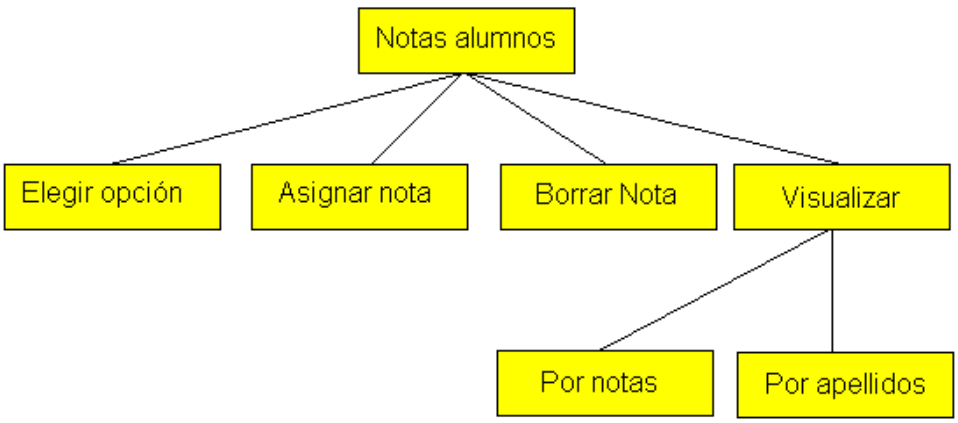
UNIDAD I PROGRAMACIÓN MODULAR

¿Qué es programación modular?

Uno de los métodos más conocidos para resolver un problema es dividirlo en problemas más pequeños, llamados subproblemas. De esta manera, en lugar de resolver una tarea compleja y tediosa, resolvemos otras más sencillas y a partir de ellas llegamos a la solución. Esta técnica se usa mucho en programación ya que programar no es más que resolver problemas, y se le suele llamar diseño descendente, metodología del *divide y vencerás* o programación *top-down*.

Es evidente que si esta metodología nos lleva a tratar con **subproblemas**, entonces también tengamos la necesidad de poder crear y trabajar con **subprogramas** para resolverlos. A estos subprogramas se les suele llamar **módulos**, de ahí viene el nombre de programación modular. En *La programación modular* disponemos de dos tipos de módulos: los procedimientos y las funciones.

Veamos un ejemplo de cómo emplear el diseño descendente para resolver un problema. Supongamos que un profesor quiere crear un programa para gestionar las **notas** de sus alumnos. Quiere que dicho programa le permita realizar tareas tales como asignar notas, cambiar notas, ver las notas según distintas calificaciones, etc. A continuación tienes un esquema que representa una de las posibles divisiones del



problema en módulos.

Los procedimientos

Un procedimiento es un **subprograma** que realiza una tarea específica. Para **invocarlo**, es decir, para hacer que se ejecute, basta con escribir su **nombre** en el cuerpo de otro procedimiento o en el programa principal. Pero, hay que tener muy en cuenta que su declaración debe hacerse antes de que sea llamado por otro módulo.

Una vez que has construido varios programillas en *La programación modular*, crear un procedimiento no es nada complicado, pues tiene prácticamente la misma estructura que un programa. Veamos las secciones que comparten y no comparten un procedimiento y un programa principal:

- Mientras que en el programa la cabecera consta de la palabra reservada *program* seguida del nombre del programa, en un procedimiento se compone de la palabra procedure seguida del **nombre** del procedimiento y una lista de **parámetros** que es opcional.

- Las secciones de declaración de constantes (const), de tipos (type) y de variables (var) también pueden aparecer en la estructura de cualquier procedimiento.
- Respecto al cuerpo del **procedimiento**, decir que al igual que el de un programa se delimita por las palabras reservadas **begin** y **end**, y en su interior puede contener sentencias simples o estructuradas.
- Por último, comentar que ambos difieren en el signo de puntuación que marca su final, ya que en un programa es el **punto** y en un procedimiento es el punto y coma.

Todas estas diferencias y similitudes que hemos comentado, puedes apreciarlas en los siguientes esquemas que representan las estructuras de un programa y de un procedimiento:

```
program nombre_programa;
const declarar_ctes;
type declarar_tipos;
var declarar_vars;
(*aquí irían los subprogramas*)
begin
   cuerpo_programa
end .
```

```
procedure nombre (lista_parametros);
   const declarar_ctes;
   type declarar_tipos;
   var declarar_vars;
   (*aquí irían los subprogramas*)
   begin
      cuerpo_procedimiento
   end ;
```

Los parámetros (argumentos)

Como habrás observado, con los procedimientos nos llega un concepto nuevo, el de los parámetros. A los parámetros también se les conoce como argumentos y tienen la misión de comunicar al procedimiento con el programa que lo llama. Por ejemplo, si quieres hacer un subprograma que multimplique dos números, lo más cómodo es que al **llamar** al procedimiento le pases los valores que participarán en la operación. Podría ser algo como:

```
procedure producto (a,b : integer; var rdo : integer) ;
   (* resto del procedimiento *)
```

En el ejemplo anterior se observan las dos clases de argumentos que existen en *La programación modular*:

- Los parámetros **por valor**
- Los parámetros **por referencia**

Los parámetros por valor tiene dicho nombre porque lo que recibe el subprograma no son más que **copias de los valores** de los datos que el programa invocador le pasa. Por tanto si en el procedimiento modificamos alguno de estos valores, los datos originales permanecerán **inalterados**. En el **ejemplo**, son a y b.

En cambio, en los parámetros por referencia lo que se pasa al procedimiento son los datos en sí. Y si éste los modifica, los cambios permanecerán una vez que la ejecución vuelva al módulo que invocó al procedimiento. Se utilizan para obtener valores de los cálculos que haga un subprograma, y en el anterior **ejemplo** es rdo.

¿Cómo se especifica que un parámetro es por valor o por referencia?

Pues es tan sencillo como anteponer la palabra reservada **var** cuando quieres que un argumento sea considerado como **referencia**. Esto se observa claramente con el parámetro rdo del ejemplo.

nota: si no tienes muy clara la diferencia entre parámetros por valor y por referencia, te aconsejo que vayas al ejemplo sobre procedimientos del que dispones en este tema, estoy seguro que te ayudará.

Las variables globales y locales

Después de estudiar los procedimientos de *La programación modular* y las diferencias entre parámetros por valor y por referencia, es hora de tratar otro aspecto clave en la programación modular: la distinción entre variables globales y locales.

Veamos un ejemplo que, como siempre, nos ayudará en la explicación de estos nuevos conceptos:

```
program varsGlobalesLocales;
var
  vGlobal : integer;

procedure nombreProc(param : integer);
var
  vLocal : integer;
begin
  vLocal := 2 * param;
  vGlobal := vLocal
end;

(*cuerpo principal del programa*)
begin
  vGlobal := 1;
  nombreProc(4);
  writeln('vGlobal vale : ',vGlobal)
end.
```

Una variable local es una variable que está declarada dentro de un subprograma, y se dice que es local al subprograma. Y lo que la caracteriza es que su valor sólo está disponible mientras se ejecuta el subprograma. Dicho de otra manera, el programa principal no tiene conocimiento alguno de las variables locales de sus procedimientos y funciones.

Las variables declaradas en la sección correspondiente a esta labor en el programa principal se denominan variables globales. Y a diferencia de las locales, su valor está disponible tanto en el cuerpo del programa principal como en el de cualquiera de los subprogramas declarados.

En el **ejemplo** anterior se declara una variable local llamada vLocal, y si se intentase usarla en el cuerpo del programa principal, por ejemplo para asignarle un valor, recibiríamos un error del compilador. También se

declara una variable global (vGlobal), cuyo valor se cambia en el cuerpo del procedimiento, y este cambio permanece después de ejecutarlo, pues el resultado que se mostraría en pantalla sería : vGlobal vale : 8.

nota: aunque en el ejemplo se hace, no es una buena práctica en programación tratar con **variables globales** en los subprogramas, porque pueden aparecer errores debidos a algún cambio no previsto en una de estas variables. Este tipo de errores son conocidos como efectos laterales y se evitan usando los parámetros en la comunicación de un programa con sus procedimientos.

Un ejemplo con los procedimientos

Después de hablar de procedimientos, parámetros por valor y por referencia y variables globales y locales, es la hora de presentarte un ejemplo con el que puedas reforzar lo que hemos visto. Con él se pretende que queden absolutamente claros estos conceptos porque son fundamentales en la programación.

En el ejemplo, tienes a la izquierda el código de un programa en el que hay un procedimiento. A la derecha, tienes tres casillas en las que debes introducir **números enteros** para variables que participan en el programa. Después de dar los valores, deberías examinar el código y determinar qué salida crees que se producirá. Por último, pulsando el botón puedes ver la salida real y compararla con la que tenías en mente.

Las funciones predefinidas

Este tema trata de la programación modular, y como ya hemos dicho, *La programación modular* nos ofrece dos tipos de módulos. Uno ya lo hemos comentado, los procedimientos, y el otro, es el que vamos a tratar ahora, las funciones.

La **división** a más alto nivel que se suele hacer con las funciones, es la que las divide según quien sea el **autor** de las mismas. Así tenemos funciones predifinidas o estándar, y funciones de usuario. Las de usuario las trataremos más adelante. Ahora nos centraremos en las predefinidas.

Las funciones predefinidas, también llamadas **estándar**, son las que el propio lenguaje *La programación modular* pone a disposición del programador. Por ejemplo, si necesitas calcular el valor absoluto de un número, no es necesario que construyas la función, pues ya se dispone de una que lo hace.

Dependiendo del tipo de compilador de *La programación modular* que uses, dispondrás de más o menos funciones estándar, pero siempre tendrás un grupo básico que comparten todos. A continuación se citan unas cuantas funciones de las básicas:

sin(*x*:real)	seno de *x*	**cos**(*x*:real)	coseno de *x*
sqr(*x*:real)	cuadrado de *x*	**sqrt**(*x*:real)	raíz cuadrada de *x*
abs(*x*:real)	valor absoluto de *x*	**ln**(*x*:real)	logaritmo neperiano de *x*
int(*x*:real)	parte entera de *x*	**frac**(*x*:real)	parte decimal de *x*
pred(*x*:tipo ordinal)	predecesor de *x*	**succ**(*x*:tipo ordinal)	sucesor de *x*
pi (*no args.*)	valor de la constante *pi*	**odd**(*x*:integer)	si *x* es o no impar

Funciones definidas por el usuario

Las funciones de usuario son, como su nombre indica, las que el propio usuario declara, de igual manera que declara procedimientos. Las funciones nacen con el propósito de ser subprogramas que siempre tienen que devolver algún valor.

Las dos principales diferencias entre procedimientos y **funciones** son:

- Las funciones siempre devuelven un valor al programa que las invocó.
- Para llamar a un procedimiento se escribe su nombre en el cuerpo del programa, y si los necesita, se incluyen los parámetros entre paréntesis. Para invocar una función es necesario hacerlo en una expresión.

Veamos el esqueleto básico que comparten las funciones:

```
function nombre [(p1,p2,...)] : tipo;
const lista_ctes;
type lista_tipos;
var lista_vars;
(*declaracion de subprogramas*)
begin
  (* cuerpo de la función *)
  nombre := valor_devuelto
end;
```

Comentemos la sintaxis de una función que aparece en el cuadro anterior:

- La lista de parámetros (p1,p2,...) está encerrada entre corchetes porque es **opcional** como en los procedimientos.
- tipo es el tipo del dato que devolverá la función. Así podemos dividir las funciones en lógicas (boolean), enteras (integer), reales (real) y de carácter (char)
- Y al final del cuerpo de la función es **obligatorio** asignarle un valor del tipo devuelto al nombre de la función, porque como ya hemos dicho una función siempre devuelve un valor.

Ahora es el momento de que vayas a un divertido ejemplo sobre las funciones disponible en este tema. Aunque más que ejemplo, se le podría llamar juego, ya que en él tienes la posibilidad de construir varias funciones pudiendo comprobar si son o no correctas.

Un ejemplo sobre las funciones

A continuación tienes un ejemplo para que refuerces lo aprendido en el punto anterior sobre las funciones. El ejemplo es como un pequeño juego con el que se pretende **construir una función**. Para ello, puedes seleccionar lo que quieres que haga la función y las **instrucciones** que la forman.

