciones.

Aunque los buenos algoritmos de ordenación necesitan del orden de n*log n comparaciones para ordenar una lista de tamaño n, en este curso estudiaremos varias técnicas sencillas de ordenación denominadas **métodos directos**, que requiern del orden de n^2 comparaciones de claves. Es por esto que los métodos de ordenación se suelen dividir en dos grandes grupos:

Directos:
- Ordenación por Inserción
- Ordenación por Selección
- Ordenación por Intercambio

Indirectos:
- Ordenación por Inserción con incrementos decrecientes (Shell)
- Ordenación Rápida
- Ordenación por Mezcla

Las razones para cubrir sólo los métodos directos son fundamentalmente las siguientes:

1. Los métodos directos son esencialmente adecuados para poner de relieve las características de los principio de ordenación más usuales.
2. Darán lugar, más adelante, a programas más cortos y sencillos. No olvidemos que los programas también ocupan memoria.
3. Aunque los métodos sofisticados requieren menos operaciones, éstas normalmente son más complejas en detalle; consecuentemente los métodos directos son más rápidos para n suficientemente pequeño, aunque no deben usarse para valores de n mayores que 100.

Ordenación por Inserción Directa

Este método es usado generalmente por los jugadores de cartas, de allí que se lo conozca también por Método de la Baraja. Los elementos (cartas) están divididos conceptualmente en dos secuencias, una secuencia destino a_1, ..., a_{i-1} y una secuencia origen a_i,...,a_n. En cada paso, 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 sitio apropiado.

Veamos un ejemplo con la lista A = (44, 55, 12, 42, 94, 18, 6, 67)

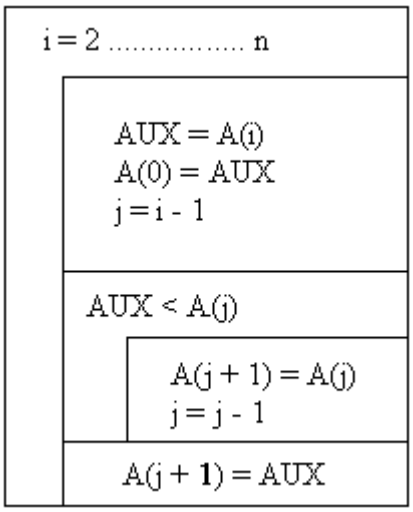
	44	55	12	42	94	18	6	67
i = 2	44	55	12	42	94	18	6	67
i = 3	12	44	55	42	94	18	6	67
i = 4	12	42	44	55	94	18	6	67
i = 5	12	42	44	55	94	18	6	67
i = 6	12	18	42	44	55	94	6	67
i = 7	6	12	18	42	44	55	94	67
i = 8	6	12	18	42	44	55	67	94

En el proceso de encontrar el sitio adecuado, conviene alternar comparaciones y movimientos, es decir "dejar caer" el elemento que se insertará, comparándolo con el inmediato precedente a_j. Como consecuencia se inserta el elemento o se mueve a_j a su derecha y se continúa con el inmediato por la izquierda, según el mismo proceso. Se observa que hay dos condiciones distintas de terminación de este proceso de "caida":

1. Se encuentra un elemento a_j menor que el elemento a insertar.
2. Se alcanza el extremo izquierdo de la secuencia destino.

Este caso típico de ciclo con dos condiciones de terminación permite aplicar la conocida técnica del centinela. Se aplica fácilmente situando como centinela a_0 = elemento a insertar.

A continuación se presenta el diagrama correspondiente:



Análisis del Método de Inserción Directa:

El número de comparaciones entre elementos en el paso de caída i, o sea C_i es:

Como máximo = i-1
Como mínimo = 1

Cabe aclarar lo siguiente: cuando se calcula el máximo C_i se consideran sólo aquellas comparaciones que dan lugar a movimientos de la lista, o sea que se cuentan las comparaciones AUX < A(j) cuyo valor de verdad es verdadero y, por lo tanto dan lugar a la ejecución del ciclo en donde se produce el movimiento A(j+1) = A(j). La última comparación es la que se hace con el centinela, pero como produce una valor de verdad falso no se la considera para el cálculo de C_i.

Cuando se calcula el mínimo C_i está presente la situación en la que un elemento debe quedar situado –momentáneamente– en la posición que ocupa, por lo tanto la primera comparación proporciona un valor de verdad falso que evita hacer los movimientos del rango del ciclo. Esta única comparación se cuenta, dando lugar al valor mínimo de 1.

Suponiendo que todas las permutaciones entre los n elementos tienen la misma probabilidad, resulta entonces:

Media = ((i-1)+1)/2=i/2

El número de movimientos Mi es:

$C_i + 2 \rightarrow$ incluyendo el centinela y la asignación de inserción.

