FACULTAD DE INGENIERIA UNAM
DIVISION DE EDUCACION CONTINUA

CLIPPER 5.2

CURSO BÁSICO

ING. CLAUDIA C. ZAVALA DÍAZ

Palacio de Minería Calle de Tacuba 5 1er. Piso Delegación Cuauhtémoc 06000 México D.F. Tel. 521-40-20 Apdo. Postal M-2385

TEMARIO

CLIPPER 5.2

I. Introducción

1. Definición de base de datos

Una base de datos es una colección de información unida, la cuál posee características comunes entre sí. La finalidad de una base de datos es el almacén de datos de tal forma que sea posible accesarlos rápida y eficientemente.

Las bases de datos sustituyeron a los archivos, debido a que el manejo de la información es mas sencilla, su acceso es rápido en las primeras.

2. Sistema Administrador de base de datos

Características principales de un sistema de administración de base de datos :

Seguridad e Integridad de los datos: Esto previenen que personas no autorizados tengan acceso a la información, en cuanto a integridad se entiende que solo una persona a la vez pueda actualizar el mismo dato al mismo tiempo.

Consulta interactiva: Permite a los usuarios obtener información de la base inmediatamente.

Entrada de datos y actualización interactiva: Permite al usuario dar entrada y actualizar información interactivamente.

Independencia de los datos: Es posible accesar a parte de la información de la base sin tener que accesar a todo al mismo tiempo.

3. Modelos de bases de datos

Las bases de datos se clasifican por su relación con otras las tenemos de los siguientes tipos:

- Jerárquica
- Árbol
- Red
- Relacional

Todas poseen diferentes características en cuanto a su relación base-dato. El tipo de base de datos actual es el Relacional, ya que nos permite manejar diferentes bases como una sola y relacionarlas por uno o varios datos, esto nos ayuda a la disminución de redundancias, conflicto de datos, etc.

3.1 Modelo relacional

El modelo relacional se ha establecido como el principal modelo de datos para aplicaciones comerciales de procesamiento de datos.

Llamaremos modelo relacional a aquel que nos permitirá crear múltiples bases de datos o tablas y crear relaciones entre ellas mediante campos clave.

Un ejemplo de ello es lo siguiente, para el manejo de la compra de artículos en un supermercado tendremos las bases que se muestran a continuación:

Almacén

Clave	Descripción	Cantidad	Precio
00001	Algodón	100000	10.00
00002	Cotonetes	10000	25.00
00003	Crema	50000	20.50

Ventas

Clave	Cantidad	Total
00002	2	50.00
00001	3	30.00
00003	1	20.50

Para el caso de este ejemplo se tendrán 2 bases de datos, la primera contendrá las existencias, descripción y precios unitarios de cada artículo, la segunda base es en la que la cajera capturará las compras por cliente, en este caso el usuario solo tecleará la clave del artículo y la cantidad de éstos comprada y el sistema le indicará de que se trata y cuanto es el costo total por artículo.

De los peligros que se tienen al diseñar utilizando el modelo de base de datos relacional se encuentran.

- Repetición de información
- Incapacidad para representar cierta información
- Pérdida de información

Por lo que un buen diseño de base de datos repercutirá cien por ciento en un buen sistema.

4. Terminología de base de datos

Registro: Conjunto de Campos

Campo: Unidad mínima de información dentro de una base de datos. Cada campo es membretado por un nombre que hace referencia al contenido de éste y poseerá características propias como son longitud y tipo.

Dato: Información que será vaciada en la base de datos

Tipo de dato: Tipo de información que podrá recibir el campo, pudiendo ser:

Carácter: Guarda letras y número (con los que no se operará, como lo son teléfono,años, etc)

Numérico: Guarda número. Se recomienda solo utilizar este tipo de dato para números operables por. ejem. cantidades porcientos, factores, etc.

Lógico: Sólo acepta valores de verdadero o falso (True y False)

Memo: Variable de memoria. Campo de memoria que almacenará información como texto, ésta estará contenida en un archivo físico con el mismo nombre de la base de datos pero con la extensión dbt.

Fecha: Información de fechas

Longitud: Tamaño del campo, para el caso de las cantidades numéricas que posean cifras decimales la longitud total será la suma de los decimales mas el punto mas el entero, por

ejemplo. para representar 9999.99 la longitud se definirá como 7.2, donde se tiene 4 enteros, un punto decimal y 2 decimales.

II. Conceptos Básicos de Clipper

5. Definición de Clipper

Compilador basado en una arquitectura abierta y en un potente lenguaje, creado por Nantucket Corporation y comprado por Computer Associates, originalmente fue creado como compilador de Dbase pero se ha alejado de éste concepto al poseer un lenguaje propio, así como herramientas que le proporcionan poder en la manipulación de base de datos. mas si posee utilerías propias que no lo permiten.

6. Versiones de Clipper

Existen diversas versiones de Clipper, haciendo básicamente referencia a su ligador, entre ellas tenemos

Autumn 86
Summer 87
Versión 5.0
Versión 5.01
Versión 5.2 la más reciente para fines de mayo se liberará la 5.3

6.1 Novedades de la versión 5.2

- Interfaz de Programación de aplicaciones de Memoria Virtual (API VM)

  Es un grupo de funciones que pueden invocarse desde rutinas escritas usando el sistema extendido y que permite la comunicación directa con el sistema VMM, éste es muy útil cuando utilice el sistema extendido como interfaz entre su aplicación y el lenguaje C o programas en lenguaje ensamblador.

- Controladores de bases de datos substituibles (RDD)

  <El controlador de Clipper DBFNTX ha sido mejorado de forma que proporciona índices condicionales y de filtro, en orden ascendente o descendente, e indexado por valores lógicos. Clipper también dispone de diversos RDD que permiten accesar a bases de datos, campos memo y formatos de archivo índice de muchos de los conocidos productos de software de base de datos como son:

  - Dbase III +
  - Dbase IV
  - Fox Pro
  - Paradox

- Interfaz de Programación de aplicaciones de controladores de bases de datos substituibles (RDD API)

  Los diseñadores de software que trabajan con otros productos pueden utilizar el interfaz de programación de controladores de bases de datos para crear nuevos controladores, que permitan la compatibilidad de las aplicaciones en Clipper con cualquier dispositivo de base de datos para el que se ha creado un controlador.

- Controladores de Terminal substituibles

  Clipper proporciona los controladores de terminal siguientes

- PCBIOS utiliza llamadas directas a la BIOS en lugar de escrituras directas en pantalla en sistemas que requieran este tipo de entrada salida.
- NOVTERM hace que las aplicaciones de Clipper se ejecuten con mayor velocidad en el software de algunos servidores de red no dedicados.
- ANSITERM proporciona soporte de terminales ANSI para los sistemas que lo requieran.

6.2 Requisitos para la instalación de Clipper

- PC compatible
- Versión de sistema operativo DOS 3.3 o superior
- 640 K de memoria RAM
- 3 MB libres en disco para la instalación mínima y 6 MB para configuración máxima

6.3 Instalación del paquete

- Insertar en la unidad de disco que se tenga el disco de sistema 1
- Escriba A: y pulse intro
- Escriba Install y pulse intro para iniciar con el procedimiento de instalación

Esta instalación posee instrucciones propias que le indicarán al usuario los pasos a seguir.

7. Estructuras de directorios

Una vez instalado Clipper creará la siguiente estructura de directorios.

Subdirectorio	Descripción
\Clipper5	Directorio raiz creado al instalar
\Bin	Aquí se encuentran los archivos ejecutables como el compilador, ligador, depurador, etc.
\ Include	Archivos de cabecera (.CH y .H)
\Lib	Archivos de biblioteca (.LIB)
\Pll	Archivos de biblioteca preenlazados (.PLL)
\Source	Directorio principal de archivos fuente (.PRG)
\DBU	Manejador de base de datos
\PE	Editor
\RL	Generador de reportes y etiquetas
\Sample	Ejemplos
\Sys	Archivos fuente de algunos archivos del sistema como GETSYS.PRG o ERRORSYS.PRG
\NG	Ayuda en línea

7.1 Modificación del Config.sys y Autoexec.bat

Una observación muy importante consiste en revisar si el autoexec.bat y el config.sys de la computadora han sido modificados.

Autoexec.bat

Debe existir en el path las siguientes rutas

c:\clipper5\bin para poder cargarlos programas ejecutables que posee clipper

c:\clipper5\lib para poder llamar a las librerías

c:\clipper5\pll para tener acceso a las librerías preenlazadas

Se deberán tener los siguientes sets

Set lib = c:\clipper5\lib

Set obj= c:\clipper5\obj

Set pll= c:\clipper5\pll

Config.sys

Se deberá tener por lo menos

Buffers=20

Files=50

para no tener error por demasiados archivos abiertos.

III. Utilerías

8. Manejo del DBU

El DBU es una utilería que le permite al usuario manipular de manera sencilla y rápida la base de datos, desde ésta se crearán bases de datos, índice, es posible borrar, copiar, adicionar información, crear relaciones, filtros y vistas.

F1	Ayuda Despliega la ayuda del Data Base Utility (DBU)
F2	Abrir Abre bases de datos e índices
F3	Crear Bases de datos e índices
F4	Salvar Salva la estructura que se está definiendo o modificando
F5	Browse Sirve para mostrar en una ventana los datos activos en la base, pudiéndolos modificar, agregar y borrar

F6	Utilerías Dentro de las utilerías se tiene lo siguiente: COPY copia a otra base o archivo APPEND Añade datos a la base actual desde otra REPLACE Reemplaza los valores de algún, algunos o todos los campos de la base PACK Borra todos aquellos registros que han sido marcados para ser borrados ZAP Borra todos los registros de la base de datos activa RUN Ejecuta un mandato de sistema operativo
F7	Mover Desplaza el apuntador de la base a través de la base de datos activa, para ello se tiene lo siguiente: SEEK Búsqueda en bases indexadas GOTO Accesa directo mediante su número de registro LOCATE Búsqueda de información en forma secuencial SKIP Salta un número determinado de registros a partir de la posición actual
F8	Set Permite la definición de filtros, relaciones y listas de campos con la base de datos activa.

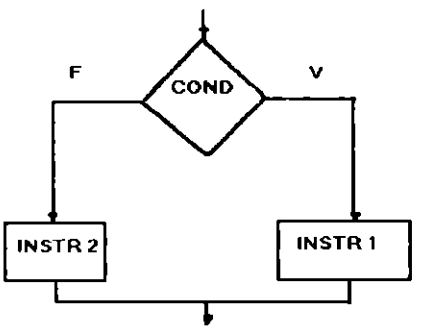
IV. Programación

9. Estructuras Lógicas

IF- THEN- ELSE- ENDIF

La estructura lógica IF nos permite realizar una serie de instrucciones dependiendo del resultado de la evaluación de una condición dada. Esto es, si la condición resulta verdadera realizará una o un grupo de instrucciones, si resulta falsa realizará otra u otro grupo.
La sintaxis es la siguiente:

```
IF <condición> THEN
   instrucciones 1
ELSE
   instrucciones 2
ENDIF
```



Las instrucciones 1 se realizarán cuando el valor de la condición sea verdadera de lo contrario realizará las instrucciones 2.
Es importante hacer notar que sólo se realizará un bloque de instrucciones, nunca realizará ambas, este tipo de estructuras no es cíclico, por lo que solo se realizará una sola vez.

