LENGUAJE DE PROGRAMACION III

SÍLABO

I.− DATOS GENERALES

CARRERA PROFESIONAL	: INGENIERIA DE SISTEMAS E INFORMATICA

II.− DESCRIPCIÓN DE LA ASIGNATURA

La asignatura es de naturaleza práctica y pretende iniciar al estudiante en el manejo de lenguajes de programacion orientados a la practica profesional. Este curso esta basado en los solidos conceptos teórico−practicos sobre algoritmos y estructuras de bases de datos adquiridos en el curso del mismo nombre que es su pre−requisto, el cual el alumno debera dominar previamente.

FoxPRO no sólo es un software de gestión de bases de datos relaciónales con una interface visual y orientada a objetos, es además, un potente lenguaje de programación estructurada, modular y de procedimientos, que permite desarrollar todo tipo de aplicaciones tanto comerciales como de las áreas de educación, producción, administración, etc. En pequeñas y medianas empresas.

La filosofía de programación que propone FoxPRO está orientada a eventos y no a menús jerárquicos y anidados como es habitual en otros programas de gestión de bases de datos.

FoxPRO 2.6 tiene mas de 600 comandos y funciones orientadas principalmente al manejo de bases de datos relaciónales, pero además cuenta con generadores de informes, pantallas, menús, etiquetas, consultas (SQL) y aplicaciones.

A pesar de ser un software basado en DOS, maneja Ventanas, Menús, desplegables (Popup) y algunos objetos; soporta el ratón (Mouse) con algo de programacion visual; tiene soporte para redes y manejo de aplicaciones multiusuario así como un excelente compilador que muestra los errores de sintaxis.

III.−OBJETIVO GENERAL

El curso esta diseñado para los alumnos de la Universidad Alas Peruanas que se inician en el desarrollo de aplicaciones, siempre con la mira hacia el desarrollo de aplicaciones de la vida real que actualmente son usadas en las empresas.

IV.− METODOLOGÍA

Durante todo el curso cada alumno dispondrá de una PC durante las clases, de manera que pueda escuchar la exposición y aplicar lo expuesto inmediatamente, reforzando y facilitando así la adquisición de conocimientos para obtener los resultados solicitados.

El alumno solicitará la ayuda del docente cuando considere que es necesario para mantener su ritmo de aprendizaje, pero debera poner de su parte practicando un minimo de 4 horas semanales frente a un computador fuera de las horas de clase.

V.− EVALUACION

Para las evaluaciones los alumnos podrán apoyarse en sus notas o apuntes del curso, separatas, libros y todo el material que consideren útil para el desarrollo de su prueba, teniendo en cuenta que deberán trabajar solo ya que la evaluación será en forma individual.

Las prácticas calificadas se desarrollarán en clase con una PC por alumno y pretenderán verificar el nivel alcanzado, planteando la solución de problemas específicos usando como herramienta al FoxPRO.

El trabajo final de curso consistirá de una aplicación práctica de tema libre, que deberá ser desarrollada fuera de horas de clase y sustentada de forma persol en la semana 16.

Los exámenes parcial y final serán pruebas prácticas, desarrolladas directamente en la PC, en un día y hora específico asignados durante las semanas 9 y 17 respectivamente.

El examen Sustitutorio, será tomado en la semana 18 del ciclo y consistirá en la evaluación práctica de todo el curso, pudiendo reemplazar la nota mas baja que el alumno haya obtenido entre el examen parcial (EP) y final (EF).

En caso la nota del examen Sustitutorio sea más baja que la nota más baja del examen parcial o del examen final, no se reemplazará ninguna de ellas, quedando el alumno con la nota obtenida hasta antes del examen Sustitutorio.

La nota final se establecerá del promedio aritmético de:

NF = 30% PP + 10% T + 30% EP + 30% EF

NF = Nota Final

PP = Promedio de prácticas

PT = Trabajos personal sustentado

EP = Examen Parcial

EF = Examen Final

VI.− CONTENIDO ANALITICO

Semana 01

Uso del Editor − Compilador − Ejecutor de programas. DO

Interfaces de usuario: barra de menú, ventanas, ayuda, linea de estado, uso del Ratón, utilidades y herramienrtas: archivador, caracteres ASCII y especiales. Tipos y manejo de archivos.

Variables de memoria, tipos: Carácter, numericas, fecha y logicas. Publicas y privadas, mostrar, guardar, restablecer y borrar.

Mostrar: ?, ??, AT, PICTURE y FUNCTION.

Operadores aritméticos: +, −, *, /, ^, % y (), Relacionales: =, #, <, >, <=, >=, ==, $ y boleanos: .NOT., .AND.,

.OR..

Semana 02

Manejo de matrices: Declarar, rellenar, copiar, guardar y recuperar (MEM).

Funciones: ACOPY(), ADEL(), ADIR(),AELEMENT(), AFDIELD(), AINS(), ALENS(), ASCAN(), ASORT(), ASUBSCRIPT().

Mostrar datos: @..SAY PICTURE, FUNCTION, @..TO, @..BOX, @..CLEAR, @..CLEAR..TO.., @..FILL TO y WAIT WIND...TIMEOUT, WAIT CLEAR.

Semana 03

Captura de datos: GET, READ, validar, consistenciar y detectar errores en entrada de datos PICTURE, FUNTION,

RANGE, VALID.

Control de flujo: Condicionales: IF...ELSE...ENDIF, IIF(), DO CASE CASE... OTHER ENDCASE, Repetitivas: DO WHILE...ENDDO y FOR...ENDFOR. Uso de LOOP y EXIT.

Semana 04

Desarrollo de programas usando: control de flujo y variables de memoria.

PRIMERA PRACTICA CALIFICADA.

Semana 05

Manejo de bases de datos: CREATE, USE, MODIFY STRUCTURE, CLOSE, DIR. Areas de trabajo: SELECT, CLOSE ALL.

Manejo y mantenimiento de registros: BROWSE, APPEND, DELETE, RECALL, PACK, SET DELETE, ZAP.

Filtros: FOR, KEY, SET FILTER TO

Semana 06

Organización de registros: Físico y lógico. Sugerencias para llaves.

Manejo de índices: tipos (.IDX o .CDX), INDEX ON, SET ORDER, REINDEX, SORT.

SEGUNDA PRACTICA CALIFICADA.

Semana 07

Mover puntero y buscar: SKIP, LOCATE, CONTINUE, SEEK, FIND, SEEK(), FOUND().

Cálculos con campos y registros: REPLACE, CALCULATE: SUM(), MAX(), MIN(), AVG(), STD(), VAR(), AVERAGE, COUNT etc.

Semana 08

Manipulación de tablas: SET RELA TO y SET SKIP TO.

Mostrar, imprimir y exportar datos: LIST ..FOR..TO.., DISPLAY.

Generador de informes.

Semana 09

EXAMEN PARCIAL

Semana 10

Programacion orientada a eventos vs orientada a menus jerarquicos.

Procedimientos y funciones de usario: PROCEDURE, FUNCTION, PARAMETERS, DO.. WITH.. IN.. RETURN, SET LIBR TO .., , uso de variables publicas y privadas.

Uso avanzado de @ GET con funciones en VALID y WHEN. READ CYCLE, SHOW GETS, CLEART GETS... SHOW OBJETS.

Inicializar y validar el ingreso de datos. Ejemplos y aplicaciones.

Semana 11

Control de objetos: Botones de Radio, Push y Check. Listas, desplegables (POPUP) e invisibles. READ CICLE, SHOW GETS, CLEAR GETS,

Activar y desactivar controles. _CUROBJ, RDLEVEL(), OBJNUM()...

Otros objetos: GETFILE, PUTFILE, GETEXP, LOCFILE(), FOPEN(), FPUTS(), FREAD(), FGETS(), FCLOSE()...

Semana 12

Ventanas: Definir, activar, mostrar, esconder, desactivar, guardar, restaurar, mover, manejar tamaños, limpiar...

Uso avanzado de BROWSE: FREEZE, KEY, LOCK, WHEN, VALID, IN WINDOW.

Semana 13

Programación de teclas: ON KEY, ON KEY LABEL, ON ERROR, ON ESCAPE, READKEY(), INKEY(), LASTKEY().

Menús: Tipos (General de sistema, barra simple, desplegable simple y multiple). Definir, activar, seleccionar (ON..). Generador de menus.

Semana 14

TERCERA PRACTICA CALIFICADA

Manejo de matrices: Declarar, rellenar, copiar. GATHER, SCATTER, ALEN()...

Campos memo: APPEND MEMO..FROM.., MODIFY MEMO, @..EDIT, MLINE(), COPY MEMO.. TO.., SAVE WIND.. TO.., REST WIND.. FROM MEMO.., SET MEMOWITDH TO..,

Semana 15

Manejo de impresora, consola y archivos: SET DEVICE TO [SCREEN ¡ PRINTER ¡ FILE <txt>[, SET PRINTER ON/OFF, SET CONSOLE ON/OFF, EJECT, PCOL(), PROW(), PRINTSTATUS(), CHR(), SET ALTERNATE ON/OFF TO <txt>, CLOSE ALTERNATE.

CUARTA PRACTICA CALIFICADA

Semana 16

Sistemas multiusuario: SET EXCLUSIVE, FLOCK(), RLOCK(), LOCK(), UNLOCK(), SET REPROCESS TO.., SET MULTILOCK, GETENV() Y NETWARE().

Macrosustitución (&) y sentencias SQL.

Semana 17

EXAMEN FINAL

Semana 18

EXAMEN SUSTITUTORIO

VII. BIBLIOGRAFÍA

- Programación en base a eventos en FoxPRO 2.6 para DOS. César A. Bustamante Gutiérrez − Consultorio Informática.
- Programación básica con FoxPRO Ramón M. Chordá Font − Editorial Rama.

- Al día en una hora en FoxPRO 2.6 José Carlos Corrales − Anaya Multimedia.
- Guía practica FoxPRO 2.6 Alejandro Domínguez − Anaya Multimedia.
- Separatas de FoxPRO: LPE−1, LPE−2, LPE−3, LPE−4 y LPE−5 − Ingº Enrique Garrido−Lecca Risco − Universidad Alas Peruanas. Disponible en pagina Web del curso.
- Ejercicios prácticos de FoxPRO, Ingº Enrique Garrido−Lecca Risco − Universidad Alas Peruanas. Disponible en pagina Web del curso.

Capitulo 1

Conceptos generales

Información

Proviene del latín informare que significa noticia, información o instrucción. Este termino afecta a todos los factores susceptibles de ser transmitidos o almacenados. Sin embargo la información no solo se transmite sino que también es procesada, que es el hecho de gestionar y transformar la información existente en una nueva.

Informática

Proviene de la contracción de las palabras Información autoMATICA. Ciencia que se encarga de todo lo referente al tratamiento de la información en cualquier medio, texto, radio, TV, computadoras etc

Computación

Ciencia que agrupa al conjunto de técnicas y métodos que nos permiten manejar información haciendo uso del computador como herramienta de trabajo.

Sistema

Conjunto de elementos que interactua para lograr la solución de un problema o situación. Ejemplos de sistema: nervioso, de transito, de irrigación, legal, de seguridad, de computo etc

Algoritmo

Conjunto de etapas o pasos que nos permiten solucionar un problema o situación.

Hardware

Es la parte física del computador, es decir la que podemos ver y tocar. Esta compuesta por: CPU o µP, memoria RAM, unidades de almacenamiento y periféricos de entrada y salida.

Software

Es la parte lógica y el lenguaje del computador. Esta formada por los programas y datos que se usan en un computador.

Firmware

Es el software cuyo código esta implementado en hardware. Por lo general se trata de algunos tipos especiales de memorias que puede mantener su contenido sin energía eléctrica. Ej: ROM, FLASH−ROM, PROM, EPROM y EEPROM.

Dígito binario o bit

Es la representación mínima de almacenamiento que puede ser un 1 o 0, si o no, on u off, verdadero o falso etc.

Byte

Es el conjunto de 8 bits con el cual se puede representar un carácter. También se la llama palabra u octeto. Normalmente para expresar valores grandes, se usan los múltiplos como:

- KiloByte = KB = 1,000 Bytes
- MegaByte = MB = 1,000 KB = 1'000,000 Bytes
- GigaByte = GB = 1,000 MB = 1'000,000 KB ...
- TeraByte = TB = 1,000 GB = 1'000,000 MB ...

ASCII (American Standar Code for Information Interchange)

Código estándar Americano para el intercambio de información. Es una tabla compuesta por 256 (8 bits ósea 28 valores) caracteres. Ej: A=65, a=97, á= 160, ñ=164 etc

Sistemas expertos

Sistema que simula el proceso de aprendizaje, memorización, razonamiento, comunicación o acción de un humano en una determinada rama de la ciencia, de forma que podría sustituirle en esas tareas con cierta garantía de éxito. Esas características le permiten almacenar datos y conocimientos, para luego en base a ellos obtener conclusiones lógicas y realizar acciones como consecuencia de lo anterior. Allá por 1977 un sistema experto era sinónimo de computación inteligente.

Inteligencia artificial

Técnicas como el reconocimiento de voz e imagen, robótica y una serie de aplicaciones que involucran conocer, comprender y razonar.

Virus

Es un programa para computadoras (software de aplicación) elaborado por alguien. Los virus tiene la capacidad de realizar ciertas tareas sin la autorización ni conocimiento del usuario. Estas tareas van desde la copia de si mismo (reproducción o contagio), destrucción de archivos, datos, hardware etc. Se propagan de varias maneras, algunos se duplican cuando se abre un archivo infectado, otros infectan el sector de arranque de los discos duros.

Multimedia

Tecnología que consiste en incluir en el computador capacidades de audio y vídeo entre otros dispositivos. Lector CD, tarjetas de sonido y vídeo de alta calidad.

Realidad virtual

Tecnología de avanzada que apoyada en la multimedia permite transportar al usuario a un mundo ficticio o no real totalmente creado por el computador, valiéndose de imágenes, sonido y sensaciones (táctiles, temperatura, movimiento etc). Este mundo imaginario es transferido al usuario a través de un equipo especial conformado por un casco, que oculta la visión del mundo real, proyectando imágenes estereoscopias (simulación de visión en 3 dimisiones) ante sus ojos y sonido sobre sus oidos. Además este casco transmite los movimientos de la cabeza hacia el computador. También se usan controles especiales, guantes y trajes para transmitir sensaciones táctiles, de temperatura, movimiento etc

Telemática

Es todo lo relacionado con las comunicaciones entre computadoras usando los medios de la telefonía, vía satélite, microondas, líneas telefónicas análogas o digitales, conmutación de paquetes etc

Red de área Local o LAN

Conjunto de computadoras interconectadas por algún medio de transmisión (alambre, microondas etc) para compartir recursos de hardware (discos duros, CDs, impresoras, módem, acceso a Internet etc), hacer procesos distribuidos (mediante aplicaciones multiusuario) o comunicarse dentro de un mismo local o edificio.

Circuito Integrado o CHIP

Rectángulo de silicio de apenas 1 a 16 mm de lado y de apenas 0.025 mm de grosor que contiene de algunas decenas a varios millones de componentes (transistores, resistencias, etc) formando un circuito lógico. Ej: Micro procesador, memoria RAM, etc.

Memoria RAM:

Random Access Memory, memoria de acceso aleatorio, es decir que se puede leer, escribir o borrar en cualquier parte de ella. Es donde deben estar las instrucciones y datos para que los use el µP. Se dice que la RAM es volátil porque pierde todo su contenido al retirarle la energía eléctrica.

Capitulo 2

Software

El computador por sí solo (hardware) no puede hacer ningún trabajo, ya que es indispensable que primero el hombre le transmita las instrucciones necesarias. A este conjunto de instrucciones especialmente escritas para ser interpretadas por un computador se le llama software.

Es decir, el software es el conjunto lógico ordenado de instrucciones y datos que usa el computador.

Tipos de software:

1. Sistema operativo: Es el que administra el funcionamiento del computadora y facilita la labor de las aplicaciones, apoyándolas en tareas básicas como leer, escribir, copiar, borrar archivos, presentaciones de pantalla, entradas por teclado, etc Ej: DOS, Windows 95, OS/2 WARP, CP/M, UNIX, System 7 etc

2. Lenguajes de programación: Software especializado que se usa para el desarrollo de programas o aplicaciones, que pueden ser usados directamente por usuarios finales.

3. Aplicaciones y utilitarios: También denominados paquetes, están escritos en algún lenguaje de programación y sirven para un fin o tarea especifico:

- **Procesadores de texto:** Para un fácil desarrollo de todo tipo de documentos. Ej: Word, WordPerfect, WordStar, etc
- **Hojas de calculo:** Presentan en pantalla una especie de hoja cuadriculada donde uno puede ingresar datos y fórmulas, que posteriormente se pueden presentar en tablas o gráficos. Ej: Excel, Quattro Pro, Lotus 123, Visi Calc etc
- **Manejadores de bases de datos:** Permiten realizar todo tipo de operaciones relacionadas con bases de datos, principalmente la búsqueda o extracción de parte de esos datos imponiendo ciertos criterios. Ej: FoxPRO, dBASE, Clipper, Paradox, R:Base, Access etc
- **Presentaciones:** Para la preparación de una presentación en diapositivas. Ej: Power Point, Harvad Graphics etc
- **Integrados:** Son varias aplicaciones reunidas en un mismo paquete. Ej: Office, Works, Perfect Office, Lotus Notes etc
- **Procesadores de calculo:** Facilitan todo tipo de cálculos de diversa índole tales como: matemáticos, estadísticos, financieros etc Ej: MathCad, Eureka, Erwin etc
- **Gráficos:** Permiten el fácil uso de todo tipo de gráficos para ingeniería, publicidad, arte, estadísticos, financieros, lineales etc. Ej: AutoCAD, PaintBrush, Corel Draw, PaintSHOP etc
- **Autoedicion:** Permiten el desarrollo de publicaciones de diseño complejo, incluidos textos en diversas fuentes, gráficos, fotos. Ej: Page Maker, Ventura, Corel Perfect, etc
- **Asistencia de proyectos:** Procesadores para planear y desarrollar proyectos. Ej: Harvard Total Proyect Managet...
- **Reconocimiento de caracteres (OCR)**
- **Herramientas CASE:** Computer Aid Software Enginering. Son paquetes que permiten programar computadores, desarrollando aplicaciones de una manera asistida.
- **Juegos:** Aplicaciones especialmente desarrolladas para el entretenimiento.

- **Antivirus y antivirus:**

Capitulo 3

Lenguajes de programación

Para que una computadora pueda realizar las funciones y operaciones que deseamos, es necesario suministrar las instrucciones adecuadas debidamente agrupadas y ordenadas. Este conjunto de instrucciones constituyen lo que se denomina un programa o aplicación.

Para que estas instrucciones sean comprensibles para el computador y debido a la propia estructura física de los mismos, estos programas deberán estar expresados como combinaciones de cero y unos o mejor dicho expresados en código binario o de lenguaje de maquina.

Debido a las graves dificultades que entraña la programación en lenguaje de maquina, los profesionales del software han desarrollado lenguajes de programación mas humanizados que permiten alejar las tareas de programación de las maquinas y acercarlas a los problemas.

CLASIFICACIONES

La clasificación de los lenguajes de programación no es fácil debido a que las categorías no son absolutamente disjuntas.

1. Por su estructura interna:

a. Bajo nivel:

Se caracterizan por poseer una estructura demasiado compleja, lo cual los hace difíciles de aprender, entender y aplicar. Ello se debe a su relación directa con el funcionamiento real de cada uno de los elementos internos del computador: µP, RAM, periféricos etc.

Son los lenguajes propios o naturales de las computadoras y por ello los programas escritos en bajo nivel nos permiten obtener la máxima velocidad de proceso y un control total de todo el hardware del computador.

1. Lenguaje de maquina: Cada instrucción esta representada por un valor numérico, el cual se describe en hexadecimal o en binario. La desventaja radica en lo difícil de su codificación, pero a cambio obtenemos alta velocidad y control. El conjunto de instrucciones que conforman un lenguaje de máquina es determinado por el microprocesador, ya que cada uno tiene un juego de intrucciones propio y diferente al resto.

2. Lenguaje Ensamblador (Assembler): Es muy similar al anterior solo que cada instrucción esta representada por una pequeña palabra (nemotecnico), mucho mas fácil de manejar para los humanos que los códigos hexadecimales, por lo que se le considera un lenguaje codificado y a cada palabra le corresponde una instrucción del microprocesador.

b. Alto nivel:

Se caracterizan por su similitud con los lenguajes humanos, por lo cual son mas fáciles de aprender, entender y usar. Sus principales objetivos son:

- Humanizar las tareas de programación, acercando los lenguajes de programación al lenguaje coloquial (de conversación)
- Hacer compatibles los distintos computadores a través de la programación: Un programa escrito en un

lenguaje de alto nivel puede ejecutarse en cualquier computador.

Como además estos lenguajes usan nombre simbólicos para representar datos, variables, direcciones de memoria, etc y sus instrucciones tienen la categoría de macro instrucciones con su uso tendremos salvada la totalidad de las dificultades que presenta la programación en código o lenguaje de maquina y lo que es mas importante, nos permite acercar las tareas de programación a los problemas alejándolos de los detalles técnicos relacionados con el computador.

Ventajas:

- Un programa escrito en un lenguaje de alto nivel puede ser usado, después de algunas modificaciones, en distintos equipos.
- El tiempo de formación de los programadores es relativamente corto, en comparación con el necesario para aprender los lenguajes de nivel inferior.
- El programador no necesita conocer cómo funciona un computador específico para poder confeccionar los programas.
- El tiempo necesario para codificar y poner a punto, es decir los cambios y correcciones posteriores, de un programa en lenguaje de alto nivel es inferior al necesario en el caso de los lenguajes menos evolucionados.
- La reducción del tiempo expresado en el punto anterior reduce también el costo de los programas.

Inconvenientes:

- El tiempo de ejecución es mayor, puesto que las instrucciones generadas por el compilador son mas numerosas que las correspondientes al mismo programa escrito directamente en lenguaje de maquina. Este incremento de tiempo es del orden del 15% en promedio.
- No se aprovechan las posibles ventajas de la arquitectura interna del sistema.
- Se incrementa el uso de memoria interna, tanto por parte del programa ejecutor (runtime) como por el programa objeto en si.
- Cada vez que se introduce un cambio es necesario compilar el programa fuente nuevamente.

c. Nivel medio:

Poseen características de alto y bajo nivel, por lo que se puede obtener velocidades de proceso muy similares al bajo nivel, control total del equipo y además facilidades de programación. Ejemplos: C++ y ADA

2. Por su potencia:

a. Primera generación: Lenguaje de maquina, no requiere traducción alguna, el computador es capaz de leerlo directamente.

b. Segunda generación: Lenguaje ensamblador dependiente de la maquina, que requiere de una traducción, aunque esta es muy simple porque cada instrucción corresponde a un código solamente.

c. Tercera generación: Lenguajes de alto nivel.

- Están diseñados para ser usados por programadores profesionales.
- Requieren especificaciones de cómo realizar una tarea.
- Se debe especificar todas las posibles opciones.
- Requieren de un numero grande de instrucciones.
- Códigos pueden ser difíciles de leer, entender, mantener y depurar.
- Originalmente desarrollados para operaciones por lote.

- Orientados hacia archivos
- Requieren de traducción y cada instrucción es convertida en varias instrucciones de maquina.
- El programador solo es enfrentado al código fuente que el mismo creo y nunca al código objeto resultante.

Ej: Fortran, Cobol, Basic, Pascal, C

d. Cuarta generación (4GL): Lenguajes más avanzados que los de alto nivel.

- Requiere la especificación de la tarea a realizar (el sistema determina cómo efectuarla)
- Ofrece opciones pre−determinadas que el usuario no necesita especificar.
- El programador no es enfrentado a ningún código, siempre usa la interface.
- Requiere traducción y cada instrucción en convertida en muchas instrucciones en lenguaje de maquina.
- Errores fáciles de localizar.
- Orientados hacia bases de datos, objetos OLE.

1. Orientados a procedimientos

Ej: dBASE, Clipper, FoxBASE...

2. Lenguajes de consulta (query) y recuperación (SQL)

Ej: SQL server

3. Lenguajes generadores (de reportes, aplicaciones, pantallas...)

Son lenguajes que brindan al programador una serie de facilidades que permiten el desarrollo de programas de una forma mas sencilla y rápida. Los lenguajes de cuanta generación poseen como característica principal, el poder generar de forma automática, el código (programas, procedimientos, funciones, sub−rutinas) encargado de la realización de algunos procesos parciales que intervendrán en un programa o inclusive generar la mayor parte del programa que se requiere. Todo ello a través de un método conversacional entre el computador y el programador.

Ejemplos: Herramientas CASE, Informix

CASE: Computer Aid Software Enginering (ingeniería de software asistida por computadora) que permiten al programador desarrollar aplicaciones en cortos periodos de tiempo debido a que las herramientas CASE generan de forma casi automática el código (la lista de instrucciones) que se realizan en un proceso muy simple para el usuario.

Programación orientada a eventos:

Programación de aplicación que responde a las entradas del usuario (seleccionando menús, botones, formularios etc) o de otras aplicaciones a tiempos regulares.

Programación orientado a procedimientos:

Método de programación que requiere de una disciplina como FORTRAN, COBOL, BASIC, C, Pascal y Xbase. El programador escribe el código en cierto orden para resolver el problema, basado en sus conocimientos del proceso y la programación. La aplicación resultante fuerza al usuario a seguir un camino predefinido desde el paso A al paso B. Un ejemplo típico es el ingreso de datos,

```
USE filex

DO WHILE !EOF()

? COD,APE,NOM

SKIP

ENDDO
```

No procedural seria:

```
USE filex

LIST OFF COD,APE,NOM
```

Lenguaje orientado a problemas:

Lenguaje de computación diseñado para manejar un problema particular. Ej: FORTRAN fue diseñado para ingeniería, COBOL para negocios y GPSS para simulaciones.

Programación visual: Programas basados en herramientas visuales como menús, botones y cualquier elemento gráfico, que se puede seleccionar de una paleta, se arrastra y suelta donde se desea sobre la pantalla. Esto también puede referirse a poder conseguir el código fuente interactuando con diagramas de flujo y gráficos lógicos asociados a códigos. Ej: Visual BASIC, Visual FoxPRO, Visual C++...

Orientado a objeto (OOP): Su objetivo es el de aumentar la productividad del programador incrementando la extensibilidad y reutilizando el software, controlando la complejidad y el costo de mantenimiento.

Ejemplos: C++, Turbo PASCAL, Power Builder...

3. Por su aplicación:

a. Científicos: Son aquellos cuya aplicación mas inmediata es resolver problemas de calculo. Históricamente son los primeros lenguajes evolucionados ya que la formulación matemática permite una fácil formación del lenguaje. Los primeros fueron el SHORT CODE, creado por el Dr Mandy en 1949, para UNIVAC y el SPEED CODING desarrollado en 1953 por Backus y Saldon para IBM. Los mas usado: FORTRAN, PASCAL, BASIC, LOGO, y APL, pero antes de llegar ellos aparecieron otros como: MATH MATIC, UNICODE, IT, GAT y FORTRANSIT.

b. De gestión: Son lenguajes orientados a la solución de problemas de tratamiento de datos para la gestión, por lo que predominan las instrucciones dedicadas a procesar instrucciones de entrada y salida. El primero fue el FLOW−MATIC desarrollado en 1955 por el Dr Hopper para UNIVAC. El lenguaje mas característico de entre los de gestión es el COBOL, seguido por el RPG−II.

c. Polivalentes o aplicación general: Son los resultados del intento de obtener un lenguaje que cubriera tanto el área científica como el área de gestión de forma equilibrada. EL primero fue JOVIAL, desarrollado en 1959 por la Strategic Air Control System. El mas conocido de estos lenguajes es el PL/1 creado en 1964. No obstante es obligatorio apuntar en esta parte que casi la totalidad de los lenguajes científicos han evolucionado hacia la gestión incluyendo entre sus estructuras otras nuevas extraídas de los lenguajes clásicos de gestión. Tal es el caso, que con lenguajes como FORTRAN, PASCAL o BASIC se puede resolver con suma facilidad cualquiera de los problemas clásicos de gestión, en principio vedados para estos lenguajes.

d. Especiales: Son lenguajes con algún propósito muy especifico:

- Lenguajes para expresiones algebraicas formales.
- Lenguajes para manejo de archivos y banco de datos.
- Control de maquinaria, equipos y herramientas.

Programas traductores

El computad es una maquina que a pesar de sus múltiples aplicaciones o posibilidades de uso, solamente puede comprender y procesar aquella información que esta expresada en lenguaje binario, es decir el lenguaje de maquina.