Se entiende por movimientos aquellas asignaciones que tienen lugar sobre la variable indizada, o sea:
A(0)=AUX $\rightarrow$ una asignación para el centinela
A(j+1) = A(j) $\rightarrow$ una asignación por cada comparación
A(j+1) = AUX $\rightarrow$ una asignación de inserción

Si sabemos que: $\sum_{i=1}^{n} i = n.(n+1)/2$ entonces :

Los números totales de comparaciones y movimientos, en función de n, son:

Valores mínimos (se presentan cuando los elementos de la lista están ordenados de origen):

$$C_{min} = \sum_{i=2}^{n} 1 = 1+1+\ldots+1 = n-1$$

$$M_{min} = 2.(n-1)$$

Valores máximos (se presentan cuando los elementos están ordenados en orden inverso)

$$C_{max} = \sum_{i=2}^{n}(i-1) = \sum_{t=1}^{n-1} t = 1+2+...+(n-1) = n.(n-1)/2$$

$$M_{max} = \sum_{i=2}^{n}((i-1)+2) = \sum_{i=2}^{n}(i-1) + \sum_{i=2}^{n} 2 = (n.(n-1)/2) + 2.(n-1) = (n^2+3.n-4)/2$$

Valores medios

$$C_{med} = (C_{min} + C_{max})/2 = (n^2+n-2)/4$$

$$M_{med} = \sum_{i=2}^{n}((i/2)+2) = (1/2).\sum_{i=2}^{n} i + \sum_{i=2}^{n} 2 = (1/2).((n.(n+1)/2)-1) + 2.(n-1) =$$

$$= (n^2+9.n-10)/4$$

La ordenación por inserción se comporta de manera realmente natural. El algoritmo es un proceso *estable* de ordenación pues no modifica el orden de los elementos que integran la secuencia destino.

Ordenación por Selección Directa

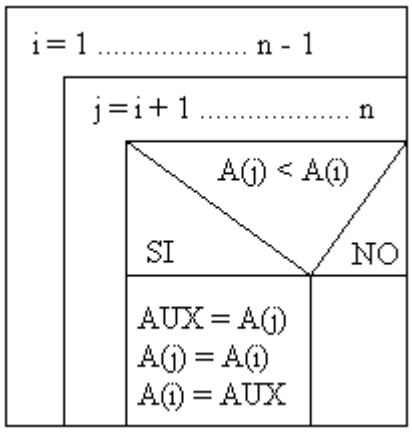
Este método se basa en los siguientes principios:

1. Seleccionar el elemento con clave mínima
2. Intercambiarlo con el primero a_1.

A continuación se repiten estas operaciones con los elementos (n-1), (n-2), etc. restantes hasta que quede un único elemento, el mayor. El método se ilustra con los mismos ocho elementos de la tabla anterior:

	44	55	12	42	94	18	6	67
i = 1	6	55	44	42	94	18	12	67
i = 2	6	12	55	44	94	42	18	67
i = 3	6	12	18	55	94	44	42	67
i = 4	6	12	18	42	94	55	44	67
i = 5	6	12	18	42	44	94	55	67
i = 6	6	12	18	42	44	55	94	67
i = 7	6	12	18	42	44	55	67	94

Este método es, en cierta forma, opuesto al de inserción directa: este considera en cada paso solamente un único elemento de la secuencia origen y todos los elementos de la secuencia destino para encontrar el punto de inserción; el método de selección directa considera todos los elementos de la secuencia origen para encontrar el que tiene la menor clave y depositarlo como elemento siguiente de la secuencia de destino. El diagrama para este algoritmo es:



Análisis del Método de Selección Directa.

Evidentemente, el número C de comparaciones entre elementos es independiente de la ordenación inicial de los mismos. En este aspecto, puede decirse que este método se comporta de forma menos natural que el de inserción directa.

En la tabla siguiente se refleja el número de comparaciones:

Pasada	Comparaciones
1	n-1
2	n-2
3	n-3
.	.
.	.
n-1	1

El número total de comparaciones es: 1 + 2 + 3 + ... + (n-3) + (n-2) + (n-1) es decir:

$$C = \sum_{i=1}^{n-1} i = (n.(n+1)/2) - n = n.(n-1)/2$$

En cuanto a los movimientos, la mínima cantidad de ellos se produce cuando los elementos están ordenados inicialmente, mientras que el máximo ocurre cuando los elementos están ordenados en orden inverso. Estos valores son difíciles de determinar a pesar de que el algoritmo es sencillo, es por ello que tales cuestiones no se abordarán aquí, pero el lector interesado puede consultar en Knuth, D. E. “Top-down Syntax Análisis” Vol 1, pp. 95-99.