Un ejemplo de ello es:

```
I=1
IF I<0  THEN
    LETRERO="NUMERO DECIMAL"
ELSE
    LETRERO="NUMERO ENTERO"
ENDIF
```

FOR-NEXT

La estructura For es una estructura cíclica dependiente de un contador que permitirá realizar una serie de instrucciones mientras el contador no llega a su límite, el desarrollador determinará el valor inicial del contador y el valor final que deberá alcanzar, así como el valor de los incrementos (por default es 1). El ciclo por si mismo irá incrementando a la variable contador sin que el usuario se preocupe por ello. Para el caso de que en lugar de incrementar se decremente el valor de los incrementos deberá de ser con signo negativo. En el momento que el contador llegué a su límite el ciclo terminará.

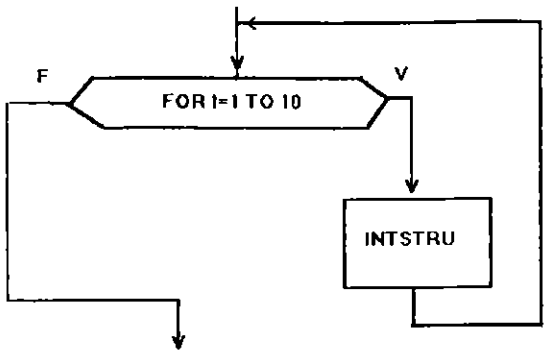
La sintaxis es la siguiente:

```
FOR contador=valor_inic TO valor_fin STEP #no_incrementos
     instrucciones
NEXT
```

Un ejemplo es el siguiente:

```
FOR  I=1  to 1000  STEP 2
    suma=suma+I
NEXT
```

En este caso el ciclo se realizará 500 veces ya que los incrementos serán de 2 en 2.

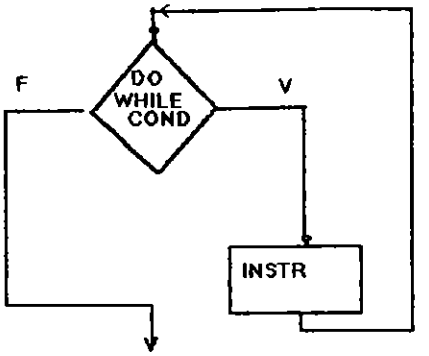


DO WHILE - ENDDO

El Do while es una estructura cíclica que realizará una serie de instrucciones mientras la condición que evaluará resulte verdadera, en este caso el número de interaciones que se realizarán no poseen un límite como en el caso del FOR.

La sintaxis es la siguiente:

```
DO WHILE <condición>
    Instrucciones
ENDDO
```



Un ejemplo de esta estructura se tiene

```
I=1
DO WHILE I<1000
      SUMA=SUMA+I
I=I+1
ENDDO
```

En este caso se utilizó el mismo ejemplo que en el For solo que aquí el número de iteraciones no está controlado y se tendrá que ir incrementando a la variable contador, es muy importante hacer notar que en la estructura WHILE se sufre el riesgo de nunca hacer falsa la condición por lo que estaríamos hablando de que el ciclo nunca terminaría y nuestro programa no pasaría de ahí, de tal forma que el programador debe de asegurarse que en todos los ciclos WHILE que maneje exista siempre una condición que rompa la estructura.

DO CASE - ENDCASE

En el caso de la estructura CASE ésta se podría sustituir por una serie de IFs anidados por lo que se respetan la mismas políticas que en el IF, solo realizará un solo bloque de instrucciones a

la vez y no es una estructura cíclica, en este caso tendremos muchas opciones a elegir y dependiendo de una variable de control se realizará uno u otro bloque de instrucciones.

La sintaxis es la siguiente

```
DO CASE
     CASE <var_opcion>= valor_a
            instrucción 1
     CASE <var_opcion>=valor_b
            instrucción 2
     CASE  <var_opción>=valor_c
            instrucción 3
     OTHERWISE
    instrucción 4
ENDCASE
```

El Otherwise funciona cuando ninguna de las opciones leídas resultó existir en la lista de opciones.

Un ejemplo es:

```
@ 10, 01 say "Teclea lo opción deseada:"  get opción
read
DO CASE
     CASE opción=1
          @ 12,02 say "ALTAS"
     CASE opción=2
          @ 12,02 say "BAJAS"
     CASE opción=3
          @ 12,02 say "MODIFICACIONES"
      CASE opción=4
          @ 12,02 say "CONSULTAS"
      OTHERWISE
           @ 12,02 SAY "SALIR"
ENDDO
```

10. Operadores

( )	Indicador de función o agrupamiento
*	Multiplicación
**	Exponenciación
+	Suma
++	Autoincremento
-	Resta
--	Autodecremento
->	Asignación
.and.	Conjunción
.or.	Disyunción
.not. !	Negación
/	División
:	Envío de información entre objetos
:=	Asignación
<	Menor que
<=	Menor o igual
>	Mayor que
>=	Mayor o igual
<> != #	Diferente
=	Igual
==	Exactamente igual
? ! ??	Despliegado de información
@	Pase de información por referencia

11. Indexados

Los índices son datos únicos, a los que se les denomina llave del archivo . Estos son utilizados para ordenar la información dentro de una base de datos o un archivo . La localización de datos dentro de éstos será rápida ya que el acceso a la información será directo, no será necesario hacer una búsqueda exhaustiva debido a la existencia de un archivo secundario en donde se

encuentran las claves y su localización dentro del archivo maestro. Para el caso de los índices en Clipper la extensión para los archivos generados será NTX.

Sencillos : Llamamos índices sencillos aquellos en los formados por un solo campo. Por ejemplo: un no. de cliente, no de cuenta, etc.
Compuestos: Se llaman compuestos a aquellos en los que lo forman de un campo. Por ejemplo: la clave y el nombre, el departamento y el nombre.

Para crear índices fuera del programa será dentro del DBU seleccionando la opción Crear índice, debiéndose tener la base de datos a indexar abierta, se le indicará el campo sobre el que se indexará, se debe de tener cuidado en que sea un dato que no se repite en ningún otro registro o por el que se quiere ordenar la información.

Para crear índices compuestos es necesario concatenar los campos cuidando que los tipos de éstos sea igual, esto es:
CLAVE+NOMBRE
DEPARTAMENTO+CLAVE

Una vez creado se utilizará en el programa.

12. Estructura de un sistema

La estructura básica de un sistema consistirá en:

- Definición de las variables de ambiente
- Selección de las bases de trabajo
- Inicialización de las variables del programa
- Cuerpo del programa
- Fin

V. Variables

13. Variables de ambiente

Son todas aquellas variables que determinan el ambiente en el que trabajara el sistema algunos de ellos son:

Set cursor on/off	apaga o enciende la visualización del cursor
Set delimiters on/off	apaga o enciende los delimitadores
Set delimiters to "[]"	define cuáles será los delimitadores
Set bell on/off	apaga o enciende la campana
Set confirm on/off	apaga o enciende la opción de confirmar con un Enter después de dar entrada a cualquier dato
Set intensity on/off	apaga o enciende la intensidad en los gets
Set wrap on/off	apaga o enciende la opción de que en los menús sea circular la navegación de las flechas
Set color to	define los colores a utilizar
Set escape on/off	apaga o enciende la opción de poder teclear esc y permitir que se salga del programa
Set console on/off	apaga o enciende los mensajes que envía el sistema a la consola

Set decimal to 0	determina el número de decimales a utilizar
Set default to c:\clipper5	define el directorio de trabajo
Set device to screen/print	direcciona la salida a pantalla o Impresora
Set message to #	Define una línea para mensajes

Set date

italian	dd-mm-aa
british	dd/mm/aa
french	dd/mm/aa
german	dd-mm-aa
japan	aa/mm/dd
usa	mm/dd/yy

14. Tipos de variables

Locales: Están disponibles para el proceso que lo crea y a los que se encuentran a un nivel inferior. Debe de ser la primera instrucción ejecutable dentro de ese programa, rutina o función.

LOCAL <lista de variables>

Estáticas: Tienen como tiempo de vida todo el programa pero solo es visible para el programa que los crea

STATIC <lista de variables>

Públicas: Son válidas en todos los programas siendo modificables en cualquier parte del sistema. Se liberan cuando se libera la memoria. Se declaran en cualquier parte del programa.

PUBLIC <lista de variables>

Privadas: Tienen vida mientras está vigente el programa en el que fueron creadas. Se declaran en cualquier parte.

PRIVATE <lista de variables>

15. Inicialización de variables

La inicialización de las variables consiste en darles un valor con el que iniciarán el programa, dependiendo de éste serán los que se utilizarán.

Carácter

Var_char=SPACE(#)	Se inicaliza con un número determinado de espacios en blanco
STORE " " TO var_char	Grabará en la variable caracter espacios en blanco, tantos como se le den

Numéricas

var_num=0	Se inicializa con cero
STORE 0 TO var_num	Se almacena un cero en la variable

Lógicas

var_log=.T. o .F.	Solo acepta valores de verdadero o falso

Fecha

var_fech=date()	Es posible inicializar con la fecha del dia
var_fech=CTOD(" / / ")	Se inicializa en blanco ayudándose de una función especial (se verá adelante)

VI. Bases de datos e índices

16. Manejo de bases e índices dentro del programa

Para la utilización de bases de datos dentro de un programa será necesario seleccionar áreas de trabajo, para ello se utilizará el comando SELECT y un número éi cuál será la etiqueta de la base de datos, de tal modo que se hará referencia a ésta por medio solo de la etiqueta. Una vez elegido el área de trabajo en la línea inmediata inferior se determinará la base de datos que poseerá dicha etiqueta, será necesario para el caso de variables con índice definirlas de éste modo.

EJ:

```
SELECT 1
USE DATOS INDEX DATOS                  // base de datos con índice
SELECT 2
USE PAIS                               // base de datos sin índice
```

Para poder tener una idea mas clara de este manejo se presentan unos ejemplos a continuación:

En este ejemplo solo se manejan las bases independientes sin ninguna relación entre ellas.

```
SELECT 1                               // Selecciona el area 1
USE  ALMACEN INDEX ALMACEN             // Selecciona la base almacén con su índice
SELECT 2                               // Selecciona el area 2
USE VENTAS INDEX VENTAS                // Selecciona la base ventas y su índice
SELECT 1                               // Utiliza el área de trabajo 1
DO WHILE !EOF()                        // Mientras no sea fin de archivo
      ? CLAVE,NOMBRE,PRECIO            // Desplegará los campos clave,nombre,precio
      SKIP                             // Brinca al siguiente registro
ENDDO
SELECT 2                               // Selecciona el area 2
DO WHILE !EOF()                        // Mientras no sea fin de archivo
```

```
    ?TICKET,CLAVE,CANTIDAD          // Desplegará los campos ticket,clave,cantidad
     SKIP                           // Brinca de registro
ENDDO
RETURN                              // Termina
```

En este caso manejaremos bases de datos que tienen relación entre sí y como se manejara ésta.