Como entenderá el computador los programas escritos en otro lenguaje ?

La respuesta es: implementados en el sistema operativo o bien soportados en unidades de almacenamiento auxiliar, se encuentran unos programas escritos en código maquina cuya misión es convertir un programa escrito en cualquiera de los lenguajes de programación (medio, alto, 3ra, 4ta generación u OOP) a lenguaje de maquina (bajo nivel).

Son los llamados programas traductores, pero existen varios traductores diferenciados fundamentalmente en su forma de actuar y en el tipo de lenguaje que permiten traducir.

1. Ensambladores: Son muy elementales, destinados a traducción de programas escritos en lenguaje ensamblador. Debido a que la estructura de una instrucción ensamblador es muy similar a una instrucción en código maquina, la traducción en código maquina, la actuación de un ensamblador se reduce a un análisis de la sintaxis de la instrucción y su descodificación a binario, generando el llamado programa ejecutable. El programa en el lenguaje original recibe el nombre de programa fuente. El ensamblador solamente proporciona el programa ejecutable a partir del fuente no detecta errores sintácticos en el análisis de dicho programa. Como puede deducirse de lo anterior, los ensambladores realizan el trabajo de traducción en una sola fase, son traductores de una etapa.

2. Compiladores: Son los mas usados, ya que traducen los programas escritos en lenguajes de alto nivel. El proceso de traducción es mas complejo que en el anterior debido a la presencia de macroinstrucciones, por lo cual, dicho proceso se desarrolla generalmente en varias etapas, normalmente dos. Una primera etapa, el compilador realiza una labor de análisis del programa fuente comprobando la corrección del mismo. Si detecta errores de tipo sintáctico, comunicará los mismos al usuario de alguna forma, dando por terminada su labor. Si no hay errores presentes en el programa, realizara la descomposición de las macroinstrucciones a operadores elementales y su traducción a código maquina, generando el llamado programa objeto. Este programa objeto no es todavía ejecutable aun estando representando en código maquina. Algunos compiladores mas complejos realizan estas operaciones en mas fases, siendo estos los compiladores de tres, cuatro etc etapas. En una segunda etapa, teniendo el programa objeto y con la colaboración de algún otro programa del sistema (linkeador), se realiza la asignación de las direcciones reales, inclusión de definiciones y procesos etc que permitan finalmente la generación del llamado programa ejecutable. Una vez concluido estos procesos, el programa podrá ejecutarse cuantas veces se desee sin necesidad de nuevas traducciones o compilaciones.

3. Intérpretes: Son también traductores destinados a los lenguajes de alto nivel, pero con una forma de trabajo distinta a la de los compiladores. Como hemos visto anteriormente un compilador actúa sobre todo el programa fuente en conjunto, mientras que un interprete lo hace instrucción a instrucción: toma una instrucción del programa fuente, analiza su sintaxis, la descompone en operaciones elementales, descodifica a código maquina y por ultimo, la ejecuta. Si durante este proceso, el interprete localiza algún error de tipo sintáctico, lo comunica al usuario interrumpiendo su acción. Como puede deducirse fácilmente de esta forma de actuación, cuando se necesita ejecutar un programa mediante interprete, es necesario realizar el proceso de

traducción completo pues como se habrá observado, el intérprete no crea el programa ejecutable, con la consiguiente perdida de tiempo que el analizar instrucción por instrucción provoca.

Programa fuente:

Es el programa que escribió el programador. Ello se realiza a través de un editor de textos.

Programa objeto:

Es el resultado de convertir el programa fuente hacia el lenguaje de maquina. Este proceso es realizado por el compilador. El proceso se denomina compilación y el código resultado, objeto o compilado.

Programa ejecutable:

Es el resultado de agregar al programa objeto: definiciones, datos, procesos y direcciones reales de ejecución que permitan al programa comportarse de forma conveniente en la memoria del computadora durante su procesamiento. Esta labor la realiza el montador (linkeador)

Algoritmo

Es el conjunto de etapas o pasos que nos permiten solucionar un problema o situación.

Criterios de evaluación de lenguajes de programación

Al momento de evaluar un lenguaje de programación se debe tener en cuenta varios características:

Variedad de tipos de datos: Un buen lenguaje de programación debe contar con un variado número de tipos de datos, que le permitan todo tipo de procesos.

Programación estructurada: Consiste en desarrollar programas en los cuales cada proceso o parte del programa está claramente definido y no colisiona con los demás procesos. Esto quiere decir que el programa debe estar desarrollado de una manera organizada que nos permita entenderlo con facilidad para poder hacer correcciones futuras sin muchas complicaciones. Un buen lenguaje de programación debe permitir programación estructurada.

Programación Orientada a objeto (OOP): Son los que permiten aumentar la productividad del programador, incrementando la extensibilidad y reutilización del software, así como también controlar la complejidad y costo del mantenimiento del software.

Transportabilidad: Un lenguaje es transportable, si es posible escribir un programa en un tipo o modelo de computador, usando una versión del lenguaje diseñada para ese computador y luego poder usar el mismo programa en otro modelo o tipo de computador, con su respectiva versión del mismo lenguaje, sin necesidad de hacer cambios en el programa o quizás muy pocos.

Generación de código: Los programas objeto y ejecutables logrados a través de un lenguaje deben ser compactos y óptimos en su velocidad de proceso.

Flexibilidad: Un buen lenguaje de programación, debe permitir la realización de correcciones, modificaciones o mejoras, en los programas que se escriben con el, de una forma flexible y sin complicaciones.

Acceso a bajo nivel: La ejecución de rutinas escritas haciendo uso del bajo nivel, permitiendo ganar

velocidad en el procesamiento, por esta razón un buen lenguaje debe permitir algunas formas de acceso hacia este nivel.

Fácil de aplicar: Un lenguaje debe ser un método entendible y practico, que brinde facilidades al programador, para el desarrollo de aplicaciones.

Enlace con otros lenguajes: Un buen lenguaje de programación, debe contener características que le permitan ejecutar rutinas escritas en otros lenguajes de programación. Esta posibilidad es muy importante cuando se realizan grandes proyectos de programación, en los cuales es común el uso de rutinas o programas escritos en diferentes lenguajes.

Generación de librerías: Un buen lenguaje debe permitir agrupar en librerías o archivos, las rutinas que va desarrollando el programador durante la realización de un programa, de tal forma que le sea posible hacer uso de ellas en nuevos programas, sin ser necesario reescribirlas, sino simplemente agregarlas de forma inmediata en el código del nuevo programa.

ALGUNOS LENGUAJES

FORTRAN

Fue desarrollado por Backus en 1954, publicando IBM el primer manual en 1956. Su nombre proviene de FORmula TRANslation y es un lenguaje típicamente de cálculo, para misiones científicas y de ingeniería. Mantuvo durante mucho tiempo las formas básicas y el estilo que lo acercaban a la gestión, es decir se ha perfeccionado haciéndose más potente, pero su estilo sigue siendo el mismo.

Los tipos de datos permitidos son: enteros reales, complejos y alfanuméricos, pudiendo formar con ellos expresiones numéricas, lógicas y alfanuméricas. Usa bucles, bifurcaciones, subrutinas externas, etc lo que le hace valido para la resolución de cualquier tipo de problema científico, así como estructura de datos típicamente de gestión, tablas y cualquier tipo de archivo.

No es necesario numerar todas las sentencias del programa, solo aquellas a que se refiere un bucle o bifurcación. Una de las ventajas de FORTRAN es que las fórmulas algebraicas se escriben prácticamente igual que las fórmulas matemáticas convencionales.

COBOL

Su nombre deriva de COmmon Business Oriented Languaje, fue diseñado expresamente para el procesamiento de datos administrativos y de índole comercial, que surge como resultado del esfuerzo de creación de un lenguaje estándar para programar computadoras que procesan datos comerciales. Sus características fueron establecidas en 1959 por un comité de trabajo integrado por representantes de los principales usuarios de lenguajes de computador, fabricantes de computadores y representantes del gobierno, denominado CODASYL (Conference On Data Systems Lenguaje). La ultima versión de COBOL, fue aceptada en 1974 por el ANSI (American National Standars Institute).

Es un lenguaje estructurado, sigue ciertos principios de diseño modular, simplificando las tareas de programación y mejorando la autodocumentacion. COBOL hace una distinción útil entre elementos y grupo de elementos, formando jerarquías de estructura de datos.

RPG (Report Program Generator)

Esta orientado al uso administrativo y por su estructura el programador no necesita conocer la maquina, confiando esta tarea al compilador.

Fue desarrollado por IBM en los años 60, para usarlo en sus propios sistemas. En 1974 se crea una nueva versión que mejoraba la potencia del lenguaje y hacia más sencilla las operaciones de entrada y salida.

En 1980 aparece el RPG−II diseñado para entornos mas sofisticados, permitiendo usar programación estructurada, bases de datos etc

LOGO

Fue desarrollado en 1968 por la National Science Fundation como parte de un proyecto de investigación, con la principal finalidad de acercar a los niños al mundo del computador.

Es un lenguaje de procedimientos interjectivos, que usando la geometría de una tortuga como base, adentra a los niños en el mundo de la ciencia, las matemáticas y el área de la construcciones de modelos intelectuales.

El LOGO se compone de órdenes que a su vez forman procedimientos, todo en un modo gráfico, que permite dibujar líneas y girar en cualquier dirección, pudiendo crear figuras, sin tener que dibujar punto a punto.

ADA

Desarrollado por Honeywell Bull en 1878, para el departamento de defensa de EEUU, con en fin de que sea usado para crear programas en las grandes computadoras, como los usados por aviones, misiles, radares, sistemas de control de armamento etc

Es muy legible, permite la programación modular, tiene facilidades para la entrada y salida de datos, permitiendo el control de muchos dispositivos y el procesamiento paralelo.

LISP (LISt Processing)

Lenguaje usado en programación no numerica para inteligencia artificial. Diseñados en 1960 por John McCarthy, su sintaxis y estructura es muy diferente a la programación tradicional. Ej: No hay diferencia en la sintaxis de los datos e instrucciones.

PASCAL

Apareció en 1971, con el propósito fundamental de enseñar la programación estructurada. Su nombre es en honor al matemático Francés del siglo XVII, Blaise Pascal.

Es un lenguaje estructurado, es decir que sin importar el tamaño del programa, esta formado por subprogramas menores, cada uno de los cuales en sí mismo es un programa estructurado, por lo que son muy legibles, se puede definir todos los tipos de datos que se desee, por lo que es aplicable a diversos problemas.

Debe declararse al inicio todas las variables y su tipo, definir el uso de cada símbolo, haciendo que presente pocos errores y fáciles de corregir. Un programa en PASCAL tiene tres partes: cabecera, declaraciones y cuerpo.

BASIC (Beginners All Purpose Simbolic Instruction Code)

Fue diseñado inspirado en FORTRAN, como una versión mucho mas simplificada, que pretende brindar a los usuarios un medio fácil y practico para realizar programas de computadores, no obstante ha llegado a ser un lenguaje tan completo, que con las modificaciones actuales orientadas a programación estructurada.

Desarrollado durante los años 1963 y 1964 en el Dartmonth College de Hannover, bajo la dirección de los

profesores John Kemeny y Thomas Kurtz, pero desde entonces se han desarrollado muchas versiones o dialectos.

Se caracteriza por ser un lenguaje de alto nivel interactivo, normalmente trabajaba mediante interpretes, pero también hay estructurados y usando compiladores. Cada instrucción esta formada por: verbo y parámetros.

Visual Basic

Versión BASIC de Microsoft con una interface gráfica que permite el diseño del código arrastrando y soltando objetos desde una barra de herramientas.

Lenguaje C

La historia de la creación del lenguaje C se remonta a 1974 en los Laboratorios Bell había sido diseñado el UNIX y cansados los programadores de reescribir una y otra vez su código para cada nueva computadora que probaban, uno de ellos Ken Thompson creó un lenguaje que llamó B, una simple versión de un lenguaje llamado BCPL (una versión de CPL).

Luego con la intensión de mejorar el B, junto a Dennis Ritchie crearon C con el que reescribieron por última vez el UNIX y en las siguientes pruebas se limitaban a crear el compilador para la nueva computadora y con el mismo código fuente generaban un nuevo objeto, es por esto que UNIX puede correr en una gran variedad de micros, minis y mainframes..

Tiene muchas ventajas sobre otros lenguajes de programación, es muy potente, tiene gran variedad de operadores y comandos, es flexible, se presta para la programación estructurada, es portátil, es decir se puede ejecutar en otro sistema con unos pocos cambios, muy veloz, casi comparable al ensamblador, útil para escribir cualquier cosa, desde sistemas operativos y compiladores hasta juegos y aplicaciones.

Este Lenguaje de alto nivel es capaz de manipular la computadora como si fuese un programa ensamblador. Durante la segunda parte de 1980, C fue el lenguaje elegido por los programadores profesionales de software comercial.

Comparado con otros lenguajes de alto nivel C aparece complicado, pero permite gran flexibilidad. Fue escandalizado por ISO y ANSI (X3J11 committee) en 1989.

Ej: Turbo C y Microsoft C

C++

Versión C orientada a objeto creada por Bjarne Stroustrup. Se volvió muy popular porque combinaba la programación tradicional en C con la capacidad OOP. Smalltalk y otros lenguajes originales OOP no proveían la estructura familiar de lenguajes convencionales como C y PASCAL.

Ej: Borland C++ y Visual C++.

El C++ es el resultado de varios años de experimentación y búsqueda, en los laboratorios Bell de AT&T, para crear un sucesor al C incluyendo todas las características de este y añadiendo nuevas facilidades especialmente como la programación orientada a objeto (OOP). Sus principales características son:

- Programación estructurada y orientada a objeto
- Economía de expresiones
- Abundancia de operadores y tipos de datos

- Codificación de alto y bajo nivel simultáneamente.
- Reemplaza ventajosamente la programación en lenguaje de máquina o ensamblador
- Uso natural de las funciones primitivas del sistema.
- Producción de código objeto y ejecutables altamente optimados.
- Facilidad de aprendizaje.

Visual C++

Creada en 1993 por Microsoft, versión orientada a objeto que permite además la programación visual.

PowerBuilder

Generador de aplicaciones cliente/servidor de Microsoft, con un lenguaje de programación parecido a BASIC. Soporta SQL varios tipos de bases relaciónales de datos incluso DB2 y Oracle, reportes QUERY y generador de gráficos.

dBASE

Fue la primera DBMS (Data Base Mannager Software) para PC. Creada por Borland. Su nombre original fue Vulcan, dBASE fue creado por Wayne Ratliff para administrar un equipo de football americano. Fue pulido en el Jet Propulsion Labs de Los Angeles.

Renombrado dBASE II cuando Hal Lashlee y George Tate formaron Ashton−Tate para comercializarlo (Ashton−Tate fue adquirido por Borland en 1991), dBASE fue muy popular por muchos años, convirtiéndose en el estándar en PC.

dBASE es un lenguaje de 4ta generación, con un intérprete, además de permitir el uso en forma interactiva.

dBASE II

Introducido en 1981, originalmente introducido para computadores con CP/M y luego en PCs con DOS.

dBASE III

Introducido en 1994, fue una mejora importante introducido solo para PCs.

dBASE III Plus

Introducido en 1986, su principal cambio introducción de menús y la opción de relacionar las tablas en forma visual.

dBASE IV

Introducido en 1988, redefinen los menús e introducen generadores de reportes, pantallas y consultas QBE de lenguaje SQL. También incluyen matrices y ventanas. Versiones para PC, UNIX, VAXs y Macintosh.

Compilador dBASE

Definitivamente uno de los puntos débiles de dBASE por lo que Fox le quito la supremacía del mercado se deba a que dBASE no tenia un compilador y el código debía ser interpretado, restando mucha velocidad de ejecución.

Esto significa que usando el código de alto nivel, escribimos el codicio en un editor de textos ASCII (PRG) que posteriormente ejecutamos (DO), entonces dBASE tiene que mirar una línea de código fuente, traduciría a lenguaje maquina y ejecutar los mandatos resultantes. El problema es que esto lo hace cada vez que ejecutamos el programa.

Clipper, Force, Quicksilver, Reflex apareció inicialmente como compiladores para código generado en dBASE, subsanando así su deficiencia, lamentablemente al producirse errores de sintaxis era bastante difícil depurarlos.

FoxBASE+ nació con un compilador que generaba objetos que al ser ejecutados por vía el módulo ejecutor (RunTime) eran notoriamente más rápidos que el simple código a ser interpretado instrucción a instrucción, sin el problema de la difícil depuración.

FoxBASE+

DBMS creado por Fox Software (comprado posteriormente por Microsoft) que gano mucha popularidad por su gran velocidad y compatibilidad con dBASE III PLUS

FoxPRO

DBMS de Microsoft para PCs. Versión mejorada del FoxBASE. FoxPRO incluye ventanas, menus, consultas QBE de lenguaje SQL y "Rushmore" tecnología para la rápida consulta de tablas grandes. Compatible dBASE IV.

PRG −−(compilador)−−> FXP −−(enlazador)−−> EXE

Access

DBMS para Windows de Microsoft que lee datos desde dBASE, Paradox, archivos Btrieve, Microsoft y SYBASE SQL y otros. Su lenguaje de programación es el BASIC y una serie de magos ayudan en la creación de reportes, consultas y gráficos.

Xbase

Nombre genérico que se le da a dBASE, Clipper, FoxPRO, Paradox... Una simple comando como LIST (mostrar los datos de una base de datos) requerirían muchos comandos en un lenguaje de 3ra generación, leer un registro, verificar el fin de archivo, poner cada campo en la pantalla y repetir este proceso hasta terminar con todos los datos. Pero no todos los comandos de Xbase son de 4ta generación, por lo que algunos dicen que es una mezcla de 3ra y 4ta generación. Consultas y generación de reportes son también 4ta generación.

Capitulo 4

Normalización de sintaxis

Usar la notación Húngara.

- Primera letra de variable una minúscula (x) el resto mayúsculas.
- Todos los comandos y funciones en mayúsculas y solo las primeras 4 letras.
- Las funciones de usuario deben tener combinación de mayúsculas y minúsculas. Ej: xApePat, xFecNac etc.

Reglas de un buen programados:

- Hágalo simple
- Los archivos temporales, cambios de variable y exceso de llamadas a subrutinas o procedimientos son una trampa en la que se debe evitar caer.
- Los nombres de programas, procedimientos, funciones de usuario, variables, tablas y campos deben ser significativos a primera vista, como: CODCLI, VALVTA, VTABRU y no RAMBO, CHICHE, XXX o SDXP5D02. Si tienes un campo llamado CODCLI lo debes capturar usando la variable xCODCLI.
- Cuando tengas dudas sobre el paso a seguir, remítase a la primera regla.

Para que la programación en grupo funcione, debemos usar todos un mismo estilo, entre los mas importantes:

- El estilo de menús, pantallas y formatos debe ser uniformes. Es decir seguir el mismo orden: entrada, preparación de entorno, apertura de tablas, operaciones de control de flujo, salida. Ojo no abusar de las confirmaciones... estas seguro de que estas seguro ?
- Reglas fijas en la escritura de código fuente:
- Títulos: encerrados en *
- Comentarios: anteponiendo *−−−−
- Comandos y funciones de Fox en mayúsculas
- Variables en minúsculas y siempre con x, si es una variable que contiene un campo debe llevar el nombre del campo.
- Funciones de usuario en minúsculas y empezando con w.
- Pensar siempre en el usuario
- Con el usuario plantear el problema y solución en forma teórica y con sus palabras (nombre de opciones etc)
- Escribir código fuente como se ha planteado sin desviase
- Someter el resultado nuevamente al usuario.
- Parametrizacion de todo aquello que pueda ser interesante de cambiar después. Discos y directorios, colores de pantalla, nombre de empresa, año de trabajo etc
- Análisis exhaustivo del tiempo de ejecución, para minimizarlo.
- Cuidado con la excesiva proliferación de variables. Las variables saturan la memoria de trabajo, sobre todo las de tipo texto usadas como títulos.
- Conforme se va desarrollando una aplicación se debe ir elaborando la documentación pertinente, que debe contener mínimo la definición de las bases de datos, sus campos y sus índices. Seria aconsejable además un diagrama de flujo de datos entre las bases por tareas.

Consejos generales:

- Para un buen manejo de la memoria se recomienda el uso de código compacto, puedes usar solo las primeras 4 letras de cada comando o función.
- Si necesitas usar TEXT...ENDTEXT varias veces para presentar pantallas, es mejor crear una base de datos con un campo C con el nombre de la pantalla y un campo MEMO que la contenga.

```
SELE 10

USE PAN

SEEK xPAN

DISP MEMO
```

- Si necesitas poner 'Pulse una tecla para continuar...' muchas veces en tu programa, mejor guarda esta cadena en una variable y haces @ x,y SAY xMEN1

- Siempre usa SPACE() y REPL('−',40) en lugar de cadenas.
- Crea un archivo llamado ESTRU.DBF con los siguientes campos:

Campo C 10 Nombre del campo

Tipo C 1 Tipo de campo (C,N,D...)

Long N 3 Longitud del campo

Fil N 2 Fila en la que se posiciona

Col N 2 Columna " " "

Mensaje C 35 Mensaje para el prompt

Mascara C 12 Picture usado

Un ejemplo seria el siguiente:

Código C 4 5 10 Introduzca el código de usuario !

Nombre C 30 7 10 Introduzca el nombre del usuario !

Sueldo C 6 9 10 Introduzca el sueldo básico E 999,999

Su uso:

```
GO TOP
DO WHILE !EOF()
camp=TRIM(campo)
DO CASE
CASE tipo='C'
x&camp=SPACE(long)
CASE tipo='N'
x&camp=0
CASE tipo='D'
x&camp=DATE()
ENDCASE
xmasc=TRIM(mascara)
@ fil,col SAY mensaje GET x&camp PICT &xmasc
```

READ

SKIP

ENDDO

Capitulo 5

Análisis y diseño de sistemas

Dentro de las organizaciones, es el proceso de examinar la situación de una empresa con el propósito de mejorarla con métodos y procedimientos mas adecuados.

El análisis de sistemas, es el proceso de clasificación e interpretación de hechos, diagnostico de problemas y empleo de la información para recomendar mejoras al sistema. Es necesario comprender en su totalidad los detalles de una situación y decidir si es deseable o factible una mejora. La selección del método, ya sea usar o no una computadora, es un aspecto secundario.

El diseño de sistemas es el proceso de planificar, reemplazar o complementar un sistema de organizaciones existente.

Sistema

Es un conjunto de componentes que interrelacionan entre sí para lograr un objetivo común. Nuestra sociedad esta rodeada de sistemas:

- Todos tenemos un sistema nervioso, por medio del cual somos capaces de tener sensaciones físicas, formados por el cerebro, medula espinal, nervios y células sensoriales bajo la piel.
- Nos comunicamos con el lenguaje, que es un sistema de palabras y símbolos que tiene significado especifico para quienes lo usamos.
- Así mismo vivimos en una sociedad en la que intercambiamos bienes y servicios por otros de valor comparable en el que los participantes obtienen un beneficio en el intercambio.
- Una organización también es un sistema donde sus componentes (producción, ventas, contabilidad, investigación, personal, despacho) trabajan juntos para generar utilidades que beneficien tanto a los empleados como a los accionistas. Cada uno de esos componentes, es un sistema, que a su vez esta compuesto por otros sistemas.

La finalidad de un sistema es la razón de su existencia, para alcanzar sus objetivos interaccionan con su medio ambiente.

Así existen sistemas de información:

- Procesamiento de transacciones (TPS): Facturación, entrega de mercadería, pago de empleados, deposito de cheques...
- Información administrativa (MIS): Los que ayudan a tomar decisiones y resolver problemas, principalmente recurriendo a los datos almacenados en TPS acumulados en periodos (semanas, meses, años) o por línea de productos.
- Soporte de decisiones (DSS):

Analista de sistemas

Cuando un analista comienza a trabajar en un proyecto de sistemas de información, a menudo tiene que

profundizar en un área de la organización con la que tiene poca familiaridad. A pesar de esto, debe desarrollar un sistema que ayude a los gerentes y personal de esa área.

El analista debe conocer la respuesta a las siguientes preguntas:

1. Qué es lo que hace el sistema y cómo lo hace ?

2. Qué procesos integran el sistema ?

3. Qué datos emplea cada proceso ?

4. Qué datos son almacenados ?

5. Qué datos ingresan y abandonan el sistema ?

6. Qué tan grande es el volumen de transacciones o decisiones ?

Cualquier nuevo sistema o conjunto de recomendaciones para cambios en el sistema existente, ya sea este manual o automatizado, debe conducir hacia una mejora. Para alcanzar este resultado, se espera que el analista de sistemas haga lo siguiente:

- Aprenda los detalles y procedimientos del sistema en uso.
- Obtenga una idea de las demandas futuras de la organización como resultado del crecimiento, del aumento de la competencia en el mercado, de los cambios en las necesidades de los consumidores, de la evolución de las estructuras financieras, de la introducción de la nueva tecnología y cambios en las políticas del gobierno entre otros.
- Documente detalles del sistema actual para su revisión y discusión por otros.
- Evalúe la eficiencia y efectividad del sistema actual y sus procedimientos, tomando en cuenta el impacto sobre las demandas anticipadas para el futuro.
- Recomiende todas las revisiones y aplicaciones del sistema actual, señalando su justificación. Si es apropiado, quizá la propuesta de un nuevo sistema completo.
- Documentar las características de un nuevo sistema con un nivel de detalle que permita comprender a otros sus componentes e interpretación del problema, de manera que permita manejar el desarrollo del nuevo sistema.
- Fomentar la participación de gentes y empleados en todo el proceso, tanto para aprovechar su experiencia y conocimientos del sistema actual, como para conocer sus ideas, sentimientos y opiniones relacionadas con los requerimientos de un nuevo sistema o de los cambios para el actual.

Parece esta lista ser demasiado exigente ? quizá abrumadora ? aunque no siempre se expresa de manera concisa, estas tareas representan las expectativas que muchas organizaciones tienen sobre el trabajo de los analistas de sistemas. Como si fuera poco normalmente se añade otro requisito:

- La tarea debe completarse con rapidez, dentro de pasos establecidos y con un mínimo de interrupción para los gerentes y empleados.

Para tener éxito, un buen analista de sistemas estructura el proceso que debe seguir para desarrollar este trabajo.

Estrategias para el desarrollo de sistemas

Hay 3 distintos métodos para desarrollar un sistema:

1. Ciclo de vida:

Incluye las actividades de investigación preliminar, determinación de requerimientos, diseño del sistema, desarrollo del software, pruebas e implantación.

- Requerimientos del sistema deben ser previsibles.
- Manejable como proyecto.
- Gran volumen de transacciones y procesamiento.
- Requiere validación de datos de entrada.
- Abarca varios departamentos y es un desarrollo en equipo.
- Tiempo de desarrollo largo.

2. Análisis estructurado:

Se enfoca en lo que el sistema o aplicación realizan sin importar la forma en que llevan a cabo su función (se abordan los aspectos lógicos y no los físicos). Emplea símbolos gráficos para describir el movimiento y procesamiento de datos. Los componentes importantes incluyen los diagramas de flujo de datos y el diccionario de datos.

- Adecuado para todo tipo de aplicación.
- Principalmente usado cuando se trata de un desarrollo único y no en equipo.
- Mayor utilidad como complemento de otros modelos de desarrollo.
- Tiempo de desarrollo corto.

3. Prototipo:

Desarrollo iterativo o en continua evolución donde el usuario participa directamente en el proceso.

- Condiciones únicas de la aplicación donde los encargados del desarrollo tienen poca experiencia o información o donde los costos o riesgos de cometer un error pueden ser altos.
- Asimismo útil para probar la factibilidad del sistema, identificar los requerimientos del usuario, evaluar el diseño de un sistema o examinar el uso de una aplicación.
- Tiempo de desarrollo indefinido.

Ciclo de vida

Es un proceso formado por las etapas de análisis y diseño, comienza cuando la administración o algún miembro del personal encargado de desarrollar sistemas, detectan un sistema de la empresa que necesita mejoras.

1. Investigación preliminar:

Tiene tres partes: la aclaración de la solicitud, estudio de factibilidad (técnica, económica y operacional) y aprobación de la solicitud.

2. Determinación de los requerimientos del sistema

− Qué es lo que se hace ?

− Cómo se hace ?

− Con qué frecuencia se presenta ?

− Qué tan grande es el volumen de transacciones o decisiones ?

− Cuál es el grado de eficiencia con el que se efectúan las tareas ?

− Existe algún problema ? que tan serio es ? cual es la causa que lo origina ?