En la izquierda, además de poder seleccionar el objetivo, tienes un cuadro que contiene las instrucciones disponibles. En la parte derecha tienes tres **selectores** para marcar que instrucciones quieres utilizar, y también tienes un cuadro para comprobar si la función es o no correcta. Suerte!!

nota: para que un programa sea considerado como correcto no basta con que *funcione*, sino que también es necesario que no contenga instrucciones que no aporten nada.

UNIDAD II **VISUAL BASIC ARREGLOS**

1.- INTRODUCCIÓN

Uno de los problemas más comunes en los diversos sistemas de información, es el tratamiento o procesamiento de una gran volumen de datos o de información.

Variables o componentes visuales manejados hasta ahora, no pueden ayudar a resolver este problema.

Las variables usadas hasta ahora reciben propiamente el nombre de variables escalares, porque solo permiten almacenar o procesar un dato a la vez.

No confundir esto, con el tipo de dato o rango de datos que una variable tiene la capacidad de almacenar.

Por ejemplo si se quiere almacenar nombre y edad de 15 personas, con el método tradicional se ocuparan 30 variables o 30 componentes visuales, y solo es nombre y edad de 15 personas, agregar mas datos y mas personas y ya es tiempo de empezar a analizar otro tipo de variables y de componentes.

Es decir, en problemas que exigen manejar mucha información o datos a la vez, variables escalares o componentes visuales de manipulación de datos normales (text, label, etc.), no son suficientes, ya que su principal problema es que solo permiten almacenar un dato a la vez.

Se ocupa entonces variables o sus correspondientes componentes visuales que sean capaces de almacenar y manipular conjuntos de datos a la vez.

Variables de tipo **arreglo** y sus correspondientes componentes visuales, si permiten almacenar y procesar conjuntos de datos del mismo tipo a la vez.

Cada dato dentro del arreglo, se llama elemento del arreglo y se simboliza y procesa (captura ,operación ,despliegue), usando el nombre del arreglo respectivo y un subíndice indicando la posición relativa del elemento con respecto a los demás elementos del arreglo, ej:

Juan
Pedro------> Nombres(2)
José
Ana--------> Nombres(4)
Carmen

18-----> Edad(1)
20
25
30----> Edad(4)

Sin embargo sus problemas son similares a los de variables normales, es decir hay que declararlos, capturarlos, hacer operaciones con ellos, desplegarlos, compararlos, etc.

Para propósitos del aprendizaje se analiza o clasifican en tres grupos diferentes los arreglos que ofrece Visual Basic, ellos son;

1.- Arreglos tradicionales (internos dentro del programa) 2.- Arreglos dinámicos (internos) 3.- Componentes Visuales de tipo Arreglo

1.- ARREGLOS TRADICIONALES VISUAL BASIC

En programación tradicional siempre se manejan dos tipos de arreglos, los arreglos tipo listas, vectores o unidimensionales y los arreglos tipo tablas, cuadros, concentrados, matrices o bidimensionales, en ambos casos son variables que permiten almacenar un conjunto de datos del mismo tipo a la vez, su diferencia es en la cantidad de columnas que cada uno de estos tipos contiene, como en los siguientes ejemplos;

2.- ARREGLOS TIPO LISTA

Juan
Pedro---->Nombres(2)
José
Ana-----> Nombres(4)
Carmen

18----> Edad(1)
20E
25
30----> Edad(4)

3.- ARREGLOS TIPO TABLAS

CIA ACME
INGRESOS POR VENTAS
(MILES DE $)
FEB MAR ABR MAY
SUC A 10 12 15 10 9
SUC B 8 7 5 9 6
SUC C 11 18 20 14 17

---

contaduría pública y de sistemas
CONCENTRADO DE CALIF
MAT FIS QUIM HIST
JUAN 5 5 5 5
JOSE 4 4 4 4
PEDRO 3 3 3 3
ANA 9 9 9 9

Como se observa, la diferencia principal entre un arreglo tipo lista, y un arreglo tipo tabla, son las cantidades de columnas que contienen.

NOTA IMPORTANTE.- LOS CONCEPTOS MANEJADOS AQUÍ ESTAN ENFOCADOS A LOS SISTEMAS DE INFORMACIÓN CONTABLES FINANCIEROS ADMINISTRATIVOS.

3.- ARREGLOS TIPO LISTA

Un arreglo tipo lista se define como una variable que permite almacenar un conjunto de datos del mismo tipo organizados en una sola columna y uno o más renglones.

También reciben el nombre de vectores en álgebra, o arreglos unidimensionales en programación.

Los procesos normales con una lista o con sus elementos, incluyen declarar toda la lista, capturar sus elementos, desplegarlos, realizar operaciones con ellos, desplegarlos, etc.

Para declarar una lista se usa el siguiente formato;

DIM nomlista(1(uno) TO Cant elementos o reng) AS tipo dato

ejemplos;

DIM EDAD(1 TO 12) AS INTEGER

DIM SUELDOS(1 TO 10) AS SINGLE

DIM MUNICIPIOS(1 TO 5) AS STRING * 20

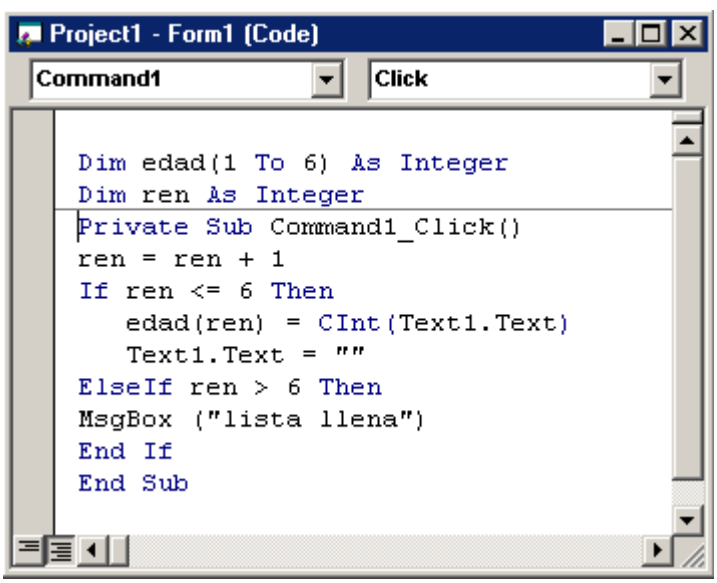
Notas:

Declaración.- Es necesario recordar, que la declaración de un arreglo tipo lista se puede hacer de dos maneras diferentes, dependiendo de si solo se usa un botón de órdenes en la pantalla, o si dos o más botones de órdenes estarán procesando el arreglo, el segundo caso, es el más común.

Si un solo botón, en toda la ventana va a realizar, todos los procesos (declaración, captura, operaciones, comparaciones, despliegue), con la lista, solo hacer la declaración de la lista, al principio del evento click, como lo muestra el programa ejemplo.

Para capturar se deberá usar un textbox y un botón de comando con el siguiente código que estará alimentando la lista en memoria:

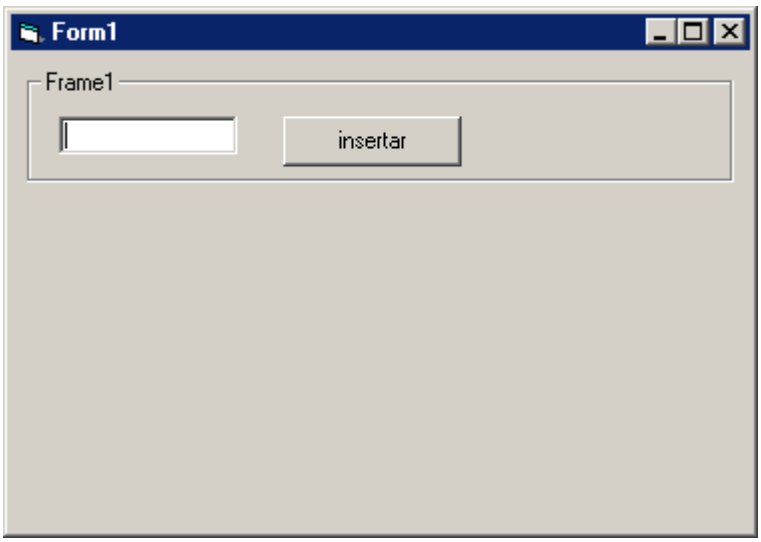
a. Programa



Observar que la declaración de la lista y la variable de control va fuera del click del botón, para que todos los botones de comando que se pongan en el programa las puedan usar.

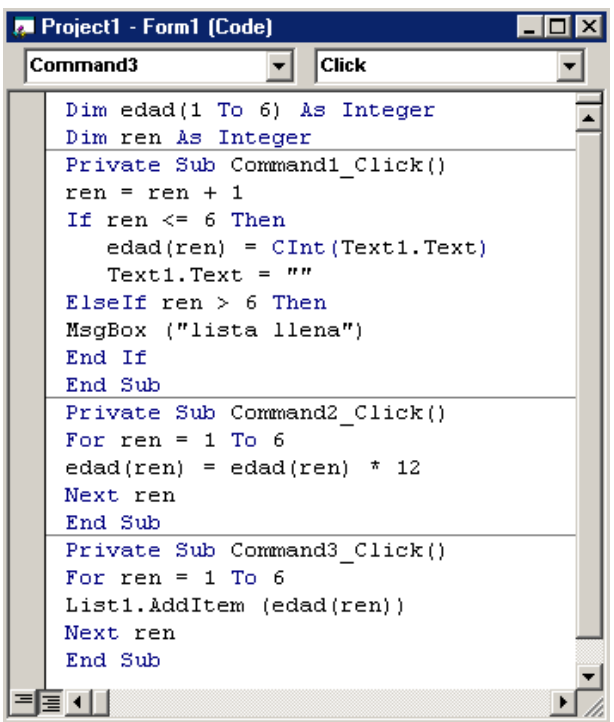
Se usa un if para evitar que se capturen datos de mas, y un segundo if con un mensagebox para avisar que ya se lleno la lista.

Pantalla de corrida



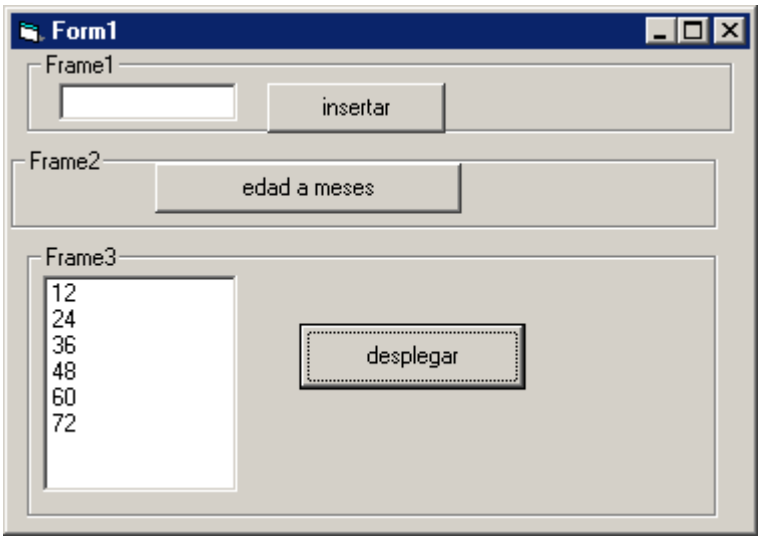
Para el caso de operaciones y comparaciones con todos los elementos de la lista a la vez, se deberá usar un ciclo for, con una variable entera llamada renglón, misma que también se usara como índice de la lista, el despliegue de la lista usara un control Listbox y el método AddItem(), para este ejemplo se pretende convertir las edades a meses:

a. Programa :



Recordar que todos los datos internos de la lista estarán almacenados en la memoria ram del computador, para desplegados se usara un componente visual que permite manipular un conjunto de datos a la vez, el ListBox, pero se tiene que usar un ciclo for para ir añadiendo o agregando elemento por elemento como se observa en el problema ejemplo que se ha venido desarrollando, en este caso se quiere desplegar las cuatro edades convertidas a meses;

B) Pantalla de salida:



TAREAS PROGRAMACION VISUAL BASIC

1.- Capturar y desplegar 5 precios de productos cualesquiera, usando dos frames, uno para capturar y uno para desplegar.