Las bases de datos estarán indexadas por el campo CLAVE, éi cuál relaciona a las bases como se vió al inicio de curso. Se simula el proceso de las ventas, le cuál le solicita a la cajera el número de la clave del artículo, una vez que se proporciona habilita la base que contiene la descripción del artículo y envía ei nombre a pantalla como medio de verificación dei mismo, selecciona la base 2 que es la de trabajo y continua pidiendo datos.

Es necesario mencionar que cuando el programador se encuentra en el caso de relacionar bases de datos indexadas es necesario que la clave que se busque no se llame igual que el campo de la base de datos, ya que el compilador se confunde y no encuentra la información, es por ello que es necesario utilizar variables de trabajo.

Ejemplo

```
SELECT 1                            // Selecciona el area 1
USE  ALMACEN INDEX ALMACEN          // Selecciona la base almacén con su índice
SELECT 2                            // Selecciona el area 2
USE VENTAS INDEX VENTAS             // Selecciona la base ventas y su índice
CONTINUA=.T.
DO WHILE CONTINUA                   // Mientras sea verdadero el continuar
    @ 10,1 SAY "TECLEE LA CLAVE : " GET WCLAVE      // Pide la clave clave,nombre,precio
    READ                            // La lee
    SELECT 1                        // Selecciona la base 1
    SEEK WCLAVE                     // Busca la clave
    @ 10,25 SAY DESCRIPCION         // Despliega la descripción
ENDDO
RETURN                              // Termina
```

En el caso siguiente se tiene una base de datos con tres índices distintos, el primero por número, el segundo por nombre y el tercero por teléfono. El usuario necesita trabajar con los tres de tal forma que pueda activarlos en el momento que lo desee. Para este ejemplo no será necesario crear 3 áreas de trabajo diferentes, tan solo se abrirá una y se le determinará cuál será el orden por medio del cual estará indexada.

El orden se determinará conforme se abrieron los índices en la selección del área de trabajo.

Cuando se cambia de orden en los índices es necesario enviar el apuntador de archivo al principio del mismo, ésto una vez seleccionado el nuevo orden.

Ejemplo

```
SELECT 1
USE  NOMBRE INDEX NOMBRE,NOMBRE1,NOMBRE2
  SET ORDER TO 1
DO WHILE !EOF()
      @10,05 SAY "ORDENADO POR NÚMERO"
       ? NUM,NOMBRE,TELEFONO
       SKIP
ENDDO
  SET ORDER TO 2
GO TOP
DO WHILE !EOF()
      @10,05 SAY "ORDENADO POR NOMBRE"
       ? NUM,NOMBRE,TELEFONO
       SKIP
ENDDO
  SET ORDER TO 3
GO TOP
DO WHILE !EOF()
      @10,05 SAY "ORDENADO POR TELÉFONO"
       ? NUM,NOMBRE,TELEFONO
       SKIP
ENDDO
```

VII. Comandos y Funciones

17. Comandos

@... SAY GET

Sintaxis

@ ren,col SAY "<texto>" picture

@ren,col GET <variable> picture <formato pict> WHEN <condición> VALID <condición> RANGE <rango>

Definición

El comando SAY su única función es desplegar un letrero o variable en pantalla dependiendo de las coordenadas especificadas.

Para el caso de la instrucción GET, ésta servirá para darle al usuario la opción de proporcionar información para alimentar al sistema. siempre deberá de ir acompañado de la instrucción READ al final de las series de GET que se utilicen en el programa ya que de esta forma los datos proporcionados realmente se instancian en las variables.

Opciones

PICTURE: Formato de pantalla y despliegue

VALID : Valida la información antes del read

WHEN : Se activa dependiendo de una condición

RANGE : Para determinar el rango valido de la respuesta

PICTURE

A SOLO LETRAS

L VALORES LÓGICOS

Y YES o NO

X CUALQUIER SÍMBOLO

D MUESTRA UNA FECHA CON EL COMANDO ESPECIFICADO

9 SÓLO DÍGITOS

LETRAS, ESPACIOS Y SIGNOS

! SÓLO MAYÚSCULAS

. POSICIÓN INICIAL

, SEPARADOR DE MILES

$ RELLENA CIFRA A LA IZQUIERDA

* RELLENA CON ASTERÍSCOS

S# SCROLL Y NUMERICO (S10)

@ Toda la longitud del campo

R DESPLIEGA EN PANTALLA PERO NO SE ALMACENA

EJEMPLO:

```
NOMBRE="AURORA"
TEXTO="**********"
VALOR=1000.20
VALOR2=-500.50
USE FUNC
@01,01 SAY "NOMBRE :"+NOMBRE
@02,01 SAY "SALARIO: "
@02,12 GET SALARIO PICTURE "999,999.99" RANGE 1,100000
          WHENNOMBRE<>' '
@03,01 SAY "DESCUENTO: "
@03,12 GET VALOR2 PICTURE "999,999.99" VALID VALOR2>0
@06,01 SAY TEXTO PICTURE "!!!!!"
@07,01 SAY TEXTO PICTURE "@!"
@15,01 SAY DATE() PICTURE "@D"
READ
```

@... BOX

Sintaxis

@ren1,col1,ren2,col2 BOX "c1c2c3c4c5c6c7c8c9"

La cadena posee las sig. características

c1 : caracter de la esquina superior izquierda

c2: caracter para la línea superior

c3: caracter para la esquina superior derecha

c4: caracter para la línea derecha

c5: caracter para la esquina inferior derecha

c6: caracter para la línea inferior

c7: caracter para la esquina inferior izquierda

c8: caracter para la línea izquierda

c9: caracter de relleno

Definición:

Dibuja una caja

Ejemplo

CLEAR

@05,10,10,20 box "12345678"

Dibujará una caja así

1222222222223

8 4

8 4

8 4

8 4

766666666665

@ ... CLEAR TO

Sintaxis

@ ren1,col1 CLEAR TO ren2,col2

Definición

Limpia la pantalla dependiendo de las coordenadas indicadas

Ejemplo

```
@10,02 SAY "Letrero"
@15,02 CLEAR TO 15,78
@15,02 SAY "Teclea datos:" GET DATOS
```

@ ... PROMPT

Sintaxis

@ ren,col PROMPT "opcion" MESSAGE <mensaje>

Definición

Comando utilizado para definir opciones de un menú dando oportunidad a enviar mensajes a pantalla sobre la opción seleccionada

Ejemplo

```
@ 1,1 prompt "altas" message "Opción de altas"
@ 2,1 prompt "bajas" message "Opción de bajas"
```

@ ... TO

Sintaxis

@ ren1,col1 TO ren2,col2 DOUBLE

Definición

Dibuja una caja dependiendo de las coordenadas dadas

Ejemplo

```
CLEAR
@5,1 TO 20,78 Double
@ 1,1 TO 3,78
@ 2,2 say "TITULO"
```

APPEND BLANK

Definición

Agrega un registro en blanco a la base de datos que se encuentra activa

Ejemplo

```
USE FUNC
@01,01 SAY "NOMBRE :"+NOMBRE
@02,01 SAY "SALARIO: "
@02,12 GET SALARIO PICTURE "999,999.99" RANGE 1,100000;
       WHENNOMBRE<>' '
@03,01 SAY "DESCUENTO: "
@03,12 GET VALOR2 PICTURE "999,999.99" VALID VALOR2>0
@06,01 SAY TEXTO PICTURE "!!!!!"
@07,01 SAY TEXTO PICTURE "@!"
@15,01 SAY DATE() PICTURE "@D"
READ
APPEND BLANK
REPLACE SAL      WITH SALARIO
REPLACE DESC   WITH VALOR2
```

APPEND FROM

Sintaxis

APPEND FROM<de donde> fields <campos> for <condición>

Definición

Agrega datos de una base a la que se tiene activa dependiendo o no de una cierta condición

Ejemplo

```
SELECT 1
USE DATOS_GR
SELECT 2
USE NOMINA
APPEND FROM DATOS_GR TO NOMINA FIELDS CLAVE,NOMBRE FOR;
CLAVE>'5000'
```

CONTINUE

Definición

Continúa con una búsqueda realizada con el comando Locate

Ejemplo

```
SELECT 1
USE  ALMACEN
SELECT 2
USE VENTAS INDEX VENTAS
CONTINUA=.T.
DO WHILE CONTINUA
      @ 10,1 SAY "TECLEE LA CLAVE : " GET WCLAVE
      READ
      SELECT 1
      DO WHILE !EOF()
            LOCATE FOR CLAVE=WCLAVE
           @ 10,25 SAY DESCRIPCION
           CONTINUE
      ENDDO
ENDDO
RETURN
```

COPY FILE

Sintaxis

COPY FILE <arch fuente> TO <arch destino>

Definición

Copia una base de datos en un archivo

Ejemplo

COPY FILE ARCHIVO1 TO ARCHIVO2

COPY STRUCTURE

Sintaxis

COPY STRUCTURE fields <campos> TO <base de datos>

Definición

Copia la estructura de la base de datos activa a otra, si no existe la crea.

Ejemplo

CLEAR

USE DATOS_GR

COPY STRUCTURE TO FIELDS CLAVE,NOMBRE TO DATOS_NOM

DELETE

Sintaxis

DELETE <scope> FOR <condición>

Definición

Borra campos de una base dependiendo o no de una condición

Ejemplo

DELETE ALL FOR CLAVE="00000"

EJECT

Definición

Alimenta una hoja de papel en la impresora

EJECT

Definición

Alimenta una hoja de papel en la impresora

Ejemplo

```
SET DEVICE TO PRINT
@1,1 SAY NOMBRE
@2,1 SAY PUESTO
@3,1 SAY SALARIO
@4,1 SAY DEPARTAMENTO
EJECT
```

ERASE

Sintaxis

ERASE <archivo>

Definición

Borra un archivo

Ejemplo

```
ERASE DATOS.DBF
```

GO TO

Sintaxis

GO TO <record>

BOTTOM

TOP

Definición

Mueve el apuntador de la base de datos a un registro determinado, al principio, o fin del a base de datos

Ejemplo

```
USE DATOS_GR
GO TOP
```

INDEX ON

Sintaxis

INDEX ON <campo> TO <archivo ntx>

Definición

Indexa una base dependiendo de un campo y lo deja en el archivo de índices con el nombre indicado

Ejemplo

```
USE DATOS_GR
INDEX ON CLAVE TO DATOS_NT
```

LOCATE

Sintaxis

LOCATE <scope> FOR <condición>

Definición

Localiza información en una base de datos no indexada

Ejemplo

```
SELECT 1
USE  ALMACEN
SELECT 2
USE VENTAS INDEX VENTAS
CONTINUA=.T.
```

```
DO WHILE CONTINUA
    @ 10,1 SAY "TECLEE LA CLAVE : " GET WCLAVE
    READ
    SELECT 1
    DO WHILE !EOF()
          LOCATE FOR CLAVE=WCLAVE
          @ 10,25 SAY DESCRIPCION
          CONTINUE
    ENDDO
ENDDO
RETURN
```