3. Diseño del sistema

4. Desarrollo del software

5. Prueba de los sistemas

6. Implementaron y evaluación

− Evaluación operacional

− Impacto organizaciones

− Opinión de los administradores

− Desempeño del desarrollo

Análisis estructurado

Es un método para el análisis de sistemas manuales o automatizados, que conducen al desarrollo de especificaciones para sistemas nuevos o para efectuar modificaciones a los que ya existen. Este análisis permite conocer un sistema, proceso o actividad en una forma lógica y manejable al mismo tiempo que proporciona la base para asegurar que no se omite ningún detalle pertinente.

El objetivo principal del análisis estructurado es organizar las tareas asociadas con la determinación de requerimientos para obtener la comprensión completa y exacta de una situación dada.

La palabra estructurado significa que el método intenta estructurar el proceso de determinación de los requerimientos comenzando con la documentación del sistema existente, intenta incluir todos los detalles relevantes, para que sea fácil verificar si se han omitido detalles, identificar los requerimientos.

Muchos confunden el análisis estructurado con el diagrama de flujo de datos que es solo una herramienta, aunque es quizá la herramienta esencial.

El análisis estructurado hace uso de las siguientes herramientas para responder las cuatro preguntas anteriormente planteadas:

1. Diagrama de flujo de datos:

Herramienta gráfica que se emplea para identificar y describir los componentes, procesos y datos de un sistema, junto con las relaciones entre estos ellos, el movimiento de los datos a través del sistema y lugares donde se almacenan dichos datos. Hay quienes dividen estos diagramas en lógicos y físicos.

2. Diccionario de datos:

Descripción de los datos y de dónde son usados en el sistema. Incluyen nombre, descripción, alias, formato de contenido, organización, donde son generados y por que procesos son usados.

3. Descripción de procesos y procedimientos:

Declaraciones formales que emplean técnicas y lenguajes que permiten a los analistas describir actividades importantes que forman parte del sistema. Describe jerarquías entre los módulos componentes y los datos que serán transmitidos entre ellos, el análisis de las transformaciones de entrada y salida de datos y el análisis de transacciones.

Herramientas para el desarrollo de sistemas

1. Para análisis

2. Para diseño

3. Para desarrollo

Preguntas de reflexión

- Cuál es el método de desarrollo correcto de un sistema ?
- Deben dos analistas desarrollar una lista idéntica de requerimientos cuando estudian en forma independiente la misma situación ?
- Para una situación dada, existe siempre un solo diseño correcto para el sistema ?
- Una vez que el analista de sistemas comprende los requerimientos de un sistema, tiene utilidad los diccionarios de datos ?

Bibliografía:

Análisis y diseño de sistemas de información

− James A. Senn − McGraw Hill

Introducción a la informática

− Jorge Rodríguez − Guía practica de Anaya Multimedia

Capitulo 1

Descripción:

FoxPRO es el más popular y completo programa de gestión de bases de datos (DBMS o Data Base Management System), combinando un potente motor con la facilidad de uso de los entornos gráficos, además de un gran numero de herramientas.

FoxPRO es además, un potente lenguaje de programación modular y de procedimientos orientado a eventos y no a menús jerárquicos y anidados (como FoxBASE, Clipper y dBASE). Esta filosofía le da nítidas ventajas, en el diseño de cualquier tipo de aplicación.

Versión de FoxPRO a ejecutar:

Los usuarios de FoxPRO 2.6 que tengan un procesador 386 SX o superior pueden elegir entre usar la versión Estándar o la versión Extendida de FoxPRO. En algunas ocasiones es mejor usar una versión en lugar de la otra, pero también hay otros casos en los que la elección no está tan clara.

Antes de decidir qué versión debe ejecutar, lea estas notas acerca del uso de memoria para las dos versiones:

Uso de la memoria en la versión Estándar

La versión Estándar se presenta en dos partes: una parte raíz (contenida físicamente en el archivo .EXE) y una serie de segmentos de código (contenida físicamente en el archivo .OVL).

Al iniciar el programa, se carga en la memoria la parte raíz y se realizan algunas asignaciones adicionales de memoria de tamaño fijo. Estas asignaciones añaden unos 290 KB de memoria. El conjunto de segmentos de código ocupa 1,1 MB aproximadamente.

Una vez cargada la parte raíz, el saldo de memoria por debajo de 640 KB, más los Bloques de memoria superior (UMB) disponibles más 64 KB de memoria EMS (si cargo EMM386) quedará a disposición de FoxPRO como pila de memoria de uso general para todos los datos de los usuarios y del sistema (variables de memoria, ventanas, BROWSE, consultas, segmentos de código, etc.), el resto de la memoria EMS se usa para los búferes de E/S y para el almacenamiento de segmentos de código inactivo.

Los segmentos de código se cargan bajo petición y flotan libremente por toda la memoria disponible, tanto en la memoria general como en la pila de memoria EMS. Los segmentos de código inactivo se descargan siguiendo el método del menos usado recientemente, según sea necesario con el fin de liberar memoria para otros fines. Si cuando se solicita un segmento de código necesario éste no está disponible, deberá cargarse desde el disco.

Puesto que el segmento raíz es relativamente pequeño, la versión

Estándar se ejecutará realmente en menos de 400 KB. Sin embargo, para la mayoría de las aplicaciones, un tamaño mínimo de memoria más realista es 512 KB. Por supuesto, una cantidad mayor de memoria puede mejorar considerablemente la velocidad de ejecución: cuanta más memoria, mejor.

Uso de la memoria en la versión Extendida

Primer MegaByte:

- Se usan 80 KB para una parte del Extender.
- Si DOSMEM=ON, el saldo de los primeros 640 KB se reasignará y quedará disponible como memoria extendida.
- Si DOSMEM=nnK, la cantidad especificada de memoria se conservará para su uso posterior mientras se reasigna el saldo de memoria y mientras queda disponible como memoria extendida.

NOTA: Si especifica un valor demasiado alto para DOSMEM=nnK y reduce el tamaño de este búfer o lo elimina completamente, quizá se degrade el rendimiento de E/S.

- De la memoria restante se asigna un búfer de 60 KB como máximo, dependiendo de la cantidad de memoria disponible. Cuando ejecute el comando RUN (!), se desasignará este búfer de forma que el programa que se ejecuta mediante RUN disponga de más memoria.
- No se usan los bloques de memoria superior (UMB).

Memoria extendida (EMS):

Todo el código para la versión Extendida, junto con el resto del Extender, se carga en la memoria y permanece residente y se hacen ciertas asignaciones de memoria fija. Para ello se requiere un total de unos 1,450 KB de memoria extendida.

Una vez realizadas estas asignaciones, el saldo de la memoria extendida quedará disponible para uso general; puede usarse para cualquier fin cuando sea necesario.

Puesto que la versión Extendida es realmente un producto a 32 bits, algunas estructuras internas de datos (variables de memoria, ventanas, etc.) requieren algo más de memoria en la versión Extendida que en la versión Estándar. Por tanto, los programas necesitarán normalmente más memoria para poder ejecutarse en la versión Extendida que en la versión Estándar.

Pros y contras:

- Con la versión Estándar, los segmentos de código FoxPRO no tienen por qué estar residentes en la memoria; se pueden cargar bajo petición. Por supuesto, no existe código del Extender. Esto significa que, para una cantidad determinada de memoria, habrá unos 1,1 MB más de memoria para búferes de E/S en la versión Estándar que en la versión Extendida.

- Pero, no todos los 1.1 MB pueden estar realmente disponibles para búferes. En aquellas situaciones reales donde hay aplicaciones grandes, se necesitaría gran parte de esa memoria para almacenar segmentos de código de FoxPRO y otros datos.

- Puesto que la velocidad de las operaciones de E/S puede depender en gran medida de la cantidad de memoria disponible, los programas que realizan muchas operaciones de E/S pueden verse enormemente beneficiados por esta memoria adicional. Obviamente, en un sistema de 3 MB sería MUY significativo un MB adicional de memoria para búferes; esto significa que, en estos sistemas menores, la versión Estándar probablemente superaría a la versión Extendida en las pruebas de E/S. Sin embargo, la ventaja marginal que proporciona esta memoria adicional para búferes se hace inapreciable a medida que se dispone de más memoria.

- Por otra parte, si su aplicación está dedicada principalmente a la entrada de datos y necesita la máxima sensibilidad de la interfaz cuando se usan muchas características nuevas, la versión Extendida será la mejor elección incluso en sistemas pequeños. Esto se debe a que en la versión Extendida todo el código de FoxPRO está residente y no hay sobrecargas al cargar el segmento.

- Además, en la versión Extendida, el tamaño de los programas, el número de ventanas, la longitud de las cadenas, el número de variables de memoria, etc. sólo están limitados realmente por la cantidad de memoria de su sistema. Tenemos sistemas en los que hay más de 30 MB de memoria disponible para la versión Extendida. Por el contrario, en la versión Estándar no puede acceder realmente a más de unos 460 KB de memoria libre total para uso general. Así, en el caso de aplicaciones complejas o de bases de datos muy grandes (especialmente cuando se usa el SQL), será obligatorio usar la versión Extendida.

Qué versión debes usar ?

- Si tu sistema tiene 4 MB o más de memoria, casi siempre será mejor usar la versión Extendida. No obstante, recuerde que independientemente de la cantidad de memoria que tenga, la versión Estándar puede tener más memoria disponible para búferes que la versión Extendida. Teniendo esto en cuenta, a veces es posible realizar pruebas de E/S en las que ganará la versión Estándar. Sin embargo con más de 4 MB, la versión Extendida casi siempre obtendrá mejores resultados en dichas pruebas.

- Entre 3 y 4 MB de memoria total, se produce un empate y lo recomendable es que pruebe ambas versiones y elija la versión más adecuada para su situación concreta.

- Si tiene 3 MB o menos, definitivamente deberá usar la versión Estándar.

Algunos limites mas saltantes:

- Máximo 1,000'000,000 de registros por tabla
- Máximo 255 campos por tabla
- Máximo 255 caracteres por campo
- Máximo 65,500 caracteres por registro

Simbología:

La simbología usada en esta separaba es la adoptada en la mayoría de los textos de FoxPRO.

<> Indica que el ítem encerrado deberá ser suministrado por el usuario al momento de usar el comando o función. Los símbolos <> no deberán ser incluidos.

[ ] Indican que el ítem encerrado es opcional. Los corchetes no son parte del comando.

| indica que se debe escoger una entre dos o mas opciones.

Capitulo 2

¡Error! Marcador no definido.

Programación en FoxPRO

Un programa FoxPRO es una secuencia de código contenido en un archivo de texto ASCII puro con la extensión PRG al que se le llama código fuente, que puede ser generado por cualquier procesador de textos.

Si opta por usar el editor incluido en FoxPRO, desde la ventana de comandos escriba: **MODIFY COMMAND <archivo>**, con lo cual si existe un archivo con ese nombre y extensión PRG se abre una nueva ventana donde se podrá editar su contenido. De no existir aún el archivo, se abre una ventana vacía.

Un vez terminado de escribir el programa, puedes hacer **<Ctrl><O>** desde el mismo editor, lo que hace que el compilador del FoxPRO verifique la sintaxis y si esta es correcta compilara el programa fuente generando el programa objeto, usando el mismo nombre y la extensión FXP.

También puedes ejecutar un programa desde:

1. La ventana de comandos con la orden: DO <archivo>

2. El prompt de DOS con: FOX <archivo> (versión iterativa)

3. El prompt de DOS con: FOXR <archivo> (versión runtime)

Nota: En las dos primeras opciones si la fecha del programa fuente (PRG) difiere de la del objeto (FXP) automáticamente se recompila la fuente generando un nuevo objeto. La opción 3 se limita a ejecutarla versión objeto si existe.

El compilador detecta cualquier error de sintaxis que exista en el archivo fuente. Los mensajes de error de compilación se guardan en un archivo de texto con el mismo nombre y extensión ERR.

Un lenguaje de programación esta fundamentado en tres soportes básicos:

- **Secuencias:** Comandos que se sitúan uno tras otro sin posibilidad de repetición cíclica alguna.
- **Selección:** Comandos que se ejecutan según cierta condición con los comandos: IF...ELSE...ENDIF o DO CASE...ENDCASE.
- **Bucles:** Comandos que se ejecutan un cierto numero de veces con los comandos: DO WHILE...ENDDO o FOR...ENDFOR.

Hay ciertos comandos básicos en FoxPRO como son:

HELP [<comando>] Solicita ayuda. También se pueda llamar con <F1>.

SET STAT ON/OFF Activa o desactiva la barra de estado.

CLEAR Limpia la pantalla

SET DEFA TO ... Define el disco y ruta de trabajo

SET PATH TO ... Al trabajar con varias rutas, es posible definir

las rutas alternativas

SET TALK ON/OFF Define el echo de los camandos sobre el escritorio.

DIR Muestra el contenido del directorio de trabajo.

RUN <comando> Ejecuta un comando directamente en el DOS, tambien puedes usar el simbolo ¡

RUN COMMAND Sale temporalmente al DOS.

* Comentario en una línea

&& Comentario a continuación de un comando en misma linea

Capitulo 3

Mostrar datos en pantalla

? | ?? <valor1>|<var1> [,<valor2>|<var2>...] [PICTURE <formato>] [FUNCTION <formato>] [AT <n>]

? y ?? evalúan expresiones y envían los resultados al escritorio.

? Un solo signo de interrogación envía un retorno de carro y un avance de línea antes de los resultados de la expresión. Los resultados se muestran en la siguiente línea del escritorio o la ventana activa definida por el usuario.

Si se omiten las expresiones, se imprime una línea en blanco.

Cuando se incluyen múltiples expresiones, se inserta un espacio entre los resultados de las expresiones.

?? Dos signos de interrogación muestran los resultados de la expresión en la línea y columna actual en el escritorio o la ventana activa definida por el usuario; no se envía retorno de carro ni avance de línea antes de los resultados.

Si está presente la cláusula PICTURE, la salida de <dato>|<var> se muestra de acuerdo con el formato especificado en <formato>, se puede incluir códigos de función anteponiendo el símbolo @, códigos PICTURE o una combinación de ambos. Es posible poner varios código de función juntos pero se deberá poner un espacio al final de los códigos de función, para diferenciarlos de los códigos picture.

Los códigos de función afectan de forma global a <dato>|<var>; los códigos PICTURE actúan solo sobre elementos (caracteres o dígitos) individuales del resultado.

AT se usa para especificar el numero de columna <n> donde se mostrará la salida. Esta opción permite alinear la salida en columnas para crear una tabla. Valores posibles entre 0 y 79.

Códigos Function:

B	Justifica los números a la izquierda
C	Pone CR después de los números positivos (Crédito)
D	Usa el formato de fecha actual según SET DATE
E	Usa el formato de fecha BRITISH
T	Elimina los espacios blancos delante y detrás
X	Pone DB en los números negativos (Débito)
Z	Presenta blanco en lugar de cero. Solo para números
(	Encierra los valores negativos entre ()
!	Convierte a mayúsculas. Solo para caracteres
^	Muestra datos numéricos usando notación científica. Solo puede usarse con datos numéricos.
$	Muestra datos en formato de moneda. El símbolo de moneda aparece antes o después del valor dependiendo de SET CURRENCY. Si CURRENCY esta en LEFT no podrá usarse el código de función $. Solo para números.

Códigos Picture:

X	Cualquier carácter
Y	Solo valores lógicos "Y", "y", "N" y "n".
!	Convierte a mayúsculas. Solo caracteres
9	Dígitos o signos
*	Muestra asteriscos a la izquierda de números.
.	Separador decimal
,	Separador de miles
$	Muestra el símbolo de moneda especificado por SET CURRENCY. Como opción predeterminada, el símbolo se sita inmediatamente antes o después del campo. Sin embargo, es posible cambiar el símbolo de moneda y su

	ubicación (SET CURRENCY), el carácter separador (SET SEPARATOR) y el carácter decimal (SET POINT).

Ejemplos:

? DATE()

? 15*(10+10)

? 'Bienvenido a ' FUNC '!'

?? 'FoxPRO'

? 'Hola',xNOM,'como estas ?'

SET DECIMALS TO [<n>]

Especifica el numero mínimo de lugares o dígitos decimales que se usaran para mostrar resultados de operaciones matemáticas, trigonométricas, financieras, etc...

<n> puede tomar un valor entre 0 y 18. El valor predeterminado es dos (2).

SET FIXED ON|OFF

Especifica si el numero de decimales usado a; mostrar datos numéricos, es fijo o no. De forma predeterminada esta en OFF.

NOTA: El numero de dígitos de precisión en los cálculos es de 16.

SET DECI TO 18

? 1000000/3	333333.333333333300000000
? 1000/3	333.333333333333300000
? 1/3	0.333333333333333300
? PI()	3.141592653589793000

SET CENTURY ON/OFF

Especifica si se usara o no los digitos de la centuria en las fechas

SET DATE TO <valor>

Especifica que formato de fecha usaras. AMERICAN mm/dd/aaaa o ITALIAN dd−mm−aaaa o JAPAN aaaa/mm/dd

Capitulo 4

VARIABLES DE MEMORIA

FoxPRO maneja cuatro tipos de variable:

- **Públicas:** Son las definidas en el programa principal y pueden ser usadas en cualquier subprograma o procedimiento.
- **Privadas:** Son las definidas en un subprograma o usando la orden PRIVATE y solo están disponibles en dicho subprograma o procedimiento.
- **De Sistema:** Son variables públicas incorporadas automáticamente por FoxPRO, no es necesario definir.
- **Arreglos, matrices o arrays:** Son variables especiales que contiene una serie de valores, en un arreglo unidireccional o bidirección. Las variables se identifican con un nombre y sus elementos con subíndices. Pueden ser públicas o privadas.

Los nombres de variables pueden contener hasta 10 caracteres significativos, formados por letras, números y subrayados, aunque es obligatorio que la primera sea una letra. Tenga en cuenta que para FoxPRO estas variables son las mismas:

xAPEPAT

xAPEMAT

El número máximo de variables de memoria que puede crear es de 1,024, pero en caso necesario este límite puede aumentarse (hasta 3,600 para la versión estándar o a 65,000 para la extendida) mediante la variable del sistema _MVCOUNT en el archivo de configuración FoxPRO.

Las variables de memoria deben ser definidas antes de ser usadas, así:

<var>=<valor>

Donde:

<var> nombre de variable conteniendo letras, números o subrayados.

<valor> FoxPRO permite cuatro tipo de valores: carácter o cadena (entre comillas), numéros, fechas (entre llaves) y lógica (entre puntos ej: .T.)

DISPLAY MEMORY [LIKE <estructura>]

Muestra el nombre, tipo, contenido y estado de todas las variables y matrices de memoria definidas actualmente, incluso del sistema. Además muestra el número de variables de memoria definidas, el número de bytes usados y el número de variables de memoria disponibles.

LIKE <estructura>

Puede mostrar información de forma selectiva sobre variables y matrices de memoria incluyendo la cláusula LIKE. Si incluye LIKE <estructura>,

FoxPRO mostrará únicamente el contenido de variables y matrices de memoria que cumplan con la <estructura>. La estructura de archivo <estructura> admite caracteres comodín. Por ejemplo, para mostrar todas las variables de memoria que comiencen por la letra A, utilice:

DISPLAY MEMORY LIKE A*

En este ejemplo, se crean varias variables de memoria y se les asigna valores. DISPLAY MEMORY muestra primero todas las variables de memoria que comienzan con "mue" y luego muestra todas las variables de

memoria que tienen seis letras y terminan con "salir".

muestra1='Adiós'

muestra2='Hola'

xSalir=.T.

mSalir=.F.

DISPLAY MEMORY LIKE mue*

DISPLAY MEMORY LIKE ?salir

CLEAR MEMORY

Libera de la memoria todas las variables y matrices, tanto públicas como privadas. Las variables del sistema no se liberan.

RELEASE <lista>

Libera de la memoria las variables y matrices cuyos nombres están contenidos en la <lista>. Separe los nombre de la lista de variables y matrices mediante comas.

SAVE TO <archivo> [ALL LIKE | EXCEPT <estructura>]

Guarda las variables y matrices en un archivo con el nombre especificado en <archivo> y de extensión MEM.

Incluya la cláusula ALL LIKE para guardar todas las variables y matrices que coincidan con la <estructura>. La estructura puede contener los caracteres comodines al estilo DOS ? y *.

Incluya la cláusula ALL EXCEPT para guardar todas las variables y matrices excepto las que coinciden con la <estructura>. La estructura puede contener los caracteres comodines al estilo de DOS ? y *.

RESTORE FROM <archivo> [ADDITIVE]

Restaura variables y matrices guardadas en un archivo de extensión MEM y las pone en la memoria.

Todas las variables y matrices se restauran como privadas cuando se usa RESTORE desde un programa y como públicas si se emite RESTORE desde la ventana de Comandos.

Cuando se emite RESTORE FROM, todas las variables o matrices que están actualmente en memoria se borrarán a menos que se incluya la palabra clave ADDITIVE. Este comando no afecta a las variables del sistema.

Si el numero de variables y matrices que se esta añadiendo con ADDITIVE, mas el numero de las variables y matrices actualmente en memoria excede el limite, se traerán a la memoria tantas variables y matrices como sea posible.

Si restaura una variable que tiene el mismo nombre que una existente, el valor de la variable de memoria existente es reemplazado por el valor de la variable restaurada.

Ejemplo:

val1=50

val2='Hola'

SAVE TO temp

CLEAR MEMORY

val3=DATE()

RESTORE FROM temp ADDITIVE

DISPLAY MEMORY LIKE val?

Elementos de la matriz:

El tamaño de una matriz determina cuántos elementos podrá contener. Cada elemento de la matriz puede almacenar una pieza sencilla de información.

Subíndices

Los elementos de una matriz se referencian mediante sus subíndices. Cada elemento de la matriz tiene un subíndice numérico único que lo identifica. Si la matriz es unidimensional, el subíndice de un elemento es igual que su número de fila. Por ejemplo, el subíndice del elemento de la tercera fila de una matriz unidimensional es 3. Los elementos de las matrices bidimensionales se referencian mediante dos subíndices:

El primer subíndice indica la fila y el segundo indica la columna en la que se encuentra el elemento. Por ejemplo, los subíndices del elemento que está en la tercera fila y en la cuarta columna de una matriz bidimensional son 3,4.

El subíndice o los subíndices del primer elemento de una matriz empiezan siempre con 1.

redimencionar matrices

Puede cambiar el tamaño y las dimensiones de una matriz emitiendo otra vez DIMENSION. El tamaño de la matriz puede aumentarse o disminuirse, las matrices unidimensionales pueden convertirse en bidimensionales y las matrices bidimensionales pueden reducirse a una dimensión.

Si el número de elementos de una matriz se incrementa, se copiará el contenido de todos los elementos de la matriz original a la matriz que se acaba de Redimensionar. El resto de los elementos de la matriz se inicializarán con el valor falso .F.

DIMENSION <matriz1> (<filas1>[,<columnas1>])

[,<matriz2> (<filas2>[,<colimnas2>])] ...

Crea o redimensiona una o más matrices.

El nombre de la matriz que se crea se especifica con <matriz1>. Pueden crearse múltiples matrices con una sola instrucción DIMENSION incluyendo nombres adicionales de matrices (<matriz2>, <matriz3> y así

sucesivamente).

Tras especificar el nombre de la matriz a crear, debe especificar el tamaño de la matriz con <filas> y <columnas>. Si incluyes solamente <filas>, se creará una matriz de una sola dimensión.

Por ejemplo, el comando siguiente crea una matriz unidimensional llamada MATRIZUNO que contiene diez filas y una sola columna:

DIMENSION matrizuno(10)

Para crear una matriz de dos dimensiones, incluya las dos expresiones: <filas> y <columnas>. El ejemplo siguiente crea una matriz bidimensional llamada MATRIZDOS que contiene dos filas y cuatro columnas.

DIMENSION matrizdos(2,4)

También es posible crear varias matrices en el mismo comando así:

DIMENSION matrizuno(10), matrizdos(2,4), matriztres(3,3)

Para enmarcar las expresiones en DIMENSION o DECLARE puede usar tanto paréntesis como corchetes. Por ejemplo, los siguientes dos comandos crean matrices idénticas:

DIME matrizuno(10), matrizdos(2,4), matriztres(3,3)

DIME matrizuno[10], matrizdos[2,4], matriztres[3,3]

Para determinar cuántos elementos contiene una matriz y cuánta información puede almacenar, multiplique el número de <filas> por el número de <columnas> de la matriz.

Los elementos de la matriz pueden contener cualquier tipo de dato y se inicializan a falso (.F.) cuando se crea la matriz por primera vez.

Puede inicializar todos los elementos de una matriz con el mismo valor mediante así:

DIME y[10,3]

y='inicial'

DIME matriz[6]

STORE 'A' TO matriz(1),matriz(2),matriz(3)

STORE 'B' TO matriz(4),matriz(5),matriz(6)

Si se disminuye el numero de elementos de una matriz, se borrarán los elementos y cualquier dato que contengan.

Cuando se redimensiona una matriz unidimensional a dos dimensiones, el contenido de la matriz unidimensional original se copia a la nueva matriz en el orden un elemento por fila.

En el ejemplo siguiente, una matriz unidimensional se convierte en una matriz bidimensional. El contenido de los elementos de la matriz unidimensional se copia a la primera fila de la matriz nueva, siguiendo por la

segunda fila y así sucesivamente. Los elementos restantes se inicializan con el valor falso (.F.).

DIME matrizuno(4)

matrizuno(1)= 'E'

matrizuno(2)= 'F'

matrizuno(3)= 'G'

matrizuno(4)= 'H'

Cuando una matriz bidimensional se convierte en unidimensional, el contenido de la matriz bidimensional original se copia a la nueva matriz en el orden fila a elemento: el primer elemento de la primera fila se convierte en el primer elemento de la matriz unidimensional, el segundo elemento de la primera fila se convierte en el segundo elemento y así sucesivamente.

Ejemplo:

En este ejemplo, se crea una matriz bidimensional y se carga con datos. Se muestran los elementos de la matriz y los datos que contienen.

DIME muestra(2,3)

muestra(1,2)= 'Adiós'

muestra(2,2)= 'Hola'

muestra(6)=99

muestra(1)=.T.

DISP MEMO LIKE muestra

Algunas funciones para matrices son:

Función	**Descripción**
ACOPY()	Copia elementos de una matriz a otra
ADEL()	Borra filas o columnas de una matriz
ADIR()	Captura nombre, tamaño, fecha, hora y atributos de archivo en ruta de trabajo en una matriz
AELEMENT()	Muestra valor de un elemento de matriz
AFIELDS()	Captura la estructura de una tabla en una matriz
AINS()	Inserta filas o columnas a una matriz
ALENS()	Muestra el numero de elementos de matriz
ASCAN()	Busca una expresión en una matriz
ASORT()	Ordena una matriz ascendente o descendente
ASUBSCRIPT()	Transforma # de elemento en fila y columna

ACOPY(<matriz1>,<matriz2>,[,<desde>[,<cuentos> [,<donde>]]])

Copia los elementos de <matriz1>, que debe existir, sobre <matriz2> que puede existir o no. Fox la crea. Devuelve el numero de lementos copiados.

<desde> # de elemento de <matriz1> desde el que se iniciara la copia, es decir numero anteriores no seran copiados.

<cuantos> # de elementos de <matriz1> a copiar. −1 si quieres copiar todos los elementos.

<donde> # de elemento de <matriz2> donde se pondran los elemetos que vienen de <matriz1>

ASORT(<matriz>[,<desde>[,<filas> [,<orden>]]])

Ordena los elementos de <matriz> alfabeticamente ascendente o descendente, basandose en una columna basica y manteniendo las filas como una unidad. Todos los elemtos a ordenar deben ser del mismo tipo. Devuelve 1 si tiene éxito y −1 en caso contrario.

<desde> # de elemento desde el que se iniciara el ordenamiento y al mismo tiempo columna sobre la que se ordenara.

<filas> # de filas que se ordenaran

<orden> 0 si quieres orde ascendente y 1 si quieres descendente.

ADIR(<matriz>[,<filtro>[,<tipo elemeto>]])

Copia a <matriz> los elementos encontrador en el directorio de trabajo y otro que se indique según el filtro y atributos impuestos. Devuelve el numero de filas de la matriz que equivale al numero de entradas capturadas.

Cada entrada contendra: nombre, tamaño, fecha, hora y atributos DOS de cada elemetos.

Los atributos son los mismos usados en DOS: A = para sacar backup, R = solo lectura, H = escondido y S = archivo de sistema.