Sin embargo, como conclusión puede afirmarse que, siendo el método de selección directa un algoritmo del orden de eficiencia de n^2, es comparable con el método de inserción directa y preferible a éste en la mayoría de los casos, excepto cuando los elementos están inicialmente ordenados, o casi ordenados, en donde el comportamiento del algoritmo de la baraja puede ser algo más rápido.

Ordenación por Intercambio Directo

La clasificación de un método de ordenación, raramente puede hacerse de forma clara. Los dos métodos descritos anteriormente pueden considerarse también de intercambio, sin embargo, el que se presenta ahora es un método cuya característica dominante es el intercambio entre pares de elementos. Se basa en el principio de comparar e intercambiar pares de elementos adyacentes hasta que todos estén ordenados.

Como en el método de selección directa, se hacen repetidas pasadas sobre la lista, moviendo en cada una de ellas el elemento de clave mínima hasta su extremo izquierdo. Si en cambio se mira la lista en posición vertical en vez de horizontal y –con alguna imaginación- los elementos se consideran como "burbujas" en un depósito de agua con "pesos" acordes con sus claves. De cada pasada sobre la lista resulta la ascensión de una burbuja hasta el nivel de peso que le corresponde, por lo que este método se conoce generalmente como Método de la Burbuja. En la tabla siguiente se clarifica el paso de la lista a través del algoritmo:

Inicial	i=2	i=3	i=4	i=5	i=6	i=7	i=8
44	6	6	6	6	6	6	6
55	44	12	12	12	12	12	12
12	55	44	18	18	18	18	18
42	12	55	44	42	42	42	42
94	42	18	55	44	44	44	44
18	94	42	42	55	55	55	55
6	18	94	67	67	67	67	67
67	67	67	94	94	94	94	94

Se deja al lector la tarea de diseñar el diagrama de bloques correspondiente.

Análisis del Método de Intercambio Directo

El número total de comparaciones es, como en el algoritmo de selección directa, obtenido de una tabla como la que se presentó para aquel método, es decir que nos conduce a:

$C = n.(n – 1) / 2$

Los números mínimo, máximo y medio de movimientos provienen de un análisis complejo, pero puede comentarse que son de orden cuadrático. El método de la Burbuja tiene escasas razones que lo hagan recomendable y su eficiencia es menor a la de los vistos precedentemente.

Búsqueda

Otro problema importante en proceso de datos, como ya se ha comentado, esla búsqueda en un conjunto de datos de un elemento específico y la recuperación de alguna información asociada al mismo.

Existen diferentes métodos de búsqueda:

Búsqueda Lineal o Secuencial
Búsqueda Binaria o Dicotómica
Búsqueda hash o por conversión de claves

Las dos primeras se pueden aplicar a listas de datos implementadas sobre variables indizadas y la tercera es más propia de otro tipo de estructuras, razón por lo cual no se abordará aquí su estudio.

Búsqueda Lineal

Esta es la técnica más simple, consiste en recorrer toda la lista buscando el elemento deseado, desde el primer elemento hasta el último de uno en uno. Si la lista contiene el elemento se devolverá su posición y, en caso contrario, un mensaje que indique el fracaso de la búsqueda.

Es notorio el gran inconveniente de este algoritmo, pues sea cual fuere el resultado de la búsqueda (existe / no existe), se recorre la lista completa. La mejora se introduce condicionando el trabajo de inspección de los elementos a una variable de control o interruptor o bandera que permita detener la búsqueda cuando el elemento es encontrado. Bajo estas condiciones, la lista será recorrida completamente sólo si el elemento buscado es el último o si no se halla entre los datos.

Análisis de la Búsqueda Lineal

En el caso más desfavorable (el elemento buscado está ubicado al final de la lista o no existe), este método requiere consultar n elementos, por lo tanto el tiempo de búsqueda es directamente proporcional al número de elementos de la lista.