MENU TO

Sintaxis

MENU TO <variable>

Definición

Crea el menú con las opciones creadas a partir del prompt

Ejemplo

```
@ 1,1 PROMPT "ALTAS" MESSAGE "Opción de altas"
@ 2,1 PROMPT "BAJAS" MESSAGE "Opción de bajas"
MENU TO OPCION
DO CASE OPCION
      CASE OPCION=1
            @1,1 SAY "ALTAS
     CASE OPCION=2
           @1,1 SAY "BAJAS"
 ENDCASE
```

PACK

Definición

Borra los registros marcados para borrar

Ejemplo

```
USE DATOS_GR
DO WHILE RECNO()>100
   DELETE
   SKIP
ENDDO
PACK
```

RECALL

Definición

Quita las marcas de borrado en los registros

Ejemplo

```
USE DATOS_GR
DO WHILE RECNO()>100
   DELETE
   SKIP
ENDDO
RECALL
```

REINDEX

Definición

Reindexa una base de datos

Ejemplo

```
USE DATOS_GR INDEX DATOS_GR
@1,1 GET WCLAVE
@2,1 GET WNOMBRE
READ
APPEND BLANK
REPLACE  CLAVE      WITH WCLAVE
REPLACE  NOMBRE     WITH WNOMBRE
REINDEX
```

RESTORE SCREEN

Sintaxis

RESTORE SCREEN FROM <nombre_pantalla>

Definición

Restaura una pantalla guardada en memoria

Ejemplo

```
CLEAR
@1,1 TO 15,70
@1,1 GET WCLAVE
@2,1 GET WNOMBRE
READ
SAVE SCREEN TO PANTALLA
CLEAR
@5,5 TO 20,78
@ 6,6 GET WPUESTO
@7,6 GET WSUELDO
RESTORE SCREEN FROM PANTALLA
```

RETURN

Definición

Regresa el control a la instrucción siguiente

Ejemplo

```
USE DATOS_GR INDEX DATOS_GR
@1,1 GET WCLAVE
@2,1 GET WNOMBRE
READ
APPEND BLANK
REPLACE  CLAVE      WITH WCLAVE
REPLACE  NOMBRE     WITH WNOMBRE
RETURN
```

SAVE SCREEN

Sintaxis

SAVE SCREEN TO <pantalla>

Definición

Salva una pantalla en memoria

Ejemplo

```
CLEAR
@1,1 TO 15,70
@1,1 GET WCLAVE
@2,1 GET WNOMBRE
READ
SAVE SCREEN TO PANTALLA
CLEAR
@5,5 TO 20,78
@ 6,6 GET WPUESTO
@7,6 GET WSUELDO
```

RESTORE SCREEN FROM PANTALLA

SEEK

Sintaxis

SEEK <variable>

Definición

Busca un valor en un archivo indexado

Ejemplo

```
SELECT 1
USE  ALMACEN INDEX ALMACEN
SELECT 2
USE VENTAS INDEX VENTAS
CONTINUA=.T.
DO WHILE CONTINUA
    @ 10,1 SAY "TECLEE LA CLAVE : " GET WCLAVE
    READ
    SELECT 1
    SEEK WCLAVE
    @ 10,25 SAY DESCRIPCION
ENDDO
RETURN
```

SELECT

Sintaxis

SELECT <etiqueta>

Definición

Selecciona una área de trabajo

Ejemplo

```
SELECT 1
USE  ALMACEN INDEX ALMACEN
SELECT 2
USE VENTAS INDEX VENTAS
SELECT 1
DO WHILE !EOF()
   ? CLAVE,NOMBRE
   SKIP
ENDDO
```

SET INDEX

Sintaxis

SET INDEX TO <arch. índice>

Definición

Toma un índice definido para la base de datos activa

Ejemplo

```
SELECT 1
USE  ALMACEN
INDEX ON CLAVE TO ALMACEN
SET INDEX TO ALMACEN
SELECT 2
USE VENTAS INDEX VENTAS
```

SKIP

Definición

Salta al siguiente registro

Ejemplo

```
SELECT 1
DO WHILE !EOF()
    ? CLAVE,NOMBRE
   SKIP
ENDDO
```

SORT ON

Sintaxis

SORT ON <campo> TO <archivo>

Definición

Sortea una base de datos sobre un campo determinado dejando el resultado en otra base de datos

Ejemplo

```
SELECT 1
USE DATOS_GR
SORT ON CLAVE TO DATOS_SOR
```

WAIT

Definición

Espera que se oprima una tecla para continuar, pudiendo desplegar un letrero o el valor de una variable, si es un letrero se deberá de marcar entre comillas si es el valor de una variable solo definir el nombre después de la palabra WAIT, en el caso de que no envíe letrero solo abrir y cerrar comillas.

Ejemplo

```
SELECT 1
USE DATOS_GR
SORT ON CLAVE TO DATOS_SOR
WAIT "TERMINO DE SORTEAR POR CLAVE"
```

ZAP

Definición

Borra el contenido de la base de datos activa

Ejemplo

```
USE DATOS_GR
ZAP
```

18. Funciones

ABS()

Sintaxis

Abs(#)

Definición

Retoma el valor absoluto del argumento. Esta función se utiliza en cálculos estadísticos y matemáticos.

Ejemplo

```
@ 1,1 SAY "TECLEE LA CANTIDAD :" GET CANTIDAD
READ
NUM_ABS=ABS(CANTIDAD)
IMPUESTO=NUM_ABS*.015
```

ALLTRIM()

Sintaxis

Alltrim("string")

Definición

Remueve los blancos iniciales y finales de la expresión suministrada como argumento.

Ejemplo

```
NOMBRE=SPACE(20)
@1,1 SAY "NOMBRE: " GET NOMBRE
READ
CADENA=   ALLTRIM(NOMBRE)
```

ASC()

Sintaxis

ASC("string")

Definición

Retorna el valor del código ASCII del primer caracter de la expresión de caracteres.

Ejemplo

```
VAR_NUM=ASC("A")
```

BOF()

Definición

Pregunta por el inicio de archivo

Ejemplo

```
USE DAT_GRAL
IF BOF()
   @1,1 SAY "ARCHIVO VACIO"
ENDIF
```

BROWSE()

Sintaxis

BROWSE(ren1,col1,ren2,col2)

Definición

Esta función visualiza varios registros en forma de una tabla, dentro de una ventana creada con coordenadas específicas.

Ejemplo

```
USE DATOS_GRAL
BROWSE(1,1,15,34)
```

CDOW()

Sintaxis

CDOW(fecha)

Definición

Retorna el nombre del dia de la semana correspondiente a la expresión de tipo fecha suministrada como argumento. Para una fecha en blanco retorna una cadena vacía.

Ejemplo

```
FECHA=DATE()
DIA=CDOW(FECHA)
@1,1 "EL DIA DE HOY ES : " + DIA
```

Ejemplo

```
USE DAT_GRAL
IF BOF()
   @1,1 SAY "ARCHIVO VACIO"
ENDIF
```

BROWSE()

Sintaxis

BROWSE(ren1,col1,ren2,col2)

Definición

Esta función visualiza varios registros en forma de una tabla, dentro de una ventana creada con coordenadas específicas.

Ejemplo

```
USE DATOS_GRAL
BROWSE(1,1,15,34)
```

CDOW()

Sintaxis

CDOW(fecha)

Definición

Retorna el nombre del dia de la semana correspondiente a la expresión de tipo fecha suministrada como argumento. Para una fecha en blanco retorna una cadena vacía.

Ejemplo

```
FECHA=DATE()
DIA=CDOW(FECHA)
@1,1 "EL DIA DE HOY ES : " + DIA
```

CHR()

Sintaxis

CHR(#)

Definición

Convierte un no. ASCII en su caracter

Ejemplo

```
VAR_CHAR=CHR(5)
```

CMONTH()

Sintaxis

CMONTH(fecha)

Definición

Retorna el nombre del mes de la fecha suministrada como argumento.

Ejemplo

```
FECHA=DATE()
MES=CMONTH(FECHA)
@1,1 "EL MES ES: "+MES
```

CTOD()

Sintaxis

CTOD(string fecha)

Definición

Convierte una fecha que está almacenada en forma de una variable de caracteres hacia la variable de tipo fecha

Ejemplo

FECHA=CTOD(" / / ")

CURDIR()

Sintaxis

CURDIR("drive:")

Definición

Visualiza el directorio actual del DOS de una determinada unidad de disco. El argumento es un expresión de caracteres que debe estar entre comillas y contiene la letra de la unidad por investigarse. Si se omite la expresión retornará el contenido en la unidad y directorio actuales.

Ejemplo

@1,1 TO 20,78
@19,2 SAY CURDIR("C:")

DATE()

Definición

Fecha del sistema

Ejemplo

@ 1,1 SAY "FECHA: " +DTOC(DATE())

DAY()

Sintaxis

DAY(fecha)

Definición

Retorna el dia del mes (número) de la expresión suministrada como argumento

Ejemplo

```
@1,1 SAY "EL DIA DE HOY ES "+STR(DAY(FECHA))
```

DBF()

Definición

Regresa la base en uso

Ejemplo

```
SELECT 1
USE DATOS_GR
? DBF()
```

DELETE()

Definición

Regresa T o F si un registro está marcado para borrar

Ejemplo

```
USE DATOS_GR
 DO WHILE !EOF()
     IF RECNO()>='1000'
        DELETE
     ENDIF
 ENDDO
 DO WHILE !EOF()
      IF DELETE()
        PACK
      ENDIF
 ENDDO
```

DISKSPACE()

Sintaxis

DISKSPACE("drive:")

Definición

Retorna la cantidad en bytes disponible en la unidad actual.

Ejemplo

@ 20,1 SAY DISKSPACE("C:")

DISPBOX()

Sintaxis

DISPBOX(ren1,col1,ren2,col2,tipo,color)

Definición

Dibuja una caja similar al comando box

Ejemplo

DISPBOX(1,1,20,56,DOUBLE)

DOTC()

Sintaxis

DTOC(fecha)

Definición

Transforma una expresión de fecha en una expresión de caracteres

Ejemplo

```
FECHA=DATE()
@1,1 SAY "LA FECHA DE HOY ES :"+DTOC(FECHA)
```

EMPTY()

Sintaxis

EMPTY(expresión)

Definición

Verifica si el contenido de una variable es nulo. Si el resultado es positivo retorna .T., en caso contrario, retorna .F.