<filtro> Puedes usar unidad (C:) ruta (\dos\) y filtro (*.prg) para filtras unicamente los elementos que quieres obtener. Puedes usar comodines como * y ? y debe estar todo en comillas.

<tipo elemento) Puedes usar los siguientes: D = Directorio, V = Volumen, H = Escondidos o S = Sistema.

AELEMENT(<matriz>,<fila>,<columna>)

Devuelve el # del elemento que corresponde a las coordenadas de fila columna de la matriz.

ASUBSCRIPT(<matriz>,<elemento>,<clave>)

Devuelve el numero de lemento. Si clave toma el valor de 1 se devuelve el numero de fila que corresponde a dicho elemento dentro de la matriz. Si clave toma el valor de 2 se devuelve el numero de columna de dicho elemento en la matriz.

ALEN(<matriz>,<clave>)

Devuelve el tamaño de la matriz. Si clave toma el valor de 1 se devuelve el numero de fila que

ASCAN(<matriz>,<expr>,<desde>,<cuantos>)

Busca la <expr> entre los elemtos de la <matriz>, iniciando la busqueda desde el elemento <desde> e incluyendo tantos elementos como indicado en <cuantos>.

Algunos otros comandos de matriz son:

SCATTER TO MEMVAR = Copia campos a una matriz

GATHER FROM MEMVAR = Copia la matriz a los campos

APPE FROM <matriz> = Añade registros a una tabla desde una matriz

COPY TO <matriz> = Copia registros desde una tabla a una matriz

Capitulo 5

Operadores

¡Error! Marcador no definido.Aritméticos:	
¡Error! Marcador no definido.+	Suma
−	Resta
*	Multiplicación
/	División
^ o **	Potencia
%	Modulo o residuo

Relacionales:	
=	Igual
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
# o <>	Diferente

¡Error! Marcador no definido.Lógicos:	
¡Error! Marcador no definido..NOT. o !	Negación
¡Error! Marcador no definido..OR.	Unión
¡Error! Marcador no definido..AND.	Intersección

Caracteres:	
+	Concatenación
−	Concatena sin espacios.
=	Igual
==	Exacto
$	Incluido
# o <>	Diferente

Prioridad de operación:		
¡Error! Marcador no definido. Grupos	()	1

Potencia	^ o **	2
Multiplicación y división	* y /	3
Suma y resta	+ y −	4
Operadores relacionales	=, >, <, #, $...	5
Operadores lógicos	! o .NOT., .OR. y .AND.	6

Ejercicio 1: Dadas las variables A=3, B=7 y C=1 determinar:

¡Error! Marcador no definido.Valor ?	**Verdadera o falsa ?**
A+B*C	A > 3
2*C+14/B	7 > B−1
1/(2*A−3*C)	11 = A+B+C
C*(A+B)	C/C^B+3 >= A
A*C+B*C−2	2*A+B = 20
2*C*(A+3*B)	A*B−C < A^A−B
B+(A−1)^A−1	3*(A^3−2*C)/7 # B−4
B^A+B^C	a $ 123
(2*B+C)/A−6	a $ 12ab345
2^A+2^B+2^C	a21 $ 12ab345

Ejercicio 2: Dadas las variables, A=5, B=8, C=2 y D=7, determinar el valor de las siguientes expresiones:

Verdadera o falsa ?
¡Error! Marcador no definido.A^C>B
4−A>C−B
D#A+C
5*A>=A+B+C+D
.NOT. A>2
! A#D−C
A−3#4 .AND. 2*C<=4
2*C=A .AND. B/C+2<D−C
A>0 .OR. B<0 .OR. C>0
3<B .OR. (A<=5 .AND. C#7)
(A#4*C .AND. 5>=A) .OR. (1−C>B−2*A)
(A>=D+1 .OR. ! B<C) .AND. (B>C−1 .OR. C#B*2)

Ejercicio 3: Dadas las variables,

A=`javier `

B=`hola'

C=`vi'

D=`JA'

Determinar el valor de las siguientes expresiones:	
A + B	C $ A
B + A	D $ A
A − B	A > D
A = C	A + B = A − B
A = D	D !$ A
C = A	? A FUNC `!'
D = A	? A PICT `!'
A == D	? A PICT `!XXXXXXXX'

Capitulo 6

CONTROL DEL FLUJO DE UN PROGRAMA

Es la lógica de un programa y la faceta mas importante de cualquier lenguaje de programación. Las sentencias **<cond>**icionales que cambian la dirección del flujo y la iterativas (bucles o lazos) que repiten un grupo de **<comandos>** un cierto numero de veces, son las herramientas que nos permiten crear programas útiles.

FoxPRO fue diseñado para que el programador dirija el flujo de control de una forma estructurada y racional, de modo que los programas resultantes sean fáciles de entender y modificar cuando sea necesario.

Es costumbre sangrar (indentar) usando la tecla **Tab** para ayudar a captar visualmente la estructura de anidamiento, aunque esto es ignorado por el compilador.

Condicional simple:

¡Error! Marcador no definido.IF <cond>

<comandos>

[ELSE

<comandos>]

ENDIF

1

Ejecuta condicionalmente un conjunto de comandos dependiendo del resultado de una expresión lógica.

Si <cond> da como resultado verdadero (.T.), se ejecutarán todas las instrucciones comprendidas entre el IF y ELSE o ENDIF (de no existir ELSE).

Si <cond> es falsa (.F.) y se incluye ELSE, se ejecutarán todas las instrucciones comprendidas entre ELSE y ENDIF. De no existir ELSE, se ignorarán todas las instrucciones entre IF y ENDIF, continuando el programa en la línea siguiente a ENDIF.

Se puede anidar un bloque IF...ENDIF dentro de otro bloque IF...ENDIF.

Ejemplo 1: Crear un programa que dado un numero, determine si este es mayo que 5 o no.

```
IF x>5

? 'x es mayor que 5'

ELSE

? 'x es menos o igual a 5'

ENDIF
```

Ejemplo 2: Crear un programa que dado un numero, determine si este es positivo, negativo o cero.

```
IF x>0

? 'el numero es positivo'

ELSE

IF x<0

? 'el numero es negativo'

ELSE

? 'el numero es cero'

ENDIF

ENDIF
```

Ejemplo 3: Crear un programa que dada una nota, responda si es aprobatoria, desaprobatoria o invalida.

```
IF nota<0 .OR. nota>20

? `Esa nota no en valida'

ELSE

IF nota>=10.5

? `Aprobado'

ELSE

? `Desaprobado'

ENDIF

ENDIF
```

Condicional múltiple:

¡Error! Marcador no definido.DO CASE

CASE <cond1>

<comandos>

[CASE <cond2>

<comandos>

...

CASE <condN>

<comandos>]

[OTHERWISE

<comandos>]

ENDCASE

2

Ejecuta el primer bloque de instrucciones entre DO CASE y END CASE cuya instrucción condicional asociada tenga el valor verdadero (.T.).

DO CASE se usa para ejecutar un conjunto de comandos dependiendo del resultado de una condición lógica. Cuando se ejecuta DO CASE, se evalúan cada uno de los casos lógicos; el resultado de la evaluación determina el conjunto de comandos que se ejecutar .

Cuando se encuentre la primera cláusula CASE cuyo valor sea verdadero (.T.), se ejecutarán las instrucciones que haya a continuación y luego salta en busca de la cláusula ENDCASE, reanudando la ejecución en el comando que hay inmediatamente a continuación de ENDCASE.

Se ejecuta una y sola una, instrucción CASE (la primera <cond> que dé resultado verdadero) y ni siquiera serán evaluados los próximos CASE.

Si todas las instrucciones CASE resultan falsas (.F.), se ejecutaran los comandos entre OTHERWISE y ENDCASE.

Si se omite OTHERWISE, la ejecución saltar al primer comando a continuación de ENDCASE.

Ejemplo 1: Crear un programa que dado un numero determine si este es positivo, negativo o cero.

DO CASE

CASE x>0

? 'el numero es positivo'

CASE x<0

? 'el numero es negativo'

OTHERWISE

? 'el numero es cero'

ENDCASE

Ejemplo 2: Crear un programa que dada una nota, responda si es aprobatoria, desaprobatoria o invalida.

DO CASE

CASE nota>20

? `Esa nota no en valida'

CASE nota>=10.5

? `Aprobado'

CASE nota>=0

? `Desaprobado'

OTHER

? `Esa nota no en valida'

ENDCASE

Repetitiva condicional:

¡Error! Marcador no definido.DO WHILE <cond>

<comandos>

[LOOP]

[EXIT]

ENDDO

3

Ejecuta un bloque de instrucciones dentro de un bucle condicional un numero de veces dependiente del estado de una condición..

Un conjunto de comandos situados entre DO WHILE y ENDDO se ejecuta mientras que el resultado de la expresión lógica <cond> sea verdadero (.T.). Una instrucción DO WHILE debe tener su correspondiente instrucción ENDDO.

<cond> es una expresión lógica y las instrucciones situadas entre DO WHILE y ENDDO se ejecutarán

mientras que dicha condición sea verdadera (.T.).

LOOP puede situarse en cualquier parte entre DO WHILE y ENDDO, devuelve el control del programa directamente a la instrucción DO WHILE, así que se vuelve a evaluar <cond>.

EXIT transfiere el control del programa desde el interior del bucle DO WHILE al primer comando a continuación de ENDDO. EXIT puede situarse en cualquier parte entre DO WHILE y ENDDO.

Ejemplo 1: Crear un programa que muestre los números en orden ascendente, del 1 al 10 inclusive.

```
x=0

DO WHILE x<10

x=x+1

? x

ENDDO
```

Ejemplo 2: Crear un programa que muestre el cuadrado de los 100 primeros números, con pausas de 10 segundos para cada pantalla.

```
x=100

y=1

DO WHILE x>0

? x PICT `999','al cuadrado es',x^2 PICT `99,999'

IF x<20

y=y+1

ELSE

WAIT WIND 'Una tecla para continuar' TIME 10

CLEAR

y=1

ENDIF

x=x−1

ENDDO
```

Repetitiva de valor fijo:

¡Error! Marcador no definido.FOR <var>=<exp1> TO <exp2> [STEP <exp3>]

<comandos>

[EXIT]

[LOOP]

ENDFOR | NEXT

4

Ejecuta los comandos que hay entre FOR y ENDFOR un numero concreto de veces.

FOR ... ENDFOR ejecuta un conjunto de instrucciones dentro de un bucle un numero especificado de veces. Se utiliza una variable de memoria o un elemento de matriz como contador para especificar cuantas veces se ejecutan las instrucciones que hay dentro del bucle.

Las instrucciones de programa que hay a continuación de FOR se ejecutan hasta que se llegue a ENDFOR o NEXT. Entonces se incrementa <var> con el valor de <exp3>. Si se omite la cláusula STEP, el contador se incrementa en 1. Entonces se compara el contador con el valor final <exp2>. Si el contador es inferior o igual al valor final <exp2>, volverán a ejecutarse las instrucciones que siguen a la cláusula FOR.

Si el contador es mayor que el valor final <exp2>, la ejecución del programa bifurca fuera del bucle FOR ... ENDFOR y continua en la siguiente línea después de ENDFOR o NEXT.

Precauciones:

- Los valores de <exp1>, <exp2> y <exp3> se leen únicamente al inicio.
- Si cambias el valor del contador <var> dentro del bucle, afectaras al numero de veces que se ejecuta el bucle.
- Si el valor de <exp3> es negativo y el valor inicial <exp1> es mayor que el valor final <exp2>, el contador disminuirá cada vez que pase el bucle.

<var> es una variable o un elemento de matriz que actúa como contador, que no es necesario que exista antes de ejecutar FOR ... ENDFOR.

<exp1> es el valor inicial del contador y <exp2> es el valor final.

<exp3> es la cantidad de incremento o decremento (si es negativa) del contador. Si se omite el contador se incrementar en 1.

<comandos> son los comandos FoxPRO a ejecutar.

EXIT transfiere el control desde dentro del bucle FOR ... ENDFOR al comando que sigue inmediatamente a la palabra clave ENDFOR. Puede situar EXIT en cualquier parte entre FOR y ENDFOR.

LOOP puede situarse en cualquier parte entre FOR y ENDFOR. La inclusión de LOOP devolverá el control directamente a la cláusula FOR sin ejecutar las instrucciones que haya entre LOOP y ENDFOR. El contador se incrementará o decrementará como si se hubiera llegado a ENDFOR.

Ejemplo 1: Mostrar todos los numero del 1 al 20.

FOR i=1 TO 20

```
? i

ENDFOR
```

Ejemplo 2: Mostrar los números pares hasta el 10 en forma descendente. Toma los valores desde variables

```
A= 10

B= 1

C= − 2

FOR i=A TO B STEP C

? i

ENDFOR
```

Ejemplo 3: Mostrar el cubo de los impares hasta el 10 en forma ascendente.

```
FOR n=1 TO 10 STEP 2

? n^3

ENDFOR
```

Ejemplo 4: Pedir se ingrese el resultado de tirar un dado mientras no se pulse la trecla Esc y luego mostrar el numero de veces que se tiro el dado.

```
CLEA

xDADO=0

n=0

DO WHILE .T.

@ 5,5 SAY `Valor:' GET xDADO pict `9' RANGE 1,6

READ

IF LASTKEY()=27

@ 7,7 SAY `Fueron `

¿? n pict `99', `veces'

EXIT

ELSE

n=n+1
```

ENDIF

ENDDO

Capitulo 7

CONTROL SAY/GET

@ <fila>,<columna> [SAY <exp>] [GET <var>/<campo>] [FUNC <formato>] [PICT <formato>] [DEFA <exp>] [MESS <exp>] [SIZE 1,n] [RANGE <min,max>] [VALID <cond> [ERROR <exp>]] [WHEN <cond>] [COLOR <pares>]

SAY visualiza la expresión en las coordenadas indicadas

GET asigna a la variable o campo el valor tecleado

La lectura se hace cuando se encuentra el comando **READ**.

FUNCTION/ PICTURE <formato>

Con <formato> se define el formato de presentación según la tabla siguiente.

Códigos Function:

A	Solo caracteres alfabéticos, no espacios ni símbolos
B	Justifica los números a la izquierda
D	Usa el formato de fecha activo según SET DATE
E	Usa el formato de fecha BRITISH
I	Centra texto
J	Justifica texto a la derecha
K	Selecciona para edición un campo entero cuando el cursor se desplaza al campo.
L	Muestra ceros a la izquierda. Solo para números
M <list>	Crea múltiples opciones predefinidas. La lista es un conjunto de elementos delimitados por comas. Los elementos individuales que aparecen dentro de la lista no pueden tener comas incrustadas. Si <var> o <campo> no contiene inicialmente uno de los elementos de la lista cuando se emite READ, se mostrará el primer elemento de la lista. Para desplazarse por la lista, presione la barra espaciadora o escriba la primera letra de un elemento. Para elegir uno de los elementos y desplazarse al siguiente control, presione <Enter> . Sólo puede usarse con datos alfanuméricos.
R	Muestra una mascara que no será grabada
S<n>	Limita el ancho a n caracteres, aunque puedes poner mas.
T	Elimina los espacios en blanco iniciales y finales.
X	Cualquier carácter
Z	Presenta blanco en lugar de cero. Solo para números
!	Convierte a mayúsculas. Solo para caracteres

’	Muestra datos numéricos usando notación científica. Solo puede usarse con datos numéricos.
$	Muestra datos en formato de moneda. El símbolo de moneda aparece antes o después del valor dependiendo de SET CURRENCY. Si CURRENCY esta en LEFT no podrá usarse el código de función $. Solo para números.

Códigos Picture:

A	Solo permite caracteres alfabéticos
L	Solo permite datos lógicos
N	Solo permite letras o dígitos
X	Permite cualquier carácter
Y	Solo permite valores lógicos "Y", "y", "N" y "n".
!	Convierte a mayúsculas. Solo caracteres
9	Solo permite dígitos o signos
#	Solo permite dígitos, blanco o signo
*	Muestra asteriscos a la izquierda de números.
.	Separador decimal
,	Separador de miles
$	Muestra el símbolo de moneda especificado por SET CURRENCY. Como opción predeterminada, el símbolo se sita inmediatamente antes o después del campo. Sin embargo, es posible cambiar el símbolo de moneda y su ubicación (SET CURRENCY), el carácter separador (SET SEPARATOR) y el carácter decimal (SET POINT).

Las expresiones de FUNCTION pueden ser introducidas en PICTURE con un símbolo @. Se puede poner varias condiciones solo hay que separarlas por ;

DEFAULT <Exp>

Si no existe esta cláusula deberá ser definida previamente la variable usada en el GET u obtendrás un error. En caso de ya existir la variable, esta cláusula es ignorada.

MESSAGE <Exp>

Muestra <Exp> como un mensaje sobre la línea 24. Este mensaje podría ser presentado en otra línea con SET MESS TO

SIZE <1>,<n>

Con <n> puedes manejar el ancho (# de columnas) de <var>/<campo>. Si no usas esta cláusula el ancho lo determina la cláusula PICTURE. Si tampoco se uso esta cláusula el ancho queda definido por el contenido actual de la variable o campo. Solo en FoxPRO para Windows será posible también definir el alto (# de filas).

RANGE <min>,<max>

Solo se usa para datos numéricos o de tipo fecha y es el rango en el cual deberá estar contenido el dato. Esta

condición solo se evalúa cuando se cambia de valor y no cuando solo aceptas el valor inicial vía <Enter>. En caso de error se presenta un mensaje de error indicando el rango valido impidiendo que se abandone el GET. Es decir es una post−condicion.

VALID <Cond>

Es una Post−condicion es decir que una vez ingresado un valor y se intente salir del GET se evaluara <Cond>. Si esta es verdadera, permitirá la salida, pero si es falsa se devuelve un mensaje de error estándar o la <Exp> de la cláusula ERROR impidiendo que se abandone el GET. Nota que se puede salir del GET con <Esc>, pero no aceptara el cambio.

Es posible devolver un numero en lugar de un valor lógico, si es 0 el curso no avanzara, quedándose en el mismo objeto, si el numero es positivo avanzara y si es negativo retrocederá según el numero.

ERROR <Exp>

Si se añade esta cláusula junto con VALID se mostrara <Exp> en lugar del mensaje estándar de FoxPRO.

WHEN <Cond>

Solo permitirá editar la <var>/<campo> si <Cond> es verdadera antes de entrar a dicho GET. Es decir es una pre−condicion.

COLOR

Permite determinar una serie de pares de color, que afectan a diversas partes según la siguiente tabla:

Colores		
N	Negro	0
W	Blanco	7
R	Rojo	4
G	Verde	2
B	Azul	1
RB	Magenta	5
BR	Cyan	3
GR	Marrón	6
GR+	Amarillo	6+
X	Invisible	

Atributos	
+	Mas intenso primer plano
*	Aclarar color de fondo

Monocromáticos	
N	Negro
W	Blanco
U	Subrayado
I	Vídeo inverso
+	Mas intenso primer plano
*	Parpadeante

Pares de color	
1	Expresión de SAY
2	Región de edición del GET
5	Texto de mensaje

Ejemplos: Los cuatro siguientes ejemplo ponen Nombre en color rojo con fondo azul y un campo para el GET azul intenso sobre rojo.

@ 5,5 SAY `Nombre:' GET xNOM COLO R/B,B+/R

@ 5,5 SAY `Nombre:' GET xNOM COLO 4/B,1+/R

@ 5,5 SAY `Nombre:' GET xNOM COLO 4/1,1+/4

Capitulo 8

@ <f1,c1> [CLEAR [TO <f2,c2>]]

Limpia parte de la pantalla, desde la esquina superior izquierda definida con <f1,c1>.

Si se emite solo @ <f1,c1> se limpia solamente la fila hasta el final de la misma línea, es decir desde: <f1,c1> hasta <f1,79>.

Si se emite solo @ <f1,c1> CLEAR se limpia un rectángulo compuesto por la esquina superior izquierda en <f1,c1> y la esquina inferior derecha, ósea <24,79>

Si se emite la cláusula completa osea: @ <f1,c1> CLEAR TO <f2,c2> se limpiara un rectángulo con la esquina superior izquierda en <f1,c1> y la esquina inferior derecha en <f2,c2>.

Ejemplo: Crear un programa que luego de rellenar toda la pantalla con algun carácter (Ejemplo *), en la primera fila pedir se ingresen las coordenas desde donde se limpiara.

CLEA

FOR F=1 TO 23

¿ REPL(`*',79)

ENDFOR

xF=0

xC=0

@ 0,0 SAY `Coordenadas:' GET xF PICT `99' RANGE 0,24

@ 0,16 SAY `,' GET xC PICT `99' RANGE 0,78

READ

@ xF,xC CLEAR

@ <f1,c1> TO <f2,c2> [DOUBLE / PANEL / <Exp>] [COLOR <par>]

Dibuja una línea o rectángulo en la pantalla entre las coordenadas especificadas.

Si unamos la cláusula DOUBLE se dibuja un rectángulo de línea doble, con PANEL se dibuja un rectángulo de línea grueso y con <Exp> se puede definir el carácter que se quiere usar.

@ <f1,c1,f2,c2> BOX [<lista>]

Construye una caja entre las coordenadas indicadas y con los códigos ASCII indicados en la <lista> en orden

horario, osea en el siguiente orden:

#	Descripción
1	Esquina superior izquierda
2	Línea horizontal superior
3	Esquina superior derecha
4	Línea vertical derecha
5	Esquina inferior derecha
6	Línea horizontal inferior
7	Esquina inferior izquierda
8	Línea vertical izquierda
9	Carácter de relleno

@ <f1,c1> FILL TO <f2,c2> [COLOR <par>]

Cambia el color del texto existente dentro de una zona de la pantalla.

Este comando cambia los colores del texto dentro de una zona rectangular del escritorio. Es posible establecer los atributos de color del primer plano y segundo plano sólo del texto ya existente.

Cualquier salida que se dirija a la misma zona después de emitir @ ... FILL se mostrar con los colores predeterminados de la pantalla.

Capitulo 9

Funciones

1. Caracteres

ALLTRIM()

Suprime los espacios blancos antes y después de la cadena.

ASC(exp)

Retorna el código ASCII del carácter mas a la izquierda de <exp>

AT(<Exp1>, <Exp2> [, <N>])

Devuelve un numero que corresponde a la posición que ocupa el primer carácter de Exp2 dentro de Exp1, contando desde el carácter más a la izquierda. Si la expresión de carácter no se encuentra, se devuelve 0.

La búsqueda distingue mayúsculas de minúsculas.

Si se incluye N, se buscará la N ava ocurrencia de Exp2 dentro de Exp1. De forma predeterminada, se busca la primera ocurrencia.

cadena='Esta es la hora de los hombres buenos'

cad_búsq='es la'

? AT(cad_búsq,cadena)

cadena='ABABAB'

cad_búsq='AB'

? AT(cad_búsq,cadena,2)

cad_búsq='ES'

? AT(cad_búsq,cadena)

USE partes

cadena='Tuerca'

LIST ALL códpart,descripc FOR AT(cadena,descrip) > 0

ATC(ExpC1,ExpC2)

Muestra la posición de una subcadera ExpC1 en otra cadena ExpC2.

ATCLINE()

Muestra el numero de línea del primer carácter de una subcadena en una cadena. No es sensible a mayúsculas / minúsculas.

ATLINE()

Muestra el numero de línea del primer carácter de una subcadena en una cadena.

CHR(n)

Carácter que corresponde al numero ASCII n

EMPTY(Exp)

Determina si Exp esta vacía o blanco

LEFT(<Exp>, <N>)

Extrae los N caracteres más a la izquierda de la cadena Exp.

Si N es mayor que la longitud de ExpC, se devolverán todos los caracteres de la expresión. Si N es negativa o 0, se devolverá la cadena nula.

? LEFT('Redmond, WA', 7)

LEN(<exp>)

Nos devuelve la longitud de la cadena <exp>

LTRIM()

Elimina espacios a la izquierda de la cadena <exp>

LOWER(<Exp>)

Convierte la cadena Exp en minúsculas.

nombre='FOX'

? LOWER(nombre)

ISLOWER(<exp>)

Devuelve .T. si el primer carácter de <exp> es una letra minúscula.

ISUPPER(<exp>)

Devuelve .T. si el primer carácter de <exp> es una letra mayúscula.

INLIST()

Determina si una cadena esta o no en una lista del mismo tipo

OCCURS()

Retorna el numero de veces que una cadena esta dentro de otra.

PADL(<Exp>, <N> [, <C>])

PADR(<Exp>, <N> [, <C>])

PADC(<Exp>, <N> [, <C>])

Devuelve Exp (que puede ser cadena, numero o fecha) rellenándola por la izquierda (PADL), la derecha (PADR), o por ambos lados (PADC) hasta llegar a la longitud especificada por N.

C es usado para el relleno. Si se omite se usan espacios.

cadena='Titulo'

? PADL(cadena, 40, '=')

? PADR(cadena, 40, '=')

? PADC(cadena, 40, '=')

RAT()

Busca la ultima ocurrencia de una cadena en otra o en un campo MEMO

REPLICATE(<exp>,n)

Repite n veces la cadena

RIGHT(<Exp>, <n>)

Extrae los n caracteres más a la derecha de la cadena Exp.

Si N es mayor que la longitud de ExpC, se devolverán todos los caracteres de la expresión. Si N es negativa o 0, se devolverá la cadena nula.

? RIGHT('Redmond, WA', 2)

STR(<n> [, <n1> [, <n2>]])

Convierte el numero n en una cadena de caracteres. Por omisión será de 10 caracteres de ancho o si se especifica una longitud de n1 caracteres y con n2 el número de lugares decimales. Si se especifica una longitud mayor que el numero de dígitos la cadera tendrá espacios en blanco antes del numérica con espacios al principio si especifica una longitud mayor que el número de a la izquierda de la coma decimal.

SPACE(n)

Devuelve una cadena de n espacios

SUBSTR(<Exp>, <n1> [, <n2>])

Extrae de la cadena Exp comenzando en la posición especificada por n1 y continuando por tantos caracteres como especifiques en n2.

El primer carácter de ExpC es la posición 1. Si se omite N2, los caracteres se extraerán hasta llegar al final de la cadena.

micadena='abcdefghijklm'

? SUBSTR(micadena, 1, 5)

? SUBSTR(micadena, 6)

STR(<exp>,n1,n2)

Convierte a formato de cadena un numero <exp> con una longitud n1 y una cantidad de decimales n2

STUFF(<exp1>,n1,n2,<exp2>)

Reemplaza desde la posición n1 de <exp1>, n2 posiciones usando <exp2>

a=`José'

b=`Antonio'

? STUFF(a,2,5,b) −−> JAnton

STRTRAN()

Busca una cadena sobre otra, y reemplaza esta con una tercera expresión

TRANSFORM(<n>, <formato>)

Formatea el numero n usando el códigos de PICTURE y FUNCTION.

precio=15.89

? TRANS(Precio, '$$$$.99')

UPPER(<Exp>)

Convierte la cadena Exp en mayúsculas.

? UPPER('abcdefgh')

2. Numéricas

ABS(n)

Muestra el valor absoluto del numero n

BETWEEN(<Exp1>, <Exp2>, <Exp3>)

Determina si el valor de Exp1 (que puede ser caracteres, números o fechas) cae dentro de los valores de otras dos expresiones del mismo tipo.

Se devuelve verdadero (.T.) si Exp1 es mayor o igual que Exp2 y menor o igual que Exp3; en caso contrario se devuelve falso (.F.).

CLOSE DATABASES

USE facturas

SCAN FOR BETWEEN(códfac,1250,1260)

? 'Impfac de ' + ALLTRIM(STR(códfac)) + ' es ' + ;

ALLTRIM(STR(impfac))

ENDSCAN

CEILING(<N>)

Devuelve el entero más próximo que es mayor o igual que la expresión numérica especificada. Es decir redondea un los números positivos con parte fraccional hasta el próximo entero mayor y los número negativo con una parte fraccional hasta la parte entera del número.

núm1=10.1

núm2=−10.9

núm3=10

núm4=−10

? CEILING(núm1)

? CEILING(núm2)

? CEILING(núm3)

? CEILING(núm4)

EXP(n)

Muestra e a la potencia n

FLOOR(<expN>)

Devuelve el entero más próximo que sea menor o igual que la expresión numérica especificada. Es decir redondea un número positivo con parte decimal reduciéndolo a la parte entera del número y redondea un número negativo con parte fraccional reduciéndolo al siguiente entero inferior.

núm1=10.1

núm2=−10.9

núm3=10

núm4=−10

? FLOOR(núm1)

? FLOOR(núm2)

? FLOOR(núm3)

? FLOOR(núm4)

WAIT WIND STR(FLOOR(núm1+núm3)) NOWA

@ 5,0 SAY FLOOR (18.1)

INT(<n>)

Devuelve la parte entera de una expresión numérica.