2.- Capturar 4 sueldos en un panel, desplegarlos aumentados en un 25% en otro panel.

3.- Capturar los datos de 5 productos comprados en una tienda, incluyendo nombre, precio y cantidad en sus 3 listas respectivas, después calcular una cuarta lista con el gasto total por cada producto desplegarlo todo en un segundo panel e incluir también el gran total.

4.- Capturar en una lista solamente 6 números múltiplos de 5, se debe de estar capture y capture números hasta que se completen los 6 múltiplos de 5.

4.- VISUAL BASIC SORTEOS U ORDENAMIENTOS

Un proceso muy común con listas, es el llamado sorteo u ordenamiento.

Este proceso consiste en reacomodar los elementos de la lista en un nuevo orden, de acuerdo a algún criterio.

Lista Original Listas Ordenadas

15	2	15
2	8	10
10	10	8
8	15	2

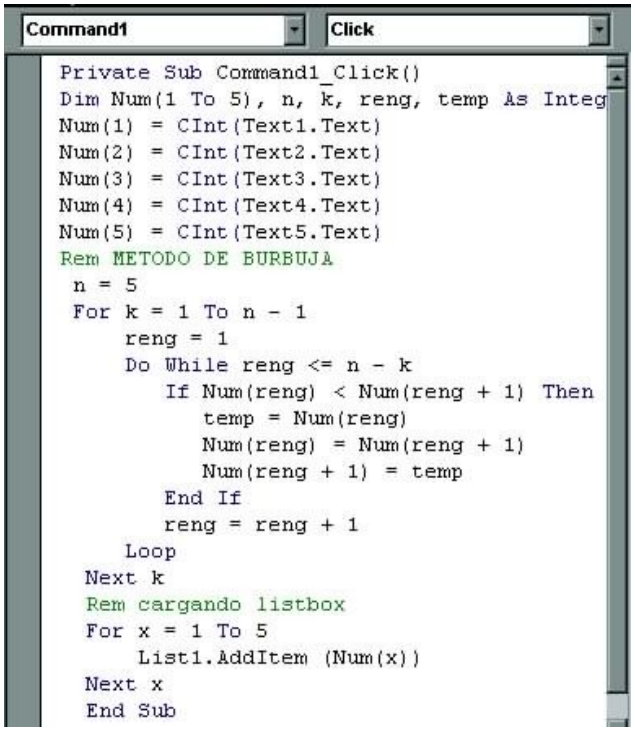
Sorteo creciente y decreciente

Existen muchos métodos u algoritmos de sorteos, el mas común de ellos, es el denominado algoritmo de burbuja, que se basa en el siguiente algoritmo:

N=CANTIDAD DE ELEMENTOS DE LA LISTA
FOR K = 1 TO N-1
RENGLÓN = 1
DO WHILE RENGLÓN <= N - K
IF LISTA(RENGLON) > LISTA(RENGLON + 1) THEN
TEMP = LISTA(RENGLON)
LISTA(RENGLON)=LISTA(RENGLON + 1)
LISTA(RENGLON + 1) = TEMP
END IF
RENGLÓN = RENGLÓN + 1
LOOP
NEXT K

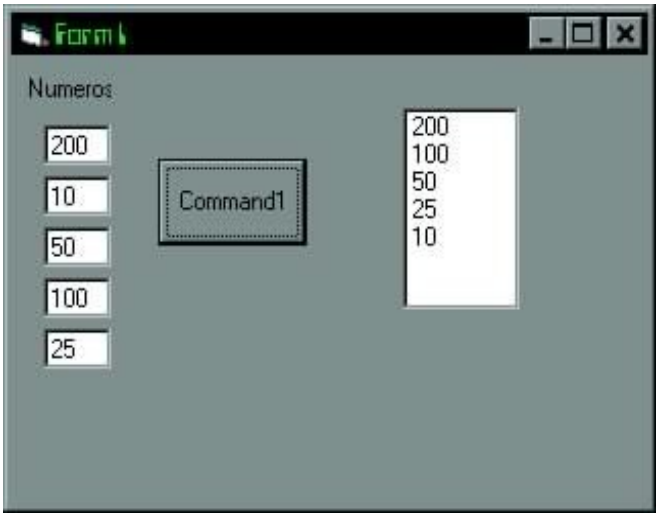
Las notas a considerar con respecto al algoritmo son:

- Las variables n, k, renglón, son variables de control y deberán ser declaradas de tipo integer.
- La variable temp, deberá ser declarada de acuerdo al tipo de dato de los elementos de la lista.
- Todas las referencias a LISTA, deberán ser cambiadas por el nombre verdadero de la lista real.
- Es el símbolo del if, quien determina el tipo de sorteo, es decir, (>)ascendente, (<) descendente. ejemplo, ordenar 6 números cualesquiera:



A) Código:

B) corrida:



TAREAS VISUAL BASIC

1.- PROGRAMACION VISUAL BASIC ORDENAR ASCENDENTEMENTE 5 MATRICULAS

2.- PROGRAMACION VISUAL BASIC ORDENAR DESCENDENTEMENTE 6 CIUDADES

3.- PROGRAMACION VISUAL BASIC ORDENAR A CRITERIO DEL USUARIO 7 ANIMALITOS

5.- ARREGLOS TIPO TABLA

Un arreglo tipo tabla se define como un conjunto de **datos del mismo tipo** organizados en dos o mas columnas y uno o mas renglones.

Para declarar un arreglo tipo tabla se usa el siguiente formato:

DIM NOMTABLA(1 TO CANTRENG, 1 TO CANTCOL) AS TIPODATO

EJEMPLOS:

DIM VTAS(1 TO 3, 1 TO 5) AS SINGLE

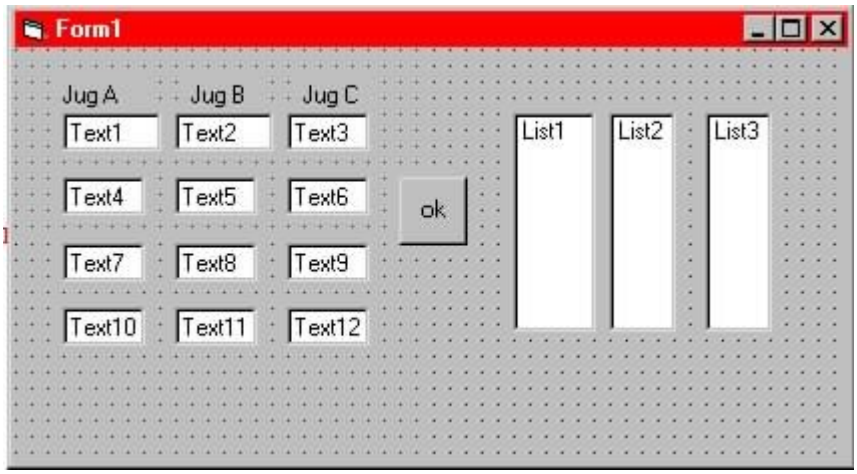
DIM CALIF(1 TO 30, 1 TO 6) AS INTEGER

Solo recordar que en capturas, se deberán usar tantos componentes Text como celdas tenga la tabla y en despliegue usar tantos controles ListBox como columnas tenga la tabla, estos métodos son provisionales mientras se analizan los componentes visuales apropiados y respectivos.

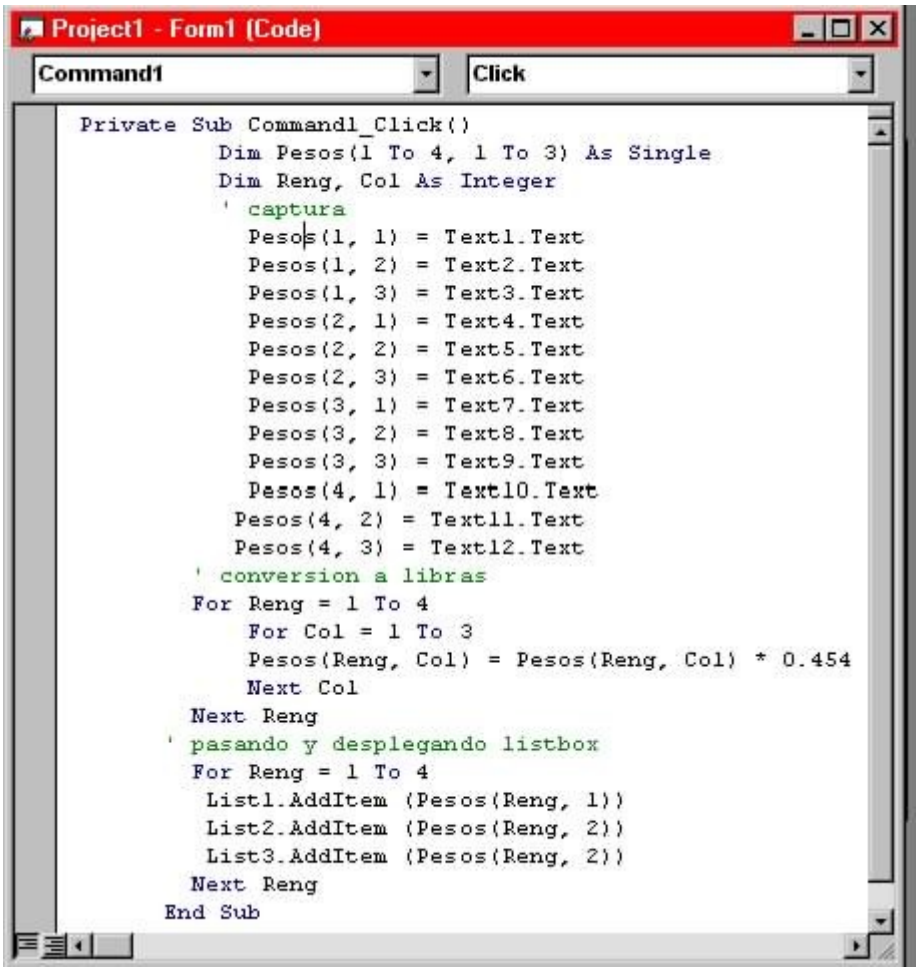
Para procesar (recordar solo operaciones y comparaciones) internamente todos los elementos de la tabla se ocupan dos ciclos for, uno externo para controlar renglón y uno interno para controlar columna.

Problema ejemplo, capturar una tabla que nos muestre el peso en lbs de los tres jugadores claves de 4 equipos de fútbol, desplegarlos en otra tabla pero convertidos a kg. (una libra = .454 kg.), el programa y la pantalla de salida son;

A. Diseño:

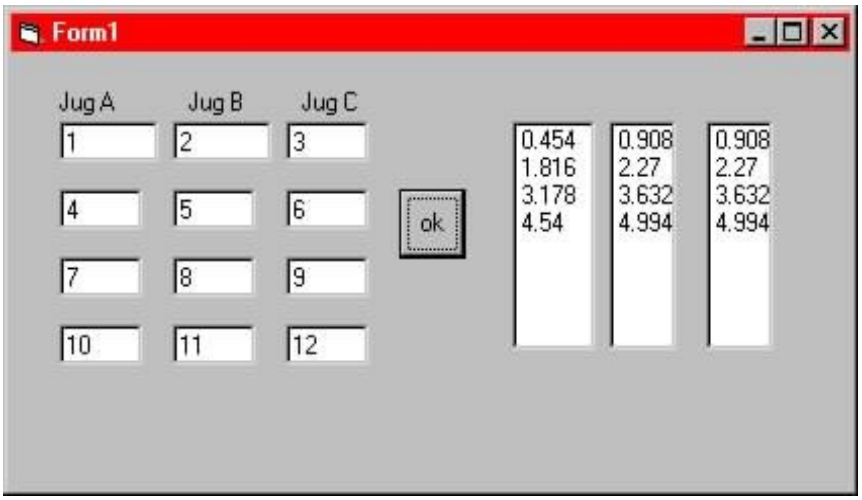


B. Código:



Observar que en procesos, son dos for's, y en despliegue, solo un for.