Ejemplo

```
IF EMPTY(CVE_NUM) THEN
     @1,1 SAY "DATO VACIO"
ENDIF
```

EOF()

Definición

Busca el fin de archivo

Ejemplo

```
SELECT 1
USE DATOS_GR
DO WHILE !EOF()
   ? CLAVE,NOMBRE
ENDDO
```

FILE()

Sintaxis

FILE("archivo")

Definición

Determina si existe un archivo o no

Ejemplo

```
SELECT 1
USE DATOS_GR
IF FILE ("DATOS_GR.NDX") THEN
     SET INDEX TO DATOS_GR
ELSE
     INDEX ON CLAVE TO DATOS_GR
ENDIF
```

FOUND()

Definición

Retorna el valor lógico .T. si un comando FIND, LOCATE, CONTINUE o SEEK encuentra un registro.

Ejemplo

```
SELECT 1
USE  ALMACEN INDEX ALMACEN
SELECT 2
USE VENTAS INDEX VENTAS
CONTINUA=.T.
DO WHILE CONTINUA
    @ 10,1 SAY "TECLEE LA CLAVE : " GET WCLAVE
    READ
    SELECT 1
    SEEK WCLAVE
```

```
        IF FOUND()
            @ 10,25 SAY DESCRIPCION
        ELSE
            @10,25 SAY "CLAVE NO ENCONTRADA"
        ENDIF
    ENDDO
    RETURN
```

INKEY()

Sintaxis

INKEY(segundos)

Definición

Detiene el proceso hasta que se presiona una tecla o ha recorrido un cierto intervalo de tiempo y retorna un número que representa la tecla presionada mas recientemente.

Ejemplo

```
USE DATOS_CR
@ 1,1 SAY "DESPLIEGUE DE INFORMACIÓN"
INKEY(0)
DO WHILE !EOF()
    ?CLAVE,NOMBRE
    SKIP
ENDDO
```

INT()

Sintaxis

INT(#)

Definición

Retorna la parte entera de la expresión numérica suministrada como argumento. La función INT() no redondea el número, simplemente no tiene en cuenta su posiciones decimales.

Ejemplo

```
@1,1 SAY "ELEMENTO 1:" GET A
@2,1 SAY "ELEMENTO 2: " GET B
READ
SUMA=A+B
ENTERO=INT(SUMA)
```

ISPRINTER()

Definición

Retorna .T. si la impresora en el puerto paralelo se encuentra conectada y lista para usarse

Ejemplo

```
IF ISPRINTER()
   SET DEVICE TOPRINT
ELSE
   @ 1,1 SAY "IMPRESORA NO EXISTENTE O FUERA DE LINEA"
ENDIF
```

LASTKEY()

Definición

Retorna el valor ASCII de la última tecla presionada.

Ejemplo

```
IF  LASTKEY() = 27
    RETURN
ENDIF
```

LASTREC()

Definición

Retorna el número de registros del archivo en uso.

Ejemplo

```
DO WHILE  RECNO()<>LASTREC()
      ? CLAVE,NOMBRE
ENDDO
```

LEFT()

Sintaxis

LEFT(string,cuantos)

Definición

Sustrae un substring a partir de la izquierda, iniciada en el primer caracter y de longitud igual al especificado por el argumento de la función.

Ejemplo

```
CADENA="EL NOMBRE ES "
SUB_CADENA=LEFT(CADENA,5)
```

LEN()

Sintaxis

LEN(string)

Definición

Determina la longitud de un string

Ejemplo

```
IF LEN(CADENA)<0
   @1,1 "LA VARIABLE ESTA VACIA"
ENDIF
```

LOWER()

Sintaxis

LOWER(string)

Definición

Contiene el contenido de una expresión de caracteres en letras minúsculas.

Ejemplo

```
MIN_CAD=LOWER("CADENA")
```

LTRIM()

Sintaxis

LTRIM(string)

Definición

Elimina los espacios en blanco a la izquierda de una cadena de caracteres

Ejemplo

```
CADENA=SPACE(20)
@1,2 SAY "TECLEE LA CADENA" GET CADENA
READ
CADENA=LTRIM(CADENA)
```

LUPDATE()

Definición

Retorna la fecha de la última actualización de la base de datos en uso

Ejemplo

```
IF LUPDATE() <>DATE()
  @1,1 SAY "DEBE DE RESPALDAR"
ENDIF
```

MONTH()

Sintaxis

MONTH(fecha)

Definición

Retorna el número del mes contenido en una variable de tipo fecha. mes

Ejemplo

```
FECHA=DATE()
@1,1 "EL NUMERO DEL MES ES : " +STR(MONTH(FECHA))
```

PAD()

Sintaxis

PAD(string,longitud,caracter de relleno)

PADC

PADR

PADL

Definición

Alinea un letrero en una longitud determinada y rellena con el caracter que se le indica

Ejemplo

```
LETRERO="TITULO DEL TEXTO"
PAD(LETRERO,80," ")
```

RECCOUNT()

Definición

Retorna el número total de registros del archivo en uso, incluidos los marcados para eliminación.

Ejemplo

```
@1,1 "EL NUMERO TOTAL DE REGISTRO ES " + STR(RECCOUNT())
```

REPLICATE()

Sintaxis

REPLICATE(string, número)

Definición

Repite un número determinado de veces un string

Ejemplo

```
@1,1 SAY REPLICATE("-",50)
@2,1 SAY TITULO
```

RESTSCREEN()

Sintaxis

RESTCREEN(r1,col1,r2,col2,nombre)

Definición

Restaura una pantalla

Ejemplo

```
@1,1 TO 20,78
@2,2 SAY TITULO
PANTALLA= SAVESCREEN(2,2,10,68)
IF LASTKEY()=27
    RESTSCREEN(2,2,10,68,PANTALLA)
ENDIF
```

RIGHT()

Sintaxis

RIGHT(string,cuantos)

Definición

Extrae un determinado número de caracteres a partir de la derecha

Ejemplo

```
CADENA="EL NOMBRE ES "
SUB_CADENA=RIGHT(CADENA,5)
```

RTRIM()

Sintaxis

RTRIM(string)

Definición

Elimina los blancos al final de una expresión de caracteres.

Ejemplo

```
NOMBRE=SPACE(20)
@1,1 SAY "NOMBRE: " GET NOMBRE
READ
CADENA=   RTRIM(NOMBRE)
```

SAVESCREEN()

Sintaxis

SAVESCREEN(r1,col1,ren2,ren2)

Definición

Salva una pantalla a un área de memoria

Ejemplo

```
@1,1 TO 20,78
@2,2 SAY TITULO
PANTALLA= SAVESCREEN(2,2,10,68)
IF LASTKEY()=27
     RESTSCREEN(2,2,10,68,PANTALLA)
ENDIF
```

STR()

Sintaxis

STR(número,tamaño,decimales)

Definición

Convierte un numérico en caracter

Ejemplo

```
@1,10 SAY "TECLEE EL NUMERO: " GET NUMERO
READ
@2,20 SAY "EL NUMERO TECLEADO: "+STR(NUMERO,5,2)
```

SUBSTR()

Sintaxis

SUBSTR(cadena,pos.inic,cuantos)

Definición

Sustrae una cadena de caracteres de otra

Ejemplo

```
CADENA="EL NOMBRE ES "
SUB_CADENA=RIGHT(CADENA,1,5)
```

TIME()

Definición

Hora del sistema

Ejemplo

```
@ 1,1 SAY "LA HORA DEL SISTEMA ES "
@2,1 SAY TIME()
```

TYPE()

Sintaxis

TYPE(string)

Definición

Determina el tipo de dato del que se envía el argumento

C CARACTER

D FECHA

I ENTERO

M MEMO

Ejemplo

```
IF  TYPE(CVE_USUARIO)="C"
   @1,1 SAY "LA CLAVE ES DE TIPO CARACTER"
ENDIF
```

UPPER()

Sintaxis

UPPER(string)

Definición

Cambia a mayúsculas la cadena

Ejemplo

```
CADENA="el letrero en minúsculas"
CADENA=UPPER(CADENA)
```

VAL()

Sintaxis

VAL(string)

Definición

Convierte de caracter a numérico

Ejemplo

```
CLAVE="00090"
 NUMERO_CH=VAL(CLAVE)
 NUMERO_CH++
 CLAVE=STR(NUMERO_CH)
```

YEAR()

Sintaxis

YEAR(fecha)

Definición

Retorna el número del año contenido en la expresión fecha del argumento

Ejemplo

```
ANIO=YEAR(DATE())
```

VIII. CAMPOS DE MEMORIA

19. *Variables de memoria*

MEMOEDIT()

Sintaxis

Memoedit(Nombre,x1,y1,x2,y,.T.)

Definición

*Guarda información en un campo Memo

Ejemplo

Nombre=memoedit(Nombre,x1,y1,x2,y2)

MEMOREAD()

Sintaxis

Memoread(Archivo de texto)

Definición

*Lee de texto y lo guarda en un campo memo

Ejemplo

Nombre=memoread("archivo.txt")

MEMOWRIT()

Sintaxis

Memowrit(Archivo de texto,Archivo)

Definición

* Escribe un campo memo en un archivo de texto

Ejemplo

memowrit("archivo.txt",Archivo)

MEMOLINE()

Sintaxis

Memoline(Nombre,long. linea,#línea)

Definición

* Extrae una línea de texto de la cadena de caracteres o campo memo.

Ejemplo

```
Local  long_linea,nlineas,linea_act
long_linea=40
use datos
nlineas=mlcount(observa,long_linea)
for linea_actual=1 to nlineas
     ?memoline(observa,long_linea,linea_actual)
next
```

MLCOUNT()

Sintaxis

Mlcount(Nombre,long. línea)

Definición

* Cuenta el número de líneas del tamaño indicado dentro de un campo memo

Ejemplo

número=mlcount(Archivo,15)

IX. ARREGLOS

20. Definición

Los arreglos son variables de memoria que alojan información con características semejantes bajo un mismo nombre, cada dato poseerá un lugar específico dentro del arreglo. Pudiendo hacer referencia a su contenido mediante la(s) coordenada(s) donde fue almacenado, por lo que contaremos con un apuntador del arreglo el cuál será la posición que ocupará el dato dentro del mismo.