? INT(12.5)

? INT(6.25 * 2)

? INT(−12.5)

número=−12.5

? INT(número)

LOG(n)

Devuelve el logaritmo natural (base e) de <n>. Solo admite valore mayores que cero.

LOG10(n)

Muestra el logaritmo común (base 10) de <n>.

MAX(<Expr1>, <Expr2> [, <Expr3> ...])

Evalúa un conjunto de expresiones y devuelve la expresión con el mayor valor ASCII o numérico o la fecha más reciente de una lista de expresiones alfanuméricas, numéricas o de fecha. Todas las expresiones deben ser del mismo tipo (caracteres, números o fechas).

? MAX(54, 39, 40)

? MAX(2^8, 10*12, PI())

? MAX({14−07−98},DATE())

MIN(<expr1>, <expr2> [, <expr3> ...])

Evalúa un conjunto de expresiones y devuelve la expresión con el menor valor ASCII o numérico o la fecha más antigua de una lista de expresiones alfanuméricas, numéricas o de fecha. Todas las expresiones deben ser del mismo tipo (caracteres, números o fechas).

? MIN(54, 39, 40)

? MIN(2^8, 10*12, PI())

? MIN({14−07−55},DATE())

? MIN('a','abc')

? MIN('a','XYZ')

MOD(<N1>, <N2>)

Divide una expresión numérica por otra expresión numérica y devuelve el resto.

? MOD(36, 10)

? MOD(4*9, 90/9)

? MOD(25.250, 5.0)

ROUND(<Exp>, <N>)

Devuelve una expresión numérica redondeada a un número especificado de lugares decimales. SET DECIMALS se ignora.

Si N es negativo, se devuelve un número entero que contiene N ceros a la izquierda del separador decimal. Por ejemplo, si N es −2, los dígitos de las decenas y centenas serán cero.

SET DECIMALS TO 4

SET FIXED ON

? ROUND(1234.1962, 3)

? ROUND(1234.1962, 2)

? ROUND(1234.1962, 0)

? ROUND(1234.1962, −1)

? ROUND(1234.1962, −2)

? ROUND(1234.1962, −3)

SET FIXED OFF

SET DECIMALS TO 2

SQRT(n)

Raíz cuadrada del numero n

VAL(n)

Convierte una cadena en numero

SIN(n) −−> Seno del angulo n en radianes

COS(n) −−> Coseno del angulo n en radianes

TAN(n) −−> Tangente del ángulo n en radianes

ASIN (n) −−> Calcula el ángulo en radianes del arco cuyo seno es n

ACOS(n) −−> Calcula el ángulo en radianes del arco cuyo coseno es n

ATAN(n) −−> Calcula el ángulo en radianes del arco cuyo tangente es n

ATN2() −−>

DTOR(n) −−> Convierte grados a radianes

RTOD(n) −−> Convierte radianes a grados

PI() −−> 3.14159

SIGN(n) −−> Devuelve −1 si n es negativo, 1 si es positivo o 0

FV()

PV()

PAYMENT()

3. Fechas

DATE() −−> Muestra la fecha del sistema

TIME() −−> Muestra la hora del sistema

SECONDS() −−> Muestra los segundos transcurridos desde medianoche

DAY(Fecha) −−> Muestra el numero del ida de la fecha

DOW(fecha) −−> Muestra el ida de la semana (1=domingo)

CDOW(Fecha) −−> Muestra el día de la semana en palabras

MONTH(Fecha) −−> Muestra el numero del mes de la fecha

CMONTH(Fecha) −−> Muestra el mes en palabras

YEAR(Fecha) −−> Muestra el año de la fecha

DMY(Fecha) −−> Convierte fecha en cadena formato dd mes aa

DTOC(Fecha) −−> Convierte fecha en cadena formato dd−mm−aa

DTOS(Fecha) −−> Convierte fecha en cadena formato aaaammdd

CTOD(Exp) −−> Convierte cadena en fecha

GOMONTH() −−> Retorna el dia que es unos meses antes o después de cierta fecha.

4. Otros

TYPE(var)

Muestra el tipo de dato de Var (numérico, carácter, lógico, fecha, memo, indefinido)

DISKSPACE(<Exp>)

Devuelve el espacio libre en bytes, que queda en la unidad especificada.

FILE(<archivo>)

Devuelve .T. o .F. dependiendo de que exista o el archivo indicado.

GETENV(<exp>)

Captura el valor actual de una variable DOS.

ISCOLOR()

Devuelve .T. si nuestra PC tiene monitor a color.

PCOL()

Devuelve el numero de la columna en que esta detenido el cabezal de la impresora.

PROW()

Devuelve el numero de la fila en que esta detenido el cabezal de la impresora.

ROW()

Devuelve la fila actual en que se encuentra el cursor actualmente

COL()

Devuelve el numero de la columna donde se encuentra el cursor.

@ 5,10 SAY `A'

? COL() −−> 10

RAND([<n>])

Devuelve un número aleatorio entre 0 y 1.

Por omisión se usa en valor semilla 100001, pero n permite especificar el valor semilla a usar con la función.

Si especificas un valor positivo para n en la primera función RAND() que emita y ninguno en las subsecuentes, devuelve siempre la misma secuencia de números aleatorios.

Si n es negativo en la primera función RAND() que emita, se creará un valor semilla a partir del reloj del sistema que es una serie más aleatoria.

Ejemplos:

1. Generar el número aleatorio semilla (hacer esto únicamente una vez) y muestra números aleatorios distribuidos uniformemente entre 0 y 1.

= RAND(−1)

FOR i=1 TO 100

? RAND()

```
ENDFOR
```

2. Genera números reales distribuidos uniformemente entre a y b

```
FOR i=1 TO 100

? (b−a)*RAND()+a

ENDFOR
```

3. Genera números entero distribuido uniformemente entre a y b

```
FOR i=1 TO 100

? INT((b−a+1)*RAND()+a)

ENDFOR
```

4. Función que devuelve un valor que simula el lanzamiento de un par de dados. Requiere la función `aleaent'. Esto no es lo mismo que 'aleaent(12)' que tiene una distribución de probabilidad bastante distinta.

```
FUNC dados

RETURN aleaent(1,6)+aleaent(1,6)
```

5. Devuelve un número aleatorio de n dígitos. Requiere la función `aleaent'

```
FUNC dígitos

PARA n

y = aleaent(0, 10^n −1) + 10^n

RETURN SUBS(STR(y,n+1),2)
```

6. Generar valores distribuidos exponencialmente de media 'a'

```
a=10

FOR i=1 TO 100

? −a*LOG(RAND())

ENDFOR
```

7. Función para devolver un valor distribuido normalmente con media 'm' y desviación estándar 'std'. Requiere la subrutina 'x_p'. Su error es < (4.5E−4 * std)

```
FUNC ndist

PARA m, std
```

```
x = x_p(RAND())

RETURN x * std + m

FUNC x_p

PARA p

IF p < 0 .OR. p > 1

RETURN 0

ENDIF

IF p < .5

adjust = .T.

ELSE

p = 1 − p

adjust = .F.

ENDIF

t = SQRT(LOG(1/(p*p)))

x = t −((t * 0.010328 + 0.802853) ;

* t + 2.515517) ;

/ (((t * 0.001308 + 0.189269) ;

* t + 1.432788) * t + 1)

RETURN IIF(adjust, −x, x)
```

8. Función que devuelve la zona bajo la curva normal con media 'm' y desviación estándar 'Std' desde −infinito a x. Su error es < (7.5E−8 * Std)

```
FUNC normal

PARA x, m, std

RETURN p((x−m)/std)

FUNC p

PARA x

IF x < 0
```

```
ajuste = .T.

x = −x

ELSE

ajuste = .F.

ENDIF

t = 1 / (1 + 0.2316419 * x)

poli = ((((1.330274429 ;

* t − 1.821255978) ;

* t + 1.781477937) ;

* t − 0.356563782) ;

* t + 0.319381530) * t

res = 1 − 0.398942280401432678 ;

* EXP(−x*x/2) * poli

RETURN IIF(ajuste, 1−res, res)
```

SYS(n)

Devuelve una cadena de caracteres con la información de la función correspondiente a n.

n	
0	Nombre y numero del PC en la red LAN
1	Fecha del sistema en juliano
2	Segundos transcurridos desde medianoche
3	Función aleatoria para producir nombres
5	Disco por emisión
12	Memoria RAM libre

TYPE()

Devuelve el tipo de expresión: Cadena Numérico, Date, Lógico, Memo o una U si es indeterminado.

```
a=`Hola como estas'

? TYPE(`a')
```

UPDATE()

Devuelve .T. cuando el ultimo READ ha cambiado de datos en sus GETs asociados.

Capitulo 9

Ejemplo 1: Crear una matriz de 60 elementos que contenga los resultados de las tablas de multiplicar del 1 al 5.

```
CLEAR

DIME tabla(12,5)

FOR f=1 TO 12

FOR c=1 TO 5

tabla(f,c)=f*c

ENDFOR

ENDFOR
```

O también:

```
CLEAR

DIME tabla(12,5)

f=1

c=1

DO WHILE f<=12

DO WHILE c<=5

tabla(f,c)=f*c

c=c+1

ENDDO

f=f+1

ENDDO
```

Ejemplo 2: Mostrar en pantalla el resultado de la función: a2+3a−4 para valores de la variable a entre 5 y 12

```
CLEAR

FOR a=5 TO 12

? a^2+3*a−4

ENDFOR
```

También puedes hacer:

```
a=5

DO WHILE a<=12

? a^2+3*a−4

a=a+1

ENDDO
```

Ejemplo 3: Mostrar en bloques de solo una pantalla a la vez, los resultados de la función: x2+x para valores de x del 1 al 100.

```
CLEAR

y=1

FOR x=1 TO 100

? x^(2+x)

IF y>20

WAIT

CLEAR

y=1

ELSE

y=y+1

ENDIF

ENDFOR
```

Ejemplo 4: Mostrar los cuadrados de los 10 primeros números.

```
xNum=1

DO WHILE xNum<=10

? `El cuadrado de ',xNum,' es ',xNum^2

xNum=xNum+1

ENDDO
```

O también:

```
xNum=1

DO WHILE .T.

? `El cuadrado de ',xNum,' es ',xNum^2

IF xNum=10

EXIT

ENDIF

xNum=xNum+1

ENDDO
```

Ejemplo 5: Mostrar los cuadrados de los 10 primeros números impares.

```
FOR xNum=1 TO 19 STEP 2

? `El cuadrado de ',xNum,' es ',xNum^2

ENDDO
```

O también:

```
xNum=1

DO WHILE xNum<20

? `El cuadrado de ',xNum PICT `99',' es ',xNum^2

xNum=xNum+2

ENDDO
```

Ejemplo 6: Mostrar el resultado de la función 2a2+3 para los números impares y de la función 3a2+2 para los pares, para valores de a entre 1 y 10.

```
FOR a=1 TO 10 STEP 2

? `Para ',a PICT `99',' la función es',2*a^2+3

? `Para ',a+1 PICT `99',' la función es',3*a^2+2

ENDFOR
```

Ejemplo 7: Mostrar los números cuyo resultado de la función x2−2x este entre 50 y 100

```
x=1

DO WHILE x<100
```

```
IF x^2−2*x>=50 .AND. x^2−2*x<=100

? `El numero ',x,' cumple la condición'

ENDIF

ENDDO
```

Ejemplo 8: Buscar si existe algún valor de x entero entre −100 y 100 que cumpla la siguiente ecuación x2−2x+10=0

```
FOR x=−100 TO 100

IF x^2−2*x+10=0

? `El numero ',x,' cumple la ecuación'

ENDIF

ENDFOR
```

Capitulo 10

Problemas propuestos 1º practica:

- Mostrar en bloques de una sola pantalla los resultados de la función 3X2+7 para valores de x entre 0 y 100
- Determinar y mostrar los 20 primeros numero de la serie: 1/2, 1/5, 1/8... (como fracción) y luego la suma total de ellos.
- Determinar los valores de x para los que la función X2+4x−8 tiene valores entre 50 y 100.
- Crear un programa que dada una edad valida (mayor a cero) y sexo de una persona, responda con alguna de las siguientes palabras: BEBE, NIÑO o NIÑA, JOVEN, ADULTO o ADULTA, TIO o TIA, ABUELO o ABUELA, según el caso: Inventa tus propios rangos de edad para cada palabra, pero toma en cuenta el sexo.
- Crear un programa que dada una nota valida (entre 0 y 20) de cierto alumno, responda con alguna de las siguientes palabras: EXCELENTE, MUY BUENO, BUENO, REGULAR, MALO, MUY MALO o PESIMO, según el caso. Inventa tus propios rangos de notas para cada palabra. La nota puede ser con decimales.
- Crear un programa que luego de pedirte tu fecha de nacimiento determine el signo del zodiaco (Aries, Tauro, Géminis, Cáncer, Leo, Virgo, Libra, Escorpio, Sagitario, Capricornio, Acuario o Piscis) a que perteneces y el grupo (Tierra, Fuego, Aire o Agua) al que pertenece dicho signo. Ej: Aries 21−03 al 22−04
- Crear un programa que devuelva el elemento mayor / menor de una matriz de n1 x n2 elementos.
- Crear un programa que almacene una matriz en un archivo y luego la pueda recuperar a voluntad nuevamente.
- Crear una matriz de n x n la cual deberá tener el numero 1 en cada una de sus diagonales y cero en los demás elementos y luego mostrar dicha matriz en pantalla, haciendo que los números 0 sean de color negro y los números 1 de color rojo.
- Crear un programa que dado un sueldo base desde la ventana de comando, determine el aumento en %, aumento neto y nuevo sueldo según la siguiente tabla:

Hasta 200	25%
entre 201 y 500	20%
entre 501 y 800	15%

entre 801 y 1000	10%
mas de 1000	5%

- Crear un programa que determine los factores de cada uno de los números entre 10 y 20 inclusive.
- Crear un programa que halle los números primos que hay entre los números 10 y 25 inclusive.
- Crear un programa que dado un numero, examine si se trata de un numero primo, de ser cierto expresarlo, de lo contrario mostrar en orden descendente todos sus factores.
- Crear un programa que halle los primeros 30 números capicúa. Nota: Un numero capicúa es el que se puede leer igual de derecha a izquierda o viceversa. Ejemplos: 11, 44, 101, 838, 5445, 12321, 981189 etc...
- Hallar todos los factores primos de cada uno de los numero entre 12 y 25.
- Crear un programa que basado en dos variables que se creen desde la ventana de comandos, una con el monto a cobrar y la otra con el monto de dinero entregado al cajero, determine el vuelto en soles y el numero de monedas y billetes que deberá entregar. Considerar billetes de 200, 100, 50, 20 y 10, monedas de 5, 2, 1, 0.50, 0.20 y 0.10

Problemas propuestos 2º practica:

- Crear un programa que pida ingresar la producción correspondiente a cada uno de los 12 meses del año y luego determine el mes de máxima y mínima producción, indicando el nombre del mes y su producción. Hallar también la producción promedio del año.
- Crear un programa que luego de solicitar se ingrese la venta de cada mes del año en números relativos entre 0 y 100, los muestre ordenadamente de forma ascendente (o descendente).
- Crear un programa que permita elegir una forma entre: triángulo, rectángulo y circulo, luego pregunte si se quiere el área y/o perímetro y según dichas respuestas pregunte por los datos pertinentes (base y altura, lados o radio) y de las respuestas solicitadas.
- Crear un programa que pida al usuario ingrese desde la pantalla un numero inicial y otro final (que deberá ser mayor al inicial) y luego mostrar en pantalla los números pares positivos entre dichos números, una pantalla a la vez.
- Crear un programa que muestre una pantalla un menú dando a elegir entre hallar el área de un triángulo, rectángulo o circulo y luego pida los datos pertinentes a dicho calculo y finalmente muestre el resultado.
- Crear un programa que después tirar un dado se ingrese el valor resultante en un programa, por 10 veces consecutivas y luego informe sobre:

    - La ocurrencia de cada numero.
    - El o los numero que se repitió mas veces.
    - El o los numero que repitió menos veces.
    - Grafique un histograma de ocurrencias (horizontal)

- Crear dos matrices 3x4 y luego presentarlas en pantalla una al lado de la otra, para que rellenemos sus elementos con un numero entre 0 y 99 y luego al lado de ambas presentar la suma.
- Crear un programa que solicite ingresar un numero entero de 4 dígitos y luego lo muestre de formas invertida, para luego pedir un nuevo numero y así hasta que pulsas tecla <Esc> Ej: 1234 −> 4321
- Crear un programa que permita ingresar 10 números enteros entre 0 y 100 inclusive y luego determine:

    - Mayor y menor valor.
    - Sumatoria y promedio.
    - Ordenar dichos números en forma ascendente.

- Crear un programa que pida ingresar el numero total de términos de la sumartoria de:

    - Pares positivos.
    - Impares positivos.
    - Términos de la serie 1/2, 1/4, 1/6...

    - Múltiplos de 7

- Crear un programa que pida ingresar el valor inicial y final (validar que inicial sea menor que final) y determinar los ternimos que estando entre dichos limites sea además:

    - Los que no son múltiplos de 4
    - Primos (1, 2, 3, 5, 7, 11...)
    - Los números inmediato siguiente a un primo (si no es primo)
    - Capricúas (44, 101, 4554...)

- Crear un programa que solicite se ingrese los valores a cada uno de los 9 elementos de dos matrices 3 x 3, luego las sume (reste, multiplique o divida) y muestre la matriz resultante en pantalla.
- Crear un programa que después de preguntar el tamaño de una matriz (a x b) y la operación que se quiera hacer (suma o resta), pida se ingrese cada elemento de ambas matrices y luego muestre el resultado (de sumar o restar) en pantalla.
- Crear un programa que convierta metros a pies, kilos a libras y litros en galones y viceversa.
- Crear un programa que después de pedir se ingrese un numero de tres dígitos sin decimales, calcule y muestre dicho numero en binario, octal y hexadecimal.
- Crear un programa que permita ingresar los últimos dos dígitos del año y su producción total en Kg, de un cierto numero de años (el cliente ingresara la cantidad de datos que desee) y luego se ordene en orden ascendente (o descendente) dicha producción. Ojo no impongas ningún limite al numero de datos deja eso completamente libre.
- Crear un programa que realice una animación de una cadena de caracteres en la pantalla (la desplace hacia un lado por ejemplo). Toma en cuenta que la pantalla es 25 x 80

Bibliografía:

Programación en base a eventos

− Cesar A. Bustamante Gutierrez − UNI

FoxPRO 2.6 para DOS y Windows a su alcance

− José Javier García Badell − McGraw Hill

Programación básica con FoxPRO

− Ramón M. Chordá − Rama

Al ida en una hora en FoxPRO 2.6

− José Carlos Corrales − Anaya Multimedia

Guía practica FoxPRO 2.6

− Alejandro Domínguez − Anaya Multimedia

Diseño de programas: 200 algoritmos y un proyectos

− Julio Vasquez Paragulla − Editorial San Marcos

Microsoft FoxPRO for DOS and Windows, lenguage reference

− Microsoft Press

Capitulo 1

Introducción a tablas DBF:

Base de datos es el conjunto de información, la cual ha sido organizada y preparada para servir a un propósito especifico. Por Sistema de Gestión de Base de Datos (DBMS) entendemos a la organización sistemática de grandes cantidades de información. La misma información esta organizada de diversas formas, de modo que nos faciliten una búsqueda, extracción, listado, reporte o gráfico.

RDBMS: Relacional Data Base Management System. Software que administra o maneja datos relacionales. Ejemplo: FoxPRO, Access, SQL, etc

BASE DE DATOS: Colección de datos relacionales sobre el mismo tema o propósito, donde se puede manejar en conjunto y de forma fácil múltiples objetos, con los que podemos guardar, mostrar, organizar y analizar datos guardados en forma de TABLAS.

TABLA: Objeto que contiene una colección de datos sobre un tema particular, almacenados en forma de filas y columnas. Por ejemplo lista de empleados, clientes, proveedores, productos, insumos, ventas, compras, etc.

QUERY: Objeto que presenta una interrogante (consulta) o define un juego de características a la BASE DE DATOS, que puede involucrar los datos contenidos en varias TABLAS y cuya respuesta es un DYNASET, una colección de REGISTROS dinámicos que responden o cumplen las condiciones o características impuestas en el QUERY o consulta.

FORMA: Objeto que arregla los datos de una o mas tablas, en forma conveniente para ver muchos CAMPOS a la vez y obtener la mejor presentación de sus datos en papel o pantalla, presentando subtotales y totales.

CONTROL: Objeto que muestra datos de un campo, variable o expresión con ciertas características especiales, propias del objeto.

OBJETO: Algo que se puede seleccionar y manipular como una unidad. Esta prediseñado y solo usaremos sus propiedades, métodos y eventos.

Hay dos tipos básicos de bases de datos:

1. Jerarquizadas:

Organizan su contenido basado en un modelo de jerarquías que establece relaciones uno a uno y en forma de árbol. Si es necesario hacer relaciones múltiples, será necesario la repetición de datos.

2. Relaciónales:

Organiza una base de datos como un conjunto de tablas rectangulares organizadas en filas (llamadas registros) y columnas (llamadas campos). A cada registro se le asigna automáticamente un numero secuencial, es decir un numero de registro y a cada campo un nombre. Cada una de estas tablas se llama relación y constituye un apoyo a la base de datos.

Campo: Unidad básica de información, respecto a determinado registro. Ejemplo: nombre, edad, sueldo, nota etc. (columnas). Pueden tener nombres de hasta 10 caracteres, iniciándose con una letra y no pueden contener espacios.

Pueden ser de varios tipos:

- **Carácter**: Son los que pueden contener hasta 255 caracteres alfanuméricos y normalmente son campos con los que no se hace operaciones. ejemplos: nombre, dirección, teléfono, # de LE etc.
- **Numérico**: Usado para introducir números que podrán ser operados posteriormente. El ancho máximo es de 19 dígitos incluido el separador decimal, los dígitos decimales y el signo (+ o −).
- **Fecha**: Para registrar fechas. Siempre de longitud 8.
- **Lógico**: Para registrar un valor lógico como .T. o .F., Si o No etc. Siempre su longitud es de 1.
- **Memo**: Usado para cuando se requiere ingresar mas de 255 caracteres en un campo o para cuando la longitud de campo debe ser variable. Siempre de una longitud de 10 caracteres, pero en realidad los datos de este campo se almacenan en un archivo aparte con la extensión FPT.

Registro: Es un conjunto de información básica dividida en campos, los cuales pertenecen a un elemento o evento. Se almacenan en el orden en que hayan sido introducidos, aunque después podremos ordenarlos como necesitemos. Ejemplo: un alumno, un curso, un empleado, una transacción etc. (filas)

Un buen ejemplo de tabla es un directorio telefónico, los datos se encuentran clasificados por orden alfabético, por numero de teléfono, dirección etc. notar que la misma información puede presentarse organizada bajo diferentes formas, de modo que nos facilite una búsqueda, una extracción o la emisión de un listado, reporte o un gráfico.

Términos básicos:

Entidad: Persona, lugar, objeto o evento de interés acerca del cual se capturan datos. Ejemplos: Alumnos, cursos, matricula...

Tabla maestro: Contiene datos que describen el estado actual de cierta entidad especifica. Siempre debe mantenerse actualizado. Ejemplo: maestro de alumnos, contiene nombre dirección, teléfono y promedio ponderado.

Tabla de transacciones: Es un archivo temporal, acumula datos de cierta entidad respecto a un periodo de tiempo. Ejemplo: matricula 99−1, contendrá los cursos en los que cada alumno esta matriculado para cierto ciclo.

Tabla cabecera: Es un archivo usado para acumular los datos generales pertinentes a un documento. Factura, boleta, guía etc

Tabla detalle: Es un archivo que complementa al anterior, donde está la descripción de los ítem de dicho documento.

Normalización de base de datos:

A través del proceso de normalización se busca simplificar la organización y evitando datos redundantes.

Formas normales:

En primer lugar se deben recolectar todos los datos de la entidad en análisis.

- Se busca agrupar datos, de modo que cada grupo represente a una entidad particular, cada grupo representa una tabla. No se debe incluir campos que no guarden relación con el criterio de grupo. Ej nombre y dirección del alumno en la tabla de matricula. Tampoco se debe incluir campos en los cuales todos los registros tengan valores iguales.

- Asignar a cada tabla una llave única que identifique cada registro. Se procede con la revisión de cada campo, si en alguno de ellos hubiera datos que no dependan de la llave primaria entonces se sugiere retirarlos.
- Se sugiere eliminar aquellos campos que se puedan conseguir a partir de otros campos existentes. Ejemplos: Total = Unidades x PU o IGV = 18% de Total., la edad ya que es mejor poner fecha de nacimiento.

En la medida de lo posible, pero sin atentar contra la claridad del sistema, se debe tratar de ahorrar espacios en los campos, esto se podrá lograr mediante la codificación de los datos o abreviar los nombres de campos a por menos el ancho del campo. Ej no usar el nombre MATRICULADO para un campo lógico (solo un carácter)

Capitulo 2

Notacion usada en este y otros libros:

Esta misma simbologia es usada por la ayuda en linea de FoxPRO.

Simbología:

La simbología usada en esta separaba es la adoptada en la mayoría de los textos de FoxPRO.

<> Indica que el ítem encerrado deberá ser suministrado por el usuario al momento de usar el comando o función. Los símbolos <> no deberán ser incluidos.

<Exp> Puede consistir en un campo, una variable o una combinación valida de campos variables y funciones. Puede ser carácter, numérico, fecha o lógico.

<Var> Se trata de una variable de memoria.

<Condición>: Expresión que deberá responder con verdadero o falso al ser analizado en cada registro.

[ ] Indican que el ítem encerrado es opcional. Los corchetes no son parte del comando.

| indica que se debe escoger una entre dos o mas opciones.

Alcance: Es una parte opcional de muchos comandos, que indica el grupo de registros sobre el cual se aplicará el comando. El parámetros FOR y WHILE establecen por defecto el alcance ALL.

- RECORD n: Un solo registro, el numero n
- NEXT n: Los siguientes n registros comenzando por el activo.
- REST: Desde el registro activo hasta el fin de archivo.
- ALL: Todos los registros de la tabla actual.

Capitulo 3

Comandos mas usados:

Este capitulo ofrece a los nuevos usuarios una descripción general de los comandos y las funciones más potentes y más usados de FoxPRO agrupados según las tareas básicas.

Manejo de tablas (DBF)

1. Crear tablas

Determinar su nombre y definir su estructura indicando el nombre, tipo y longitud de cada campo.

CREA <dbf / ?> [FROM <archivo>]

2. Tablas

Abrir una tabla y activar un índice en determinada área.

USE [<dbf>] [ALIAS <alias>] [IN <area>] [ORDE <tag>] [ASCENDING|DESCENDING] [AGAIN] [EXCLUSIVE/SHARED] [NOUPDATE]

CLOSE DATABASE = Cierra todas las tablas activas y selecciona como activa el area numero 1.

Algunas funciones útiles:

DBF([<area>|<alias>]) da el nombre de la tabla en uso.

ALIAS([<area>]) da el nombre del alias de una tabla.

SELECT(0/1) Con 0 retorna el # del area activa y con 1 retorna el # del area mas alta no usada aun.

USED() Da .T. si el area esta ocupada

FCOUNT([<area>|<alias>]) da número de campos.

FIELD(<#>[<area>|<alias>]) da el nombre del campo.

FSIZE(#) Da el tamaño de los campos que corresponden a un registro

HEADER(#) La longitud de la cabecera de una tabla

RECCONT() Numero total de registros de la tabla

RECNO() Numero de registro actual

3. Estructura

MODI STRU = modifica la estructura de la tabla activa

COPY STRU TO <archivo> FIELD <campos> [EXTENDED]

LIST STRU [NOCONSOLE] [TO PRIN | TO FILE <txt>]

DISP STRU [NOCONSOLE] [IN <area>] [TO PRIN | TO FILE <txt>]

4. Buscar tablas

DIR [ON <d:>][LIKE[<ruta>][<skel>]][TO PRINTER | TO FILE <txt>]

5. Areas de trabajo

FoxPRO puede manejar 225 áreas (solo 25 en versión estándar), en cada una de las cuales puede abrir una

tabla y mantener separadamente la posición del puntero.

De forma predeterminada, el área de trabajo número 1 (A) está activa cuando se inicia FoxPRO. Puede especificar un área de trabajo distinta con: SELE <#>. Si <#> es 0, se seleccionará el área de trabajo libre cuyo número sea inferior.