C)Salida:



Recordar que en este nivel de instrucción, solo se pretende, entender los conceptos asociados a arreglos, mejores maneras de procesarlos existen, como se vera mas adelante, pero ya se puede:

TAREAS VISUAL BASIC

1.- TAREAS PROGRAMACION VISUAL BASIC CONSTRUIR UN CUADRO QUE CONTENGA LOS COSTOS FIJOS DE CUATRO PRODUCTOS CUALESQUIERA, QUE SE PRODUCEN EN TRES PLANTAS DIFERENTES DE UNA EMPRESA MAQUILADORA.

2.- TAREAS PROGRAMACION VISUAL BASIC CONSTRUIR UN CUADRO QUE CONTENGA LOS INGRESOS MENSUALES POR VENTAS DURANTE LOS TRES PRIMEROS MESES DEL AÑO DE CUATRO SUCURSALES DE UNA CADENA DE AUTO REFACCIONES, AGREGAR AL FINAL UNA LISTA QUE MUESTRE LOS INGRESOS MENSUALES TOTALES POR MESES Y UNA SEGUNDA LISTA QUE MUESTRE LOS INGRESOS MENSUALES TOTALES POR SUCURSAL.

3.-TAREAS PROGRAMACION VISUAL BASIC CONSTRUIR UN CUADRO QUE CONTENGA LAS COMISIONES GANADAS POR TRES VENDEDORES, DE LOS 5 TIPOS DE LINEA BLANCA DE CONOCIDA MUEBLERIA, ADEMAS LISTAS DE COMISIONES TOTALES Y PROMEDIOS GANADAS POR LOS VENDEDORES, ASI COMO LISTAS DE COMISIONES TOTALES Y PROMEDIOS POR TIPO DE LINEA BLANCA.

ANALIZAR ESTE CODIGO:

```
' PARA TOTALES Y PROMEDIOS POR RENGLON
FOR R = 1 TO 4
FOR C = 1 TO 3
TOTRENG(R) = TOTRENG(R) + TABLA(R,C)
NEXT C
PROMRENG(R) = TOTRENG(R)/3
NEXT R
'PARA TOTALES Y PROMEDIOS POR COLUMNA
FOR C = 1 TO 3
```

FOR R = 1 TO 4

TOTCOL(C)=TOTCOL(C) + TABLA(R,C)

NEXT R

PROMCOL(C) = TOTCOL(C) / 4

NEXT C

SUGERENCIA: CONSTRUIR PRIMERO LOS CUADROS EN PAPEL.

6.- VISUAL BASIC ARREGLOS DINÁMICOS

Hasta ahora los arreglos vistos son de tipo estáticos, es decir, son de tamaño fijo, ya definido o declarado.

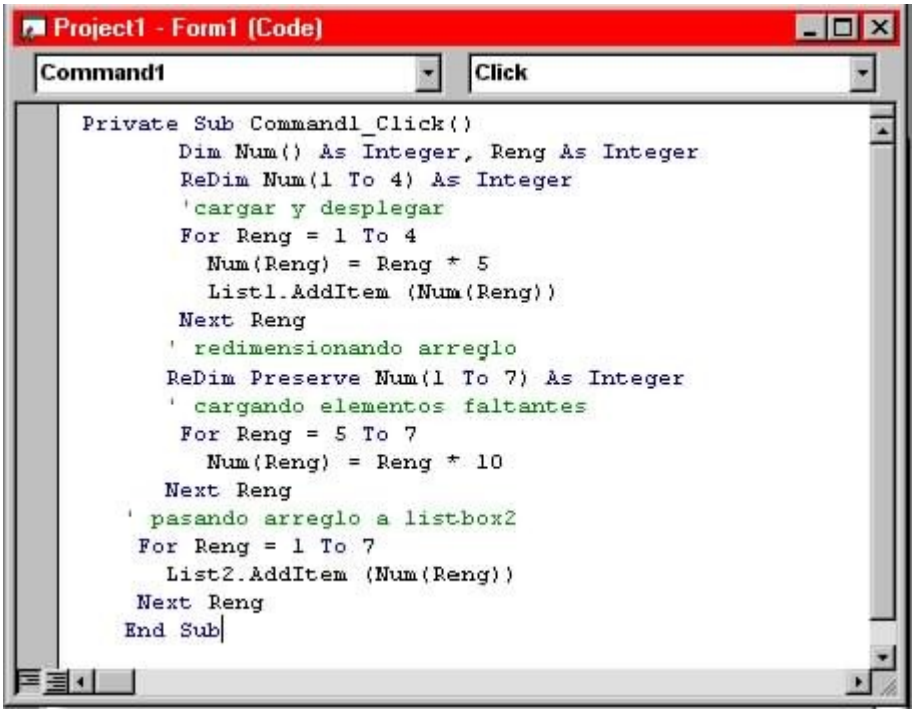
Visual Basic, contiene los mecanismos apropiados para crear arreglos dnamicos, es decir, arreglos que tienen la capacidad de ir creciendo, para ajustarse a las necesidades del problema, incluso si perder los elementos de datos que ya contenga.

Para esto:

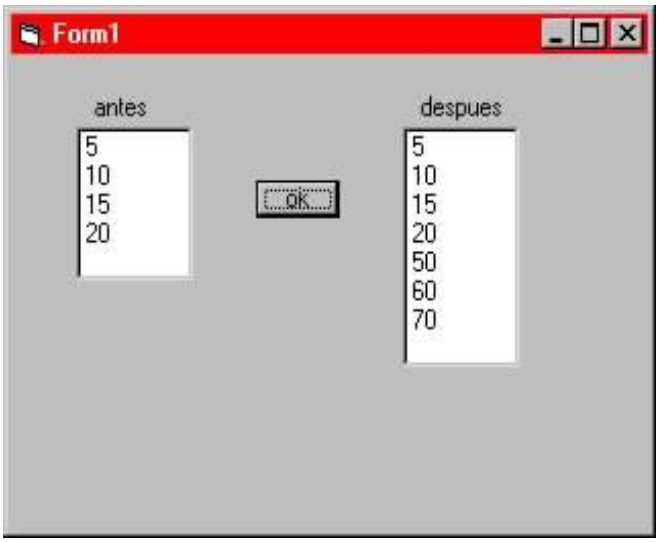
A) Declarar primero el arreglo sin tamaño fijo:

B) Usar la instrucción siguiente cada vez que se desee redimensionar.

REDIM [PRESERVE] NOMARREGLO(1 TO REN,[1 TO COL]) AS TIPODATO



Pantalla de salida:



7.- CONTROLES VISUALES DE TIPO ARREGLO

8.- CONTROLES VISUALES TIPO LISTA

9.- CONTROL LISTBOX VISUAL BASIC

Este componente permite procesar visualmente un conjunto de elementos de tipo string.

Su primer y mas importante aspecto a recordar, cuando se procese o programe, es que el primer indice de la lista, es el indice numero 0(cero).

Este componente, contiene muchas propiedades y métodos que facilitan el trabajo con datos, entre ellas se encuentran:

PROPIEDAD ACCIÓN O SIGNIFICADO

AddItem(item, index) Inserta un elemento en posición indicada

Columns Para desplegar en una o mas columnas

Clear Elimina todos los elementos de la lista

List(index) Para accesar un elemento en posición

ListCount Regresa la cantidad de elementos en lista

RemoveItem(index) Elimina ítem en posición indicada

Sorted=true Ordena los elementos de la lista usada solo al tiempo de diseño

---

Notas:

a. Capturas: Solo se ocupara un Text, el evento click del TextBox, y el método AddItem del ListBox.

b. Procesos: Se ocupara un ciclo for , y los métodos list y listcount de ListBox

c. Despliegues: No se ocupa, porque todos los cambios son visibles.

d) Despliegues: Pero si se quiere pasar de un ListBox a otro ListBox, entonces ciclo for, list y listcount

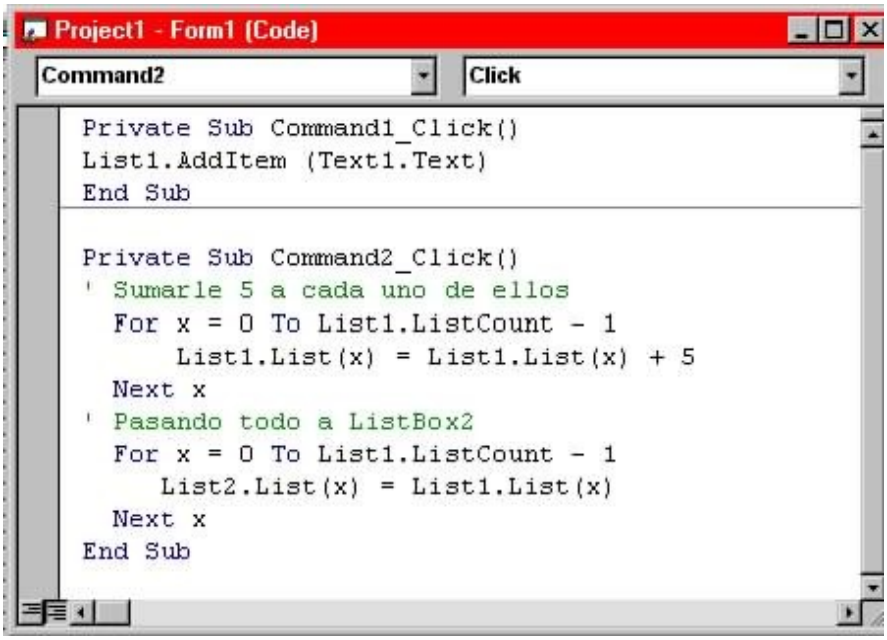
Ejemplo:

1ro.- Capturar números enteros en un ListBox

2do.- Sumarles 5 a cada uno de ellos

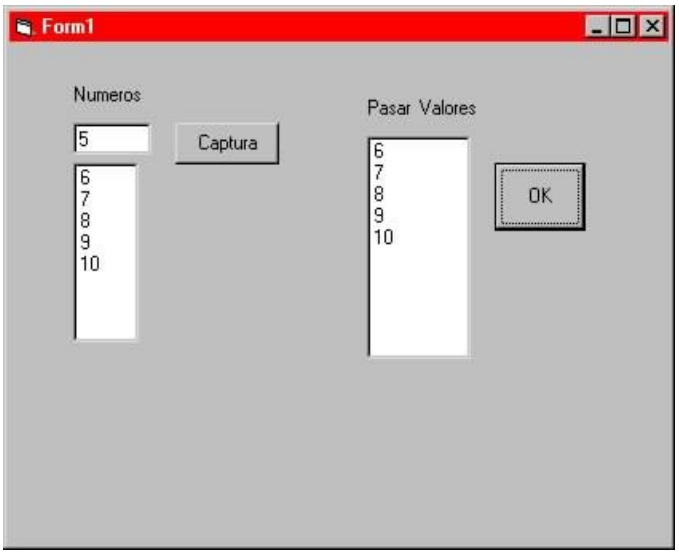
3ro.- Pasarlos a un segundo ListBox

a. Código Fuente



Recordar que el primer índice en un ListBox es el cero, por eso el ciclo va desde el cero, hasta la cantidad de elementos menos uno.

b. Pantalla de salida:



TAREAS PROGRAMACION VISUAL BASIC

1.- CAPTURAR EN UNA LISTA LOS SUELDOS DE 6 EMPLEADOS DE UN CASINO Y DESPLEGARLOS EN UNA SEGUNDA LISTA AUMENTADOS EN UN 30%

2.- CAPTURAR EN UNA LISTA LOS PESOS EN KILOGRAMOS DE 6 PERSONAS DESPLEGARLOS EN UNA SEGUNDA LISTA CONVERTIDOS A LIBRAS Y ADEMAS SOLO LOS MAYORES DE 100 LIBRAS.

3.- CAPTURAR EN SUS 4 LISTAS RESPECTIVAS MATRICULA, NOMBRE Y DOS CALIFICACIONES DE 5 ALUMNOS, DESPUÉS CALCULAR UNA LISTA DE PROMEDIOS DE CALIFICACIONES.

4.-CAPTURAR EN SUS LISTAS RESPECTIVAS NUMEMPLEADO, NOMEMPLEADO, DÍAS TRABAJADOS Y SUELDO DIARIO DE 5 EMPLEADOS, DESPLEGAR EN OTRA PANTALLA O PANEL LA NOMINA PERO SOLO DE AQUELLOS EMPLEADOS QUE GANAN MAS DE $300.00 A LA SEMANA.