Tipos

- Unidimensionales: Poseen una sola dimensión
- Multidimensionales: Poseen dos o mas dimensiones

Definiciones

Declare Arreglo[i]

Arreglo = Array{i}

21. Funciones

AADD()

Sintaxis

AADD(Arreglo,campo)

Definición

* Añade elementos al arreglo modificando su tamaño

Ejemplo

Arreglo={1,2,3,4,5}

AADD(Arreglo,8)

Arreglo={8,2,3,4,5}

ACLONE()

Sintaxis

ACLONE(Arreglo_fuente,Arreglo_destino)

Definición

*Copia los elementos de un arreglo en otro

Ejemplo

Arreglo1={1,2,3,4,5}

Arreglo2={6,7,8,9,0}

Arreglo2=ACLONE(Arreglo1)

Arreglo2={1,2,3,4,5}

ACOPY()

Sintaxis

ACOPY(Arreglo1,Arreglo2,pos. Inic.,cuantos elementos)

Definición

"Sustituye en el arreglo 2 lo del arreglo 1 dentro del rango especificado, perdiéndose los valores originales del segundo arreglo

Ejemplo

Arreglo1={1,2,3,4,5}

Arreglo2={6,7,8,9,0}

ACOPY(Arreglo1,arreglo2,1,3)

Arreglo2={1,2,3,9,0}

ADEL()

Sintaxis

ADEL(Arregio,posición)

Definición

"Borra el elemento seleccionado dejando en la última posición un valor nulo

Ejemplo

```
Arreglo1={7,6,2,4,5}
ADEL(Arreglo1,3)
Arreglo1={7,6,4,5,NIL}
```

ADIR()

Sintaxis

ADIR("archivos",nombre,tamaño,fechas,hora,atributos)

Definición

*Llena arreglos con los elementos de un directorio

Ejemplo

```
CLEAR
 declare arreglo[ADIR("*.prg")]
 ADIR("*.PRG",ARREGLO)
 n=len(arreglo)
 ren=5
 for i =1 to n
     @ ren,5 say arreglo[i]
      ren=ren+1
  next
```

AFILL()

Sintaxis

AFILL(arreglo,elemento de relleno)

Definición

* Llena con un acaracter el arreglo

Ejemplo

AFILL(arreglo,"*")

AINS()

Sintaxis

AINS(arreglo,posición)

Definición

* Inserta un elemento al arreglo, pero lo recorre desapareciendo el último

Ejemplo

AINS(arreglo,5)

ASIZE()

Sintaxis

ASIZE(arreglo,nuevo tamaño)

Definición

*Redimensiona un arreglo

Ejemplo

ASIZE(arreglo,7)

ASORT()

Sintaxis

ASORT(arreglo,elemento inicial,# elementos a ordenar)

Definición

*Sortea los elementos del arreglo

Ejemplo

ASORT(arreglo,1,7)

ASCAN()

Sintaxis

ASCAN(arreglo,posición inicial,#elementos)

Definición

* Busca un elemento en el arreglo

Ejemplo

ASCAN(arreglo,1,5)

X. LIBRERIAS PREENLAZADAS

22. Definición

Directorio PLL (Pre Linked Library) Es donde se encuentran las llamadas a las librerías . La ventaja de ligar con las librerías preenlazadas es que debido a que no las incluye dentro del ejecutable, el volumen de éste es menor, la desventaja es que para portar el sistema deberá ser ligado con las librerías ya que sino las buscará en el momento de correr y sino las encuentra no ejecutará el programa.

Existen 3 archivos

BASE50.PLT

BASE50.LNK

BASE50.PLL

En el caso de solo existir el BASE50.LNK deberá de ser ligado para obtener los otros dos archivos. Se ligará de la siguiente forma:

RTLINK @BASE50

Una vez que se ha asegurado de tener los tres archivos se ligarán los programas con la opción PLL.

RTLINK FI <PROGRAMA> /PLL:BASE50

En el Autoexec de la computadora deberá estar activo en el ambiente del sistema operativo.

SET PLL=C:\CLIPPER5\PLL

XI. FUNCIONES Y PROCEDIMIENTOS

23. Definición

Las funciones definidas por el usuario son un recurso nuevo que ofrece Clipper y que permite economizar muchas líneas de código.

Los beneficios inmediatos del uso de UDF, son

- Creación de bibliotecas de funciones que pueden usarse por cualquier sistema desarrollado en Clippper
- Aumento de seguridad en la evaluación de consistencia de los datos por la utilización de las mismas funciones en todo un sistema.
- Reducción notable del trabajo de mantenimiento del sistema.
- Reducción de código redundante.
- Estimula el uso de la programación estructurada, lo que facilita la construcción y mantenimiento de sistemas.

24. Funciones y Procedimientos

Las funciones y procedimientos componen las UDFs, la diferencia principal entre ambas es el manejo de los valores de retorno. Las funciones por lo general envían parámetros de retorno, mientras que los procedimientos son procesos que se realizan repetidamente y a los cuáles bastará con llamarlos para realizar las operaciones.

Componentes

La estructura básica de una función es la siguiente:

FUNCTION <nombre de la función> (lista de parámetros)

o

PARAMETERS(lista de parámetros)

definición de variables

RETURN(valor de retorno)

Para el caso de los procedimientos

PROCEDURE <nombre del procedimiento> (lista de parámetros)

definición de variables

RETURN

Parámetros:

Los parámetros no son mas que variables de memoria utilizadas para recibir los datos o argumentos pasados hacia una función o procedure.

Ejemplo:

```
a=10
b=20
c=5
d=2
numcalc(a,b,c,d)

FUNCTION NUMCALC
PARAMETERS v1,v2,v3,v4
LOCAL nval
nval=v1*v2+(v3*1.3+v4)
RETURN(nval)
```

La manera más segura de trabajar con funciones es asegurar que no ocurrirán errores por falta de parámetros o por el paso de parámetros impropios de tal manera que aseguremos su cantidad y tipo.

Parámetros por referencia y por valor

Un argumento(parámetro) puede pasarse hacia una función de dos maneras: por referencia y por valor.
Cuando un argumento se pasa por valor (que es el modo normal de una función), su contenido original no se alterará, pues la función copia el contenido de la variable pasada hacia una variable local (dentro de la función). Esta variable se procesa y al final, el valor se retorna.

Ejemplo:

```
vr=60
?quad(vr)
?vr

FUNCTION quad
PARAMETERS num
num=num*num
RETURN (num)
```

En este ejemplo el programa principal visualizará una línea como resultado de la función y otra con el valor original de la variable vr. Notaremos que este valor se mantiene sin modificaciones después del procesamiento de la función, ya que pasó hacia la función una copia de su contenido y éste se asignó a una variable local.

Cuando se pasa una variable por referencia hacia una función, lo que pasa es su dirección de memoria en cuanto a su contenido, esto es , el comando PARAMETERS no creará una variable local dentro de la función, sino un alias para la variable utilizada como argumento, de esta forma la variable pasada sufrirá las modificaciones.

Para indicar el hecho de que una variable debe pasarse como referencia, se debe preceder del signo "@" y el nombre de la variable.

Ejemplo:

```
ml=50                    // Argumento pasado por valor retorna 2500
?quad(ml)          // Retorna 50
?ml
?
?"_____"
?quad(@ml)               // Argumento pasado por referencia, retorna 2500
?ml                      // Retorna 2500
```