SELECT(0) Da el # del área de trabajo activa

SELECT(1) Da el # del área de trabajo libre mas alta

USED([<#>]) da .T. si área está ocupada o .F. si esta libre.

6. Formas de visualización

SET STATUS OFF/on Muestra o no la barra de estado

SET HEADER ON/off Determina si se usara el encabezado o no en LIST, DISP y CALC

SET DATE TO [DMY / MDY / YMD] −−> Especifica el formato de las fechas

SET CENTURY OFF/on Determina si usa 2 (default) o 4 dígitos para representar los años.

SET DECIMALS TO <#> Especifica el numero de decimales para las expresiones numéricas (0 a 18)

SET FIXED

SET MESSAGE TO [<#> / LEFT / CENTER / RIGHT] [WINDOW <nombre>]

SET STICKY ON/off Especifica como el Mouse mostrara los menús de FoxPRO.

SET TALK ON/off Especifica si se mostrara o no los comandos ejecutados desde la ventana de comandos.

Capitulo 4

Manejo de registros:

1. Añadir registros

APPE [BLANK][FROM<dbf>[FIEL<campos>][FOR<cond>]]

[DELIMITED [WITH TAB | WITH <delimit> | WITH BLANK]

[TYPE WK1|WK3|WKS|XLS||]

SET CARRY ON/OFF Si esta en ON copia los datos del registro anterior al nuevo registro añadido.

2. Modificar registros

3. Borrar registros

3.1 Marcar

DELETE [ALL / NEXT<#> / RECORD <#> / REST] [FOR<cond>] [WHILE<cond>] [NOOPTIMIZE]

3.2 Desmarcar

RECALL [ALL / NEXT<#> / RECORD <#> / REST] [FOR<cond>] [WHILE<cond>] [NOOPTIMIZE]

3.3 Empaquetar

PACK [MEMO] [DBF]

4. Mostrar o no los registros marcados

Mientras que con SET DELETE OFF se podrá ver los registros marcados para borrar con un rombo. Con SET DELETE ON ya no se mostraran dichos registros.

5. Borrar todos los registros

ZAP

6. Algunas funciones útiles

RECCONT() Numero total de registros de la tabla

RECNO() Numero de registro actual

HEADER(#) La longitud de la cabecera de una tabla

RECSIZE()

FSIZE(#) Da el tamaño de los campos que corresponden a un registro

DELETED() Devuelve verdadero si el registro actual está marcado para su eliminación.

Capitulo 5

Organización de registros

Los registros de una tabla se pueden ordenar alfabética, cronológica o numéricamente y existen dos tipos de ordenamiento:

Físico: Que consiste en generar una nueva tabla a partir de la tabla activa, con la misma estructura, cuyos registros se encuentran físicamente ordenados respecto a un campo o conjunto de campos llamada llave o clave. Este método solo es recomendable para tablas estables, es decir las que rara vez se modifican sus registros, ya que una modificación obligaría a generar un nuevo archivo.

SORT ON <campo> TO <tabla> [FOR <cond>] [WHILE<cond>] [ASCE | DESC] [FIELDS [<campos> | LIKE<mascara> | EXCEPT<mascara>]][<alcance>]

Lógicos: Que consiste en la generación de un archivo índice, que contiene el orden bajo el cual debe de verse los registros de la tabla. El campo o campos por los que se hace dicho ordenamiento se llama llave o clave. Este archivo índice es pequeño comparado al tamaño de la tabla que contiene los registros y mientras esta activo la modificación o añadido de registros la actualiza automáticamente.

El FoxPRO maneja básicamente dos tipos de índices:

Simples (IDX): Guardan una sola clave, hay que abrirlos manualmente cada vez que se usa la tabla, para que actualice las modificaciones en los registros de la tabla. Usados en FoxBASE.

Compuestos (CDX): Pueden almacenar hasta 25 claves, cada una con una etiqueta (TAG), se abre automáticamente al usar la tabla a la que están asociados, por lo que no hay peligro de desactualizarlo al modificar la tabla y miden aproximadamente 1/6 del tamaño en KB de un archivo IDX. Usados en FoxPRO.

1. Crear

INDEX ON <campo> TO <idx> | TAG <tag> [FOR <exp>] [ASCENDING | DESCENDING] [UNIQUE] [ADDITIVE]

2. Abrir IDX

SET INDEX TO <idx1>,<idx2>,<idx3>

3. Activar o cambiar

SET ORDE TO <tag>|<idx> [IN <area>|<alias>] [ASCE | DESC]

Si usas SET ORDE TO se desactivan todos los índices y quedara la tabla en el orden original, por numero de registro.

4. Reconstruir índices

Reconstruye todos los índices activos.

REINDEX

5. Eliminar índices

DELETE TAG <etiqueta>

6. Poner un filtro

SET FILTER TO <exp>

7. Algunas funciones útiles

CDX([<area>|<alias>]) −−> da el nombre del archivo índice asociado.

ORDER() −−> Da el nombre del IDX o etiqueta (TAG) activa.

TAG([<#>][,<area>|<alias>]) −−> Da el nombre de la etiqueta.

KEY([<#>|<tag>][,<area>|<alias>]) −−> da la expresión índice clave.

FILTER() −−> Da la condición de filtro

8. Comandos que afectan los resultados

SET COLLATE TO `spanish'

Afecta la forma como se ordenan los registros (ejemplo: a, b, c, ch, d...) cuando activamos un índice y la forma de compara cadenas, según el idioma.

? `A' = `a'	.T.	`spanish' o `general'
? `A' = `a'	.F.	`machine'

SET EXACT OFF/on

Afecta la forma de comparar cadenas de diferente longitud.

Si esta en OFF (default) las cadenas se consideran iguales si coinciden carácter a carácter hasta el final de la que esta a la derecha.

Si esta en ON la cadena mas corta se rellena con espacios hasta igualar la longitud de la mas larga.

Ejemplo:

? `Juancho' = `Juan'

Si esta en OFF .T.

Si esta en ON .F.

Capitulo 6

Mostrar, imprimir y exportar datos:

- **Mostrar en ua ventaa**

El comando BROW muestra los datos en una ventana, donde se pueden modificar los datos existentes. Para moverse usar flechas de navegacion, <PgUp>, <PgDn>, <Tab>, <Shift><Tab>. Usar <Ctrl><W> o <CTRL><End> para guardar y salir. Solamente la tecla <Esc> para salir sin salvar. <Ctrl><G> añadir registro y <Ctrl><R> marcar/desmarcar para borrar.

BROW [LAST] [FIELDS <campo1,compo2,...>]

[FOR <cond>] [KEY <exp1>,<exp2>]

[VALID [:F] <exp> [ERROR <mensaje>]] [WHEN <cond>]

[FREEZE <campo>] [LOCK <exp>] [TITLE <titulo>][WIDTH <ancho>] [TIMEOUT <#>] [IN WINDOW <ventana>][COLOR <par1>,<par2>...,<par8>]

[NOAPPEND] [NODELETE] [NOMODIFY] [NOLGRID] [NORGRID] [NOLINK] [NOMENU] [NOWAIT] [NOCLEAR] [NOMENU]

Si se usa la cláusula FIELDS, es posible indicar las características de cada campo de la siguiente manera:

Comando	Descripción
:#	Ancho del campo en caracteres.

:R	Solo lectura, no será posible modificar. (Read only)
:P=<Exp>	Picture
:H=<Exp>	Titulo del campo
:B=m,M	Rango en el que debe estar (Between)
:V=<Cond>	Si la condición es T el valor es aceptado, sino será rechazado. Solo verifica si hay algún cambio. (Post−Condición)
:E=<Exp>	Mensaje de error. (usado con :V)
:F	Fuerza la validación al salir del campo (aunque no cambie) (Forced valid)
:W=<Cond>	Si la condición es falsa no se podrá mover el cursos a dicho campo (Pre−condición)

Campos calculados: Es un campo creado con una expresión, de solo lectura, usando el formato: <nombre> = <exp>

FOR <cond>: Permite filtrar registros, de manera que solo quedaran visibles los que cumplan la condición.

KEY <exp1>,<exp2>: Permite filtrar registros entre un rango basándose en la llave índice activa.

VALID [:F] <cond> [ERROR <mensaje>]: Evalúa <cond> si se efectúo un cambio o si usaste :F, si la condición es verdadera o 1 te deja continuar, si es falsa o 0 envía el mensaje de ERROR. Es una post−condición.

WHEN <cond>: Evalúa <cond> si es verdadera o 1 te permite posicionarte en el registro, si es falsa o 0 se salta al siguiente registro. Es una pre−condicion.

TITLE <exp>: Permite poner un titulo a la ventana

WIDTH <ancho>: Permite limitar el ancho máximo de cada uno de los campos de forma general.

TIMEOUT <#>: Determina cuantos segundos se esperara a que el usuario ingrese algo. Transcurrido dicho tiempo la ventana del BROWSE se cierra automáticamente.

IN WINDOW <ventana>: El browse se abre dentro de la ventana previamente definida por el usuario.

COLOR <par1,par2...,par8>

# Par	**Elemento**
1	Resto de registros
2	Campo activo
3	Borde
4	Titulo si Browse esta activo
5	Titulo si Browse no esta activo
6	Texto seleccionado
7	Registro activo
8	Sombra

CHANGE y EDIT funcionan de manera muy parecida a BROWSE y usan las mismas cláusulas

2. List

LIST [OFF][<campo1>,<campo2>...][FOR<cond>]

[WHILE<cond>][TO PRIN | FILE <txt>]

[NOCONSOLE]

3. Display

Muestra registros o expresiones de una tabla.

DISP [ALL][OFF][<campo1>,<campo2>...][FOR<cond>]

[WHILE <cond>][TO PRINTER | FILE <txt>]

[NOCONSOLE]

DISP STATUS Muestra el estado del entorno FoxPro.

DISP STRUCTURE Muestra la estructura de una tabla.

4. Uno a uno

? | ?? <exp1>,<exp2>,<exp3>...[PICT <exp>][FUNC <exp>][AT <exp>]

??? <exp> directo a la impresora

5. Comandos adicionales

SET HEADING ON/OFF

SET FILTER TO <cond> = Usado para hacer visible SOLO a los registros que cumplan con la condición.

SET FILTER ON/OFF/LOCAL/GLOBAL

Capitulo 7

Mover puntero y buscar registros:

1. Mover

.1 Absoluto:

Se pone el numero del registro al que queremos ir o mediante el comando: GO <#> [IN <area> | <alias>]

.2 Relativo:

Se queremos movernos a través de los registros de forma relativa y según el índice activo:

GO TOP | BOTTOM [IN <area> | <alias>]

SKIP [<exp>] [IN <area> | <alias>]

RECNO(<area>|<alias>) numero de registro activo

RECCOUNT(<area>|<alias>) numero total de registros

2. Búsqueda Secuencial

Búsqueda secuencial es la que partiendo del primer registro de una tabla, comprueba uno a uno si cumplen la condición impuesta y se para en el primer registro que la cumpla.

LOCATE [ALL][NEXT <exp>] [FOR <cond>][WHILE <cond>]

Se usa CONTINUE para buscar el próximo registro. Esta búsqueda puede ser lenta en caso de muchos registros (mas de 1'000,000) si no se usa la tecnología RUSHMORE, basada en el uso de la condición FOR sobre un campo indexado con una expresión básica optimizable del tipo:

<exp índice> <operador> <constante>

donde:

<exp índice> debe coincidir exactamente con el índice activo y no debe contener alias.

<operador> debe ser uno de los siguientes: <, >, =, < =, > =, < >, #, !=

<constante> puede ser una variable de memoria o un campo.

3. Búsqueda en indexados

SEEK <var> −−> para buscar en base a una variable o si se usa una cadena deberá estar entre comillas.

FIND <expC> −−> para buscar en base a una cadena o valor. No es necesario el uso de comillas.

Nota 1: La expresión de búsqueda no es una condición, sino simplemente el valor a buscar en coincidencia con la clave índice en uso en ese momento. Es indispensable que este indexada la tabla para que funcione alguno de estos comandos.

Nota 2: Es posible consultar el éxito de una búsqueda con FOUND(<area>|<alias>), en caso no tener éxito una búsqueda el puntero queda al final de la tabla, lo compruebas con EOF(<area>|<alias>).

4. Algunas funciones útiles

BOF(<area>|<alias>) Inicio de archivo

EOF(<area>|<alias>) Fin de archivo

FOUND() Si la busqueda fue positiva

SEEK(<var>,<area>|<alias>) Función avanzada que hace la búsqueda al mismo tiempo que da un .T. si la búsqueda fue exitosa.

Capitulo 8

Operaciones con campos y registros:

1. Operadores Lógicos:

Para agrupar usa (), para negar usa (.NOT. o !), conjunción (.AND.) y disyunción (.OR.)

2. Operadores Aritméticos

Sumar (+), restar (−), multiplicar (*), dividir (/), potenciar (^ o **).

3. Operadores Relaciónales

Menor que (<), mayor que (>), igual (=), distinto (# o <>), menor o igual (<=), mayor o igual; (>=), exacto (==).

4. Calculo con campos

Hacer cálculos entre campos sobre un mismo registro.

REPLACE<campo> WITH <exp> [ALL] [FOR<cond>] [WHILE<cond>]

5. Calculo con registros

Hacer cálculos sobre un mismo campo en todos los registros.

CALCULATE <operador> [FOR<cond>][TO<var>]

Donde los operadores pueden ser: CNT(), SUM(campo), MAX(campo), MIN(campo), AVG(campo), STD(campo), VAR(campo), NPV(campo), etc donde campo debe ser de tipo numérico.

Antiguamente se usaba:

SUM <campo>[FOR<cond>][TO<var>]

AVERAGE <campo>[FOR<cond>][TO<var>]

COUNT [FOR<cond>][TO<var>]

Capitulo 9

Manipular múltiples tablas

Como consecuencia de la normalización es probable que para cierto caso tengamos un diseño de base de datos compuesto de varias tablas y es necesario conectarlas para lograr los resultados deseados. Esto se hace relacionando las tablas.

El proceso exige abrir una tabla principal (padre) la cual puede o no estar indexada en un área cualquiera y en otra área diferente una segunda tabla (hijo) la cual deberá estar indexada por el campo que determinara la relación. Luego desde el área donde esta el archivo padre, se emite el siguiente comando:

SET RELATION [TO <campo> INTO <area/alias>] [OFF]

Desde este momento obtendremos una conexión entre estas tablas, de manera que cualquier movimiento en la tabla padre se refleja automáticamente en la tabla hijo y para referirte a un campo de la tabla hijo solo debes anteponer el área o alias delante del nombre del campo de la tabla hijo.

Ej: LIST COD,B.NOMBRE,FECHA

Aquí se hace un listado mostrando el campo COD y FECHA que pertenecen a la tabla padre y NOMBRE que pertenece a la tabla hijo y esta ubicada en el área 2.

Se llama relación uno a uno, si un registro del padre corresponde a un registro en el hijo.

Se llama relación muchos a uno, si varios registros del padre corresponden al mismo registro hijo.

Se llama relación uno a muchos, si un registro en el padre corresponde a muchos registros en el hijo y entonces al posicionarse en cierto registro del padre, solo será mostrado el primer registro correspondiente en el hijo, de manera que no es posible ver los demás. para solucionar este problema hay que emitir el siguiente comando en el área del padre:

SET SKIP TO <area>

Funciones:

RELATION() Muestra la relación entre dos tablas

TARGET() Muestra el alias de la tabla hija

Capitulo 10

Funciones

Tablas y areas:

DBF(<area>) Muestra el nombre de la tabla activa

USED() Determina si tabla esta en área activa o no

ALIAS() Muestra el sobrenombre de la tabla

SELECT() Muestra el área de trabajo activa

Relaciones:

RELATION() Muestra la relación entre dos tablas

TARGET() Muestra el alias de la tabla hija

Indices:

CDX() Muestra el nombre del archivo de índice activo

TAG() Muestra la etiqueta del índice activo

KEY() Muestra la expresión de indexación activa

Campos:

FCOUNT(<area>) Muestra el numero de campos

FIELD(n,<area>) Muestra el nombre del campo n

Registros:

BOF(<area>) Devuelve verdadero si el puntero de registros se encuentra al principio de una tabla.

EOF(<area>) Devuelve verdadero si el puntero de registros se encuentra al final de una tabla.

RECNO(<area>) Numero de registro activo

RECCOUNT(<area>) Numero total de registros de la tabla

RECSIZE(<area>) Longitud en bytes del registro

FLUSH() Escribe el contenido del buffer en disco

DELETE(n) Responde .T. si el registro esta marcado para borrar

Otros:

FILE(archivo) Indica si existe o no un archivo

HEADER() −−> Muestra tamaño en bytes del encabezado de tabla

Capitulo 11

Ejemplos:

- USE maealu IN c ALIAS padre

Aquí abrimos la tabla MAEALU en el area 3 con el alias PADRE

- INDE ON cod TAG código

Indexamos la tabla en uso por el campo COD y asignamos a este orden el nombre CODIGO.

- SET ORDE TO código DESC

Activamos la llave código en forma descendente

- ? KEY()

Preguntamos por la llave de índice activa.

- BROW FIEL nom:5,cod:R,nota,edad

Aquí el campo NOM solo tendrá un ancho de 5 caracteres y el campo COD será de solo lectura, los campos NOTA y EDAD no tienen ningún condición especial.

- BROW FIEL cod,nota:B=0,20:E=`La nota debe ser entre 0 y 20'

Aquí obligamos a que la NOTA ingresada este en el rango de 0 a 20, si no es así da un mensaje de error. El campo COD es normal.

- BROW FIEL cod,edad:V=edad>18:F:E=``Debe ser mayor de edad

Con :V imponemos la condición, con :F forzamos a que siempre valide y con :E imponemos el mensaje de error.

- BROW FOR edad<21 .AND. nota>10 NOAP NODE

Aquí mostraremos todos los campos de los registros que cumplan la condición impuesta. Los alumnos que tengan menos de 21 años y al mismo tiempo estén aprobados en el curso. Además no será posible añadir ni borrar ningún registro.

- BROW FIEL cod,apepat,apemat,nom,nota:R WHEN nota<11

Solo muestra los campos mencionados de todos los registros de la tabla, pero solo será posible modificar los apellidos y el nombre de los que estén desaprobados en el curso. La NOTA nunca podrá ser modificada.

- BROW FIEL cod,apepat,apemat,nom,nota FREE cod

Aquí el cursor siempre permanecerá en el campo COD.

- LIST OFF cod,nombre FOR PROM>10

Muestra por pantalla sin numero de registro, los campos COD y NOMBRE, de los registros con nota promedio aprobatoria.

- SET ORDE TO nombre

DISP nombre FOR (DATE()−nacio)/365>18

Muestra por pantalla con pausa, los nombres de los alumnos ordenados alfabéticamente, con mas de 18 años.

Ejercicios prácticos:

- Crear una tabla con el nombre MAEALU que contenga los siguientes campos: (ingresar por lo menos 5 registros)

Nombre	**Tipo**	**Tam**	**Descripción**
COD	Carácter	5	Código del alumno
NOM	Carácter	20	Nombres del alumno
APE	Carácter	40	Apellidos del alumno
FEC	Fecha	8	Fecha nacimiento
PRO	Numérico	5.2	Promedio ponderado
P	Lógico	1	Esta al día en pagos o no

- Indexar la tabla MAEALU en los siguientes ordenes: código del alumno, apellidos y nombre, fecha de nacimiento y nota promedio.
- Como se donde se encuentra el puntero de registro en cierto momento ?
- Buscar alguien de apellido PAR...
- Colocar el puntero en el registro # 4 y visualice el registro completo (todos sus campos)
- Mueva el puntero dos registros hacia atrás según el índice activo y márquelo para borrarlo.
- Borrar los registros cuya nota sea menor que 8.
- Mostrar por pantalla con pausa los apellidos y nombres y nota promedio de los alumnos que están al Día en sus pagos, en orden de edad de mayor a menor.
- Cuantos registros tiene la tabla abierta en el área 2 ?
- Crear un archivo de texto que contenga los registros completos pero sin numero de registro, de los alumnos matriculados en este curso, en orden alfabético.
- Con dos ventanas BROW muestra los cursos en los que esta matriculado cada alumno.
- Con dos ventanas BROW muestra los alumnos que están matriculados en cada curso−seccion.
- Saque un promedio de notas de todos los alumnos
- Cuantos alumnos con promedio aprobado hay ?
- Cuantos alumnos llevan este curso ?

Bibliografía:

Programación en base a eventos

− Cesar A. Bustamante Gutierrez − UNI

FoxPRO 2.6 para DOS y Windows a su alcance

− José Javier García Badell − McGraw Hill

Programación basica con FoxPRO

− Ramón M. Chordá − Rama

Al dia en una hora en FoxPRO 2.6

− José Carlos Corrales − Anaya Multimedia

Guia practica FoxPRO 2.6

− Alejandro Dominguez − Anaya Multimedia

Diseño de programas: 200 algoritmos y un proyectos

− Julio Vasquez Paragulla − Editorial San Marcos

Microsoft FoxPRO for DOS and Windows, lenguage reference

− Microsoft Press

Ejemplos y ejercicios practicos en FoxPRO 2.6

− Ing Enrique Garrido−Lecca Risco − Universidad Alas Peruanas

Capitulo 1

Programación orientada a eventos

La principal característica de FoxPRO es la filosofía de programación orientada a eventos que propone.

Si te esfuerzas por adoptar esta filosofía, lograras nítidas ventajas respecto a la programación tradicional, orientada a menús jerárquicos y anidados como ocurre con sus antecesores (FoxBASE, dBASE, Clipper y otros DBMS)

Esta filosofía esta apoyada básicamente por la programación de teclas (ON KEY LABEL) y el uso de las cláusulas VALID y WHEN de algunos comandos como GET y BROW básicamente con funciones de usuario (FDU).

Procedimientos y funciones de usuario

Un programa es un conjunto de códigos que realiza una tarea especifica, la cual se puede llamar con el comando **DO <nombre>** ya sea desde la ventana de comando o desde otro programa. Es por eso que a los procedimientos también se les llama subprogramas.

El inconveniente de usar subprogramas es el tiempo que demora FoxPRO en acceder al disco en busca del código.

Para solucionar este problema FoxPRO permite usar **procedimientos**, que son subprogramas ubicados en el mismo archivo que el programa principal, de manera que son cargados en memoria junto con el programa principal y FoxPRO los lee muy rápidamente desde memoria.

También es posible usar procedimientos contenidos en otros archivos con el comando **DO <nombre> IN <archivo>** lo cual no aporta nada en cuanto a velocidad, por lo que es mucho mejor usar el concepto de librería que es un archivo PRG donde se ponen todos los procedimientos que requerimos sean llamados desde cualquiera de nuestros programas de una aplicación, el cual se carga en memoria con el comando **SET PROCEDURE TO <archivo librería>** normalmente ubicado en el programa principal.

Existen dos maneras de transferir información desde y hacia los procedimiento:

1. Usando mismas variables:

Inicializamos una serie de variables y luego tenemos el cuidado de usar exactamente las mismas en el procedimiento, para que al volver mantengan los valores calculados en el procedimiento.

PRINCIPAL.PRG	
xNUM=2	**********
xPOT=3	PROC OPERA
RESP=0	**********
DO OPERA	RESP=xNUM^xPOT
? RESP	RETURN
RETURN	

Nota: Las variables creadas en un procedimiento padre, son publicas para los procedimientos llamados desde el. Las variables creadas en un procedimiento hijo, NO existirán al retornar al procedimiento padre.

2. Pasando parámetros:

Pasándole paramentos al procedimiento, para que opere los valores y devuelva el valor, sin importar las variables que es cada lado se usen.

PRINCIPAL.PRG	
xNUM=2	**********
xPOT=3	PROC OPERA
RESP=0	**********
DO OPERA WITH xNUM,xPOT,RESP	PARA X,Y,Z
? RESP	Z=X^Y
RETURN	RETURN

FUNCIONES DE USUARIO

Una función se reconoce porque lleva () al final de su nombre y porque retorna un valor que puede ser numérico, carácter, fecha o lógico.

Su estructura es muy similar a la de los procedimientos solo que se usara FUNCTION en lugar de PROCEDURE para definirla.

Para ejecutar una función no se usa DO, simplemente se le llama por su nombre con el () al final.

1. Usando mismas variables:

PRINCIPAL.PRG	**********
xNUM=2	FUNC OPERA
xPOT=3	**********
? OPERA()	RETURN xNUM^xPOT
RETURN	

2. Pasando parámetros:

Para transferir parámetros no se usa la cláusula WITH simplemente se introducen dichos parámetros en el paréntesis.

PRINCIPAL.PRG	**********
xNUM=2	FUNC OPERA
xPOT=3	**********
? OPERA(xNUM,xPOT)	PARA X,Y
RETURN	RETURN X^Y

Quizás donde son mas interesantes de usar las funciones de usuario sea en la validación, osea cláusulas VALID y WHEN de comandos como GET y BROW.

Justamente el uso de esta filosofía es a la que se le da el nombre de **programación orientada a eventos** y no a menús jerárquicos o anidaciones, como es habitual en un principiante de FoxPRO o al que usa FoxBASE, Clipper y otros gestores de bases de datos del mercado.

Elección entre procedimientos y funciones

Algunas veces nos encontraremos en la duda de crear un procedimiento o una función de usuario, para realizar una tarea concreta. Existen algunas reglas para determinar el uso de uno u otro:

Usemos procedimientos cuando sea necesario:

ð Dividir un programa en módulos: Ya que es mas fácil mantener el código posteriormente, debido a la separación de operaciones y el reducido tamaño de los programas, acelerando la ejecución y se documenta mejor.

ð Realizar tareas repetitivas: Los formatos empleados para mostrar datos en pantalla con un modelo determinado son un ejemplo del uso de procedimientos.

Usemos funciones cuando sea necesario:

ð Validar datos o procesos: Dado que las funciones se llaman directamente por su nombre sin requerir del comando DO.

ð Realizar cálculos: Las operaciones en general son mas cómodas de hacer con funciones.

ð Sumar procesos: En informes es el único camino para sumar procesos.

Capitulo 2

CONTROL SAY/GET

@ <fila>,<columna> SAY <Exp> GET <var>/<campo>

SAY visualiza la expresión en las coordenadas indicadas

GET asigna a la variable o campo el valor tecleado

La lectura se hace cuando se encuentra el comando **READ**.

FUNCTION/PICTURE <Exp>

Ambas clausulas nos proporcionan un formateo especial de presentacion de la variable, la diferencia fundamental entre ambas clausulas es que FUNCTION actua de forma global, miestras que PICTURE actua elemento a elemento.

En <Exp> se usa un codigo especial definido en las tablas siguientes:

Function:

A	Solo caracteres alfabéticos, no espacios ni símbolos
B	Justifica los números a la izquierda
D	Usa el formato de fecha activo
L	Muestra ceros a la izquierda
I	Centra texto
T	Elimina los espacios en blanco iniciales y finales.

S<n>	Limita el ancho a n caracteres, aunque puedes poner mas.
J	Justifica texto a la derecha
X	Cualquier carácter
Z	Presenta blanco en lugar de cero
!	Convierte a mayúsculas

Picture:

A	Solo caracteres alfabéticos
L	Solo datos lógicos
N	Solo letras o dígitos
X	Cualquier carácter
!	Convierte a mayúsculas
9	Dígitos o en blanco
#	Dígitos, blanco o signo
*	Muestra asteriscos a la izquierda de números.
.	Separador decimal
,	Separador de miles

Los codigos de FUNCTION pueden ser introducidas en PICTURE con un símbolo @. Se puede poner varias condiciones solo hay que separarlas por ;

DEFAULT <Exp>

Si no existe esta cláusula deberá ser definida previamente la variable usada en el GET u obtendrás un error. En caso de ya existir la variable, esta cláusula es ignorada.

MESSAGE <mensaje>

Muestra <Exp> como un mensaje sobre la línea 24. Este mensaje podría ser presentado en otra línea con SET MESS TO

SIZE <Exp1>,<Exp2>

Con <Exp1> manejamos al alto (# de filas), pero siempre debera ser de 1 fila para FoxPRO DOS y con <Exp2> puedemos manejar el ancho (# de columnas) de <var>/<campo>. Si no usas esta cláusula el ancho lo determina la cláusula PICTURE. Si tampoco se uso esta cláusula el ancho queda definido por el contenido actual de la variable o campo.

RANGE <Exp1>,<Exp2>

Solo se usa para datos numéricos o de tipo fecha y es el rango en el cual deberá estar contenido el dato. Esta condición solo se evalúa cuando se cambia de valor y no cuando solo aceptas el valor inicial vía <Enter>. En caso de error devuelve el error estándar o la <Exp> de la cláusula ERROR impidiendo que se abandone el GET. Es decir es una post−condicion.