10.- CONTROL MSFLEXGRID VISUAL BASIC

Este control, no aparece entre los veinte controles de default que trae Visual Basic, importarlo al Tool Box, siguiendo el procedimiento que se dio en el ultimo tema de la primera UNIDAD VISUAL BASIC (Componente Animatión), la librería que lo contiene se llama Microsoft FlexGrid Control 5.0

Este componente es de los mas importantes, para el procesamiento de muchos datos, permite concentrar, procesar y mostrar gran cantidad de información para la vista del usuario.

Este componente presenta, manipula y procesa conjuntos de datos de tipo strings en forma tabular, es decir en forma de tablas, matrices, cuadros concentrados, ejemplo;

CIA ACME

INGRESOS POR VENTAS MENSUALES

MILLONES DE PESOS

ENE FEB MAR ABR

SUC A 1 2 3 4

SUC B 5 6 4 5

SUC C 6 7 8 9

Recordar que son los datos numéricos internos quienes se procesan (es decir, se capturan, se realizan operaciones con ellos, se despliegan, etc.), es la información externa quien le da sentido.

Algunas de sus propiedades y métodos mas interesantes son:

Cols.- Determina la cantidad de columnas que contendrá la tabla.

Recordar que para efectos de programación, la primera de ellas es la columna 0.

Rows.- Determina la cantidad de renglones que contendrá la tabla.

Recordar que para efectos de programación, el primero de ellos es el renglón 0.

Fixedcols , Fixedrows.- Determinan la cantidad de columnas y renglones fijos o de encabezado, estas propiedades ponerlas en 0.

Col, Row.- Al tiempo de ejecución del programa, regresan la posición de la celda actual, no confundir con Cols, Rows.

TextMatrix(Row,Col) = String, Es la propiedad mas importante, porque permite el acceso a cualquier celda de la tabla, ejemplos.

ejemplos.:

MsFlexGrid1.TextMatrix(2,4) = "PATO"

Observar que para acceder y manipular una celda, se debe primero indicar, el renglón y la columna adecuadas.

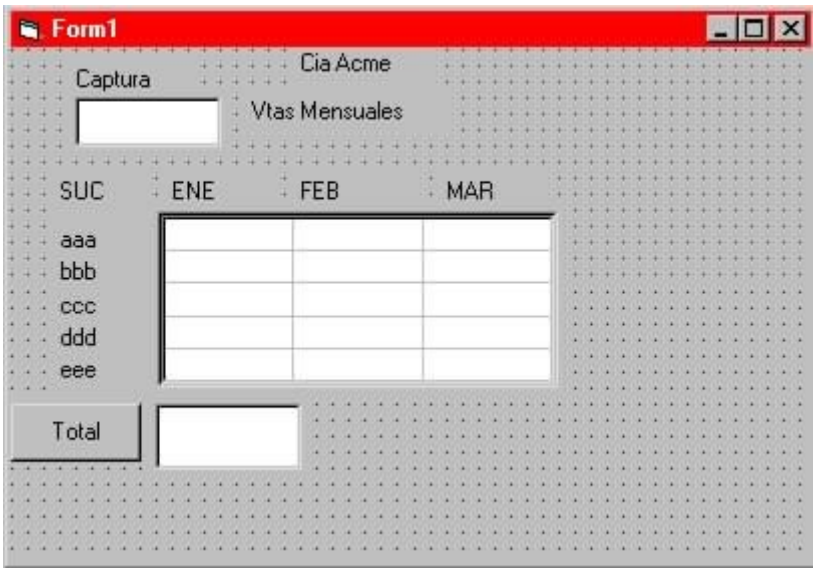
Otro aspecto importante a recordar, es que MSFlexGrid no permite edición directa por parte del usuario de sus celdas, por ese motivo se usara un componente externo TextBox para capturas, así como el evento click de MSFlexGrid.

Para procesar **todos** los elementos de la tabla, solo recordar que se deben usar dos ciclos for, **uno externo para controlar renglones, y uno interno para controlar columna**.

Si solo se quiere procesar un solo renglón o columna, entonces solo se ocupara el ciclo **contrario,** y el renglón o columna original se darán como constantes, ver programa ejemplo.

Ejemplo, Capturar una tabla de ingresos por ventas de la CIA Acme y obtener el total de las ventas del primer mes:

a. Pantalla de Diseño:

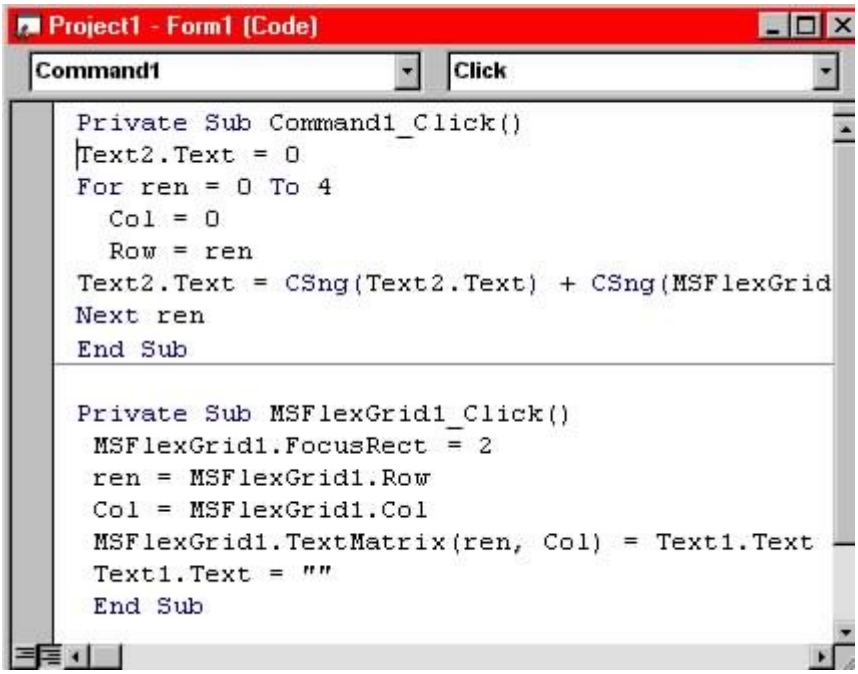


El Click del MSFlexGrid, usa la propiedad FocusRect, para graficar un rectángulo alrededor de la celda.

Se usa la propiedad MatrixText, para cargar la celda con el dato que se encuentra en el TextBox, observar que la posición, renglón, columna de MatrixText se obtienen usando las propiedades Row Y Col, al final se deja en blanco la caja TextBox, para que el usuario capture otro dato.

El Click del Command, primero se asegura de que este en 0(cero) la caja Text2 y luego se usa un ciclo renglón, porque como ya se indico , se quiere procesar una sola columna, misma que se dejo como constante, dentro de la operación

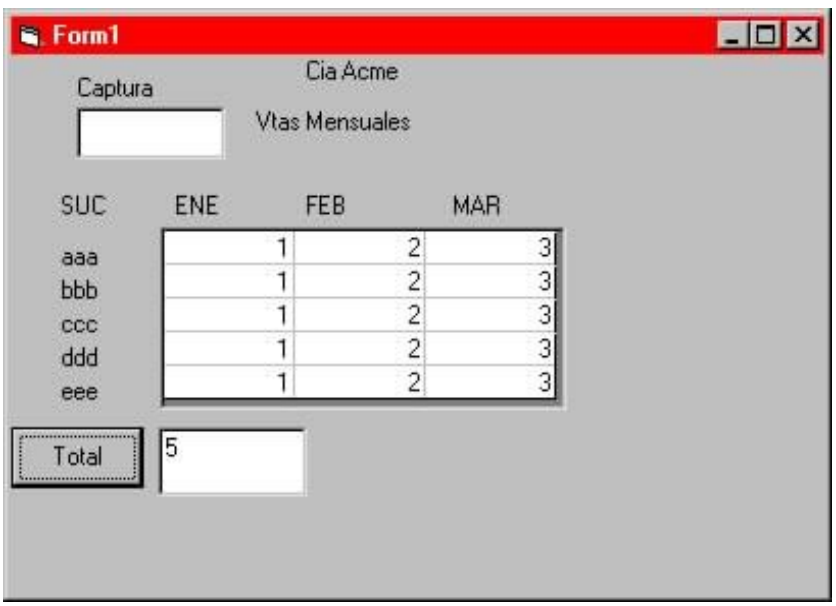
b. Código:



.

Se esta usando el concepto de acumulador(Acum=Acum+NvoDato), para acumular el resultado.

c. Pantalla de Ejecución:



Un proceso muy común con tablas, cuadros y concentrados es agregarles listas de totales y promedios ya sea por columna o por renglón, o ambas , por ejemplo;

CIA ACME

INGRESOS MENSUALES

(MILES DE PESOS)

ENE FEB MARZO TOTALSUC PROMSUC

SUC A 1 2 3 6 2

SUC B 4 5 6 15 5

SUC C 7 8 9 24 8

SUC D 10 11 12 33 11

TOTMES 22 26 30

PROMMES 5.5 6.5 7.8

En este ejemplo aparte de la tabla se ocupan 4 listas, dos para totales y dos para promedios.

El Codigo, para este tipo de problemas ya se dio en el tema de arreglos normales tipo tabla.

TAREAS PROGRAMACION VISUAL BASIC

1.- Construir un concentrado que despliegue los costos fijos de tres diversos productos que se fabrican en cuatro sucursales de una empresa MAQUILADORA.

2.- Construir un concentrado que contenga los ingresos por ventas mensuales de los 4 primeros meses del año de tres sucursales de una cadena refaccionaría, agregar listas de ingresos totales por mes e ingresos promedios por sucursal.

3.- Construir un cuadro que contenga las calificaciones de 5 materias de cuatro alumnos cualesquiera, incluir promedios de calificaciones por materia y por alumno.

CUESTIONARIO

1.- Cuando se usan arreglos

2.- Que es variable escalar

3.- Que es variable arreglo

4.- Que es elemento de un arreglo

5.- Como se simboliza o manipula un elemento de un arreglo

6.- Que es un arreglo tipo lista

7.- Que es un arreglo tipo tabla

8.- Formato para declarar una lista

9.- Cuantos for's se usan para manipular una lista

10.- Cuando se usa PUBLIC en la declaración de una lista

11.- Que es sorteo

12.- Algoritmo de sorteo usado en este curso

13.- Formato de declaración de un arreglo tipo tabla

14.- Cuantos ciclos se usan para manipular una tabla

15.- Que son arreglos dinámicos

16.- Como se declaran arreglos dinámicos en V.B.

17.- Que es control ListBox

18.- Numero del primer indice en un control ListBox

19.- Cinco propiedades importantes del control ListBox

20.- Que es control MSFlexGrid

21.- Propiedades importantes del control MSFlexGrid

UNIDAD III ALGORITMOS DE ORDENAMIENTO

¿Qué es ordenamiento?

Es la operación de arreglar los registros de una tabla en algún orden secuencial de acuerdo a un criterio de ordenamiento.

El ordenamiento se efectúa con base en el valor de algún campo en un registro.

El propósito principal de un ordenamiento es el de facilitar las búsquedas de los miembros del conjunto ordenado.

Ej. de ordenamientos:

Dir. telefónico, tablas de contenido, bibliotecas y diccionarios, etc.

El ordenar un grupo de datos significa mover los datos o sus referencias para que queden en una secuencia tal que represente un orden, el cual puede ser numérico, alfabético o incluso alfanumérico, ascendente o descendente.

¿Cuándo conviene usar un método de ordenamiento?

Cuando se requiere hacer una cantidad considerable de búsquedas y es importante el factor tiempo.

Tipos de ordenamientos:

Los 2 tipos de ordenamientos que se pueden realizar son: los internos y los externos.

Los internos:

Son aquellos en los que los valores a ordenar están en memoria principal, por lo que se asume que el tiempo que se requiere para acceder cualquier elemento sea el mismo (a[1], a[500], etc).

Los externos:

Son aquellos en los que los valores a ordenar están en memoria secundaria (disco, cinta, cilindro magnético, etc), por lo que se asume que el tiempo que se requiere para acceder a cualquier elemento depende de la última posición accesada (posición 1, posición 500, etc).