A continuación incluimos una funciones que serán de gran ayuda para la programación de sus sistemas.

```
SET DATE BRIT
CLEAR
MODFECHA()
INKEY(0)

CLEAR
DATA=DATE()
@ 10,10 SAY "FECHA : " GET DATA VALID BUSCAFECHA(DATA)
READ
INKEY(0)

CLEAR
? "FECHATEX()"
@ 10,10 SAY FECHATEX(DATE())
@ 11,10 SAY FECHATEX(CTOD("07/09/91"),"MEXICO")
@ 12,10 SAY FECHATEX(DATE()+3)
INKEY(0)
```

```
CLEAR
? "DIATEX()"
@ 10,10 SAY DIATEX(DATE())
@ 11,10 SAY DIATEX(CTOD("07/09/91"))
@ 12,10 SAY DIATEX(DATE()+3)
INKEY(0)

CLEAR
? "FIESTAS()"
@ 10,10 SAY FIESTAS(DATE())
@ 11,10 SAY FIESTAS(CTOD("07/09/91"))
@ 12,10 SAY FIESTAS(DATE()+3)
INKEY(0)


CLEAR
? "FINANO()"
@ 10,10 SAY FINANO(DATE())
@ 11,10 SAY FINANO(CTOD("07/09/61"))
@ 12,10 SAY FINANO(DATE()+3)
DATA=CTOD("10/12/91")
@ 13,10 SAY FINANO(DATA+30)
INKEY(0)


CLEAR
? "INIANO"
@ 10,10 SAY INIANO(DATE())
@ 11,10 SAY INIANO(CTOD("07/09/61"))
@ 12,10 SAY INIANO(DATE()+3)
DATA=CTOD("10/12/91")
@ 13,10 SAY INIANO(DATA+30)
INKEY(0)
```

```
CLEAR
? "FINMES()"
@ 10,10 SAY FINMES(DATE())
@ 11,10 SAY FINMES(CTOD("07/09/61"))
@ 12,10 SAY FINMES(DATE()+3)
DATA=CTOD("10/12/91")
@ 13,10 SAY FINMES(DATA+30)
@ 14,10 SAY FINMES(CTOD("07/02/91"))
INKEY(0)
```

```
CLEAR
? "INIMES()"
@ 10,10 SAY INIMES(DATE())
@ 11,10 SAY INIMES(CTOD("07/09/61"))
@ 12,10 SAY INIMES(DATE()+3)
DATA=CTOD("10/12/91")
@ 13,10 SAY INIMES(DATA+30)
@ 14,10 SAY INIMES(CTOD("07/02/91"))
INKEY(0)
```

```
CLEAR
? "FINSEM()"
@ 10,10 SAY FINSEM(DATE())      //
@ 11,10 SAY FINSEM(CTOD("07/09/61"))
@ 12,10 SAY FINSEM(CTOD("18/09/91")-4)
DATA=CTOD("21/04/91")           // DOMINGO
@ 13,10 SAY FINSEM(DATA)
@ 14,10 SAY FINSEM(DATA-1)      // SABADO
INKEY(0)
```

```
CLEAR
? "INISEM"
@ 10,10 SAY INISEM(DATE())     //
@ 11,10 SAY INISEM(CTOD("07/09/61"))
@ 12,10 SAY INISEM(CTOD("18/09/91")-4)
DATA=CTOD("21/04/91")          // DOMINGO
@ 13,10 SAY INISEM(DATA)
@ 14,10 SAY INISEM(DATA-1)     // SABADO
INKEY(0)


CLEAR
? "INIMESSIG()"
@ 10,10 SAY INIMESSIG(DATE())              // 18/04/91
@ 11,10 SAY INIMESSIG(CTOD("07/09/61"))
@ 12,10 SAY INIMESSIG(DATE()+3)
DATA=CTOD("10/12/91")
@ 13,10 SAY INIMESSIG(DATA+30)
INKEY(0)


CLEAR
? "MESTEX"
@ 10,10 SAY MESTEX(DATE())
@ 11,10 SAY MESTEX(CTOD("07/09/91"))
@ 12,10 SAY MESTEX(DATE()+3)
INKEY(0)


CLEAR
? "SABDOM()"
DATAREC=CTOD("")
@ 10,10 SAY "FECHA DEL RECIBO..: " GET DATAREC ;
            VALID .NOT. SABDOM(DATAREC)
READ
```

```
? SABDOM(CTOD("31/03/91"))    // .T.
? SABDOM(CTOD("01/04/91"))    // .F.
INKEY(0)


CLEAR
? "LUNES()"
?
? LUNES(DATE())      // 18/04/91   JUEVES
? LUNES(DATE()+3)
? LUNES(DATE()+2)
INKEY(0)


CLEAR
? "VIERNES()"
? VIERNES(DATE())       //
? VIERNES(DATE()+3)
? VIERNES(DATE()+2)
INKEY(0)


CLEAR
? "SUMAMES()"
DATA=CTOD("21/04/91")
@ 13,10 SAY SUMAMES(DATA)
@ 14,10 SAY SUMAMES(DATA ,1)
@ 15,10 SAY SUMAMES(DATA ,12)
INKEY(0)

*********************************************************
FUNCTION INIMESSIG(dATA1)
ANO= YEAR(dATA1 )
MES= MONTH(dATA1 )+1
dIA= "01"
```

```
IF MES=13
   ANO=ANO+1
   MES=1
ENDIF
ANO=STR(ANO-1900,2)
MES=STR(MES,2)
final=CTOD((dia+"/"+MES+"/"+ANO))
RETURN ( final)

*********************************************************
FUNCTION FINMES(DATA1)
ANO= YEAR(DATA1 )
MES= MONTH(DATA1 )+1
dia= "01"
IF MES=13
   ANO=ANO+1
   MES=1
ENDIF
ANO=STR(ANO-1900,2)
MES=STR(MES,2)
final=CTOD((dia+"/"+MES+"/"+ANO))
RETURN ( final-1)

*********************************************************
FUNCTION INIMES(DATA1)
RETURN (DATA1 - DAY(DATA1 )+1)

*********************************************************
FUNCTION INISEM(DATA1)
IF DOW(DATA1 )=1     &&  SI FUERA DOMINGO RESTAR 6 DIAS
   final=DATA1 -6
ELSE
   final= DATA1 -(DOW(DATA1 )-2)
ENDIF
RETURN (final)
```

```
*****************************************************
FUNCTION FINSEM(data1)
IF DOW(data1 )=2
   final = data1 +4
ELSE
   final=data1 + (6-DOW(data1 ))
ENDIF
RETURN (final)

*****************************************************
FUNCTION FINANO(data1)
ano=YEAR(data1 )-1900
final=CTOD("31/12/"+STR(ano,2))
RETURN (final)

*****************************************************
FUNCTION INIANO(data1)
ano=YEAR(data1 )-1900
final=CTOD("01/01/"+STR(ano,2))
RETURN (final)

*****************************************************
FUNCTION DIATEX(data1)
dia=DOW(data1 )
SEMANA={"Domingo","Lunes","Martes","Miercoles",;
        "Jueves","Viernes","Sabado"}
RETURN (SEMANA[DIA])

*****************************************************
FUNCTION FECHATEX(data1,cidade)
IF CIDADE=NIL
   CIDADE="SAO PAULO"
ENDIF
mes=MONTH(data1 )
m:={"Enero","Febrero","Marzo","Abril","Mayo",;
```

```
    "Junio","Julio","Agosto","Septiembre","Octubre","Noviembre",;
    "Diciembre"}
RETURN (cidade+", "+STR(DAY(data1 ),2)+" de "+m[mes]+" de ";
   +STR(YEAR(data1 ),4))

************************************************************
FUNCTION MESTEX(data1)
mes=MONTH(data1)
m:={"Enero","Febrero","Marzo","Abril","Mayo",;
    "Junio","Julio","Agosto","Septiembre","Octubre","Noviembre",;
    "Diciembre"}
RETURN (m[mes])

************************************************************
FUNCTION LUNES(data1)
IF DOW(data1 )=7
   data1=data1 +2
ENDIF
IF DOW(data1 )=1
   data1=data1 +1
ENDIF
RETURN (DTOC(data1))
************************************************************
FUNCTION VIERNES(data1)
IF DOW(data1 )=7
   data1=data1 -1
ENDIF
IF DOW(data1 )=1
   data1=data1 -2
ENDIF
RETURN (DTOC(data1))

************************************************
```

```
FUNCTION FIESTAS(data1 ,aumenta)
IF AUMENTA=NIL
   aumenta=.F.
ENDIF
feniados=ARRAY(10)
ano=RIGHT(DTOC(DATA1),2)
feniados[1]=CTOD("01/01/"+ano)
feniados[2]=CTOD("07/09/"+ano)
feniados[3]=CTOD("15/11/"+ano)
feniados[4]=CTOD("12/10/"+ano)
feniados[5]=CTOD("21/04/"+ano)
IF ascan(feniados,data1 )=0
   datanova=data1
ELSE
   datanova=data1 +IIF(aumenta,1,-1)
ENDIF
RETURN (datanova)

************************************************
FUNCTION SABDOM(data1)
IF DOW(data1)= 1 .OR. DOW(data1 )=7
   RETURN (.T.)
ELSE
   RETURN (.F.)
ENDIF

************************************************************
FUNCTION MODFECHA
SAVE SCREEN
CLEAR
IF .NOT. FILE("C:\COMMAND.COM")
   @ 09,19,14,68 BOX REPL(chn(219),8)+" "
   @ 11,21 SAY "IMPOSIBLE EJECUTAR, FALTA EL ARCHIVO"
   @ 12,21 SAY "COMMAND.COM NO ESTA DIRECTORIO ESPECIFICADO"
   INKEY(2)
```

```
   RESTORE SCREEN
   RETURN(.F.)
ENDIF
IF MEMORY(2)<100
   @ 09,19,14,61 BOX REPL(CHR(219),8)+" "
   @ 11,21 SAY "IMPOSIBLE EJECUTAR, FALTA MEMORIA"
   @ 12,21 SAY "100 kB SON NECESARIOS. EXISTEN: "+STR(MEMORY(2),3)
   INKEY(2)
   RESTORE SCREEN
   RETURN(.F.)
ENDIF
RUN DATE
RESTORE SCREEN
RETURN (.T.)

*****************************************
FUNCTION BUSCAFECHA(data1 ,lin1,col1)
IF LIN1=NIL
   lin1=ROW()
ENDIF
IF COL1=NIL
   col1=COL()
ENDIF
SET CURSOR OFF
teste=0
DO WHILE .T.
   @ lin1,col1 GET data1
   CLEAR GETS
   teste=INKEY(0)
   IF teste=13 .OR. teste=27
      EXIT
   ENDIF
   IF teste=5 .OR. teste = 24
      data1= data1 +IF(teste=5,1,-1)
   ENDIF
```

```
   IF TESTE=18 .OR. TESTE=3
      DATA1=DATA1+IF(TESTE=18,30,-30)
   ENDIF
ENDDO
SET CURSOR ON
VARIAVEL=READVAR()
&VARIAVEL=DATA1
RETURN .T.
*********************************************
FUNCTION SUMAMES(DATA1 ,MESES)
IF MESES=NIL
   MESES = 0
ENDIF
MES=MONTH(DATA1 ) + MESES
DIA= DAY(DATA1 )
ANO=YEAR(DATA1 )
DO WHILE MES>12
   MES=MES-12
   ANO=ANO+1
ENDDO
NOVADATA=CTOD(STR(DIA,2)+'/'+STR(MES,2)+'/'+STR(ANO,4))
RETURN (NOVADATA)

*************************************************************
FUNCTION EXPANDE(LIN,COL,TEXTO)
LOCAL TEXTONUEVO:=" "
FOR i = 1 TO LEN(TEXTO)
   TEXTONUEVO+=SUBSTR(TEXTO,i,1)+SPACE(1)
NEXT
IF LIN=NIL .AND. COL=NIL
   RETURN (LTRIM(TEXTONUEVO))
ELSE
   @ LIN,COL SAY LTRIM(TEXTONUEVO)
   RETURN NIL
ENDIF
```

```
*********************************************************
FUNCTION COLUMNA(texto,h_lin,h_col)
FIN:=LEN(TEXTO:=TRIM(TEXTO))
IF FIN > (24-h_lin)  // NO CABE EN LA PANTALLA
   RETURN (.F.)
ENDIF
FOR I = 0 TO FIN-1
   @ h_lin+1,h_col SAY SUBSTR(texto,I+1,1)
NEXT
RETURN NIL

*************************************************
FUNCTION BOXTEXTO(f_lin1,f_col1,f_lin2,f_col2,texto)
IF PCOUNT()<4
   RETURN(.F.)
ENDIF
IF TEXTO=NIL
   texto="*"
ENDIF
longitud=f_col2-f_col1+1
altura=f_lin2-f_lin1
texto=TRIM(texto)+" "
textobox=REPLICATE(texto,INT(80/LEN(texto)+3))
@ f_lin1,f_col1 SAY (LEFT(textobox,longitud))
pos=1
pos2=longitud
DO WHILE pos <= altura
   @ f_lin1+(pos),f_col1 SAY SUBSTR(textobox,pos+1,1)
   @ f_lin1+(pos),f_col2 SAY SUBSTR(textobox,pos2+pos,1)
   pos=pos+1
ENDDO
@ f_lin2,f_col1 SAY (SUBSTR(textobox,altura+1,longitud))

RETURN NIL
```

```
*********************************************************

FUNCTION CENTRA
PARAMETERS texto,long_linea
texto=TRIM(texto)
IF LONG_LINEA=NIL
   long_linea=80      && SUPONE LA LONGITUD DE LA PANTALLA
ENDIF
RETURN ((long_linea-LEN(texto))/2)

*************************************************

FUNCTION CENTRACAD(LIN,COL,texto,long_string,COLOR)
texto=TRIM(texto)
IF LONG_STRING=NIL
   RETURN(.F.)
ENDIF
nuevapos=(long_string-LEN(texto))/2
final=SPACE(nuevapos-1)+texto
final=final+SPACE(long_string-LEN(final))
IF COLOR<>NIL
   COLORANTIG=SETCOLOR()
   NUEVOCOLOR=COLOR
   SETCOLOR("&NUEVOCOLOR.")
   @ LIN,COL SAY FINAL
   SETCOLOR("&COLORANTIG.")
ELSE
   @ LIN,COL SAY FINAL
ENDIF
RETURN NIL

*********************************************************
```

```
FUNCTION MUEVEIZQ(LIN1,COL1,TEXTO)
CURSTAT=SETCURSOR()
SETCURSOR(0)
IF LIN1=NIL
   LIN1=22
ENDIF
IF COL1=NIL
   COL1=20
ENDIF
IF TEXTO=NIL
   TEXTO="PULSE UNA TECLA PARA CONTINUAR..."
ENDIF
TV=SAVESCREEN(LIN1-1,COL1-1,LIN1+1,COL1+LEN(TEXTO)+1)
@ LIN1-1,COL1-1 TO LIN1+1,COL1+LEN(TEXTO)+1
PARADA=0
DO WHILE PARADA=0
   PARADA=INKEY(.1)     && controla la velocidad
   @ LIN1,COL1 SAY SUBSTR(TEXTO,1,40)
   TEXTO=SUBSTR(TEXTO,2,40)+SUBSTR(TEXTO,1,1)
ENDDO
RESTSCREEN(LIN1-1,COL1-1,LIN1+1,COL1+LEN(TEXTO)+1,TV)
SETCURSOR(CURSTAT)
RETURN NIL

*********************************************************
FUNCTION MUEVEDER(LIN1,COL1,TEXTO)
CURSTAT=SETCURSOR()
SETCURSOR(0)

IF LIN1=NIL
   LIN1=22
ENDIF
IF COL1=NIL
   COL1=20
ENDIF
```

```
IF TEXTO=NIL
   TEXTO="PULSE UNA TECLA PARA CONTINUAR..."
ENDIF
TAMANHO=LEN(TEXTO)
TV=SAVESCREEN(LIN1-1,COL1-1,LIN1+1,COL1+TAMANHO+1)
@ LIN1-1,COL1-1 TO LIN1+1,COL1+LEN(TEXTO)+1
PARADA=0
DO WHILE PARADA=0
   PARADA=INKEY(.1)     && controla la velocidad
   @ LIN1,COL1 SAY SUBSTR(TEXTO,1,40)
   TEXTO=RIGHT(TEXTO,1)+SUBSTR(TEXTO,1,TAMANHO-1)
ENDDO
RESTSCREEN(LIN1-1,COL1-1,LIN1+1,COL1+TAMANHO+1,TV)
SETCURSOR(CURSTAT)
RETURN NIL


*****************************************************
FUNCTION GENERACAR (VAR1)
IF VALTYPE(VAR1)="U"
   RETURN (.F.)
ELSE
   TIPO=VALTYPE(VAR1)
ENDIF
DO CASE
CASE TIPO="C"
   FINAL=VAR1
CASE TIPO="D"
   FINAL=DTOC(VAR1)
CASE TIPO="N"
   FINAL=STR(VAR1)
CASE TIPO="L"
   FINAL=IIF(VAR1,"Si","No")
CASE TIPO="M"
   FINAL=VAR1
```

```
CASE TIPO="A"
   FINAL="MATRIZ"
CASE TIPO ="B"
   FINAL="BLOQUE"
ENDCASE
RETURN(FINAL)
```