VALID <Cond>

Es una post−condicion es decir que una vez ingresado un valor y se intente **salir** del GET se evaluara <Cond>. Si esta es verdadera, permitirá la salida, pero si es falsa se devuelve un mensaje de error estándar o la <Exp> de la cláusula ERROR impidiendo que se abandone el GET. Nota que se puede salir del GET con <Esc>, pero no aceptara el cambio.

Es posible devolver un numero en lugar de un valor lógico, si es 0 el curso no avanzara, quedándose en el mismo objeto, si el numero es positivo avanzara y si es negativo retrocederá según el numero.

ERROR <mensaje>

Si se añade esta cláusula junto con VALID se mostrara <mensaje> en lugar del mensaje estándar.

WHEN <Cond>

Solo permitirá editar la <var>/<campo> si <Cond> es verdadera antes de **entrar** a dicho GET. Es decir es una pre−condicion.

COLOR

Permite determinar una serie de pares de color, que afectan a diversas partes según la siguiente tabla:

Colores	
N	Negro
W	Blanco
B	Azul
R	Rojo
G	Verde
GR+	Amarillo
RB	Magenta
BR	Cyan
GR	Marrón
X	Invisible

Atributos	
+	Mas intenso primer plano
*	Aclarar color de fondo

Monocromáticos	
N	Negro
W	Blanco
U	Subrayado
I	Vídeo inverso
+	Mas intenso primer plano
*	Parpadeante

Pares de color	
1	Expresión de SAY
2	Región GET
5	Texto de mensaje
6	Región GET seleccionado
10	Región GET desactivado

WINDOWS <nombre>

Esta cláusula es usada para editar un campo MEMO. La ventana deberá ser creada previamente y debes usar <Ctrl><W> para guardar los cambios.

ENABLE/DISABLE

Permite acceder o no al GET. Normalmente se activa desactiva desde una función de usuario que esta como VALID de otro GET vía el comando SHOW GET <var> ENABLE/DISABLE

SHOW

SHOW GETS [ENABLE/DISABLE] [LEVEL <#>] [OFF / ONLY] [LOCK] [WINDOW <nombre>] [COLOR <pares>]

Refresca todos los objetos pendientes y ejecuta la rutina SHOW del READ activo.

Cuando se incluye OFF no se refrescan los objetos, solo se ejecuta la rutina de SHOW del READ activo. Con ONLY solo se refrescan los objetos y no se ejecuta la rutina SHOW del READ activo.

Cuando se incluye LOCK se bloquea el registro si se uso <campo> en lugar de <var>

SHOW GET <var>/<campo> [,<#> [PROMPT <exp>]] [ENABLE/DISABLE] [LEVEL <#>] [COLOR <pares>]

Vuelve a mostrar el objeto representado por la <var> o <campo> de manera que actualiza la pantalla si se cambio su valor o cambia el texto de un boton (<#> especifica la opcion en caso de un boton multiple como RADIO o PUSH) o cambiar el estado de abilitado a desabilitado y viseversa.

Tipicamente se usa una funcion de usario en una clausula VALID de otro boton, la clausula WHEN del mismo boton o la clausula en READ.

SHOW OBJECT <#> [ENABLE/DISABLE] [LEVEL <#>] [PROMPT <exp>][COLOR <pares>]

Funciona igual que SHOW GET solo que la referencia no se hace con <var> o <campo> sino por el numero de objeto segun OBJNUM() o _CUROBJ.

CLEAR

CLEAR GETS

Cancela todas los GET pendientes

CLEAR READ [ALL]

Cancela el READ actual y retorno el control al antior o cancela todos de incluir la clausula ALL.

RDLEVEL()

Retorna el nivel READ actual.

OBJNUM() y _CUROBJ

Capitulo 3

BOTONES OBJETO

BOTON DE RADIO (*R)

Son un conjunto de opciones precedidas de paréntesis y solo una de ellas podrá seleccionarse a la vez, vía un punto. Por omision no termina el READ CYCLE.

@ f,c GET <var>/<campo> [FUNC <formato> / PICT <formato>] [DEFA <exp>] [SIZE <n1,n2,n3>] [ENAB / DISA] [MESS <mensaje>] [VALID <cond>] [WHEN <cond>] [COLOR <pares>]

SIZE

Con esta clausula manejamos el tamaño. n1 determina el alto en filas (siempre 1 en DOS), n2 determina el ancho y n3 determina el espacio entre las opciones.

Pares de color	
5	Texto de mensaje
6	Botón seleccionado
7	Tecla caliente
9	Botón habilitado
10	Botón desactivado

x=1

@ 5,5 GET x PICT `@*R \<Masculino;\<Femenino'

READ

IF x=1

? `Varón'

ELSE

? `Mujer'

ENDIF

o también:

x=`Masculino'

@ 5,5 GET x PICT `@*R \<Masculino;\<Femenino'

READ

? x

BOTON DE CONFIRMACION o CHECK BOX (*C)

Representa una sola opción independiente que puede tomar el valor de verdadero o falso, 1 o 0, vía una aspa. Por omision no termina el READ CYCLE.

@ f,c GET <var>/<campo> [FUNC <formato> / PICT <formato>] [DEFA <exp>] [SIZE <n1,n2>] [ENAB / DISA] [MESS <mensaje>] [VALID <cond>] [WHEN <cond>] [COLOR <pares>]

SIZE

Con esta clausula manejamos el tamaño. n1 determina el alto en filas (siempre 1 en DOS), n2 determina el ancho en columnas.

Pares de color	
5	Texto de mensaje
6	Botón seleccionado
7	Tecla caliente
9	Botón habilitado
10	Botón desactivado

```
STORE .F. TO x1,x2,x3

@ 5,5 GET x1 PICT `@*C Uno'

@ 6,5 GET x2 PICT `@*C Dos'

@ 7,5 GET x3 PICT `@*C Tres'

READ

? `Quiere: '

IF x1

?? `Uno'

ENDIF

IF x2

IF x1

?? ` y `

ENDIF

?? `Dos'

ENDIF

IF x3

IF x1 .OR. x2

?? ` y `

ENDIF

?? `Tres'

ENDIF
```

```
IF !(x1 .OR. x2 .OR. x3)

?? ` ninguno'

ENDIF
```

Nota: También puedes inicializar las variables con 0 o 1.

BOTON PULSADOR o PUSH BUTTONS (*)

Es un conjunto de opciones que por omision terminan el GET CYCLE cuando se selecciona alguna.

@ f,c GET <var>/<campo> [FUNC <formato> / PICT <formato>] [DEFA <exp>] [SIZE <n1,n2,n3>] [ENAB / DISA] [MESS <mensaje>] [VALID <cond>] [WHEN <cond>] [COLOR <pares>]

SIZE

Con esta clausula manejamos el tamaño. n1 determina el alto en filas (siempre 1 en DOS), n2 determina el ancho en columnas y n3 el espaciamiento entre elementos.

Pares de color	
5	Texto de mensaje
6	Opción seleccionada
7	Tecla caliente
9	Opción habilitado
10	Opción desactivado

```
ok=1

DO WHILE .T.

@ 5,5 GET ok PICT `@* \!\<Ejecutar;\?\<Cancelar'

READ

IF ok=2

EXIT

ENDIF

? `Bien, continuamos...'

ENDDO
```

Nota: Hay dos opciones predeterminadas:

	Descripcion	Activa con
\!	Opcion por defecto, que estara resaltada	<Ctrl><Enter>
\?	Opcion de escape	<Esc>

MENU DESPLEGABLE o BOTON POPUP (^)

Un marco en cuyo interior están las opciones en forma vertical. Por omision no termina el READ CYCLE. Se puede incluir SIZE <ignorado>,<ancho>

@ f,c GET <var>/<campo> [FROM <matriz> [RANGE <n1[,n2]>]] / [POPUP <nombre>] [FUNC <formato> / PICT <formato>] [DEFA <exp>] [SIZE <n1,n2>] [ENAB / DISA] [MESS <mensaje>] [VALID <cond>] [WHEN <cond>] [COLOR <pares>]

RANGE

Por omision el POPUP toma la primera columna de la matriz. Tu puedes cambiar esto determinado de que elemento quieres empezar con n1 y opcionalmente tambien puedo determinar cuantos elementos tomar con n2.

Por ejemplo si tenemos una matriz 3 x 3 asi:

a	b	c
d	e	f
g	h	i

Si uso n1=3, el POPUP toma los elementos c, f, i.

Si uso n1=5, el POPUP toma los elementos e, h.

Si uso n1=2 y n2=2, el POPUP toma los elementos b, e.

SIZE

Con esta clausula manejamos el tamaño. n1 es ignorado y n2 determina el ancho en columnas.

Pares de color	
1	Opción desactivada
2	Opción habilitada
3	Borde
5	Texto de mensaje
6	Opción seleccionada
7	Tecla caliente
9	POPUP habilitado
10	POPUP desactivado

Hay tres maneras de definir las opciones de un popup:

ð Desde la clausula FUNC o PICT

ð Desde una matriz

ð Desde un menu POPUP

X=1

@ 5,5 GET x PICT `@^ opc1; opc2; opc3'

READ

@ 5,5 GET y PICT `@^' FROM w

READ

Nota: Si w fuera una matriz bidireccional se mostraran solo los valores de la primera columna de la matriz.

@ 5,5 GET z PICT `@^' FROM w RANGE 2

READ

Nota: Si w fuera una matriz bidireccional se mostraran solo los valores de la columna indicada en RANGE <n1>.

@ 5,5 GET z PICT `@^' FROM w RANGE 3,3

READ

Nota: Si w fuera una matriz bidireccional se mostraran solo los valores de la columna indicada con RANGE <n1> y el numero de elementos con <n2>.

Ejemplo 1:

USE alumnos

DIMM xSTRU(FCOUNT())

FOR a=1 to FCOUNT()

xSTRU(a)=FIELD(a)

ENDFOR

@ 5,5 GET y PICT `@^' FROM xSTRU SIZE 1,20

READ

RETURN

Ejemplo 2:

USE alumnos

=AFIELDS(xSTRU)

COPY TO ARRAY xCAMP

@ 2,5 GET x PICT '@^' FROM xSTRU DEFA 1 VALID VER()

@ 5,5 GET y PICT '@^' FROM xCAMP RANGE x DEFA 1

@ 3,30 GET xOK PICT '@* Salir' DEFA 1

READ CYCLE

CLEA

RETURN

*–––––––

FUNC VER

*–––––––

SHOW GETS

RETURN

Una interesante opción aquí es el uso del parámetro POPUP, donde previamente se debió definir dicho POPUP con:

DEFI POPUP <nombre> PROMPT	FILES [LIKE <esqueleto>]
	STRUCTURE
	FIELD <campo>

SIZE <f>,<c>	Alto en filas y ancho en columnas del popup
MARG	Añade un margen adicional
SCROLL	Permite manejar una barra de desplazamiento
MARK <Exp>	Simbolo a usar al seleccionar
SHADOW	Pone una sobra
TITLE <Exp>	Pone <Exp> como titulo del popup
FOOT <Exp>	Pone <Exp> como pie del popup

para luego llamar:

@ 5,5 GET <var> POPUP <nombre> SIZE <alto>,<ancho>

READ

BOTON DE LISTAS (&)

Es muy parecido al anterior, solo que aquí todas las opciones son visibles todo el tiempo. Por omision no termina el READ CYCLE.

@ f,c GET <var>/<campo> FROM <matriz> [RANGE <n1,n2>] [FUNC <formato> / PICT <formato>] [DEFA <exp>] [SIZE <n1,n2>] [ENAB / DISA] [MESS <mensaje>] [VALID <cond>] [WHEN <cond>] [COLOR <pares>]

SIZE

Con esta clausula manejamos el tamaño. n1 determina el alto en filas y n2 determina el ancho en columnas.

Pares de color	
1	Opción desactivada
2	Opción habilitada
3	Borde y barra de desplazamiento
5	Texto de mensaje
6	Opción seleccionada
9	Lista habilitado
10	Lista desactivado

@ 5,5 GET x PICT `@&' FROM w SIZE 5,20

READ

BOTON INVISIBLE (*I)

Es un conjunto de opciones de pantalla que pone en vídeo inverso lo que hay en esa posición, permitiendo que sea cualquier cosa lo que se muestre. Después de *Y hay un espacio seguido de tantos ; como botones −1 hay. Aquí es muy importante usar la cláusula SIZE <alto>, <ancho>, <separación>. Por omision no termina el READ CYCLE.

@ f,c GET <var>/<campo> [FUNC <formato> / PICT <formato>] [DEFA <exp>] [SIZE <n1,n2,n3>] [ENAB / DISA] [MESS <mensaje>] [VALID <cond>] [WHEN <cond>] [COLOR <pares>]

SIZE

Con esta clausula manejamos el tamaño. n1 determina el alto en filas, n2 determina el ancho en columnas y n3 determina el espacio entre las opciones en columnas.

Pares de color	
5	Texto de mensaje
6	Botón seleccionado

FOR y=0 TO 32 STEP 8

@ 5,5+i TO 8+i,9

ENDFOR

@ 5,5 GET x PICT `@*I ;;;' SIZE 3,4,5

READ

Nota: 4 rectángulos de 3 caracteres de alto, 4 de ancho, separados 5 caracteres entre ellos.

Opciones:

M	Multiples opciones en el mismo GET
S	Simple. Una sola opcion por cada GET
T	Termina: Después de seleccionada una opción termina el READ
N	No termina: Continua el READ CYCLE
H	Horizontal: Acomoda las opciones una bajo otra.
V	Vertical: Acomoda las opciones una al lado de otra.
\<	Tecla caliente: Se ilumina y calienta la letra siguiente a este símbolo.
\!	Por defecto: Se marca con << >> y un <Ctrl><Enter> la activa.
\?	Salida por defecto: Cuando se pulsa <Esc> se ejecutara esta opción.
\\	Desactivada: Opción desactivada

Resumen de botones objeto:

Botón	**Picture**	**Tipo**	**Omisión**		**Especiales**
Radio	@*R	M	NV	TH	\<, \\
Check	@*C	S	N	T	\<, \\
Push	@*	M	TV	NH	\<, \\, \!, \?
Popup	@^	M	N	T	\<,\
Lista	@&	M	T	N	
Invisible	@*I	M	NV	TH	\\

Capitulo 4

Activar botones: READ

Activa todos los objetos creados desde el último READ o CLEAR GETS. Los objetos son campos GET, casillas de verificación, listas, emergentes, botones de comando, botones de radio y regiones de edición de texto definidas con @ ... GET.

Un usuario puede salir de un READ de varias formas:

ð Moviéndose hacia adelante/atrás mas allá del primero/último objeto. Solo si **no** se añadió la cláusula CYCLE.

ð Al presionar <Esc>, <Ctrl><W> o elegir un botón que estaba definido para terminar el READ.

ð También se temían el READ emitiendo CLEAR READ o debido a la cláusula TIMEOUT.

Notas:

ð Un READ puede abarcar varias ventanas.

ð Cuando mueva el cursor de objeto a objeto, los objetos se activaran en el orden en el que se crearon, independientemente de la ventana en la que estén situados.

ð Cuando mueva el cursor a un objeto, la ventana que lo contiene se convertirá en la ventana de salida.

ð Los READ anidados se crean definiendo objetos y emitiendo READ en una rutina que se llamará mientras que el READ esté activo. Los comandos READ pueden anidarse hasta cinco niveles de profundidad. RDLEVEL() devuelve el nivel READ actual.

Orden de ejecución de los eventos:

1. Cláusula WHEN del nivel READ

2. Primer GET de la ventana

3. Cláusula ACTIVATE del nivel READ

4. Cláusula SHOW del nivel READ

5. Cláusula WHEN del nivel GET para el primer GET

La lista siguiente muestra el orden en el que se llama a las cláusulas READ cuando se activa una ventana nueva:

1. Cláusula VALID del campo del que se sale

2. Se desactiva la ventana del campo del que se sale

3. Se activa la ventana del nuevo campo GET

4. Cláusula DEACTIVATE del nivel READ

5. Cláusula ACTIVATE del nivel READ

6. Cláusula WHEN del campo nuevo

ACTIVATE <ExpL1>

Esta cláusula se ejecuta cuando se emite el READ y cada vez que cambia la ventana. Puede verse como una cláusula WHEN a nivel de ventana.

La expresión lógica <ExpL1> es normalmente una función definida por el usuario, que usa WOUTPUT() para determinar qué ventana está activa y puede desactivar objetos @ ... GET de otras ventanas, ocultar ventanas, mostrar un mensaje, etc.

DEACTIVATE <ExpL2>

Esta cláusula se ejecutará siempre que intente traer hacia adelante cualquier otra ventana (esto es, siempre que cambie WONTOP()). Puede verse como una cláusula VALID a nivel de ventana.

La expresión lógica <ExpL2> es normalmente una función definida por el usuario, que puede usarse para validar el contenido de campos de una ventana antes de permitirle poner delante otra ventana. El READ terminara o no dependiendo de la respuesta de la función: verdadero (.T.) o falso (.F).

MODAL

Esta cláusula impide que se activen las ventanas que no están involucradas en el READ.

WITH <lista títulos de ventana>

De forma predeterminada, todas las ventanas interjectivas (ventanas Examinar, accesorios de escritorio y ventanas abiertas con MOD FILE, MODI REPORT, etc.) pueden participar en un READ. Incluya la cláusula WITH para restringir las ventanas que participan en el READ. Las ventanas incluidas en la cláusula WITH pueden incluir ventanas del sistema (Examinar, accesorios y ventanas de edición texto y memo). La inclusión de la cláusula WITH crea automáticamente un READ MODAL.

Incluya el título de la ventana Examinar (de forma predeterminada, el alias de la tabla) en la lista de ventanas para que esté accesible la ventana Examinar.

Incluya el alias de la tabla en la lista de ventanas para que esté accesible la ventana Memo.

Si el título de una ventana contiene un carácter que no es alfabético, numérico o un subrayado, incluya la parte del título hasta el primer carácter que no sea alfabético, numérico o subrayado en la lista de ventanas. Por ejemplo, si el título de la ventana es 'Lista de clientes', incluya Lista en la lista de ventanas (el espacio entre 'Lista' y 'de' no es alfabético, numérico o un subrayado).

Si el título de una ventana contiene un carácter que no es alfabético, numérico o un subrayado, puede situar el título entre comillas en la lista de ventanas. Por ejemplo si el título de la ventana es 'Lista de clientes', puede incluir "Lista de clientes" en la lista de ventanas.

SHOW <ExpL3>

Esta cláusula se ejecuta siempre que se emite SHOW GETS. El valor devuelto por una rutina SHOW se ignora. Una rutina SHOW puede usarse para refrescar @ ... SAY o para activar o desactivar objetos.

VALID <ExpL4> | <ExpN1>

Esta cláusula se evalúa cuando intenta salir del READ actual o cuando se emite READ sin una cláusula @ ... GET que excede el tiempo. La cláusula VALID puede especificar una expresión lógica o numérica o una función definida por el usuario (FDU) que devuelve un valor lógico o numérico.

El READ se terminará si la expresión <ExpL4> se evalúa verdadera (.T.). Si VALID <ExpL4> se evalúa como falsa (.F.), el objeto actual permanecerá activo, si es posible.

Si el objeto no puede permanecer activo (quizá la rutina VALID desactivó el objeto), se moverá al primer objeto.

Si la cláusula VALID devuelve un número, se moverá al objeto correspondiente. El READ se terminará si no existe un objeto correspondiente al número. Una cláusula VALID que devuelve un

valor que no sea de tipo Lógico o Numérico tiene el mismo efecto que si devolviera verdadero (.T.).

WHEN <ExpL5>

Esta cláusula determina si se ejecuta el READ. <ExpL5> debe evaluarse verdadera (.T.) al emitir el READ para que se ejecute el READ. Si <ExpL5> se evalúa falsa (.F.), se ignorará el READ y la ejecución del programa continuará con el comando que sigue inmediatamente a READ.

OBJECT <ExpN2>

Incluya esta cláusula para especificar el objeto que se selecciona inicialmente cuando emita READ. <ExpN2> determina el objeto que estará seleccionado inicialmente. Los números de los objetos se determinan por el orden en el que se crean los objetos.

Cada botón de comando, botón radio e invisible individual se considera un objeto. En el ejemplo de programa siguiente, se crean un campo y tres botones radio. El botón radio central se selecciona inicialmente incluyendo la cláusula OBJECT 3 con el READ. El campo NOMBRE es el objeto número 1, el primer botón radio es el objeto número 2, el segundo botón radio es el objeto número 3 y el último botón radio es el objeto número 4.

```
radio=1

nombre=SPACE(10)

CLEAR

@ 2,2 SAY 'Introduzca un nombre: ' GET name

@ 4,2 GET radio PICTURE '@*R Manzanas;Naranjas;Limones'

READ CYCLE OBJECT 3
```

TIMEOUT <ExpN3>

Esta cláusula determina cuanto tiempo estará en efecto un READ. <ExpN3> especifica el número de segundos que pueden transcurrir sin que el usuario introduzca nada antes de que se termine el READ. Si el READ se termina por una cláusula TIMEOUT, READKEY() devuelve 20 si no se han hecho cambios a ninguno de los objetos. Si se han hecho cambios, READKEY() devolverá 276.

Cuando se termina READ por una cláusula TIMEOUT, se descartarán todos los cambios hechos en el campo que se estaba editando cuando terminó el READ. Sin embargo, se guardan los cambios hechos en otros campos.

SAVE

Todas las definiciones de objetos se eliminan cuando se termina un READ a menos que se incluya la palabra clave SAVE. Puede remitir READ sin recrear los objetos si incluye SAVE con el READ previo.

NOMOUSE

Incluya la palabra clave NOMOUSE para impedir que se seleccionen objetos con el Mouse. Debe utilizar el teclado para moverse de objeto en objeto. Puede, no obstante, utilizar el Mouse dentro de los campos para cortar, copiar y pegar y para posicionar el cursor.

LOCK | NOLOCK

Las palabras clave LOCK y NOLOCK le permiten especificar si un registro que contiene campos especificados en los objetos se bloquea automáticamente durante un READ cuando la tabla está abierta para uso compartido en una red.

Si se incluye LOCK, se intentarán bloqueos de registro para cada registro utilizado en los objetos. Si los bloqueos se realizan con éxito, podrá editar los registros. SET REPROCESS determina cuantas veces o durante cuanto tiempo se intentarán los bloqueos. Si no se incluye LOCK o NOLOCK, READ asumirá, de forma predeterminada READ LOCK.

Si se incluye NOLOCK, no se bloquearán los registros utilizados en los objetos. En su lugar, todos los objetos que utilicen campos de los registros se harán de solo lectura y no podrá seleccionarse el control y se mostrará en colores desactivados.

Si emite READ NOLOCK y modifica un campo memo, el registro se bloqueará automáticamente.

READ de fundación

FoxPRO te permite crear fácilmente interfaces controladas por eventos (como el interfase FoxPRO) para sus aplicaciones.

Con versiones anteriores de FoxPRO, algunos usuarios sofisticados usaban un bucle de eventos para controlar las opciones de menú y las selecciones de ventana al crear una interfaz controlada por eventos. El bucle de eventos debía sondear constantemente el estado del sistema y luego actuar basándose en la ventana que estaba encima o en la opción de menú elegida por el usuario.

DO WHILE se usa para crear el bucle de eventos (a menudo un bucle grande). Deben evitarse los bucles de eventos por las razones siguientes:

Un solo READ con varias ventanas

Es muy raro necesitar programación de bucles de eventos en FoxPRO. Puede usarse un solo READ que soporte varias ventanas para eliminar la mayoría de los bucles de eventos. Un solo READ con múltiples ventanas proporciona versatilidad y flexibilidad sin programación compleja.

Sugerimos que delimite todas las ventanas con las que quiere interactuar. FoxPRO puede coordinar ventanas que contienen objetos (@..GET, casillas de verificación, botones radio, emergentes, etc.) con otras ventanas que no están asociadas habitualmente con un READ. De forma predeterminada, todas las ventanas interactivas (ventanas Examinar, accesorios del escritorio, ventanas de edición memo y ventanas abiertas con MODIFY FILE, MODIFY REPORT) pueden participar en un READ.

Coordinar ventanas con un solo READ

Para coordinar mediante programa ventanas que contienen objetos @ ... GET con un solo READ:

1. Cree las ventanas que contendrán los objetos @ ... GET.

2. Active una de las ventanas y emita los @ ... GET que crean los objetos de la ventana.

3. Active otra ventana y emita los @ ... GET que crean los objetos de esa ventana.

4. Continúe de esta forma con el resto de las ventanas.

5. Emita READ.

El código de programa de conjunto de pantalla generado por GENSCRN proporciona un buen ejemplo de esta secuencia.

Las ventanas creadas y activadas están ahora coordinadas por un solo READ. Cada uno de los objetos GET aparece en su ventana respectiva y el usuario puede moverse entre los objetos @...GET de cada una de las ventanas. Cuando el usuario se mueve de un objeto a otro con el teclado, los objetos se atraviesan en el orden en que se han emitido los @...GET, independientemente de la ventana en la que estén. Una ventana se activa y se convierte en la ventana de salida actual cuando el objeto actual está en la ventana.

Cuando el usuario presiona <Tab>, <Enter> o Flecha abajo cuando el cursor está posicionado en el último objeto de una ventana, el cursor se posicionará sobre el primer objeto @...GET de la ventana siguiente. Cuando el usuario presiona Mayús+Tab o Flecha arriba en el primer objeto de una ventana, el cursor se posicionará en el último objeto @...GET de la ventana anterior. El usuario pude moverse también entre objetos @...GET diferentes de ventanas diferentes haciendo clic sobre los objetos.

El acceso a otras ventanas durante un solo READ puede controlarse con la cláusula READ MODAL.

Al incluir MODAL en READ impide la interacción con las ventanas que no pertenezcan al conjunto de ventanas coordinadas del READ. Cuando se incluye MODAL con READ, el usuario no podrá cerrar, ampliar, minimizar o mover ventanas que no estén involucradas en el READ. Al hacer clic sobre una ventana no involucrada en el READ sonará el tono, si BELL está SET ON.

De forma predeterminada, todas las ventanas interactivas (ventanas Examinar, accesorios del escritorio y ventanas abiertas con MODIFIY FILE, MODIFY REPORT, etc.) pueden participar en un READ.

Puede indicar explícitamente las ventanas que participan en el READ especificando sus nombres con la cláusula WITH. El comando READ se hace automáticamente MODAL si incluye la cláusula WITH para especificar otras ventanas involucradas en el READ. Las ventanas incluidas en la cláusula WITH pueden incluir ventanas del sistema FoxPRO (ventanas Examinar, accesorios del escritorio y ventanas de edición texto y memo).

ACTIVATE y DEACTIVATE

Cuando una ventana involucrada en el READ se trae hacia delante, la acción realizada puede controlarse con las cláusulas ACTIVATE y DEACTIVATE. El usuario podrá traer una ventana hacia adelante haciendo clic sobre la ventana, recorriendo cíclicamente las ventanas con Ctrl+F1 o eligiendo un control o una opción de menú que active la ventana.

Cuando una ventana que contiene el objeto esté encima del todo e intente poner encima otra ventana, la cláusula DEACTIVATE se ejecuta después de que se traiga hacia adelante la nueva ventana.

El valor devuelto por la cláusula DEACTIVATE determina si el READ está terminado. Si la cláusula DEACTIVATE devuelve verdadero (.T.), se termina el READ. Si la cláusula devuelve falso (.F.), el READ no se terminará.

Cuando se ejecuta la cláusula DEACTIVATE, WONTUP() devuelve el nombre de la nueva ventana que se ha puesto delante y WLAST() el nombre de la ventana que estaba antes delante del todo. Las funciones de ventana (WONTOP(), WOUTPUT() ...) devolverán falso (.F.) si incluye el nombre de una ventana que no existe.

Si su cláusula DEACTIVATE deja que la nueva ventana permanezca delante y la ventana nueva contiene objetos, se ejecutará la cláusula ACTIVATE.

Sugerencia Cuando se trae una ventana hacia adelante, puede retornar la ventana a su posición anterior emitiendo el comando siguiente en la cláusula DEACTIVATE.

ACTIVATE WINDOW (WLAST())

La función WREAD() facilita la manipulación de ventanas que participan en un READ. WREAD() le permite determinar si una ventana específica está participando en el READ actual.