Eficiencia en tiempo de ejecución:

Una medida de eficiencia es:

Contar el # de comparaciones (C)

Contar el # de movimientos de items (M)

Estos están en función de el #(n) de items a ser ordenados.

Un "buen algoritmo" de ordenamiento requiere de un orden *nlogn* comparaciones.

La eficiencia de los algoritmos se mide por el número de comparaciones e intercambios que tienen que hacer, es decir, se toma n como el número de elementos que tiene el arreglo o vector a ordenar y se dice que un algoritmo realiza O(n2) comparaciones cuando compara n veces los n elementos, n x n = n2

Algoritmos de ordenamiento:

Internos:

- Inserción directa.
  - Inserción directa.
  - Inserción binaria.
- Selección directa.
  - Selección directa.
- Intercambio directo.

- Burbuja.
- Shake.
- Inserción disminución incremental.
- Shell.
- Ordenamiento de árbol.
- Heap.
- Tournament.
- Sort particionado.
- Quick sort.
- Merge sort.
- Radix sort.
- Cálculo de dirección

Externos:

- Straight merging.
- Natural merging.
- Balanced multiway merging.
- Polyphase sort.
- Distribution of initial runs.

Clasificación de los algoritmos de ordenamiento de información:

El hecho de que la información está ordenada, nos sirve para poder encontrarla y accesarla de manera más eficiente ya que de lo contrario se tendría que hacer de manera secuencial.

A continuación se describirán 4 grupos de algoritmos para ordenar información:

Algoritmos de inserción:

En este tipo de algoritmo los elementos que van a ser ordenados son considerados uno a la vez. Cada elemento es INSERTADO en la posición apropiada con respecto al resto de los elementos ya ordenados.

Entre estos algoritmos se encuentran el de INSERCION DIRECTA, SHELL SORT, INSERCION BINARIA y HASHING.

Algoritmos de intercambio:

En este tipo de algoritmos se toman los elementos de dos en dos, se comparan y se INTERCAMBIAN si no están en el orden adecuado. Este proceso se repite hasta que se ha analizado todo el conjunto de elementos y ya no hay intercambios.

Entre estos algoritmos se encuentran el BURBUJA y QUICK SORT.

Algoritmos de selección:

En este tipo de algoritmos se SELECCIONA o se busca el elemento más pequeño (o más grande) de todo el conjunto de elementos y se coloca en su posición adecuada. Este proceso se repite para el resto de los elementos hasta que todos son analizados.

Entre estos algoritmos se encuentra el de SELECCION DIRECTA.

Algoritmos de enumeración:

En este tipo de algoritmos cada elemento es comparado contra los demás. En la comparación se cuenta cuántos elementos son más pequeños que el elemento que se está analizando, generando así una ENUMERACION. El número generado para cada elemento indicará su posición.

Los métodos simples son: Inserción (o por inserción directa), selección, burbuja y shell, en dónde el último es una extensión al método de inserción, siendo más rápido. Los métodos más complejos son el quick-sort (ordenación rápida) y el heap sort.

A continuación se mostrarán los métodos de ordenamiento más simples.

METODO DE INSERCIÓN.

Este método toma cada elemento del arreglo para ser ordenado y lo compara con los que se encuentran en posiciones anteriores a la de él dentro del arreglo. Si resulta que el elemento con el que se está comparando es mayor que el elemento a ordenar, se recorre hacia la siguiente posición superior. Si por el contrario, resulta que el elemento con el que se está comparando es menor que el elemento a ordenar, se detiene el proceso de comparación pues se encontró que el elemento ya está ordenado y se coloca en su posición (que es la siguiente a la del último número con el que se comparó).

Procedimiento ***Insertion Sort***

Este procedimiento recibe el arreglo de datos a ordenar **a[]** y altera las posiciones de sus elementos hasta dejarlos ordenados de menor a mayor. **N** representa el número de elementos que contiene a[].

paso 1: [Para cada pos. del arreglo]	For i <- 2 to N do
paso 2: [Inicializa v y j]	v <- a[i] ; j <- i.
paso 3: [Compara v con los anteriores]	While a[j-1] > v AND j>1 do
paso 4: [Recorre los datos mayores]	Set a[j] <- a[j-1],
paso 5: [Decrementa j]	set j <- j-1.
paso 5: [Inserta v en su posición]	Set a[j] <- v.
paso 6: [Fin]	End.

Ejemplo:

Si el arreglo a ordenar es a = ['a','s','o','r','t','i','n','g','e','x','a','m','p','l','e'], el algoritmo va a recorrer el arreglo de izquierda a derecha. Primero toma el segundo dato 's' y lo asigna a *v* y *i* toma el valor de la posición actual de *v*.

Luego compara esta 's' con lo que hay en la posición j-1, es decir, con 'a'. Debido a que 's' no es menor que 'a' no sucede nada y avanza i.

Ahora v toma el valor 'o' y lo compara con 's', como es menor recorre a la 's' a la posición de la 'o'; decrementa j, la cual ahora tiene la posición en dónde estaba la 's'; compara a 'o' con a[j-1] , es decir, con 'a'. Como no es menor que la 'a' sale del for y pone la 'o' en la posición a[j]. El resultado hasta este punto es el arreglo siguiente: a = ['a','o','s','r',....]

Así se continúa y el resultado final es el arreglo ordenado :

a = ['a','a','e','e','g','i','l','m','n','o','p','r','s','t','x']

MÉTODO DE SELECCIÓN.

El método de ordenamiento por selección consiste en encontrar el menor de todos los elementos del arreglo e intercambiarlo con el que está en la primera posición. Luego el segundo mas pequeño, y así sucesivamente hasta ordenar todo el arreglo.

Procedimiento ***Selection Sort***

paso 1: [Para cada pos. del arreglo]	For i <- 1 to N do
paso 2: [Inicializa la pos. del menor]	menor <- i

paso 3: [Recorre todo el arreglo] For j <- i+1 to N do
paso 4: [Si a[j] es menor] If a[j] < a[menor] then
paso 5: [Reasigna el apuntador al menor] min = j
paso 6: [Intercambia los datos de la pos. min y posición i]; Swap(a, min, j).
paso 7: [Fin] End.

Ejemplo:

El arreglo a ordenar es a = ['a','s','o','r','t','i','n','g','e','x','a','m','p','l','e'].

Se empieza por recorrer el arreglo hasta encontrar el menor elemento. En este caso el menor elemento es la primera 'a'. De manera que no ocurre ningún cambio. Luego se procede a buscar el siguiente elemento y se encuentra la segunda 'a'.

Esta se intercambia con el dato que está en la segunda posición, la 's', quedando el arreglo así después de dos recorridos: a = ['a','a','o','r','t','i','n','g','e','x','s','m','p','l','e'].

El siguiente elemento, el tercero en orden de menor mayor es la primera 'e', la cual se intercambia con lo que está en la tercera posición, o sea, la 'o'. Le sigue la segunda 's', la cual es intercambiada con la 'r'.

El arreglo ahora se ve de la siguiente manera: a = ['a','a','e','e','t','i','n','g','o','x','s','m','p','l','r'].

De esta manera se va buscando el elemento que debe ir en la siguiente posición hasta ordenar todo el arreglo.

El número de comparaciones que realiza este algoritmo es :

Para el primer elemento se comparan n-1 datos, en general para el elemento i-ésimo se hacen n-i comparaciones, por lo tanto, el total de comparaciones es:

la sumatoria para i de 1 a n-1 (n-i) = 1/2 n (n-1).

MÉTODO BURBUJA.

El bubble sort, también conocido como ordenamiento burbuja, funciona de la siguiente manera: Se recorre el arreglo intercambiando los elementos adyacentes que estén desordenados. Se recorre el arreglo tantas veces hasta que ya no haya cambios. Prácticamente lo que hace es tomar el elemento mayor y lo va recorriendo de posición en posición hasta ponerlo en su lugar.

Procedimiento ***Bubble Sort***

paso 1: [Inicializa i al final de arreglo] For i <- N down to 1 do
paso 2: [Inicia desde la segunda pos.] For j <- 2 to i do
paso 4: [Si a[j-1] es mayor que el que le sigue] If a[j-1] < a[j] then
paso 5: [Los intercambia] Swap(a, j-1, j).
paso 7: [Fin] End.

MÉTODO DE SHELL.

Ordenamiento de disminución incremental.

Nombrado así debido a su inventor Donald Shell.

Ordena subgrupos de elementos separados K unidades (respecto de su posición en el arreglo) del arreglo original. El valor K es llamado incremento.

Después de que los primeros K subgrupos han sido ordenados (generalmente utilizando INSERCION DIRECTA), se escoge un nuevo valor de K más pequeño, y el arreglo es de nuevo partido entre el nuevo conjunto de subgrupos. Cada uno de los subgrupos mayores es ordenado y el proceso se repite de nuevo con un valor más pequeño de K.

Eventualmente el valor de K llega a ser 1, de tal manera que el subgrupo consiste de todo el arreglo ya casi ordenado.

Al principio del proceso se escoge la secuencia de decrecimiento de incrementos; el último valor debe ser 1.

"Es como hacer un ordenamiento de burbuja pero comparando e intercambiando elementos."

Cuando el incremento toma un valor de 1, todos los elementos pasan a formar parte del subgrupo y se aplica inserción directa.

El método se basa en tomar como salto N/2 (siendo N el número de elementos) y luego se va reduciendo a la mitad en cada repetición hasta que el salto o distancia vale 1.

Procedimiento **Shell Sort**;

```
const
        MAXINC = _____;
incrementos = array[1..MAXINC] of integer;
var
        j,p,num,incre,k:integer;
begin
        for incre := 1 to MAXINC do begin /* para cada uno de los incrementos */
        k := inc[incre];        /* k recibe un tipo de incremento */
        for p := k+1 to MAXREG do begin /* inserción directa para el grupo que se encuentra cada K posiciones */
                                num := reg[p];
                                j := p-k;
                                while (j>0) AND (num < reg[j]) begin
                                        reg[j+k] := reg[j];
                                        j := j - k;
                                end;
                                reg[j+k] := num;
                        end
        end
end;
```

Ejemplo:

Para el arreglo a = [6, 1, 5, 2, 3, 4, 0]

Tenemos el siguiente recorrido:

Recorrido	***Salto***	***Lista Ordenada***	**Intercambio**
1	3	2,1,4,0,3,5,6	(6,2), (5,4), (6,0)
2	3	0,1,4,2,3,5,6	(2,0)
3	3	0,1,4,2,3,5,6	Ninguno
4	1	0,1,2,3,4,5,6	(4,2), (4,3)
5	1	0,1,2,3,4,5,6	Ninguno

UNIDAD IV ACCESO Y ORGANIZACIÓN DE ARCHIVOS

Conceptos generales

- Cuando hablamos de archivos lo que estamos tratando de hacer posible es mantener los datos de una manera persistente.
- De manera que cuando un dato se encuentra en memoria es posible almacenarlo en disco y después con otro programa leerlo y reconstruirlo en memoria.
- Los archivos pueden estar estructurados de distintas formas:
  - registros (secciones 5.2 - 5.7)
  - modelos abstractos (sección 5.8)

Archivos de Registros, métodos de almacenamiento manteniendo la identidad de los campos

- La unidad básica de datos es el campo (field), el cual contiene simplemente valor del dato
- Fields están organizados en grupos, cuando hablamos de listas con campos similares nos referimos a un arreglo (array) y cuando hablamos de listas con campos diferentes estamos refiriéndonos a registros (record)

En términos de programación cuando hablamos de un record en memoria nos estamos refiriendo a un objeto, el cual tiene sus miembros o atributos; sin embargo cuando se almacena en disco dicho objeto entonces hablamos simplemente de un registro.