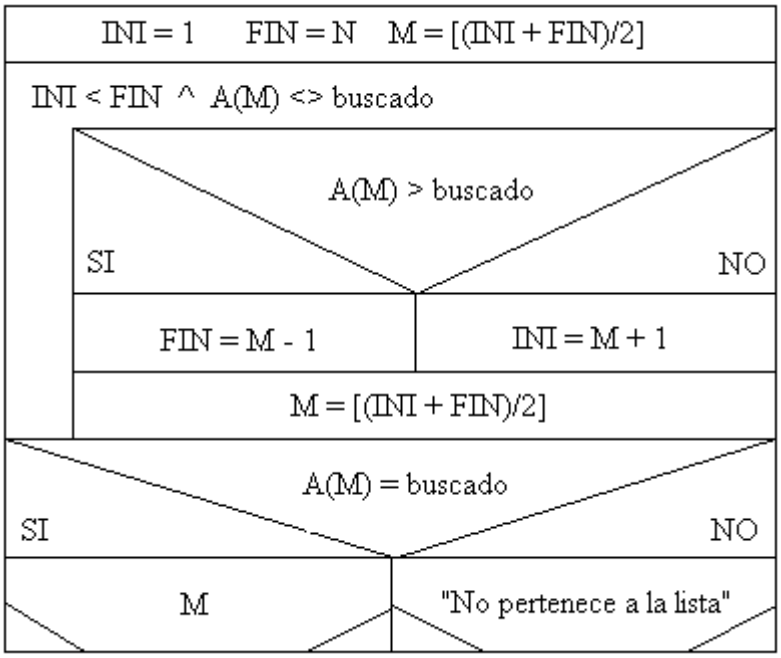
Búsqueda Binaria

La búsqueda lineal, por su simplicidad, es buena para listas pequeñas, pero para grandes listas es ineficiente; la búsqueda binaria es el método idóneo. Se base en el conocido concepto *divide y vencerás*.

Este método tiene una clara expresión en la operación que se lleva a cabo cuando se busca una palabra ene el diccionario. La búsqueda no comienza en la página 1 y se sigue secuencialmente, sino que se abre el diccionario en una página donde aproximadamente se piensa que puede estar la palabra buscada, es decir, se divide el diccionario en dos partes; al abrir se observa si se acertó con la página o sino, se determina en cuál de las dos partes se deberá seguir buscando. Se repite este proceso hasta que, por aproximaciones sucesivas se encuentra la palabra o se concluye que no está contenida en el diccionario.

Debe notarse que, al hacer una partición en dos de la lista y sucesivas sublistas, habrá un elemento "central" que actúa como límite entre ambas partes. Dado que este elemento es el primero en compararse con el que se está buscando, al momento de considerar la nueva sublista con la cual se reiterará el proceso, dicho elemento central no es incluido para evitar una futura comparación redundante. Por ello, las sublistas que se originan de la partición, cuyo elemento medio es el de posición m, tendrán como límite derecho el elemento ubicado en posición (m-1) y como límite izquierdo el elemento ubicado en posición (m+1).

Diagrama: teniendo en cuenta que la variable de entrada *buscado* contiene el contenido del elemento que se está buscando



Cuando n es impar estas particiones resultan de tamaño entero, pero cuando n es par, la división proporciona un número real, lo cual no tiene sentido porque está denotando el tamaño de una lista. Por lo tanto se aplica la función parte entera para subsanar este problema, de esta forma, las dos listas resultan del mismo tamaño o, a lo sumo, difieren en una unidad.

Pruebas de escritorio:
Prueba 1: lista A() = (5, 9, 21, 33, 512); buscado = 33

buscado	INI	FIN	M	A(M)	Salida
33	~~1~~	5	~~3~~	21	
	4		4	33	4

Prueba 2: lista A() = (5, 9, 21, 33, 512); buscado = 2

buscado	INI	FIN	M	A(M)	Salida
2	1	~~5~~	~~3~~	21	
		~~2~~	~~1~~	5	
		0	0		No pertenece a la lista

Análisis de la Búsqueda Binaria

En cada comparación efectuada se divide en dos mitades el tamaño de la lista. Si n es el tamaño de la lista, los tamaños sucesivos de las sublistas serán:

n/2, n/4, n/8, ... o sea: $n/2^1$, $n/2^2$, $n/2^3$, ...

El proceso terminará, en el caso más desfavorable, cuando el tamaño se hace menor o igual a 1. Por consiguiente, si $C_{máx}$ es el número mayor de comparaciones:

$n/ C_{máx} < \mathbf{1}$ o bien $n < 2^{Cmáx}$

Tomando logarítmos en base 2 en ambos miembros de la desigualdad, se obtiene:

$\log_2 n < C_{máx} . \log_2 2 = C_{máx}$ o sea: $\log 2\ n < C_{máx}$

Por consiguiente, la eficiencia de la búsqueda binaria es del orden de $\log_2 n$ lo que lo hace mucho más rápido que la búsqueda lineal.

Por ejemplo, para una lista de 50.000 elementos, la búsqueda lineal requiere, en el peor de los casos 50.000 comparaciones y 25.000 por término medio, mientras que la búsqueda binaria nunca requerirá más de $\log_2$ 50.000, es decir, alrededor de dieciséis comparaciones. A fin de ser sinceros, al tiempo de búsqueda binaria habría que sumarle el tiempo destinado a ordenar previamente la lista.