Coordinar READ múltiples/READs de fundación

La sección anterior sobre un solo READ con múltiples ventanas recomienda una solución al control de múltiples ventanas con un solo comando READ. La solución mediante un solo READ funciona en la mayoría de las aplicaciones.

Sin embargo, en algunas aplicaciones puede ser necesario un READ de fundación. Un READ de fundación es un READ sin objetos. Un READ de fundación se utiliza para coordinar varios READ en una sola sesión interactiva.

Un READ de fundación incorpora normalmente código en la rutina de su cláusula VALID que activa los controles de los READ hijos. Los READ hijos también se denominan READ anidados.

Un READ de fundación tiene normalmente una cláusula VALID, La cláusula VALID se ejecuta cuando se produce el evento que en otro caso terminaría con el READ. Estos son los eventos que terminan un READ de fundación:

Cualquier clic de Mouse o pulsación de teclas que no sea una selección de menú o no ejecute una instrucción ON KEY LABEL.

Durante un READ de fundación, el menú del sistema FoxPRO está accesible.

Seleccionar un elemento de un menú o ejecutando un comando ON KEY LABEL no termina un READ de fundación. Si se activa un menú pero no se hace una selección del menú, el READ de fundación no se terminará.

Al terminar un READ hijo. Sin embargo, el READ de fundación no se terminará si se termina un READ nieto.

El valor devuelto por la cláusula READ VALID determina si se termina o no un READ de fundación.

Si la cláusula READ VALID devuelve verdadero (.T.), se terminará el READ de fundación; si devuelve falso (.F.), el comando READ de fundación permanecerá activo.

READKEY() facilita la gestión de los comandos READ. Especificando un argumento numérico en READKEY(), podrá determinar si se terminó el último READ. Por ejemplo, el valor puede indicar que el

último READ se terminó cerrando una ventana, por un CLEAR READ, por una cláusula READ VALID o DEACTIVATE que devuelva el valor lógico verdadero (.T.).

Estos son los valores que devuelve READKEY(<expN>) con la causa de terminación de READ

correspondiente:

Valor	Causa
1	Ninguna de las siguientes
2	Se ha emitido CLEAR READ
3	Se ha elegido el control de terminación
4	Se ha cerrado la ventana READ
5	La cláusula DEACTIVATE ha devuelto .T.
6	Se ha excedido el tiempo del READ

Capitulo 5

Otros objetos en FoxPRO

GETFILE(C1[,C2] [,C3] [,N])

Muestra el cuadro de dialogo `Abrir archivo' y devuelve la unidad, ruta, nombre y extensión del archivo elegido o una cadena nula si presionas <Esc> o elige el botón "Nada" o "Cancelar". Si eliges el botón "Nuevo", devolverá 'Anónimo' como nombre de archivo

C1: Designa la extensión de filtro, si no se pone ninguna mostrara todos los archivos. Ej:

ð PRG solo mostrara los archivos con esa extensión.

ð PRG;FXP mostrara los archivos de ambas extensiones, a no ser que tengan el mismo nombre en cuyo caso solo se mostrara el que lleva la extensión PRG.

ð PRG|FXP Muestra los archivos con ambas extensiones.

ð *P Muestra todas las extensiones que cumplen ese filtro. También puedes usar ? como comodín igual que DOS.

ð ; Mostraran solo los archivos que no tengan extensión.

ð Si no especificar nada, se mostraran todos los archivos

C2: Es el titulo en la parte superior del cuadro de dialogo.

C3: Texto en lugar del botón principal (Abrir por omisión)

N1: Numero y tipo de botón a usar.

ð 0 = <C3 o Abrir> y Cancelar

ð 1 = <C3 o Abrir>, Nuevo y Cancelar

ð 2 = <C3 o Abrir>, Nada y Cancelar

Ejemplo:

```
xDBF=GETFILE(`DBF','Ver una tabla','Ver',1)

DO CASE

CASE `Anonimo'$xDBF

CREATE ?

CASE EMPTY(xDBF)

RETURN

OTHER

USE &xDBF

BROW

ENDCASE
```

PUTFILE(C1[,C2] [,C3])

Muestra el cuadro de dialogo `Guardar como' y devuelve la unidad, ruta y nombre del archivo seleccionado.

Si no se introduce un nombre de archivo, se devuelve el nombre de archivo predeterminado (C2) y la extensión (C3).

Si se elige el botón de comando "Cancelar" o se presiona <Esc>, se devuelve la cadena nula.

C1 es el titulo en la parte superior del cuadro de diálogo.

C2 es el nombre de archivo predeterminado que se muestra en el cuadro de texto.

La primera extensión en C3 se añade automáticamente al nombre del archivo si no se incluye una extensión en C2. Unicamente se mostrarán los archivos con las extensiones especificadas.

ð PRG solo mostrara los archivos con esa extensión.

ð PRG;FXP mostrara los archivos de ambas extensiones, a no ser que tengan el mismo nombre en cuyo caso solo se mostrara el que lleva la extensión PRG.

ð PRG|FXP Muestra los archivos con ambas extensiones.

ð *P Muestra todas las extensiones que cumplen ese filtro. También puedes usar ? como comodín igual que DOS.

ð ; Mostraran solo los archivos que no tengan extensión.

ð Si no especificar nada, se mostraran todos los archivos

Este ejemplo crea un archivo de datos delimitado a partir de una tabla existente elegida por el usuario. GETFILE() se usa para encontrar y abrir una tabla y PUTFILE() se usa para devolver el nombre del archivo objetivo.

```
xDBF=GETFILE('DBF', 'Abrir tabla:')

USE (xDBF)

xNOM = ALIAS() + '.TMP'

xSAL = PUTFILE('Archivo de salida:', xNOM, 'TMP')

*−−− Se ha presionado Esc

IF EMPTY(xSAL)

CANCEL

ENDIF

*−−− Crear archivo delimitado

COPY TO (xSAL) DELI

MODI FILE (xSAL) NOEDIT
```

GETDIR(C1 [,C2])

Muestra el cuadro de diálogo Seleccionar directorio y devuelve el nombre de un directorio que se elija como una cadena de tipo Carácter.

Si no elige ningún directorio, ya sea que haces <Esc> o presionar el boton "Cancelar" se devolverá la cadena nula.

C1 especifica el directorio que se mostrará inicialmente en el cuadro de diálogo. Cuando no se especifica, se mostrara el directorio de trabajo.

C2 es el titulo en la parte superior del cuadro de diálogo.

LOCFILE(C1[,C2] [,C3])

Busca el archivo especificado y devuelve su ruta y nombre. Si no lo encuentra llama al cuadro de dialogo abrir (GETFILE()).

C1 especifica el nombre del archivo a buscar

C2 si no se especifico una extension en C1, esta es la lista de extensiones que se usaran para la busqueda

C3 es el titulo en la parte superior del cuadro de diálogo.

CREATE [<nombre>/?]

Muestra el cuadro de dialogo de crear una tabla.

GETEXPR C1 TO <var> [TYPE C2 [,C3]] [DEFAULT C4]

Llama el cuadro de dialogo `Generador de expresiones' y guarda la expresión construida por el usuario en la variable <var>.

C1 es el titulo que aparece en el cuadro, normalmente usado para recordar la clase de expresión a construir.

La expresión que se cree se guardará como una cadena de caracteres en la variable <var>. Si la variable de memoria no existe todavía, se creará.

Si sale del cuadro via "Cancelar", se almacenará la cadena C4 (si existe) o una cadena nula en la variable.

C2 especifica el tipo de expresión: C para carácter, N para numérico, D para fecha y L para lógico.

C3 es un mensaje de error que puedes usar solo si se uso tambien C2 y la expresión no es válida. C2 y C3 deben separase con punto y coma (;) y toda esta combinación debe encerrarse entre comillas.

C4 le permite mostrar la expresión inicial predeterminada, la cual se puede aceptar o sobrescribir una propia. Si se incluye la cláusula DEFAULT, C4 se almacenará en la variable de memoria si se sale del cuadro presionando ESC o eligiendo "Cancelar".

Ejemplo:

En este ejemplo, GETEXPT se usa para obtener una expresión LOCATE del tipo apropiado. Si LOCATE tiene éxito, se muestra el nombre de la empresa; en caso contrario, se muestra un mensaje.

```
CLOSE DATA

USE clientes

GETEXPR 'Escriba la condición a localizar ' TO x;

TYPE 'L;Error debe ser una condicion';

DEFA 'EMPRESA = " "'

LOCATE FOR &x

IF FOUND()

? 'Empresa:', empresa

ELSE

? 'La condición', x, ' no se encontró'

ENDIF
```

Capitulo 6

VENTANAS

DEFINE WINDOW

Crea una ventana y especifica sus atributos.

DEFINE WINDOW <nombre>

FROM <f1,c1> TO <f2,c2> / AT <f3,c3> SIZE <f4,c4>

[IN <ventana> | IN SCREEN]

[TITLE <Exp>]

[FOOTER <Exp>]

[DOUBLE | PANEL | NONE | SYSTEM | <cadena borde>]

[FLOAT]

[GROW]

[ZOOM]

[MINIMIZE]

[CLOSE]

[SHADOW]

[FILL <Exp>]

[COLOR <lista de pares de colores>]

El número de ventanas definidas por el usuario que puede crear está limitado únicamente por la cantidad de memoria y los recursos del sistema disponibles.

Las definiciones de ventana se sitúan en memoria y los nombres de las ventanas definidas por el usuario aparecen listados en la parte inferior del menú emergente Ventana. El nombre de la ventana de usuario creada en primer lugar aparece en la parte superior de la lista, seguido del nombre de cada ventana definida por el usuario adicional. El nombre de la ventana de usuario a la que se dirige actualmente la salida está marcado con un rombo. Si no hay ventana de salida, la salida se dirige al escritorio.

Especifique el nombre de la ventana a crear con <nombre>. Los nombres de ventana pueden tener hasta 10 caracteres de largo, debiendo empezar con una letra o subrayado y no pueden comenzar con un número. Pueden contener cualquier combinación de letras, números y subrayados.

La esquina superior izquierda de la ventana sobre el escritorio viene determinada por las coordenadas especificadas con <f1,c1> y la esquina inferior derecha se especifica con <f2,c2>. Estas coordenadas determinan el tamaño de la ventana.

Es posible definirse una ventana con coordenadas que caigan fuera del borde o que sean mayor que el escritorio. También pueden definirse ventanas situadas dentro de otras ventanas definidas por el usuario.

AT y SIZE pueden usarse también para especificar la posición y tamaño de una ventana.

IN <ventana>

Esta cláusula situar una ventana en otra ventana. Una ventana hija no puede moverse fuera de la ventana madre. Si se mueve la ventana madre, la ventana hija se moverá con ella.

Las coordenadas de la ventana hija especificadas con las cláusulas FROM y TO o AT y SIZE son relativas a la ventana madre, no al escritorio.

TITLE <Exp> asigna un título a la ventana, el cual se centra sobre el borde superior de la ventana. Si el título es más ancho que la ventana, se truncará el título.

FOOTER <Exp> asigna un pie a la ventana, el cual se centra sobre el borde inferior de la ventana. Si el pie es más ancho que la ventana, se truncará el pie.

DOUBLE | PANEL | NONE | SYSTEM | <cadena borde>

Puede especificar un borde incluyendo la cláusula DOUBLE, PANEL, NONE, SYSTEM o <cadena borde>. El borde predeterminado es una línea sencilla.

Nota: Si incluye otras cláusulas determinadas (GROW, ZOOM y así sucesivamente), los controles de ventana apropiados se situarán en el borde superior de la ventana.

FLOAT

Para mover una ventana, elija Mover del menú Ventana, presione <CTRL><F7> y use las teclas de flecha o con el Mouse puede arrastrar una ventana desde el borde superior o titulo.

Omita FLOAT para impedir que pueda moverse una ventana.

GROW

Para cambiar el tamaño de una ventana, elija Tamaño del menú Ventana o presione <Ctrl><F8>. Para cambiar el tamaño de una ventana con el Mouse, coloque el puntero sobre cualquier lado o ángulo de la ventana y arrástrelo.

Omita GROW para impedir que pueda cambiarse el tamaño de una ventana.

CLOSE

Si se incluye la cláusula CLOSE, podrá cerrar una ventana definida por el usuario desde el menú de Archivo, con las teclas <Ctrl><F4> o con un click en la esquina superior izquierda.

Si la ventana se creó con la cláusula SYSTEM, aparecera un cuadrado en el ángulo superior izquierdo, si no es asi se dispondrá del mismo cuadrado para cierre solo que "oculto" en el mismo lugar.

Al cerrar una ventana se quitará del escritorio o la ventana madre definida por el usuario y su definición se quitará de memoria.

Omite CLOSE para impedir que se cierre una ventana.

ZOOM

Una ventana definida por el usuario puede agrandarse hasta llenar por completo el escritorio. Si una ventana está agrandada a tamaño completo, también puede reducirse a su tamaño original.

Para agrandar una ventana elija Zoom flecha arriba del menú Ventana o presione <Ctrl><F10>.

Si la ventana se creó con la cláusula SYSTEM, puede hacer clic sobre el control de Zoom del ángulo superior derecho del borde de la ventana, si no es asi se dispondrá de un control Zoom "oculto" en el ángulo superior derecho del borde de la ventana. Al hacer clic sobre ese ángulo la ventana se agrandará hasta llenar todo el escritorio.

Repita estos pasos para volver la ventana a su tamaño original. Para agrandar una ventana hasta que llene por completo la ventana principal de FoxPRO, elija Maximizar del menú de control de la ventana, presione <CTRL><F10>, haga clic sobre el botón "Maximizar" de la ventana o haga doble clic sobre el borde superior de la ventana.

Para volver la ventana a su tamaño original, elija Restablecer del menú de control, presione <Ctrl><F5> o haga clic sobre el botón "Restaurar".

MINIMIZE

Para minimizar una ventana haga doble clic sobre el borde superior de la ventana, presione <Ctrl><F9> o elija Zoom hacia abajo del menú Ventana. Para volver la ventana a su tamaño original, repita los mismos pasos. Puede minimizar una ventana y "apartarla" en el ángulo inferior derecho del escritorio presionando la tecla MAYÚSCULAS cuando haga doble clic sobre el borde superior de la ventana.

SHADOW

De forma predeterminada, las ventanas no tienen sombra. Puede cambiar el estado predeterminado de las sombras con SET SHADOWS. Cualquier texto u objeto cubierto por la sombra será, no obstante, visible.

FILL <Exp>

Con esta cláusula se rellena con un carácter el fondo de la ventana. Si <Exp> contiene más de un carácter, únicamente se usará el primer carácter como carácter de relleno. También puede especificar un carácter de relleno con CHR().

COLOR <pares> / SCHEME <#>

De forma predeterminada, los colores de las ventanas se controlan mediante el esquema de color 1.

ACTIVATE WINDOW

Muestra y activa una o más ventanas definidas por el usuario.

ACTIVATE WINDOW [<nombre1> [,<nombre2> ...]] | ALL

[IN <ventana> | SCREEN]

[BOTTOM | TOP | SAME]

[NOSHOW]

Las ventanas activadas permanecen en el escritorio o en la ventana madre hasta que se emita DEAC WIND o HIDE WIND. Esto quita la ventana del escritorio o ventana madre pero no de la memoria. Las ventanas pueden volver a mostrarse con ACTI WIND o SHOW WIND.

Activar una ventana la convierte en la ventana frontal y dirige hacia ella toda la salida. La salida puede dirigirse a una sola ventana a la vez. Una ventana continúa siendo la ventana activa de salida hasta que se desactiva con DEAC WIND, se libera con RELE WIND o hasta que se activa otra ventana o el escritorio. Si no hay ventana de salida activa, la salida se dirigirá al escritorio.

Nota: Para asegurarse de que los resultados se dirigen a una ventana específica cuando se desactiva la ventana de salida, deberá activar explícitamente la ventana a la que desee enviar los resultados con ACTI WIND.

Todas las ventanas activadas se muestran hasta que se emite DEAC WIND o HIDE WIND para eliminarlas de la vista. La emisión de estos comandos elimina las ventanas de la vista, pero no de la memoria. Es posible volver a mostrar las ventanas emitiendo ACTI WIND o SHOW WIND.

Para eliminar las ventanas de la vista y de la memoria, use CLEAR WIND, RELE WIND o CLEAR ALL. Las ventanas que se eliminen de la memoria deberán redefinirse para volverlas a traer al escritorio.

ALL activará todas las ventanas definidas. La última ventana activada será la ventana de salida activa.

IN <ventana>

Cuando se incluye esta cláusula, la ventana se sitúa y activa en la ventana madre especificada con <nombre3>. La ventana activada se denomina ventana hija. Una ventana madre puede tener múltiples ventanas hijas. Una ventana hija activada dentro de una ventana madre no se puede desplazar fuera de ella. Si se desplaza la ventana madre, la ventana hija se moverá con ella.

Nota: La ventana madre debe estar visible para que cualquiera de sus ventanas hijas sean visibles.

IN SCREEN

Con esta cláusula es posible situar explícitamente una ventana en el escritorio, pero por omisión las ventanas se activan en el escritorio.

Es posible usar ACTI WIND para situar los accesorios de escritorio de FoxPRO y las ventanas del sistema en el escritorio o en una ventana madre. FoxPRO tiene los siguientes accesorios: Archivador, Calculadora, Calendario / Agenda, Caracteres especiales, Tabla ASCII y Puzzle. Ej: ACTI WIND calculadora

BOTTOM | TOP | SAME

Cuando activamos una ventana esta se convierte en la ventana frontal. Es posible incluir BOTTOM, TOP o SAME para especificar dónde se activan las ventanas respecto a otras ventanas previamente activadas.

Incluya NOSHOW para activar y dirigir la salida a una ventana sin mostrar la ventana.

Nota: Active una ventana usando NOSHOW para dibujar una ventana de entrada. Una vez dibujada la pantalla, emita SHOW WINDOW para que la ventana abierta "emerja". Esta técnica mostrará la ventana

rápidamente.

DEACTIVATE WINDOW

Desactiva ventanas definidas por el usuario o ventanas de sistema quitándolas de la pantalla pero no de la memoria.

DEAC WIND <nombre1>[,<nombre2> ...] | ALL

En el escritorio puede situarse más de una ventana a la vez, pero la salida se dirige solamente a la ventana activada más recientemente. Cuando está presente más de una ventana, la desactivación de la ventana de salida actual limpia el contenido de la ventana, quita la ventana de la pantalla y envía la salida subsecuente a la ventana activada con anterioridad. Si no hay ventana de salida, la salida se dirige al escritorio.

Si incluye ALL se desactivarán todas la ventanas activas.

HIDE WINDOW

Oculta una o mas ventana definida por el usuario o de sistema que esté activa. No es lo mismo ocultar una ventana que cerrarla. Cuando se oculta una ventana, permanece residente en memoria y activa. La salida puede enviarse a una ventana oculta, pero no podrá verla.

HIDE WIND <nombre1> [,<nombre2>] ... | ALL

[IN <ventana> | IN SCREEN]

[BOTTOM | TOP | SAME]

[SAVE]

IN <ventana>

Las ventanas pueden ocultarse en una ventana madre. Puede especificar la ventana madre en la que se ocultará una ventana hija incluyendo IN WIND.

IN SCREEN

También puede ocultar específicamente una ventana en el escritorio.

BOTTOM | TOP | SAME

Incluya BOTTOM, TOP o SAME para especificar donde se ocultan las ventanas, con respecto a otras ventanas.

ð BOTTOM sitúa la ventana detrás de las otras ventanas

ð TOP la sitúa delante de todas las demás ventanas.

ð SAME oculta la ventana sin afectar su posición actual.

ð SAVE muestra una imagen de una ventana en el escritorio o en una ventana definida por el usuario después de que la ventana se ha ocultado.

Normalmente, las ventanas se quitan del escritorio o de una ventana definida por el usuario después de ocultarlas. Use CLEAR para quitar una imagen de ventana del escritorio o de una ventana definida por el usuario. No puede guardar una imagen de la ventana de salida actual.

RELEASE WINDOWS [<lista de ventanas>]

Esto deja disponible la memoria que tuvieran asignada previamente. Las variables de memoria del sistema no pueden liberarse. Las ventanas quitadas de memoria deberán definirse de nuevo para volver a mostrarse.

CLEAR WINDOWS

Libera de la memoria todas las definiciones de ventanas definidas por el usuario y borra las ventanas del escritorio.

SAVE WINDOWS

Guarda las definiciones de ventanas en un archivo o campo memo para su uso posterior.

RESTORE WINDOWS

Repone en memoria las definiciones de ventanas previamente almacenadas en un archivo o campo memo.

En el ejemplo siguiente, se crea una ventana madre, PRUEBA1. Una ventana hija, PRUEBA2, se sitúa en la ventana madre.

```
CLEAR

−−−ventana madre −−−

DEFI WIND prueba1 FROM 1, 1 TO 20, 30 TITLE "Prueba1"

ACTI WIND prueba1

−−− ventana hija −−−

DEFI WIND prueba2 FROM 1, 1 TO 20, 20 TITLE "Prueba2" IN WIND prueba1

ACTI WIND prueba2

ACTIV SCRE

WAIT WIND 'Presione una tecla para borrar las ventanas.'

RELE WIND prueba1, prueba2

CLEAR
```

USO DE BROWSE EN UNA VENTANA

Esta es quizás una de las mejores maneras de manejar un browse ya que este toma las dimensiones de la ventana en la que fue metida:

Ejemplo 1: En este ejemplo se muestra los registros cuyos nombres empiecen con las letras de la A a la D inclusive, ordenados alfabeticamente, no se permitira modificar (NOMO) ni añadir (NOAP).

```
USE prueba ORDE nom

DEFI WIND W1 FORM 0,0 TO 10,40 TITLE `El titulo aqui' FOOT 'El pie aqui'

ACTI WIND W1

BROW IN WIND W1 FIEL nom,ape,fono KEY `A','D' NOMO NOAP

RELE WIND W1
```

Ejemplo 2: En este ejemplo la informacion es mostrada en un BROW que termina inmediatante (NOWA) pero no se borra (NOCL), queda visible.

```
USE prueba

DEFI WIND W2 FORM 5,20 TO 15,60

ACTI WIND W2

BROW IN WIND W2 FIEL nom,ape,fono NOWA NOCL
```

FUNCIONES DE VENTANA

WEXIST() = Dice si ventana estas definida o no

WONTOP() = Dice si ventana es la primera o no

WOUTPUT() = Dice si datos se dirigen o no a ventana

WVISIBLE() = Dice si esta visible o no una ventana

WCOLS() = # de columnas disponibles en ventana

WROWS() = # de filas disponibles en ventana

WLCOL() = Localización de columna mas a la izquierda

WLROW() = Localización de fila mas arriba

MWINDOW() = Da el nombre de ventana donde esta el Mouse

MDOWN() = Cuando haces clic, da un .T.

MCOL() = Columna donde esta el puntero del Mouse (0 a 79)

MROW() = Fila donde esta el puntero del Mouse (0 a 24)

ACCESORIOS DE ESCRITORIO

Es posible activar los accesorios disponibles desde el escritorio desde dentro de un programa con el comando:

ACTI WIND <nombre>

Accesorio	**Nombre**
Archivador	FILER
Calculadora	CALCULATOR
Calendario/Agenda	CALENDAR
Caracteres especiales	SPECIAL
Tabla ASCII	ASCII
Rompecabezas	PUZZLE

Nota: No sera necesario definir estas ventanas para usarlas.

VENTANAS DE SISTEMA

Tambien es posible activar ventanas del sistema por el mismo metodo para eso usar los nombre de la siguinte tabla.

Ventana	**Nombre**
Comandos	COMMAND
Depuracion	DEBUG
Seguimiento	TRACE
Presentacion	VIEW

Nota: No sera necesario definir estas ventanas para usarlas.

Capitulo 7

Practica

1. Crear un programa que muestre las opciones Siguiente, Anterior, Primero, Ultimo y luego la opcion <Salir>. Luego usando una tabla cualquiera, verificar su funcionamiento mirando el STATUS.

2. Basado en el programa anterior, añadir una opcion <Ver> junto al boton salir, de modo que al pulsarlo se muestren los campos del registro actual.

3. Crear un programa que presente en un primer POPUP la estructura de una tabla cualquiera (nombre de sus campos) y luego en un segundo POPUP todos los registros del campo elegido en el POPUP anterior.

4. Crear un programa que muestre las tablas (DBF) existentes en el disco, permita abrir una, luego muestre sus campos permitiento seleccionar uno de ellos, para luego mostra todos los registros del campos seleccionado.

5. Crear un programa que muestre las tablas (DBF) existentes en el disco, permita abrir una, luego muestre todas sus etiquetas de indice, para luego de seleccionar una de ellas me informe sobre el campos o expresión que la componen.

6. Crear un programa que pida el monto a pagar, el monto entregado, las monedas y billetes disponibles, para determinar el vuelto y el numero de monedas y billetes a entregar (segun la disponibilidad)

7. Crear un programa que sea capaz de convertir metros a pies, kilos a libras, litros a galones y viseversa, segun la eleccion del usuario.

8. Crear un programa que muestre en una primera ventana las regiones, luego de elegida una region se muestren los departamentos de dicha region, luego sus provincias y por ultimo sus distrutos.

9. Crear un programa que muestre en una primera ventana el codigo, nombre y sexo del alumnos, en una segunda ventana los codigos, nombres y nota de cursos en los que esta matriculado, en una tercera ventana el total de creditos aprobados y el promedio ponderado y en una cuarta ventana las opciones: Siguiente, Anterior, Primero, Ultimo, Buscar y Salir. la opcion buscar sera por nombre de alumno.

Bibliografía:

Programación en base a eventos

− Cesar A. Bustamante Gutierrez − UNI

FoxPRO 2.6 para DOS y Windows a su alcance

− José Javier García Badell − McGraw Hill

Programación basica con FoxPRO

− Ramón M. Chordá − Rama

Al dia en una hora en FoxPRO 2.6

− José Carlos Corrales − Anaya Multimedia

Guia practica FoxPRO 2.6

− Alejandro Dominguez − Anaya Multimedia

Diseño de programas: 200 algoritmos y un proyectos

− Julio Vasquez Paragulla − Editorial San Marcos

Microsoft FoxPRO for DOS and Windows, lenguage reference

− Microsoft Press

COMO APRENDER LPE ?

Se tomaran 4 practicas y 2 exámenes, que se calificaran de forma personal, entre 0 y 20 según lo transcrito en el papel y eso representa el 90% de la nota final del curso.

Al final del curso entregaras un trabajo libre personal, en el que entran en consideración además de tu esfuerzo, asistencia a clases, presentación, orden etc. Esto representa el 10% de la nota final del curso.

A continuación algunos consejos generales que te pueden ser útiles si tomas en cuenta.

No basta solo asistir a clases:

Es importante una buena base en algoritmos y no basta con asistir a clases para aprender a programar, porque no es un curso que se pueda aprender de memoria.

Antes de cada clase:

Es recomendable que leas en cualquier libro propuesto en la bibliografía o la separata del curso, el tema correspondiente a la clase. Los temas están detallados en el silabus del curso. Todo el material es disponible en la pagina Web de UAP.

Practicar fuera de horas de clase:

Según mi experiencia el alumno debe "practicar" Fox en una PC por lo menos unas 4 horas semanales, puede ser solo o en grupo, volviendo a resolver los ejercicios desarrollados en clase, las practicas dirigidas o calificadas (que las resuelve el profesor en la siguiente clase) y luego los ejercicios propuestos que se encuentra en las separatas del curso y en el archivo de ejemplos y ejercicios de FoxPRO, ambos disponibles en la pagina Web de la UAP. Consulta al profesor (dentro o fuera de clases) o asesor (en el horario publicado) del curso, sobre los ejercicios que no te salgan o tengas dudas.

Antes de la practica o examen:

Lee tu apuntes de clase y en la separata del curso, todos los capítulos hasta el tema de la prueba.

Que preguntas vendrán en la practica o examen ?

Todos son ejercicios prácticos que deberás resolver en tu PC y luego transcribir a un papel. Nunca pongo preguntas de teoría.

Algunos ejercicios son simples variantes de los ejercicios resueltos en clase o practicas dirigidas. Otros son los que están propuestos en la separata del curso y archivo de ejercicios de FoxPRO, ambos disponibles en la pagina Web. Resuélvelos !

En la practica o examen:

Ya que este no es un curso que se pueda aprender de memoria, recuerde que puede usar todo el material de consulta que quiera durante las prueba, solo se le pide que su trabajo sea personal. No esta permitido compartir dicho material.

Atentamente,

Ing Enrique Garrido−Lecca Risco

Universidad Alas Peruanas

CARRERA PROFESIONAL DE INGENIERIA DE SISTEMAS E INFORMATICA