Nota: cuando el objeto se guarda completamente, incluyendo sus métodos estamos hablando de "Persistencia de Objetos" (Object Serialization)

Problema:

Deseamos almacenar la siguiente información que tenemos en memoria en disco

Mary Ames	Alan Mason
123 Maple	90 Eastgate
Stillwater OK 74075	Ada OK 74820

los guardaremos en un archivo de manera consecutiva, tal como se fue especificando, pero ahora tenemos un gran problema. La información no se puede separar para distinguir los distintos campos que componen cada registro.

```
AmesMary123 MapleStillwaterOK74075MasonAlan90 EastgateAdaOK74820
```

Solución:

Existen distintas maneras de agregar estructuras de datos a archivos y mantener la identidad de los campos:

- Forzar a los campos a tener una longitud fija
- Comenzar cada campo con un indicador de la longitud del campo
- Colocar un delimitador al final de cada campo para separarlo del siguiente
- Usar una pareja "keyword=value" para identificar cada campo y su contenido
-

Forzar a los campos a tener una longitud fija

Los campos (nombre, dirección, estado) de nuestro ejemplo anterior tenían longitudes variables. Pero nosotros podemos pensar en establecer una medida fija para cada uno de los campos en el caso del ejemplo de Person cada campo tiene una longitud y el tamaño total de un registro siempre sera de 67 bytes (11+11+16+16+3+10)

De manera que el archivo quedaría de la siguiente manera (recordar cubrir los espacios vacíos):

Ames	Mary	123 Maple	Stillwater	OK	74075
Mason	Alan	90 Eastgate	Ada	OK	74820

Para recuperar la información se puede hacer matemáticamente leyendo el número de bytes/chars correspondientes a cada campo

Desventajas

- El archivo tiende a hacerse demasiado grande
- Qué sucede si algún campo en un registro necesita más espacio que lo acordado ?, necesitamos arreglar la longitud de ese campo en todos los registros, desperdiciando espacio y tiempo.

Ventaja

- Cuando de antemano sabemos que nuestros campos siempre tendrán la misma longitud es una buena opción

Comenzar cada campo con un indicador de la longitud del campo

Se coloca antes de cada campo su longitud, generalmente si los campos no son muy largos esta medida ocupará un solo byte (256 bytes de longitud)

04Ames04Mary09123 Maple10Stillwater02OK0574075
05Mason04Alan1190 Eastgate03Ada02OK0574820

Colocar un delimitador al final de cada campo para separarlo del siguiente

Escribir un caracter especial entre cada campo de manera que se pueda preservar la identidad La decisión de cual caracter "especial" utilizar es muy importante ya que no debe utilizarse en ninguno de los campos.

En algunos casos los espacios en blanco, el salto de línea o el tabulador pueden ser una buena opción.

Ames|Mary|123 Maple|Stillwater|OK|74075
Mason|Alan|90 Eastgate|Ada|OK|74820

Métodos de Almacenamiento manteniendo la identidad de los Registros

Lo anterior es muy útil e interesante pero pierde de vista el concepto original que teníamos de "registro", en el ejemplo cada registro esta compuesto por 6 campos.

Un registro es un conjunto de campos que permanecen juntos cuando el archivo es visto en términos de organización de alto nivel.

En términos de programación lo que se busca es poder leer "registros" de archivos como un todo (buffer en memoria) para poder separar cada uno de sus campos.

Métodos para organizar registros en archivos

Requerir que los registros tengan una longitud fija (bytes)

- Requerir que los registros tengan un número fijo de campos
- Comenzar cada registro con un indicador de la longitud (suma de todos los bytes de cada campo en el registro)
- Utilizar un segundo archivo para mantener una bitácora del byte de inicio donde comienza cada registro
- Colocar un delimitador al final de cada registro para separarlo del siguiente

Requerir que los registros tengan una longitud fija (bytes)

Un archivo de registros de longitud fija es aquel en el cual todos los registros contienen el mismo número de bytes

La manera de reconocer a los registros es muy similar a la forma que se utiliza para campos, aritméticamente

Ames	Mary	123 Maple	Stillwater	OK74075
Mason	Alan	90 Eastgate	Ada	OK74820

Nota: este espacio que no es utilizado pero se debe preservar para mantener la longitud de los campos, a esto se le conoce como "Fragmentación Interna"

Es necesario mencionar que un registro de longitud fija puede tener campos de longitud variable

Por otro lado es importante resaltar que se puede tener un número variable de campos, siempre y cuando la suma de su longitud (registro) sea la misma

Ames|Mary|123 Maple|Stillwater|OK|74075 <----------Unused space---------->
Mason|Alan|90 Eastgate|Ada|OK|74820 <----------Unused space---------->

Requerir que los registros tengan un número fijo de campos

De manera diferente al método anterior, también se puede tener un número fijo de campos por registros Cuando leemos sabremos que después de leer seis campos empezaremos un nuevo registro Lo anterior se realiza a través de la función "modulo" (%) , cual éste sea 0 hablaremos de un registro nuevo

Ames|Mary|123 Maple|Stillwater|OK|74075|Mason|Alan|90 Eastgate|Ada|OK|74820

Comenzar cada registro con un indicador de la longitud

(suma de todos los bytes de cada campo en el registro + separadores)

Esto es de gran utilidad cuando hablamos de registros de longitud variable

40Ames|Mary|123 Maple|Stillwater|OK|74075|36Mason|Alan|90 Eastgate|Ada|OK|74820

Utilizar un segundo archivo para mantener una bitácora del byte de inicio donde comienza cada registro

El uso de un índice nos sirve para conocer el desplazamiento (offset) de cada registro en el archivo original

Ames|Mary|123 Maple|Stillwater|OK|74075|Mason|Alan|90 Eastgate|Ada|OK|74820

00	40

Colocar un delimitador al final de cada registro para separarlo del siguiente

Al igual que con los campos el caracter que se escoja para separar los registros debe ser un caracter especial que no se utilice dentro de la información

Ames|Mary|123 Maple|Stillwater|OK|74075|#Mason|Alan|90 Eastgate|Ada|OK|74820

Acceso Secuencial y Acceso Directo

Existen dos maneras de accesar y buscar registros de una archivo:

Secuencial

- Características
  - Consecutivamente
  - Respetando el orden de aparición en el archivo
  - El orden de complejidad será O(n) lo cual implica que es demasiado lento para grandes volúmenes de datos, O(n/2) en promedio.
- Se utiliza cuando:
  - Se está buscando en un archivo de texto algun patrón (pattern)
  - Archivos con pocos registros
  - Archivos que no necesitan "búsquedas" por ejemplo los respaldos en cintas
  - Cuando de antemano se sabe que se recuperarán muchos resultados (vale la pena la espera)

Existe una técnica que permite aumentar la velocidad de estos accesos llamada "Blocking"

Se basa en leer bloques de registros en lugar de leer uno por uno

El leer un bloque es más lento porque se traen en cada viaje al disco más datos, pero nuevamente la separación se hace en memoria donde la velocidad es mucho mayor y ahí se gana tiempo.

Aunque el rendimiento mejora considerablemente en realidad no es algo considerable ya que el número de las comparaciones para buscar el patrón o valor que se requiere sigue siendo el mismo.

Directo

- Características
    - El orden de complejidad será O(1)
    - Se basa en las funciones de seek
    - Para obtener un buen rendimiento se deben hacer los registros de una longitud cuyo múltiplo sea del tamaño de un sector del disco. Si el sector es de 512 bytes y nuestro registro mide 30, lo mas adecuado es que mida 32, ya que 32x16=512 (16 registros en un viaje al disco)
- Se utiliza cuando:
    - Los registros son de longitud fija
    - Tenemos una manera de saber en que posición del archivo está un registro,

Ejemplo. campo id=número del registro en el archivo

Reutilizando el espacio en Archivos

Hasta el momento hemos hablado de organización de archivos, de agregar datos y poder distinguir los distintos campos y registros que componen la información.

Qué sucede si eliminamos un registro ??

- Necesitamos recorrer los demás para mantener consistente el archivo
    - Esta suena bastante bien, pero tiene la desventaja de perder mucho tiempo en realizar esta operación
- Dejar el espacio (hueco) y reutilizarlo posteriormente
    - El problema aquí es el desperdicio de espacio, pero si tomamos en cuenta que se reutiliza entonces no habrá mucho problema.

De manera que tenemos 3 operaciones que modifican la organización de nuestro archivo:

- Agregar Registros
- Actualizar Registros
- Eliminar Registros

Si el archivo que se pretende utilizar sólo se utiliza para guardar información (agregar) entonces no hay mayor complejidad en su manipulación; se vuelve más interesante cuando hablamos de actualización y eliminación tanto en archivos de registros de tamaño fijo, como aquellos con registros de tamaño variable. Algo importante que debemos recordar es que se puede ver a la actualización como un "borrar y agregar" así que nos concentraremos primeramente en la eliminación de registros.

Eliminación de Registros y Compactación de Espacio

Mencionábamos anteriormente que una solución para el borrado de registros consiste en recorrer todos los demás registros para evitar que queden espacios. A este procedimiento se le conoce como "Compactación de espacio".

Aunque su desventaja es el tiempo requerido para compactar el archivo este esquema llega a ser el más adecuado para archivos no muy grandes.

Por otro lado también decíamos que otra estrategia del borrado consiste en mantener el hueco de aquel registro que hemos eliminado.
Para fines prácticos lo que se hace es colocarle una marca al registro para identificarlo como "eliminado"; esta marca puede estar en algún campo del registro o designar algún campo en específico para este fin.

Ames|Mary|123 Maple|Stillwater|OK|74075 <----------Unused space---------->

*|son|Alan|90 Eastgate|Ada|OK|74820 <----------Unused space---------->

Brown|Martha|625 Kimbark|Des Moines|IA|50311 <----------Unused space---------->

Nota: una de las ventajas de este esquema es que se pueden recuperar los registros eliminados (ej, utilizando un campo especial para indicar si el registro está activo o no)

Ya que podemos identificar aquellos registros que se han eliminado ahora la pregunta es:

Cómo se reutiliza el espacio ??

Compactando el archivo periódicamente

- (Ya sea a través de una calendarización o bien cuando se alcance cierto límite de "huecos")

- Técnicas de reclamación de espacio

Reclamación de espacio

Antecedente:

Existen aplicaciones donde la reutilización de espacio es crítica, y se debe de hacer lo antes posible.

Recordar los dos tipos de archivos: con registros de longitud fija y los de longitud variable

Los primeros son más sencillos ya que podemos aprovechar el espacio agregando simplemente otro registro en el hueco.

Para poder reclamar el espacio disponible se necesita:

- Identificar aquellos registros que se han eliminado (ej, con una marca especial)
- Poder encontrar esos registros rápidamente sin tener que recorrer todo el archivo, si no hay huecos se insertará al final
- Una manera de saber inmediatamente si hay huecos en el archivo
- Una forma de saltar directamente a uno de los huecos existentes.

La tarea de saber si existen huecos y dónde se encuentran no es tarea fácil

La primer idea que se nos viene a la mente es usar una "lista ligada" (linked list)donde colocamos los números de los registros que se van eliminando.

Esta idea es bastante buena, pero una mejoría es ver a la lista como una "pila" (stack) donde vamos colocando "arriba" aquellos registros que se van eliminando, esto facilita el manejo de la lista de espacio disponible.

header -->	libre4 -->	libre3 -->	libre2 -->	libre1--> null

Stack con las referencias de los registros disponibles

Una limitante de nuestro programa que manipula archivos es que al iniciar tendríamos que leer todo el archivo, verificar cuales son los huecos y crear la pila

Pero esto afortunadamente no es así, la pila se encuentra "inmersa" en el archivo como se muestra a continuación (registros de longitud fija):

Head de la pila: 5

0	1	2	3	4	5	6
Edwards	Bates	Wills	*-1	Masters	*3	Chavez

Eliminamos el registro 1

Head de la pila: 1

0	1	2	3	4	5	6
Edwards	*5	Wills	*-1	Masters	*3	Chavez

Agregamos 3 registros nuevos

Head de la pila: -1

0	1	2	3	4	5	6
Edwards	NUEVO_1	Wills	NUEVO_3	Masters	NUEVO_2	Chavez

Notas:

-En el ejemplo estamos usando como referencia el número del registro, pero en la práctica lo que se utiliza es el offset para desplazarnos en el archivo.

-El head se puede almacenar en otro archivo o bien en el "header" del mismo archivo (sección 5.8)

Lo anterior es un mecanismo bastante útil pero recordemos que se refiere a registros de longitud fija, donde los registros eliminados y los nuevos son del mismo tamaño; en el caso de registros de longitud variable los espacios libres y los nuevos registros pueden o no ser del mismo tamaño.