XII. COMPILACIÓN Y LIGADO

25. *Opciones de compilación y ligado*

Clipper posee un serie de opciones de compilación con diferentes fines, los mas importantes son los siguientes:

-m	Compila solo el programa referido
-s	Checa sintaxis solamente
-w	Genera mensajes de warning para referencias ambiguas de variables
-p	Contiene información del precompilador genera un archivo de salida de extensión ppo y con el mismo nombre del compilado
-b	Para accionar al debbuger

XIII OVERLAYS

26. Definición

Los overlays se utilizan para el manejo de la memoria. A partir de la versión 5.0 ya no fueron necesarios ya que esta versión de Clipper implementó los overlays dinámicos. Anteriormente los overlays se llamaban estáticos ya que el usuario los creaba a su gusto y criterio. En este curso se ve el manejo de los overlays estáticos como una liga con las versiones anteriores.

A continuación se verá la manera de crearlos

+ Se compila cada programa o programa principal por separado con la opción -m, la cuál compilará al programa sin la compilación de los subprogramas.

```
c:\Clipper5  Prueba -m
c:\Clipper5  prueba1
c:\Clipper5  prueba2
```

+ Se crea un archivo de enlace

```
ejemplo.lnk
```

+ Se teclea lo siguiente en el archivo de enlace

```
File Prueba
  Begin area
    Section  file Prueba1
    Section file Prueba2
  enarea
```

+ Se liga

```
RTLINK @ejemplo
```

Esto particionará a la memoria en ejecutables, pero no se verá físicamente el cambio, para poder particionar al ejecutable físicamente se hará lo siguiente:

+ Se modifica el archivo de enlace

```
File prueba
Beginarea
  Section into prueba1 file prueba1
  Section into prueba2 file prueba2
endarea
```

Lo anterior creará archivos con extensión OVL de cada uno de los programas, con lo cuál se ejecutará el sistema por partes. El problema es que en el caso de que algún archivo OVL sea borrado marcará error en la ejecución y se tendrá que compilar nuevamente.

+ En el caso de que se tengan rutinas utilizadas en todo el programa el archivo Ejemplo.lnk quedará

```
File Prueba,funciones
Begin area
  section into prueba1...
  ...
Endarea
```

XIV MACROS

27 Definición

Una macro es un símbolo que avisa al compilador para que asuma el contenido de una variable como si fuera un literal. Se utiliza normalmente cuando se desea escribir alguna rutina genérica o reutilizable.

Ejemplo

```
CLEAR
ARCHIVO=SPACE(30)
@2,2 SAY "DAME EL NOMBRE DEL ARCHIVO A DESPLEGAR" GET ARCHIVO
READ
ARCHIVO=ALLTRIM(ARCHIVO)
SELECT 1
USE &ARCHIVO
BROWSE(1,1,16,45)□
RETURN
```

Una limitación que existe es el hecho de que no es posible que una macro contenga comas que separen sus elementos dentro de una cadena. Si se tuviesen tres índices para abrir una base sería necesario crear tres variables de macros separadas por comas. Una nueva opción, disponible a partir de la versión 5.0, sería la creación de una matriz con los nombres de los índices y la utilización de una expresión extendida.

```
archivo="archivo"
indice="ind_1","ind_2","ind_3"
use &archivo index (indice[1],indice[2],indice[3])
```

XV DIRECTIVAS

28 Definición

Las directivas son órdenes que se incluirán en el programa fuente. El preprocesador las traduce antes de realizar la compilación ordinaria. Se deberán de incluir en los programas como las primera instrucción ejecutable que se tenga.

Se tienen las siguientes

#COMMAND Especifica un comando definido por el usuario

#COMMAND patrón de entrada =patrón resultante

#COMMAND

#DEFINE Define el valor de una constante

#DEFINE esc 27

#IFDEF Sirve para indicar al compilador que el código comprendido entre #ifdef y #else o #endif se procese solo si existe el identificador.

```
#IFDEF  identificador
  instrucciones
#ELSE
  instrucciones
#ENDIF
```

#INCLUDE Incluye un archivo de definiciones en el programa donde se está llamando. Los archivos ya preestablecidos por Clipper se encuentran en el subdirectorio Clipper5 Source y su extensión es ch, teniéndose Inkey.ch, box.ch, setcurs.ch, etc

#INCLUDE "archivo.ch"

#UNDEF Retira una definición realizada con #DEFINE

Ejemplo

```
SET TALK OFF
SET STATUS OFF
#DEFINE ESC 27
#DEFINE RESTA(A,B) A-B
#COMMAND BLANCO=>CLEAR
BLANCO
DECLARE ARREGLO[5]
```

```
ARREGLO={3,5,6,7,8}
#IFDEF RESTA
    IF LASTKEY() # ESC
       FOR  I = 1  TO 5
           ? ARREGLO[I]
           ? RESTA(ARREGLO[I],I)
              NEXT
          ENDIF
    #ELSE
? 'EXPRESIÓN NO DEFINIDA'
    #ENDIF
    RETURN
```

XVI OTROS

29. Creación de JOINS

Los joins sirven para relacionar bases de datos que poseen por lo menos un campo en común, siendo este el campo llave por medio de esto desplegar la información

Se tienen dos bases de datos unidas por medio de un campo común num, que es la clave del trabajador, en la base de datos Nomb se encuentran los datos generales del trabajador y en Suel el sueldo que tiene se mezclarán ambos archivos para conocer los datos del trabajador incluyendo su sueldo, dejándose esto en una base de datos nueva llamada Nuevo. A continuación se hace un browse de esta última base para desplegar los campos de num, nombre y sueldo, los dos primeros pertenecientes a la primera base y sueldo perteneciente a la segunda.

Ejemplo

```
SELECT 1
USE NOMBRE INDEX NOMBRE ALIAS NOMB
SELECT 2
USE SUEL DO INDEX SUELDO ALIAS SUEL
SELECT 1
JOIN WITH SUEL TO NUEVO FOR NUM=SUEL->NUM FIELDS
NUM,NOMBRE,
 SUEL->SUELDO,SUEL->DEPTO
USE NUEVO
BROWSE(5,5,18,50)
CLOSE ALL
```

Ejemplo

```
SELECT 1
USE NOMBRE INDEX NOMBRE ALIAS NOMB
SELECT 2
USE SUELDO INDEX SUELDO ALIAS SUEL
SET RELATION TO NUM INTO NOMB
SET RELATION TO NUM INTO NOMB
DISPLAY NUM,NOMB- NOMBRE,DEPTO,SUELDO
CLOSE ALL
```

30 Ayudas en línea

Las ayudas en línea serán aquellas que el usuario genere para ayudas particulares en cuanto al error o consulta de algún campo en particular. Para ello se definirá una tecla como tecla de ayuda, para esto se utilizará la instrucción SET KEY valor de la tecla en ASCII TO procedimiento

SET KEY 28 TO AYUDA

***** PROGRAMA AYUDA.PRG

```
SET KEY 28 TO AYUDA
DO WHILE .T.E
  COD= SPACE(3); NOMBRE=SPACE(30);VALOR=0
  OCUPACION=SPACE(3);COMENT1:=COMENT2:=SPACE(60)
  CLEAR
  CONTROL=1
  @10,10 SAY "CODIGO ..." GET COD
  @12,10 SAY "NOMBRE ..." GET NOMBRE
  @14,10 SAY "SALARIO .." GET VALOR
  READ
  IF LASTKEY()=27
     EXIT
  ENDIF
  CLEAR
  CONTROL=2
  @10,10 SAY "OCUPACION ..." GET OCUPACION
  @12,10 SAY "COMENTARIO .." GET COMENT1
  @14,10 SAY "            ..." GET COMENT2
  READ
ENDDO

PROCEDURE AYUDA(PROC,LINEA,VAR)
  PANTALLA=SAVESCREEN(11,40,17,73)
  @11,40 CLEAR TO 17,73
  @11,40 TO 17,73
```

```
DO CASE
  CASE VAR="COD"
      @13,41 SAY "El codigo debe de estar compuesto"
      @14,41 SAY " por tres digitos y no"
      @15,41 SAY " puede contener espacios"
      inkey(0)

  CASE VAR="VALOR"
      @13,41 SAY "Escriba el valor pagado hasta "
      @14,41 SAY " antes de la salida"
      inkey(0)
  CASE VAR="COMENT1" .OR. VAR="COMENT2"
      @13,41 SAY "El campo destinado a comentarios"
      @14,41 SAY " diversos sobre el funcionario"
      inkey(0)
  OTHERWISE
      @13,43 SAY "NO EXISTE AYUDA ESPECIFICA"
      @14,43 SAY "PARA ESTA VARIABLE"
      @15,43 SAY "EN CASO DE DUDA CONSULTE CON"
      @16,43 SAY "EL CENTRO DE INFORMACION"
      inkey(0)
ENDCASE
RESTSCREEN(11,40,17,73,PANTALLA)
```

BIBLIOGRAFÍA

+ Clipper 5.01

José Antonio Ramalho

Ed. McGraw Hill

+ 111 Funciones en Clipper

José Antonio Ramalho

Ed. McGraw Hill

+ Manual de usuario

Clipper 5.2

+ Fundamentos de bases de datos

Henry K. Korth

Ed. McGraw Hill

+ Análisis y diseño de sistemas

Kendall y Kendall

Ed. Prentice Hall