Para registros de longitud variable se necesita:

- Una manera de conocer aquellos registros que han sido eliminados
- Un algoritmo que permita agregar los registros eliminados a una lista de disponibles
- Un algoritmo para encontrar y recuperar registros de la lista de disponibles para reutilizarlos

El más fácil:

- Tener una lista (no como pila)similar a la que usamos para registros de longitud fija solo que incluyendo la longitud de cada registro
- Cuando se desea agregar un nuevo registro se recorre la lista buscando aquel hueco cuyo tamaño sea >= al del registro que se pretende almacenar.

Size=47 -->	Size=38 -->	Size=72 -->	Size=68 --> null

Reutilizando el registro de tamaño 72

Size=47 -->	Size=38 -->	Size=68 --> null

Lista de disponibles para registros de longitud variable

Head de la lista: 5

0	1	2	3	4	5	6
Edwards	*-1, Size=68	Wills	*1, Size=72	Masters	*3, Size=38	Chavez

Eliminando el registro 6

Head de la lista: 6

0	1	2	3	4	5	6
Edwards	*-1, Size=68	Wills	*1, Size=72	Masters	*3, Size=38	*5,Size=47

Insertando un registro nuevo de 50 bytes

Head de la lista: 6

0	1	2	3	4	5	6
Edwards	*-1, Size=68	Wills	NUEVO	Masters	*1, Size=38	*5,Size=47

Notas:

-En el ejemplo estamos usando como referencia el número del registro, pero en la práctica lo que se utiliza es el offset para desplazarnos en el archivo.

-El head se puede almacenar en otro archivo o bien en el "header" del mismo archivo (sección 5.8)

Esta solución es muy común y bastante utilizada, pero tiene una ligera mejoría:

- Si queremos almacenar un registro de 50 bytes y el primer espacio donde se puede guardar es de 120 bytes estamos desperdiciando 70 bytes (donde podría almacenarse otro registo)
- De modo que un cambio adicional seria dividir ese registro de 120 bytes en 2 registros, uno de 50 (donde se agregaría el nuevo registro) y un hueco disponible de 70
- El único inconveniente es lo que se conoce como "Fragmentación Externa"
    - ej que el hueco fuera de 58 bytes, agregamos el nuevo de 50 y dejamos un hueco de 8, hueco que jamás se ocupará si los registros son mayores

Llaves (Keys)

Hemos mencionado como se almacena la información y como se elimina, pero es importante aclarar algunos conceptos.

- Cuando buscamos algún registro en particular, ya sea para actualizarlo o eliminarlo lo debemos hacer a través de ciertos campos "claves" o "llaves" que sirven para distinguir de manera única a cada registro.

Lo anterior no quiere decir que para hacer búsquedas en general estamos restringidos a esas llaves, podemos buscar en cualquier campo que sea necesario (ej. fechas), pero para el caso de actualizar y eliminar lo más conveniente es tener un campo(s) único(s) para poder movernos rápidamente en el archivo al momento de hacer las comparaciones (ej. ID).

- A la combinación de campos que distinguen a cada registro de manera única se le conoce como Primary Key o Llave Primaria

- Una característica de las llaves primarias es que casi nunca cambian en su ciclo de vida, Ej, ID, RFC, CURP, Nombre (dependiendo del dominio).
- Se pueden utilizar otros campos para diferenciar los registros, Secondary Keys o Llaves Secundarias

Ordenamiento Interno y Búsqueda Binaria

- Es necesario poder hacer búsquedas de registros en los archivos, siguiendo algún criterio o patrón.
- Cuando hacemos una búsqueda de algun registro, ésta se hace principalmente a través de las llaves primarias.
- Como se mencionó en la sección 5.4 se pueden realizar búsquedas secuenciales o de acceso directo.

Las búsquedas secuenciales son demasiado lentas ya que tienen que recorrer todo o casi todo el archivo; aún cuando se encuentre algún resultado podríamos saber de antemano que existe la posibilidad de encontrar más registros que cumplan con el criterio de búsqueda, de ahí que se deba continuar recorriendo todo el archivo.

Por otro lado las búsquedas por acceso directo son extremadamente rápidas, ya que podemos ir al "offset" deseado dentro del archivo, desafortunadamente esto requiere un método o fórmula para saber ubicar donde se encuentra un registro en el archivo.

De manera que si buscamos el registro de la persona con ID 35 sabremos que es el registro 35 del archivo y solo habría que multiplicarlo por su longitud (para el caso de registros de longitud fija) para obtener el desplazamiento que debemos hacer para leer del disco dicho registro.

Pero desafortunadamente este "método" no es muy adecuado en todos lo casos ya que no hay muchas maneras de garantizar que cada registro tendrá una posición única.

Ej. Si queremos hacer búsquedas de registros que no tienen números y solo texto (nombre, dirección, ciudad), la manera de relacionarlos con el lugar que ocupan en disco puede volverse demasiado complicado, aunque sigue siendo una opción.

Por otro lado cuando se habla de búsquedas en memoria primaria, donde los accesos son mucho más rápidos, se han desarrollado algoritmos que garantizan el encontrar registros ágilmente. Tal es el caso de la Búsqueda Binaria cuyo tiempo de búsqueda es de O(log2 n).

El algoritmo de Búsqueda Binaria se basa en la metodología "divide y vencerás", de manera que dado un arreglo de registros "ordenados" podemos buscar rápidamente alguno partiendo el arreglo a la mitad y viendo en que mitad podría estar dicho registro, y a su vez esa mitad es dividida en 2 y así sucesivamente hasta encontrar el registro.

Para el caso de un archivo este algoritmo sería de gran utilidad:

Si por ejemplo tenemos un archivo de 2000 registros y quisiéramos hacer una búsqueda secuencial, ésta nos tomaría en promedio:

- Para una búsqueda secuencial 1/2 n = 1000 comparaciones
- Para una búsqueda binaria 1 + log2 2000 = 11 comparaciones

A simple vista luce mucho más atractiva la búsqueda binaria, pero hay un "precio" que pagar, el tener que ordenar el archivo.

Nota: recordemos que no es lo mismo ordenar un arreglo en memoria , que un conjunto de registros en disco, lo segundo es considerablemente más lento.

Ordenamiento de un archivo en memoria.

Como mencionábamos el ordenar un archivo es muy lento ya que se requerirían muchas "pasadas" por los mismos registros varias veces para poder hacer el ordenamiento; sin olvidar que tenemos: posicionar, leer, posicionar, escribir y esto de repite muchas veces.

Una solución es lo que se conoce como "internal sorting" u "ordenamiento interno", el cual consiste en leer todo el archivo en memoria, secuencialmente, ordenarlo en la memoria primaria y entonces volver a bajarlo a disco. Esto teóricamente es lo ideal ya que el tiempo que toma hacerlo es demasiado corto, pero desafortunadamente para nuestro fin (búsqueda binaria) tiene sus inconvenientes:

a) La Búsqueda Binaria requiere más de 1 o 2 accesos para encontrar en el archivo

- Para archivos grandes, ej 1,000 registros el promedio de accesos es de 9.5 y para uno de 100,000 sería de 16.
- Recordemos que nuestra meta cuando hablamos de archivos es tratar de ir lo menos posible al disco, así que lo anterior no es alentador.

b) Mantener el archivo ordenado es demasiado caro

- Si tenemos el proceso de agregar nuevos registros es demasiado frecuente
- Podemos mantener parte del archivo y parte sin ordenar, pero en esa parte se necesitará un búsqueda secuencial.

Algunas soluciones:

En aplicaciones donde se puede acumular la nueva información esta se guarda en un archivo separado y luego a través de algún ordenamiento tipo "merge sort" se actualizan los datos.

Las soluciones deben cumplir los siguientes criterios:

- No involucrar el reordenamiento de los registros en el archivo cuando uno nuevo es agregado
- Deben estar asociados con estructuras de datos que permitan reordenar eficientemente el archivo.
- De esto se hablará más adelante

c) El Ordenamiento Interno sólo funciona para archivos pequeños

Si el archivo es demasiado grande no tendremos la suficiente memoria ram para tener todo el archivo

- Es importante notar que cuando la memoria ram se acaba no nos marcará un error, se empezará a utilizar memoria virtual, la cual al estar basada en disco nos conduce al mismo problema.

Keysorting

Una solución para el ordenamiento interno es algo muy simple, el lugar de mantener todo el registro en memoria lo que almacenamos es la "llave" y su posición en el archivo.

Arreglo		**Archivo**	
Key	Posición	Posición	Registro

Itaca	1	1	Itaca | 56 | Chicago
Kellog	2	2	Kellog | 77 | New York
Harris	3	3	Harris | 99 | California
Bell	k	k	Bell | 32 | Denver

De manera que el procedimiento sería el siguiente

1) Leer cada registro secuencial mente y mantener en un arreglo en memoria las llaves de dichos registros, así como su posición dentro del archivo

2) Ordenar el arreglo (obviamente por llave)

3) Recorrer el arreglo:

3.1) Para cada registro del arreglo se lee del disco el registro asociado (ya que de antemano sabemos su posición)

3.2) En otro archivo por separado se van guardando los registros leídos

4) Al final eliminamos el archivo original y nos quedamos con el que ya está ordenado.

Arreglo

Key	Posición
Bell	k
Harris	3
Itaca	1
Kellog	2

Archivo

Posición	Registro
1	Itaca | 56 | Chicago
2	Kellog | 77 | New York
3	Harris | 99 | California
k	Bell | 32 | Denver

Ordenar el arreglo

Arreglo

Key	Posición
Bell	k
Harris	3
Itaca	1
Kellog	2

Archivo

Posición	Registro
1	Bell | 32 | Denver
2	Harris | 99 | California
3	Itaca | 56 | Chicago
k	Kellog | 77 | New York

Reordenar el archivo

Limitaciones del Keysorting

- El proceso de lectura de los registros de disco se realiza 2 veces
- El número de "seeks" de lectura es igual al número de registros

- Lo anterior empeora cuando después de cada seek de lectura hay otro de escritura, por lo cual la aguja del disco se tiene que desplazar muchas veces.

La solución:

Para qué volver a escribir todos los registros ??

Mejor únicamente escribir el "arreglo" que ya se tiene en memoria a disco

En dicho **archivo índice** (index file) si podemos realizar las búsquedas binarias como se pretendía.

Archivo Indice (index file)

Key	Posición
Bell	k
Harris	3
Itaca	1
Kellog	2

Archivo

Posición	Registro
1	Itaca | 56 | Chicago
2	Kellog | 77 | New York
3	Harris | 99 | California
k	Bell | 32 | Denver

Este es el origen de lo que se conoce como Indexamiento, que se verá en la siguiente sección.

5.8 Archivos que no se basan en Registros

Antecedentes

- No todos los archivos se basan en registros de datos
- Algunos archivos pueden contener registros pero cada uno con campos diferentes

Ejemplo más comunes:

- Archivos de Imágenes: contiene información del tamaño de la imagen, los colores y específicamente de cada píxel.
- Archivos de Música: Contiene información de la grabación, si es stereo, copyright, fuente, cantante, título y la información de la canción.

Abstract Data Models

- Estos archivos poseen lo que se conoce como "Abstract Data Models"
- Estos modelos indican que el contenido de un archivo estará orientado a la aplicación y no al medio
- En otras palabras, son formatos estándar de aplicaciones específicas que se emplean en todas las plataformas.
- Cada uno de estos formatos se identifica por la extensión del archivo, ej. gif, png, mp3, avi, etc...

Estos modelos o formatos definen la estructura del archivo, de manera que se tienen:

Headers:

- Cuando hablamos de registros, tenemos que están compuestos por campos. Estos campos se identifican por lo que se conoce como "metadatos" (información acerca de los datos), en el caso de los campos el metadato será el nombre de dicho campo.
- Para el caso de archivos que no se basan en registros el archivo presenta al inicio (a veces al final) una serie de bytes que representan los metadatos del archivo, es decir, si por ejemplo el archivo es

una imagen al inicio tendremos bytes que nos digan el largo, el ancho, el número de colores, etc, estos bytes se encuentran especificados en el "formato" del archivo

Offset	Name	Description
0	magic	magic number
4	width	image width in pixels
8	height	image height in pixels
12	depth	bits per pixel
16	length	image size in bytes
20	type	encoding type
24	maptype	type of colormap
28	maplength	lenght of colormap

Sun Rasterfile Header